Job Management Partner 1 Version 10

Job Management Partner 1/Advanced Shell Description, User's Guide, Reference, and Operator's Guide

3021-3-362-10(E)

Notices

■ Relevant program products

P-1M12-B1AL Job Management Partner 1/Advanced Shell 10-51 (for AIX V6.1, AIX V7.1)

P-8112-B1AL Job Management Partner 1/Advanced Shell 10-51 (for Red Hat Enterprise Linux 5 Advanced Platform (x86), Red Hat Enterprise Linux 5 (x86), Red Hat Enterprise Linux 5 Advanced Platform (AMD/Intel 64), Red Hat Enterprise Linux 5 (AMD/Intel 64), Red Hat Enterprise Linux Server 6 (32-bit x86), and Red Hat Enterprise Linux Server 6 (64-bit x86_64))

P-1J12-B1AL Job Management Partner 1/Advanced Shell 10-51 (for HP-UX 11i V3 (IPF))

P-9D12-B1AL Job Management Partner 1/Advanced Shell 10-51 (for Solaris 10 (SPARC) and Solaris 11 (SPARC))

P-2412-B1AL Job Management Partner 1/Advanced Shell 10-51 (for Windows Server 2012, Windows 8, Windows 7, Windows Server 2008, Windows Vista, Windows Server 2003, Windows Server 2003 (x64), and Windows XP)

P-2612-B2AL Job Management Partner 1/Advanced Shell - Developer 10-51 (for Windows Server 2012, Windows 8, Windows 7, Windows Server 2008, Windows Vista, Windows Server 2003, Windows Server 2003 (x64), and Windows XP)

■ Export restrictions

If you export this product, please check all restrictions (for example, Japan's Foreign Exchange and Foreign Trade Law, and USA export control laws and regulations), and carry out all required procedures.

If you require more information or clarification, please contact your Hitachi sales representative.

■ Trademarks

Active Directory is either a registered trademark or a trademark of Microsoft Corporation in the United States and/or other countries.

AIX is a trademark of International Business Machines Corporation in the United States, other countries, or both.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc., in the United States and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

HP-UX is a product name of Hewlett-Packard Development Company, L.P. in the U.S. and other countries.

Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.

Microsoft is either a registered trademark or a trademark of Microsoft Corporation in the United States and/or other countries.

Oracle and Java are registered trademarks of Oracle and/or its affiliates.

Red Hat is a trademark or a registered trademark of Red Hat Inc. in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Win32 is either a registered trademark or a trademark of Microsoft Corporation in the United States and/or other countries.

Windows is either a registered trademark or a trademark of Microsoft Corporation in the United States and/or other countries.

Windows Server is either a registered trademark or a trademark of Microsoft Corporation in the United States and/or other countries.

Windows Vista is either a registered trademark or a trademark of Microsoft Corporation in the United States and/or other countries.

Other company and product names mentioned in this document may be the trademarks of their respective owners.

The following program product contains some parts whose copyrights are reserved by Oracle and/or its affiliates: P-9D12-B1AL.

The following program product contains some parts whose copyrights are reserved by UNIX System Laboratories, Inc.: P-9D12-B1AL.

Other product and company names mentioned in this document may be the trademarks of their respective owners. Throughout this document Hitachi has attempted to distinguish trademarks from descriptive terms by writing the name with the capitalization used by the manufacturer, or by writing the name with initial capital letters. Hitachi cannot attest to the accuracy of this information. Use of a trademark in this document should not be regarded as affecting the validity of the trademark.

■ Microsoft product name abbreviations

This manual uses the following abbreviations for Microsoft product names:

Full name or meaning	Abbreviation	
Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2012 Standard (x64)	Windows Server 2012	Windows Server[#1]
Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2012 Datacenter (x64)		
Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2012 Standard		
Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2012 Datacenter		
Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2012 R2 Standard		
Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2012 R2 Datacenter		
Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2008 Enterprise	Windows Server 2008	
Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2008 Standard		
Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2008 R2 Datacenter		
Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2008 R2 Enterprise		
Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2008 R2 Standard		
Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2003, Enterprise Edition[#2]	Windows Server 2003[#3]	
Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2003, Standard Edition		
Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2003 R2, Enterprise Edition[#2]		
Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2003 R2, Standard Edition		
Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2003, Enterprise x64 Edition[#2]	Windows Server 2003 (x64)[#3]	
Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2003, Standard x64 Edition		
Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2003 R2, Enterprise x64 Edition[#2]		
Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2003 R2, Standard x64 Edition		
Windows$^{(R)}$ 8.1 Pro	Windows 8	Windows[#1]
Windows$^{(R)}$ 8.1 Enterprise		

Full name or meaning	Abbreviation	
Windows$^{(R)}$ 8.1 Pro 32-bit version	Windows 8	Windows[#1]
Windows$^{(R)}$ 8.1 Enterprise 32-bit version		
Windows$^{(R)}$ 8.1 Pro Pro 64-bit version		
Windows$^{(R)}$ 8.1 Enterprise Pro 64-bit version		
Windows$^{(R)}$ 8 Pro		
Windows$^{(R)}$ 8 Enterprise		
Windows$^{(R)}$ 8 Pro 32-bit version		
Windows$^{(R)}$ 8 Enterprise 32-bit version		
Windows$^{(R)}$ 8 Pro 64-bit version		
Windows$^{(R)}$ 8 Enterprise 64-bit version		
Microsoft$^{(R)}$ Windows$^{(R)}$ 7 Enterprise	Windows 7	
Microsoft$^{(R)}$ Windows$^{(R)}$ 7 Professional		
Microsoft$^{(R)}$ Windows$^{(R)}$ 7 Ultimate		
Microsoft$^{(R)}$ Windows Vista$^{(R)}$ Business	Windows Vista	
Microsoft$^{(R)}$ Windows Vista$^{(R)}$ Enterprise		
Microsoft$^{(R)}$ Windows Vista$^{(R)}$ Ultimate		
Microsoft$^{(R)}$ Windows Vista$^{(R)}$ Business 64-bit version		
Microsoft$^{(R)}$ Windows Vista$^{(R)}$ Enterprise 64-bit version		
Microsoft$^{(R)}$ Windows Vista$^{(R)}$ Ultimate 64-bit version		
Microsoft$^{(R)}$ Windows$^{(R)}$ XP Professional Operating System	Windows XP	

#1: Windows Server and Windows are sometimes referred to collectively as *Windows*.

#2: These products are sometimes referred to collectively as *Windows Server 2003 Enterprise Edition*.

#3: These products are sometimes referred to collectively as *Windows Server 2003*.

■ Restrictions

Information in this document is subject to change without notice and does not represent a commitment on the part of Hitachi. The software described in this manual is furnished according to a license agreement with Hitachi. The license agreement contains all of the terms and conditions governing your use of the software and documentation, including all warranty rights, limitations of liability, and disclaimers of warranty.

Material contained in this document may describe Hitachi products not available or features not available in your country.

No part of this material may be reproduced in any form or by any means without permission in writing from the publisher.

■ Issued

February 2015: 3021-3-362-10(E)

■ Copyright

Summary of amendments

The following table lists changes in this manual (3021-3-362-10(E)) and product changes related to this manual.

Changes	Location
(Windows only) The `who` command (script format) can now be used in UNIX-compatible commands. In addition, the names of the sample script files for the `chmod` and `su` commands were changed.	*2.1.1(1)*, *2.1.1(4)*, *2.6.6(2)*, *8.2.2(2)*, *8.5*
The following UNIX-compatible commands are now supported: • `dirname` • `expand` • `getopt` • `stat` Options were also added to the following UNIX-compatible commands: • `cut` • `date` • `diff` • `expr` • `ls`	*2.1.1(4)*, *2.1.2(3)*, *2.5*, *8.2.2(1)*, *cut command*, *date command*, *diff command*, *dirname command*, *expand command*, *expr command*, *getopt command*, *ls command*, *stat command*
A setting for suppressing creation of spool job directories during job execution is now supported (spool job creation suppression functionality). As a result, the following environment setting parameter was added: • `SPOOLJOB_CREATE` parameter	*2.2.3(2)*, *2.6.8(1)*, *7.2.1(2)*, *SPOOLJOB_CREATE parameter*, *adshfile command*, *#-adsh_spoolfile command*
UTF-8 character encoding is now supported in AIX and HP-UX.	*2.2.4*, *2.8.4(1)*
The following UNIX-compatible commands now support options in the long-option format: • `cut` • `date` • `diff` • `ls`	*2.5*, *8.1.2*, *cut command*, *date command*, *diff command*, *ls command*, *Appendix C*
A job definition script that does not satisfy the default definition for the `CHILDJOB_SHEBANG` parameter can now be run as a child job. In connection with this implementation, the following changes were made: • A message concerning child jobs that can be suppressed by the `JOBLOG_SUPPRESS_MSG` parameter was added. • The sample script file execution method was changed from the function format to the child job format.	*2.6.5*, *2.6.6(2)*, *3.1.2(1)*, *3.2.2*, *3.2.3(1)*, *5.1.11(3)*, *5.1.11(4)*, *CHILDJOB_EXT parameter*, *CHILDJOB_PGM parameter*, *CHILDJOB_SHEBANG parameter*, *JOBLOG_SUPPRESS_MSG parameter*, *Appendix C*
An output mode that minimizes the types of output messages (minimum output mode) was added. As a result, operands were added to the following environment setting parameters: • `OUTPUT_MODE_CHILD` parameter • `OUTPUT_MODE_ROOT` parameter	*2.6.8(1)*, *2.6.8(2)*, *3.3.1*, *3.3.2*, *3.3.4*, *3.4*, *3.4.1*, *3.4.4*, *JOBEXECLOG_PRINT parameter*, *OUTPUT_MODE_CHILD parameter*, *OUTPUT_MODE_ROOT parameter*, *OUTPUT_STDOUT parameter*, *adshexec command*, *adshscripttool command*, *11.2*

Changes	Location
Options were also added to the following commands: • adshexec command • adshscripttool command	2.6.8(1), 2.6.8(2), 3.3.1, 3.3.2, 3.3.4, 3.4, 3.4.1, 3.4.4, JOBEXECLOG_PRINT parameter, OUTPUT_MODE_CHILD parameter, OUTPUT_MODE_ROOT parameter, OUTPUT_STDOUT parameter, adshexec command, adshscripttool command, 11.2
The maximum number of subscripts for an array was increased from 1,023 to 65,535.	2.6.15(1), 5.1.3, set command, adshread command, 11.3
A shell can now be coded directly in the adshexec command that executes a job.	3.2.2, 3.2.4, 5.5.3(1), adshexec command
User-specific postprocessing can now be performed when a job is terminated forcibly.	1.5, 2.6.21, 3.10.2, 4.3.2, 4.4.6(4), 4.4.6(5), 5.5.3, 6.1.5, 7.2.1(2), TRAP_ACTION_SIGTERM parameter (in 7.3), 9.2.1, trap command (in 9.3)
(Windows only) Lowercase letters can now be used in environment variable names. As a result, the following environment setting parameter was added: • VAR_ENV_NAME_LOWERCASE parameter	5.1.2(1), 5.5, 7.2.1(2), VAR_ENV_NAME_LOWERCASE parameter, export command, typeset command, #-adsh_file command, #-adsh_file_temp command, #-adsh_spoolfile command
Substring expansion is now supported in variable substitution.	5.1.2(3), 5.1.6(6), 11.3
A method for specifying batch creation of multiple array elements was added.	5.1.3
Jobs can now store information about executing functions as an array (array of functions information). As a result, the following environment setting parameter was added: • VAR_SHELL_FUNCINFO parameter	5.5.1, 5.5.3, 7.2.1(2), VAR_SHELL_FUNCINFO parameter, #-adsh_step_start command, #-adsh_step_error command, #-adsh_step_end command, 11.3, Appendix C
In the formats used to perform variable substitution to the length of the character string constituting the value of the variable or to the number of array elements, the units can now be specified for the length of the character string to be replaced. As a result, the following environment setting parameter was added: • VAR_SHELL_GETLENGTH parameter	7.2.1(2), VAR_SHELL_GETLENGTH parameter
Changes were made to what is output as a job definition script file name when job contents are specified directly in the -r option in the adshscripttool command.	adshscripttool command
The following messages were added: KNAX0235-E, KNAX0474-E, KNAX1880-E, KNAX6058-E, KNAX6072-E, KNAX6097-E, KNAX6385-E, KNAX6718-I, KNAX7073-I, KNAX7128-E	11.2, 11.3
Changes were made to the explanations of the following messages: KNAX0411-E, KNAX0441-E, KNAX0445-E, KNAX0449-E, KNAX1873-E, KNAX5407-E, KNAX6007-E, KNAX6008-E, KNAX6022-E, KNAX6226-E, KNAX6241-E, KNAX6382-I, KNAX6997-E, KNAX6710-I, KNAX7450-I, KNAX7451-I, KNAX7901-I, KNAX7902-I	11.3
Changes were made to the text for the following messages: KNAX9000-E, KNAX9001-E	11.3
The description about a trap action was added to Glossary.	Appendix C

In addition to the above changes, minor editorial corrections were made.

Preface

This manual explains how to use Job Management Partner 1/Advanced Shell to create and execute job definition scripts for batch jobs.

In this manual, Job Management Partner 1 is abbreviated as *JP1*.

JP1/Advanced Shell consists of the following products:

- JP1/Advanced Shell (script execution base for batch jobs)
- JP1/Advanced Shell - Developer (script development base for batch jobs)

This manual uses the terms *execution environment* and *development environment* to distinguish between the JP1/Advanced Shell environment and the JP1/Advanced Shell - Developer environment, respectively.

■ Intended readers

This manual is intended for individuals interested in using JP1/Advanced Shell to develop, execute, or manage batch jobs. Readers of this manual must be familiar with the following:

- Windows and UNIX
- JP1/AJS
- JP1/Base
- JP1/IM

■ Organization of this manual

This manual is organized into the following parts, chapters, and appendixes:

PART 1: Overview

1. Overview of JP1/Advanced Shell

JP1/Advanced Shell is a product for creating and executing job definition scripts for batch jobs. Chapter 1 describes the purpose of JP1/Advanced Shell, provides an example of its application to a business operation, explains the overall system configuration and general procedures, and provides an overview of JP1/Advanced Shell's operation and functionality in a cluster system.

PART 2: Setup

2. Preparations for Using JP1/Advanced Shell

Chapter 2 discusses the conditions and requirements for using JP1/Advanced Shell, including the program installation directory, the main programs, prerequisites, installation, environment information settings, custom job registration, user-reply functionality settings, and environment information settings for cluster operation.

PART 3: Operation

3. Executing Batch Jobs

Chapter 3 explains how to execute batch jobs and the batch job processing in JP1/Advanced Shell (execution environment).

4. Using JP1/Advanced Shell - Developer (Windows Only)

Chapter 4 explains how to employ JP1/Advanced Shell - Developer so that you can use JP1/Advanced Shell Editor to develop job description scripts in a Windows environment. The chapter also explains how to use the editor to debug job definition script files.

5. Creating Job Definition Scripts

Chapter 5 explains the syntax for job definition scripts.

6. Debugging Job Definition Scripts

Chapter 6 describes the debugger functions of JP1/Advanced Shell.

PART 4: Reference

7. Parameters Specified in the Environment Files

Chapter 7 provides details about the description format used for parameters and commands. You define in environment files information such as return codes, coverage, system execution logs, and directory paths. Export parameters are used to define environment variables. Conditional parameters are used to apply desired environment setting parameters or export parameters specifically to the physical host or specifically to a particular logical host.

8. Commands Used During Operations

Chapter 8 describes the syntax and details of the commands used for operations.

9. Job Definition Script Commands and Control Statements

Chapter 9 describes in detail the description formats for the standard shell commands, extended shell commands, extended script commands, script control statements, and reserved script commands used in job definition scripts.

PART 5: Troubleshooting

10. Troubleshooting

Chapter 10 describes troubleshooting, including how to respond when problems occur, the types of log information, the troubleshooting information that needs to be collected, and how to collect it.

11. Messages

Chapter 11 lists the messages output by JP1/Advanced Shell and provides detailed information about errors that might occur.

A. Coverage Information That Is Acquired

Appendix A describes the coverage information that JP1/Advanced Shell acquires.

B. Reference Material for This Manual

Appendix B provides reference information such as a list of related manuals and an explanation of the abbreviations used in this manual.

C. Glossary

Appendix C is a glossary that explains the terms used in this manual.

■ Conventions: Administrators permissions

This manual uses the term *Administrators permissions* to refer to the Administrators permissions for a local PC. The actions of a user who has Administrators permissions for a local PC are no different from those for a local user or domain user, or for a user working in an Active Directory environment.

■ Conventions: Fonts and symbols

The following table explains the text formatting conventions used in this manual:

Text formatting	Convention
Bold	Bold characters indicate text in a window, other than the window title. Such text includes menus, menu options, buttons, radio box options, or explanatory labels. For example: • From the **File** menu, choose **Open**. • Click the **Cancel** button. • In the **Enter name** entry box, type your name.
Italic	Italic characters indicate a placeholder for some actual text to be provided by the user or system. For example: • Write the command as follows: `copy` *source-file target-file* • The following message appears: `A file was not found. (file =` *file-name*`)` Italic characters are also used for emphasis. For example: • Do *not* delete the configuration file.
`Monospace`	Monospace characters indicate text that the user enters without change, or text (such as messages) output by the system. For example: • At the prompt, enter `dir`. • Use the `send` command to send mail. • The following message is displayed: `The password is incorrect.`
<u>`underline`</u>	The underline indicates the default value among two or more values enclosed in selection symbols.

The following table explains the symbols used by this manual in syntax explanations:

Symbol	Convention
|	A vertical bar separates multiple items, and has the meaning of OR. For example: `A|B|C` means `A`, or `B`, or `C`.
{ }	Curly brackets indicate that only one of the enclosed items is to be selected. For example: `{A|B|C}` means only one of `A`, or `B`, or `C`.
[ ]	Square brackets indicate that the enclosed item or items are optional. A vertical bar is used to delimit multiple items. For example: `[A]` means that you can specify `A` or nothing. `[B|C]` means that you can specify `B`, or `C`, or nothing.
< >	Single angle brackets enclose the syntax element that must be used to specify an item.

Symbol	Convention
+	The plus sign indicates that the immediately preceding item can be specified multiple times. It is also used to indicate that the items before and after it are specified together. **Examples:** {A|B}+ Indicates that A or B can be specified multiple times in any order. CR+LF Indicates that the carriage return character (CR) and the linefeed character (LF) are specified together.
*	The asterisk indicates that the immediately preceding item can be omitted or that it can be specified one or more times. **Example:** {A|B}* Indicates that A or B can be specified one or more times in any order or that A and B can both be omitted.
~	A swung dash indicates that the syntax element enclosed by the single angle brackets (< >), double angle brackets (<< >>), or double parentheses ((( ))) that follow must be used to specify the item that precedes the swung dash.
<< >>	Double angle brackets enclose the default value for an item.
(( ))	Double parentheses enclose the permissible range of values that can be specified.
...	An ellipsis indicates that the immediately preceding item can be repeated as many times as necessary. For example: A, B, B, ... means that, after you specify A, B, you can specify B as many times as necessary.
Δ	Denotes a single-byte space. Δ_0: Denotes zero or more spaces (spaces can be omitted). Δ_1: Denotes one or more spaces (at least one space is required).

The following table explains the syntax elements used in this manual:

Syntax element	Characters that can be specified
<numeric characters>	0|1|2|3|4|5|6|7|8|9
<uppercase alphabetic characters>	A|B|C|...|Z
<lowercase alphabetic characters>	a|b|c|...|z
<alphabetic characters>	<uppercase alphabetic characters>|<lowercase alphabetic characters>
<special characters>	,|.|/|'|(|)|*|&|+|-|=| Δ (space)|\
<octal>	<0|1|2|3|4|5|6|7> +
<decimal>	<numeric characters> +
<hexadecimal>	0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F
<integer>	A series of signed or unsigned numeric characters
<unsigned integer>	<numeric characters> +
ss<symbolic name>	{<alphabetic characters>|<numeric characters>|@|#|_ (underscore)} + Used in: Job names

Syntax element	Characters that can be specified
<environment variable name>	{<alphabetic characters> | _ (underscore)}{<alphabetic characters> | _ (underscore)} | <numeric characters)} * Used in: Environment variable file definition names, environment variable names, and extended script commands
<path name>	A character string that conforms to the path naming conventions of UNIX or Windows
<command name>	A path name consisting of permitted characters other than the path separator
<logical host name>	{<alphabetic characters> | <numeric characters> | - (hyphen)} +
<any character string>	A string of characters consisting of any combination of alphabetic characters. Note the following: • JP1/Advanced Shell does not check the character type. • Character strings with a meaning appropriate for the location where they are used must be specified. • We recommend that you use characters in the range permitted for the symbolic name in which they are used.
<ASCII character string>	A character string consisting exclusively of characters in the ASCII character set, other than ASCII control characters (a character string in the range from `0x20` to `0x7E`)

■ Conventions: common application data folder

This manual uses the term *common application data folder* to refer to one of the following folders, depending on the Windows version being used:

Windows Server 2003, Windows Server 2003 (x64), or Windows XP:

system drive`:\Documents and Settings\All Users\Application Data`

Windows Server 2012, Windows 8, Windows 7, Windows Server 2008, or Windows Vista:

system drive`:\ProgramData`

■ Conventions: Shared documents folder

This manual uses the term *shared documents folder* to refer to one of the following folders, depending on the Windows version being used:

Windows Server 2003, Windows Server 2003 (x64), or Windows XP:

system drive`:\Documents and Settings\All Users\Documents`

Windows Server 2012, Windows 8, Windows 7, Windows Server 2008, or Windows Vista:

system drive`:\Users\Public\Documents`

■ Conventions: The JP1/Advanced Shell installation folder in Windows

In this manual, *installation folder* refers to the folder in which JP1/Advanced Shell has been installed, unless otherwise stated. The following shows the installation folder when the product is installed with the default settings.

x86 environment:

system-drive`:\Program Files\Hitachi\JP1AS` x64 environment:

system-drive`:\Program Files(x86)\Hitachi\JP1AS`

■ Conventions: Windows menu names used in the manual

The Windows menu names used in this manual assume that you are using one of the following OSs:

Windows 7, Windows Server 2008, Windows Vista, Windows Server 2003, or Windows XP

In Windows 8 and Windows Server 2012, no **Start** menu is displayed. Instead, you must use the **Start** window that can be opened from the lower left corner of the window.

Contents

Part 1: Overview

Part 2: Setup

Part 3: Operation

Part 4: Reference

Part 5: Troubleshooting

1 Overview of JP1/Advanced Shell

JP1/Advanced Shell is a product for creating and executing job definition scripts for batch jobs. This chapter describes the purposes of JP1/Advanced Shell, provides an example of its application to a business operation, explains the overall system configuration and general procedures, and provides an overview of the product's functions.

1.1 Purposes of JP1/Advanced Shell

JP1/Advanced Shell is a product for improving development productivity and the operational efficiency of batch applications. It enables you to efficiently create and execute job definition scripts (*shell scripts*) for batch jobs.

JP1/Advanced Shell has the features described below.

1.1.1 Inheriting assets between the OSs of batch applications

- Using existing assets

  You can use shell scripts created in a UNIX environment to develop job definition scripts in a Windows environment.

  Because the job definition scripts used in JP1/Advanced Shell employ language specifications that have standard shell compatibility, it is easy to learn the language and migrate from existing shell scripts.

- Cross-platform support

  Cross-platform means applicability to multiple OS bases. This feature enables you to use cross-platform functions.

  - You can execute job definition scripts developed in a Windows environment in both Windows and UNIX environments.
  - You can use UNIX-compatible commands in both Windows and UNIX environments.

1.1.2 Expediting the configuration of batch applications

- Controlling job execution

  JP1/Advanced Shell extends job definition scripts so that you can automate and concisely code processes that are used repetitiously in batch applications.

  You can reduce the volume of coding in job definition scripts and improve readability and maintainability of job definition scripts by doing the following:

  - Specifying job step execution conditions
  - Using variables that are valid in job steps
  - Outputting error messages and setting return codes when batch jobs terminate with errors
  - In the event a batch job terminates with an error, automatically terminating child processes forcibly and deleting temporary files used by the batch job

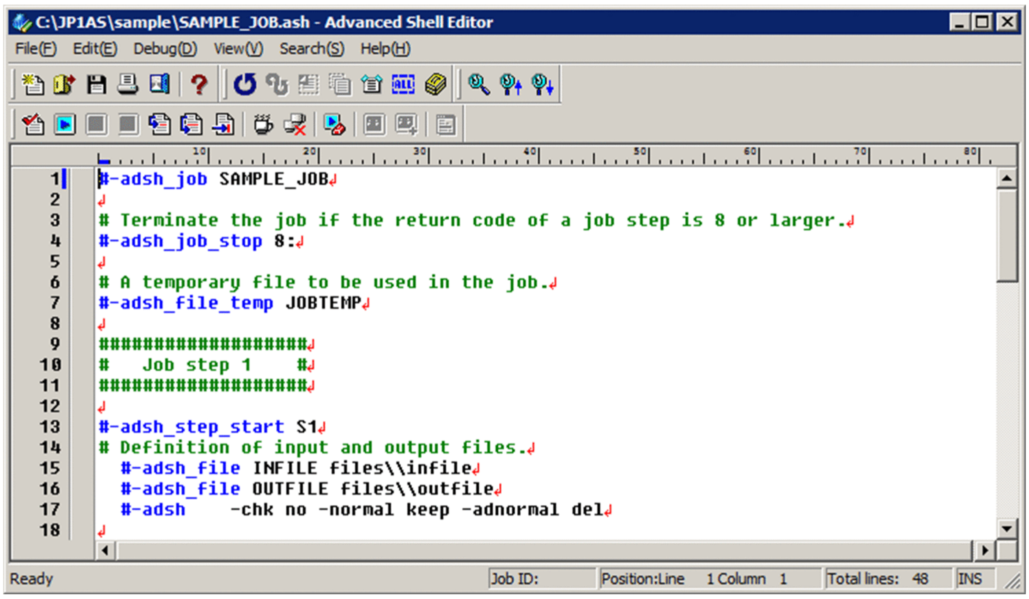
- Using an editor to develop job definition scripts (development environment)

  In the development environment, you can use the JP1/Advanced Shell Editor (a dedicated editor with debugging functions) of the Graphical User Interface (GUI) to develop and debug job definition scripts.

  - You can execute job definition scripts in job steps, and set breakpoints.
  - You can accumulate coverage information for job definition scripts.

  The following figure shows the JP1/Advanced Shell Editor window.

Figure 1–1: JP1/Advanced Shell Editor window



```
#-adsh_job SAMPLE_JOB

# Terminate the job if the return code of a job step is 8 or larger.
#-adsh_job_stop 8:

# A temporary file to be used in the job.
#-adsh_file_temp JOBTEMP

####################
#   Job step 1     #
####################

#-adsh_step_start S1
# Definition of input and output files.
  #-adsh_file INFILE files\\infile
  #-adsh_file OUTFILE files\\outfile
  #-adsh    -chk no -normal keep -adnormal del

```

- Efficient file allocation and postprocessing

  You can automate and concisely code processes, such as checking for regular files, and allocating and deleting temporary files.

  - You can automatically allocate temporary files during batch job execution and delete them once the batch job has terminated.
  - You can check for regular files during batch job execution and perform appropriate postprocessing on files depending on job step or job processing results.

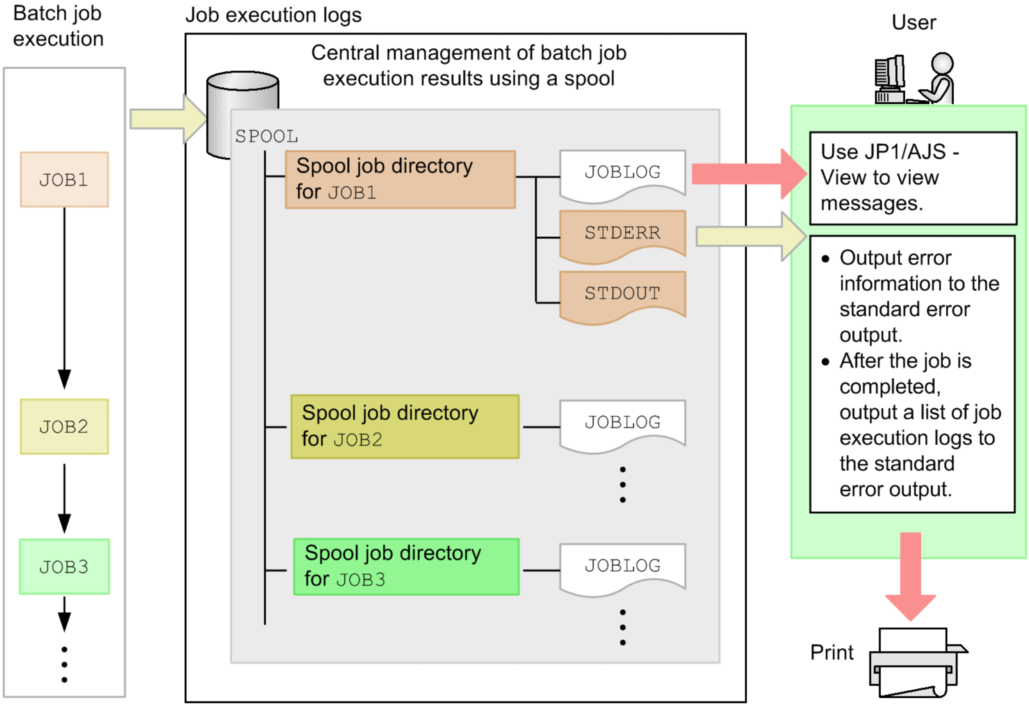
1.1.3 Improving serviceability and maintainability by central management of batch job execution results

Maintainability of batch applications can be improved by automatically outputting job execution logs in the event of an error and managing such logs centrally.

In conventional open systems, management of batch job execution results is complicated because the results are not stored at one central location. JP1/Advanced Shell enables you to collect batch job execution results on a spool as job execution logs, and to manage them centrally. By using JP1/AJS - View, you can execute batch jobs on a periodic basis and reference the results by automatically executing job definition scripts.

Each job's execution results are output to a spool job directory under the spool directory. The following figure illustrates central management of batch job execution results.

Figure 1–2: Central management of batch job execution results


Batch job execution
Job execution logs
Central management of batch job execution results using a spool
User
JOB1
JOB2
JOB3
SPOOL
Spool job directory for JOB1
Spool job directory for JOB2
Spool job directory for JOB3
JOBLOG
STDERR
STDOUT
JOBLOG
JOBLOG
Use JP1/AJS - View to view messages.
• Output error information to the standard error output.
• After the job is completed, output a list of job execution logs to the standard error output.
Print

For details about the output contents of the job execution logs, see *3.4 Job execution log*.

Lastly, troubleshooting support enables you to handle problems through collection of various types of data, including job execution logs, system execution logs, and trace logs. For details, see *10. Troubleshooting*.

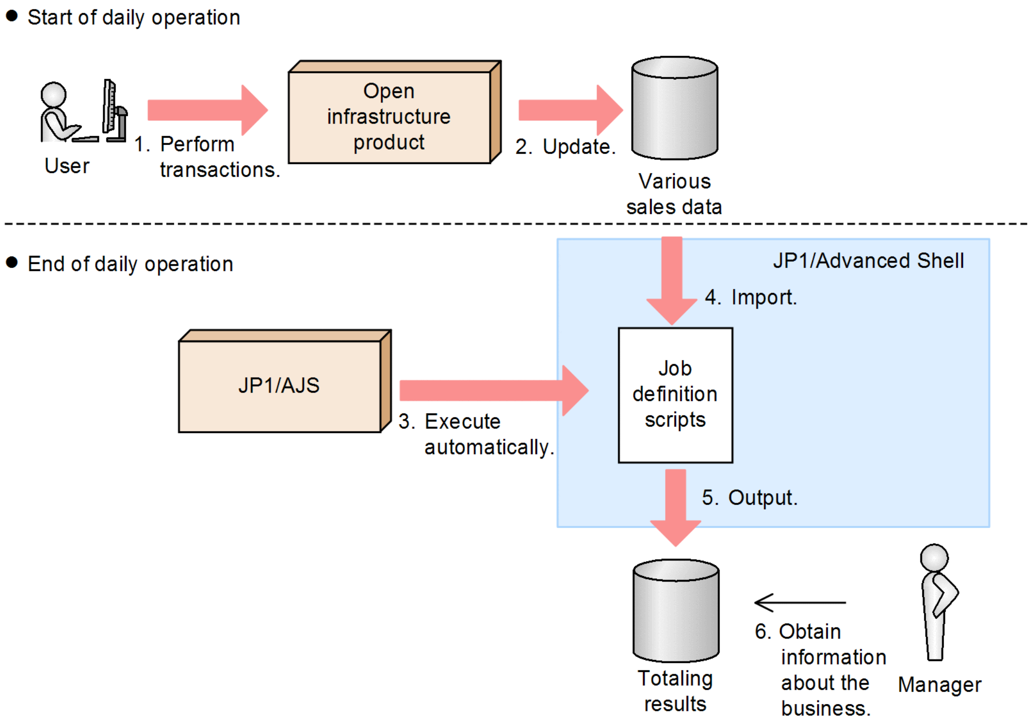
1.2 Example of application to a business operation

You can apply JP1/Advanced Shell to the following type of business operation.

In the case of an operation that involves many transactions in an online system during the daytime and totaling of the transactions at night, you can develop and execute batch jobs that obtain totals, including sales figures, number of products sold, and inventory updates. You can also develop and execute batch jobs for obtaining rolling totals, such as daily, monthly, and term-end processing, as well as batch jobs that have specific purposes and that are used for special occasions.

The following figure shows an example of JP1/Advanced Shell operation (for obtaining daily operation totals)

Figure 1–3: Example of JP1/Advanced Shell operation (obtaining daily operation totals)



To run JP1/Advanced Shell:

1. Start daily operation and perform transactions involving products.
2. The open infrastructure product updates the various sales data.
3. Daily operation ends and JP1/AJS issues instructions to execute job definition scripts automatically at specified times.
4. JP1/Advanced Shell executes job definition scripts to process the various sales data.
5. JP1/Advanced Shell outputs the execution results of the job definition scripts.
6. The manager can obtain information, including totals and changes in product sales, based on the execution results.

1.3 General procedures

JP1/Advanced Shell consists of the execution environment (JP1/Advanced Shell) and the development environment (JP1/Advanced Shell - Developer). Job definition scripts created in the development environment are executed in the execution environment.

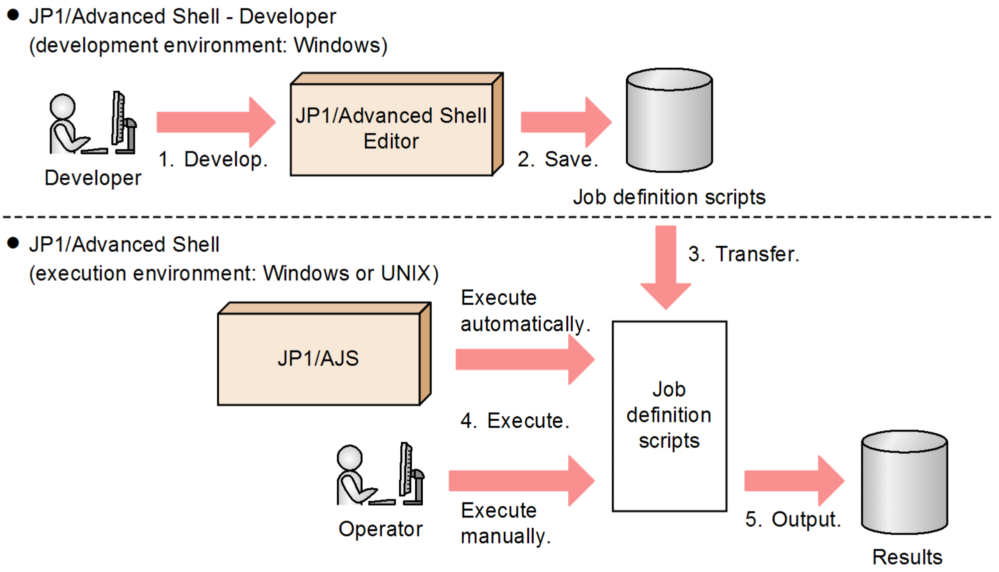
The users of JP1/Advanced Shell are classified as system administrators and general users, based on their system permissions. The following table explains the roles of these two classes of users.

Table 1–1: Classification of JP1/Advanced Shell users

JP1/Advanced Shell user		Role
System administrator		This user is responsible for system operations. The superuser permission must have been assigned to this user beforehand. The system administrator manages an environment that can run JP1/Advanced Shell and registers the general users who will use JP1/Advanced Shell.
General user	Developer	This user's responsibilities include creation and debugging of job definition scripts.
	Operator	This user defines and runs JP1/Advanced Shell, checks the execution results, and handles JP1/Advanced Shell execution errors, if any. For details about the operator's tasks when JP1/AJS is used, see *3.1.1 Operator's tasks in JP1/AJS jobs*.

The following figure shows the overall system configuration of JP1/Advanced Shell.

Figure 1–4: Overall system configuration of JP1/Advanced Shell



To use a JP1/Advanced Shell system:

1. The developer uses JP1/Advanced Shell Editor to develop job definition scripts; this is always done in the Windows development environment.

2. The job definition scripts from JP1/Advanced Shell Editor are saved.

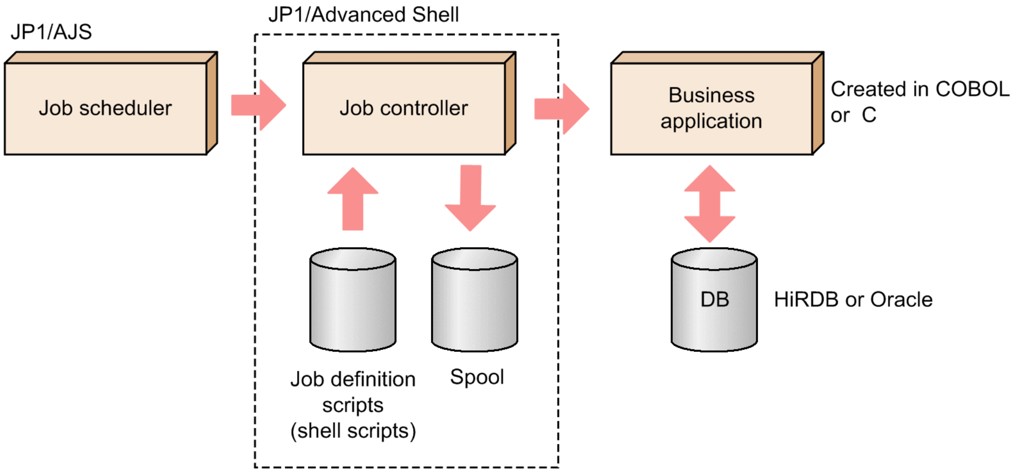
3. The job definition scripts are transferred to the execution environment.

4. In the execution environment, the operator uses the following methods to send instructions to execute the job definition scripts:
    - Automatic execution using JP1/AJS
    - Manual execution from the command prompt and UNIX shell
5. The job definition script execution results output by JP1/Advanced Shell are checked.

1.3.1 Procedure for executing batch jobs automatically (working with JP1/AJS)

When you use JP1/Advanced Shell to run batch jobs, you can call the execution environment from the job scheduler's JP1/AJS so that the batch jobs can execute automatically. JP1/Advanced Shell provides job controller functions that manage execution of a user's business applications. The following figure shows the positioning of JP1/Advanced Shell for business applications.

Figure 1–5: Positioning of JP1/Advanced Shell for business applications

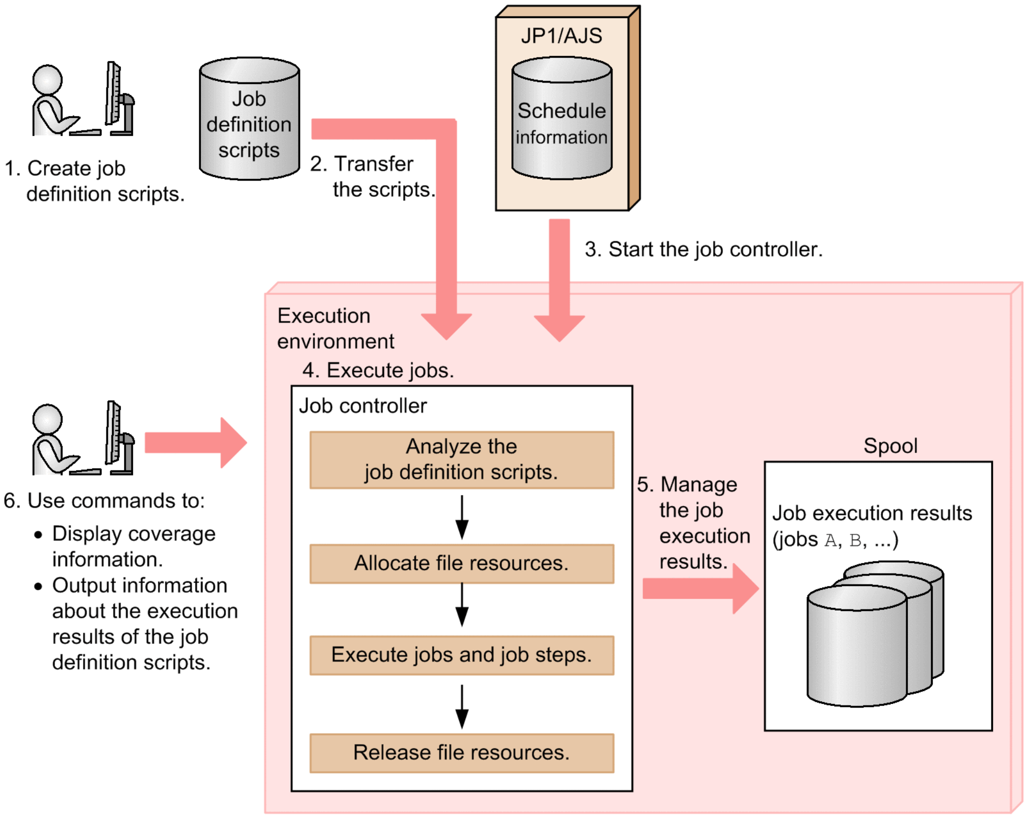


When you link JP1/Advanced Shell to JP1/AJS, you can register batch job execution schedules for executing batch jobs.

A job definition script containing job definitions is analyzed by the job controller. The job controller controls execution and termination of batch jobs by allocating and releasing input and output devices and various system resources. JP1/Advanced Shell achieves central management by executing this job definition script and collecting the execution results on the spool.

The figure below shows the JP1/Advanced Shell operation procedure. In the figure, the processing performed by JP1/AJS is identified by the number 3, and the processing performed by JP1/Advanced Shell is identified by the numbers 4 through 6.

Figure 1–6: JP1/Advanced Shell operation procedure (working with JP1/AJS)


Job definition scripts
1. Create job definition scripts.
2. Transfer the scripts.
JP1/AJS
Schedule information
3. Start the job controller.
Execution environment
4. Execute jobs.
Job controller
Analyze the job definition scripts.
Allocate file resources.
Execute jobs and job steps.
Release file resources.
5. Manage the job execution results.
Spool
Job execution results (jobs A, B, ...)
6. Use commands to:
• Display coverage information.
• Output information about the execution results of the job definition scripts.

To run JP1/Advanced Shell:

1. Create job definition scripts.
2. Transfer the job definition scripts to JP1/Advanced Shell's execution environment.
3. The job controller starts according to a schedule registered in JP1/AJS.
4. The job controller executes batch jobs using the procedure shown below according to the contents of a job definition script created in step 1.
   Analyze the job definition script ➜ Allocate file resources ➜ Execute jobs and job steps ➜ Release file resources
5. Collect the batch job execution results on the spool for central management.
6. If necessary, use commands to display coverage information and to output information about the execution results of the job definition scripts.

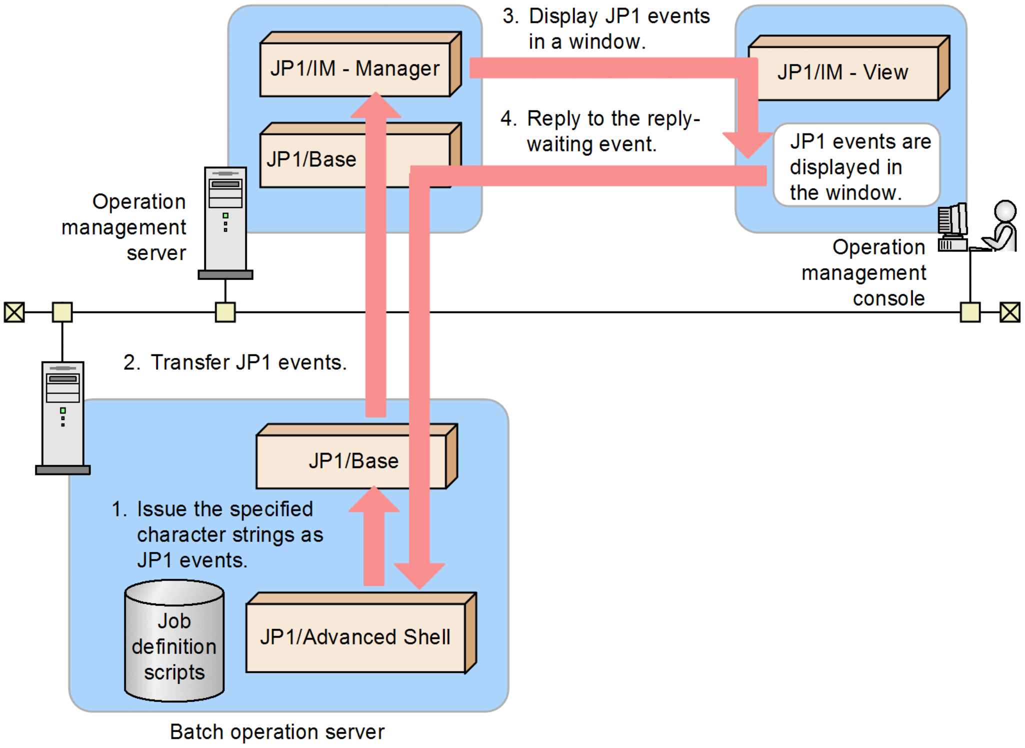
1.3.2 Procedure for using the user-reply functionality

The user-reply functionality enables the following operations to be performed from job definition scripts:

- Notifying the operator of batch job information
- Enabling the operator to reply to job definition scripts

Linked with JP1/IM, the user-reply functionality issues specified character strings as JP1 events. The following figure shows the procedure for using the user-reply functionality.

Figure 1–7: Procedure for using the user-reply functionality


3. Display JP1 events in a window.
JP1/IM - Manager
JP1/IM - View
4. Reply to the reply-waiting event.
JP1/Base
JP1 events are displayed in the window.
Operation management server
Operation management console
2. Transfer JP1 events.
JP1/Base
1. Issue the specified character strings as JP1 events.
Job definition scripts
JP1/Advanced Shell
Batch operation server

To use the user-reply functionality:

1. If a command in which a character string is specified is executed by means of a job definition script, the specified character string is issued as a JP1 event.

2. The issued JP1 event is transferred by JP1/Base to a specified operation management server.

3. The specified character string is displayed in the JP1/IM - View window.

4. If it's a reply-waiting event, the operator can enter a reply.
   The reply entered from the operator is stored in a shell variable specified in the job definition script.

For details, see *3.7 Using the user-reply functionality*.

1.4 Overview of operation in a cluster system

A cluster system consists of multiple server systems that are configured in such a manner that if a failure occurs on one of the servers, applications can continue on another server. In a cluster system, hosts are classified as follows:

- Server running applications (*active server*)
- Server on standby that can inherit applications if the active server fails (*standby server*)

If a failure occurs on the active server that is running applications, the applications can be inherited to the standby server. The function for inheriting applications in the event of a failure is called *system switchover*. A logical server that is the unit of failover during system switchover is called a *logical host*.

In a cluster system, the applications must be run in a logical host environment so that they can continue their processing even after system switchover. You can run applications that are to be run on a logical host on any server, regardless of the physical server being used.

A logical host consists of the components explained below. The applications that are run as daemons or services store data on the shared disk and use a logical IP address to communicate.

Table 1–2: Components of a logical host

Component of logical host	Description of component
Daemon or service	Daemons and services are the applications that are run in a cluster system, such as JP1/AJS and JP1/Advanced Shell. If a failure occurs on the active server's logical host, the daemons or services with the same names are started on the standby server's logical host.
Shared disk	This is a disk unit that is connected to both the active server and the standby server. If this disk stores the information that is to be inherited during system switchover (such as definition information and execution status), the standby server inherits the connection to the shared disk in the event of a failure on the active server's logical host.
Logical IP address	This is an IP address allocated while a logical host is running. If a failure occurs on the active server, the standby server inherits the same logical IP address allocation. This enables the client to use the same IP address as if the same server is always running.

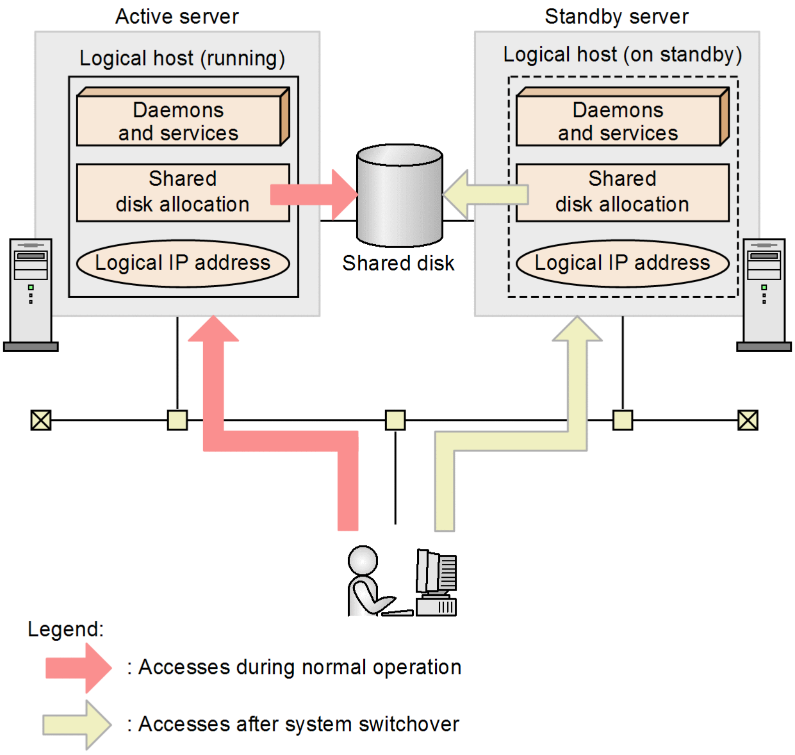
Important note

In this manual, a logical server that is the unit of failover during system switchover is called a *logical host*. However, some cluster software products and applications use different terms, such as *group* or *package*. Check the appropriate terminology in your system by referencing your cluster software's documentation.

A logical server that is the unit of failover during system switchover is called a *logical host*, while the physical server is called a *physical host*.

The following figure shows accesses during normal operation and after system switchover.

Figure 1–8: Accesses during normal operation and after system switchover



The following explains the figure.

- Accesses during normal operation

  While the active server is running, the shared disk and logical IP address are allocated on the active server where the daemons or services are running.

- Accesses after system switchover

  If a failure occurs on the active server, the standby server inherits the shared disk and logical IP address and starts the same daemons or services that were running on the active server. Even though the physical server has changed as a result of system switchover, it looks to the client as if the server with the same IP address is running because the standby server inherits the shared disk and logical IP address.

To run JP1/Advanced Shell in a logical host environment, you must have a shared disk to store the data that needs to be inherited during system switchover and a logical IP address. If you will be using the user-reply functionality, you must also set up the cluster software so that it can control the start, stop, and operation monitoring of the user-reply functionality's management daemon or service.

A JP1/Advanced Shell that is running in a logical host environment can inherit the job execution environment from the active server to the standby server during system switchover by using data stored on the shared disk. This means that JP1/Advanced Shell must store the spool on the shared disk. Note that execution of a job that was executing at the time system switchover occurred does not continue.

For details about the JP1/Advanced Shell settings required for cluster operations, see *2.9 Running in a cluster configuration*.

1.5 Overview of functionality

The following table describes the functionality supported by JP1/Advanced Shell.

Table 1–3: Functionality of JP1/Advanced Shell

Functionality	Related item	Section
Defining a job execution environment	Specifying the environment variables required for job execution	*2.5*
	Specifying environment files.	*2.6*, *7.*
Creating job definition scripts according to the syntax for shell scripts	Basic components of job definition scripts	*5.1*
	Conditional	*5.2*, *5.4*, *9.6*
	Arithmetic operations	*5.3*, *5.4*
	Shell variables	*5.1.2*, *5.5*
	Shell options	*5.6*
Using files from job definition scripts	Regular files	*5.9.1*, *9.5*
	Temporary files	*5.9.2*, *9.5*
	Program output data files	*5.9.3*, *9.5*
Controlling job execution	Declaring job names	*5.8.1*, *9.5*
	Defining job end conditions	*5.8.2*, *9.5*
	Starting or ending job steps	*5.8.3*, *9.5*
	Specifying a definition so that any command is always treated as having terminated normally	*5.8.4(2)*, *9.5*
	Defining return codes for extended script commands	*2.6.9*, *7.3*
	Calling external scripts	*5.8.6*, *9.5*
	Starting child jobs	*2.6.5*, *3.1.2(1)*, *3.2.3*, *5.1.11*, *7.3*
	Forcibly terminating jobs	*2.6.21*, *3.10*, *7.3*
Acquiring job information within shell scripts	Using shell variables for which a job step return code has been specified	*5.5.1*
	Using environment variables for which job information has been specified	*2.5*, *5.7*
Using the editor to create job definition scripts[#1]		*4.*, *5.*
Executing commands	Shell operation commands	*8.3*
	UNIX-compatible commands	*2.6.6*, *8.4*, *8.5*
	Standard shell commands	*9.3*
	Extended shell commands	*9.4*
	Extended script commands	*9.5*
	Script control statements	*9.6*
	Reserved script commands	*9.7*
Registering custom jobs[#2]		*2.7.1*

Functionality	Related item	Section
Using the user-reply functionality	Issuing any character string as a JP1 event	*9.4*
	Issuing any character string as a reply-waiting event	*9.4*
	Starting the user-reply functionality's management daemon or service	*8.3*
Collecting the operation information of job definition scripts[#2]	Accumulating the operation information of job definition scripts	*7.3*
	Outputting the operation information of job definition scripts	*8.3*
Using the same job definition scripts on different platforms	Converting job definition scripts so that they can be used in both Windows and UNIX	*2.6.2*, *2.6.3*, *2.6.4*, *2.6.12*, *7.3*
	Using the UNIX-compatible commands	*2.6.6*, *8.4*, *8.5*
Deleting spool jobs		*3.8* , *8.3*
Collecting coverage information	Acquiring coverage information	*3.9* , *8.3*, *Appendix A*
	Displaying coverage information	*3.9* , *8.3*
	Merging coverage information	*3.9* , *8.3*
	Always enabling acquisition of coverage information	*3.9*, *8.3*
Debugging job definition scripts	Using CUI for debugging[#3]	*6.1.2*, *6.1.4*, *6.1.5*, *6.2*
	Using GUI for debugging[#1]	*4.2.2*, *4.4.6*, *6.1.1*, *6.1.3*, *6.1.5*
Outputting job execution logs		*1.1.3*, *3.4*
Troubleshooting	Collecting data, such as job execution logs, system execution logs, and trace logs	*10.*
	Replying to reply-request messages when JP1/IM is not available	*8.3*, *10.*

#1

Available only in the development environment

#2

Available only in the execution environment

#3

Available only in the UNIX edition

Part 2: Setup

2 Preparations for Using JP1/Advanced Shell

This chapter discusses the conditions and requirements for using JP1/Advanced Shell, including the program installation directory, the main programs, prerequisites, installation, environment information settings, and custom job registration.

2.1 Program installation directory

The program installation directory for JP1/Advanced Shell depends on the OS being used. In a Windows environment, you can change the default installation directory. In a UNIX environment, the program is installed in a fixed directory.

This section explains the organization of JP1/Advanced Shell's program installation directory and the directories used to store various files that are output and referenced by JP1/Advanced Shell.

2.1.1 Installation folder (Windows only)

(1) Installation folder

In Windows, you can install JP1/Advanced Shell in any folder. The following folders for the environments are created under the specified installation folder.

Environment to be installed	Installation target	Remarks
Execution environment	*installation-folder*`\JP1ASE`	We recommend that you install this environment on a server.
Development environment	*installation-folder*`\JP1ASD`	We recommend that you install this environment on a client PC.
Custom job definition programs included in the execution environment (JP1/Advanced Shell - Custom Job)	*installation-folder*`\JP1ASV`	Install these programs at the operation management console on which JP1/AJS - View is installed.

The organization of the installation folder is shown below. Only folders for the environments are created.

```
installation-folder#1
  |---JP1ASD            : Development environment folder
  |    |---bin          : Program folder
  |    |---cmd          : Folder for UNIX-compatible commands
  |    |---doc--en--help--INDEX.HTM     : Help (manual)
  |    |---maintenance  : Folder used for handling errors
  |    |---readme.txt   : readme file
  |    |---sample       : Folder for sample data
  |          |-Adapter_HITACHI_JP1_AS_ASD_USERREPLY.conf
  |          | : Adapter command configuration file used for the user-reply
functionality
  |          |-hitachi_jp1_as_C_attr_cn.conf
  |          | : definition file for the extended event attributes (English)
  |          |-hitachi_jp1_as_C_attr_en.conf
  |          | : definition file for the extended event attributes (English)
  |          |-hitachi_jp1_as_EUC_attr_ja.conf
  |          | : definition file for the extended event attributes
(Japanese EUC)
  |          |-hitachi_jp1_as_SJIS_attr_ja.conf
  |          | : definition file for the extended event attributes
(Japanese Shift JIS)
  |          |-hitachi_jp1_as_UTF8_attr_ja.conf
  |          | : definition file for the extended event attributes
(Japanese UTF-8)
  |          |-sample.ase               : Templates for environment files
  |          |-sample.ash               : Sample job definition script file
  |          |-script_0        : Common sample script that does nothing
  |          |-script_chmod1     : Sample script file for chmod
  |          |-script_chmod2     : Sample script file for chmod
  |          |-script_chmod3     : Sample script file for chmod
  |          |-script_su1        : Sample script file for su
  |          |-script_who1          : Sample script file for who
  |---JP1ASE            : Execution environment folder
  |    |---bin          : Program folder
  |    |---cmd          : Folder for UNIX-compatible commands
  |    |---doc--en--help--INDEX.HTM     : Help (manual)
  |    |---maintenance  : Folder used for handling errors
  |    |---readme.txt   : readme file
  |    |---sample       : Folder for sample data
  |    |     |-Adapter_HITACHI_JP1_AS_ASE_USERREPLY.conf
  |    |     | : Adapter command configuration file used for the user-reply
functionality
  |    |     |-hitachi_jp1_as_C_attr_cn.conf
  |    |     | : definition file for the extended event attributes (English)
  |    |     |-hitachi_jp1_as_C_attr_en.conf
  |    |     | : definition file for the extended event attributes (English)
  |    |     |-hitachi_jp1_as_EUC_attr_ja.conf
  |    |     | : definition file for the extended event attributes
(Japanese EUC)
  |    |     |-hitachi_jp1_as_SJIS_attr_ja.conf
  |    |     | : definition file for the extended event attributes
(Japanese Shift JIS)
  |    |     |-hitachi_jp1_as_UTF8_attr_ja.conf
  |    |     | : definition file for the extended event attributes
(Japanese UTF-8)
  |    |     |-sample.ase               : Templates for environment files
  |    |     |-sample.ash               : Sample job definition script
  |    |    |-script_0            : Common sample script that does nothing
  |    |    |-script_chmod1               : Sample script file for chmod
  |    |    |-script_chmod2               : Sample script file for chmod
  |    |    |-script_chmod3               : Sample script file for chmod
  |    |    |-script_su1                  : Sample script file for su
  |    |    |-script_who1                 : Sample script file for who
  |    |---util--setup.exe                 : Installer for the custom job
definition program
  |
  |---JP1ASV            : Custom job definition program folder
       |---bin          : Program folder
       |---doc--en--help--INDEX.HTM    : Help (manual)
       |---image--custom              : Folder for custom job icons
       |            |-CUSTOM_PC_ADSHPC.gif : Custom job icons for PC jobs#2
       |            |-CUSTOM_PC_ADSHUX.gif : Custom job icons for UNIX jobs#2
       |---maintenance  : Folder used for handling errors
```

#1

Do not use any of the following characters in the installation folder name: `& [ ] { } ^ = ; ! ' + , ` ~ # %`. The product will not function normally if it is installed in a folder whose name contains any of these characters.

#2

If a version of JP1/AJS3 - View that is earlier than 09-50 is installed and the custom job definition program is to be installed, the custom job icons must be copied to the following folder:

```
JP1/AJS-View-installation-folder\image\custom
```

(2) Trace output folder and folder for creating a system environment file

The trace output folder and the folder for creating the system environment files are created under the Common application data folder.

```
Common-application-data-folder
  |---Hitachi--JP1AS--JP1ASD     : Development environment folder
                |      |---conf  : System environment file storage folder
                |      |---trace : Trace output folder
                |      |---uxpl  : Log folder
                |-----JP1ASE     : Execution environment folder
                |      |---conf  : System environment file storage folder
                |      |---trace : Trace output folder
                |      |---uxpl  : Log folder
                |-----JP1ASV     : Custom job definition program folder
                |      |---trace : Trace output folder
                |-----misc       : Folder for libraries common to all
products
                       |---trace : Trace output folder
                       |---uxpl  : Log folder
```

(3) System execution logs, spool, and temporary files

The folders for system execution logs, spool, and temporary files are created in the shared documents folder.

```
shared-documents-folder
  |---Hitachi--JP1AS--JP1ASD     : Development environment folder
                |      |---log   : Folder for system execution logs
                |      |---spool : Spool folder
                |      |---temp  : Folder for temporary files
                |-----JP1ASE     : Execution environment folder
                |      |---log   : Folder for system execution logs
                |      |---spool : Spool folder
                |      |---temp  : Folder for temporary files
                |-----misc       : Folder for libraries common to all
products
                       |---log   : Log folder
```

(4) List of programs

The following table lists and describes the storage locations and file names of the main programs used in JP1/Advanced Shell.

Table 2–1: Main programs used in JP1/Advanced Shell (Windows only)

Storage folder	File name	Overview of program (icon)	Description
installation-folder\each-environment's-folder[#1]\bin	`adshchmsg.exe`	Manual response to reply-request messages when an error occurs	Command that is used to respond manually to reply-request messages when an error occurs. This command must be used by a user with Administrators permissions.

Storage folder	File name	Overview of program (icon)	Description
installation-folder\each-environment's-folder[#1]\bin	`adshctmj.exe`	JP1/Advanced Shell execution definition program ( )	Program that defines the JP1/Advanced Shell execution environment in a custom job definition program.
	`adshctmjpc.bat`	JP1/Advanced Shell execution definition program for PC jobs	Program that defines the JP1/Advanced Shell execution environment for PC jobs in a custom job definition program.
	`adshctmjunix.bat`	JP1/Advanced Shell execution definition program for UNIX jobs	Program that defines the JP1/Advanced Shell execution environment for UNIX jobs in a custom job definition program.
	`adshcvmerg.exe`	Merging coverage information	Command that merges coverage information. This program can be used in both environments (execution and development).
	`adshcvshow.exe`	Displaying coverage information from commands	Command that displays coverage information. This program can be used in both environments (execution and development).
	`adshcvview.exe`	Displaying coverage information from the editor	Program that displays coverage information. This program enables coverage information to be displayed from the editor in the development environment.
	`adshedit.exe`	JP1/Advanced Shell Editor ( )	Editor used to edit job definition scripts in the development environment. Double-clicking the icon opens the JP1/Advanced Shell Editor.
	`adshesub.exe`	Debugging in the editor	Program that debugs job definition scripts in the development environment. This program is started automatically from `adshedit.exe`.
	`adshevtout.exe`	Outputting job definition script operation information	In an execution environment, the command that outputs job definition script operation information to a CSV file.
	`adshexec.exe`	Executing batch jobs	Command that starts the job controller that analyzes and controls execution of job definition scripts.
	`adshexecsub.exe`		Command that executes batch jobs in the execution environment. This command is started automatically from `adshexec.exe`.
	`adshfile.exe`	Registration of file postprocessing	Command that defines how a specified file is to be processed when a job step or job is terminated. This program can be used in both environments (execution and development).
	`adshhk.exe`	Deleting spool jobs	Command that deletes spool jobs. This command can be used in both environments (execution and development).
	`adshlsmsg.exe`	Displaying a list of reply-request messages when an error has occurred	Command that outputs job definition script operation information to a CSV file in the execution environment.
	`adshmsvcd.exe`	User-reply functionality's management service	Command that registers the service that manages shared memory for the user-reply functionality. This command is used in the development environment. It must be used by a user with Administrators permissions.
	`adshmsvce.exe`	User-reply functionality's management service	Command that registers the service that manages shared memory for the user-reply functionality. This command is used in the execution environment. It must be used by a user with Administrators permissions.

Storage folder	File name	Overview of program (icon)	Description
installation-folder\each-environment's-folder#1\cmd	awk.exe, basename.exe, cat.exe, cmp.exe, cp.exe, cut.exe, date.exe, diff.exe, dirname.exe, egrep.exe, expand.exe, expr.exe, find.exe, getopt.exe, grep.exe, head.exe, hostname.exe, ls.exe, mkdir.exe, mv.exe, paste.exe, rm.exe, rmdir.exe, sed.exe, sleep.exe, sort.exe, split.exe, stat.exe, tail.exe, touch.exe, uname.exe, uniq.exe, wc.exe, which.exe	UNIX-compatible commands#2	Commands that are used mainly in UNIX batch applications but can also be used in a Windows environment. These commands can be used in both environments (execution and development).
installation-folder\each-environment's-folder#1\maintenance	adshcollect.bat	Collecting data	Command that collects troubleshooting data. This program can be used in both environments (execution and development).

#1

each-environment's-folder is JP1ASD for the development environment and JP1ASE for the execution environment and JP1ASV for the custom job definition program.

#2

The UNIX-compatible commands also include chmod, su, and who. If you will be using any of these three commands, edit beforehand each of the applicable sample script files provided by JP1/Advanced Shell using the procedure described in *2.6.6(2) Preparations for using the script-format UNIX-compatible commands (Windows only)*.

2.1.2 Installation directory (UNIX only)

(1) Installation directory

The UNIX execution environment is installed in a fixed directory (/opt/jp1as). There is no development environment for a UNIX environment.

The organization of the installation directory is as follows:

```
/opt/jp1as
  |---bin                    : Program directory
  |---cmd               : Directory for UNIX-compatible commands
  |---conf              : System environment file storage directory
  |---instlog           : Installation log information directory
  |---lib               : Library directory
  |    |---nls          : Message catalog directory
  |---log               : Directory for system execution logs
  |---maintenance       : Directory used for handling errors
  |---sample            : Directory for sample data
  |    |---Adapter_HITACHI_JP1_AS_USERREPLY.conf
  |    | : Adapter command configuration file used for the user-reply
functionality
  |    |---hitachi_jp1_as_C_attr_cn.conf
  |    | : definition file for the extended event attributes (English)
  |    |---hitachi_jp1_as_C_attr_en.conf
  |    | : definition file for the extended event attributes (English)
  |    |---hitachi_jp1_as_EUC_attr_ja.conf
  |    | : definition file for the extended event attributes (Japanese EUC)
  |    |---hitachi_jp1_as_SJIS_attr_ja.conf
  |    | : definition file for the extended event attributes (Japanese
Shift JIS)
  |    |---hitachi_jp1_as_UTF8_attr_ja.conf
  |    | : definition file for the extended event attributes (Japanese
UTF-8)
  |    |---jp1_as_md    : Automatic start and stop script file
  |    |                  for user-reply functionality management daemon#
  |    |---sample.ase   : Templates for environment files
  |    |---sample.ash   : Sample job definition script file
  |---sbin              : Program directory for system administrator
  |---system
  | : Directory used by the user-reply functionality management daemon
  |---trace             : Trace directory
  |---util--setup.exe   : Installer for the custom job definition program
```

#

Installed for the Linux, HP-UX, and Solaris versions.

(2) Spool directory and directory for temporary files

The spool directory and the directory for temporary files are created in the following directories:

```
/var/opt/jp1as--spool : Spool directory
            |---temp  : Directory for temporary files
```

(3) List of programs

The following table lists and describes the storage locations and file names of the main programs used in JP1/Advanced Shell.

Table 2–2: Main programs used in JP1/Advanced Shell (UNIX only)

Storage directory	File name	Overview of program	Description
/opt/jp1as/bin	adshcvmerg	Merging coverage information	Command that merges coverage information.

Storage directory	File name	Overview of program	Description
/opt/jp1as/bin	adshcvshow	Displaying coverage information	Command that displays coverage information.
	adshevtout	Outputting job definition script operation information	Command that outputs job definition script operation information to a CSV file.
	adshexec	Executing batch jobs	Command that executes batch jobs.
	adshfile	Registration of file postprocessing	Command that defines how a specified file is to be processed when a job step or job is terminated.
	adshhk	Deleting spool jobs	Command that deletes spool jobs.
/opt/jp1as/cmd	awk, basename, cat, cmp, cp, cut, date, diff, dirname, egrep, expand, expr, find, getopt, grep, head, hostname, ls, mkdir, mv, paste, rm, rmdir, sed, sleep, sort, split, stat, tail, touch, uname, uniq, wc, which	UNIX-compatible commands	UNIX-compatible commands that can be used from job definition scripts.
/opt/jp1as/maintenance	adshcollect	Collecting data	Command that collects troubleshooting data.
/opt/jp1as/sbin	adshchmsg	Manual response to reply-request messages when an error occurs	Command that is used to respond manually to reply-request messages when an error occurs. This command must be used by a user with the superuser permissions.
	adshlsmsg	Displaying a list of reply-request messages when an error has occurred	Command that displays a list of reply-request messages when an error has occurred. This command must be used by a user with the superuser permissions.
	adshmdctl	User-reply functionality's management daemon	Command that starts and stops the daemon for managing shared memory for the user-reply functionality. This command must be used by a user with the superuser permissions.

2.2 Evaluations prior to installation

This section explains the evaluations that need to be made prior to installation, including the system configuration, prerequisite programs, related programs, and files to be used.

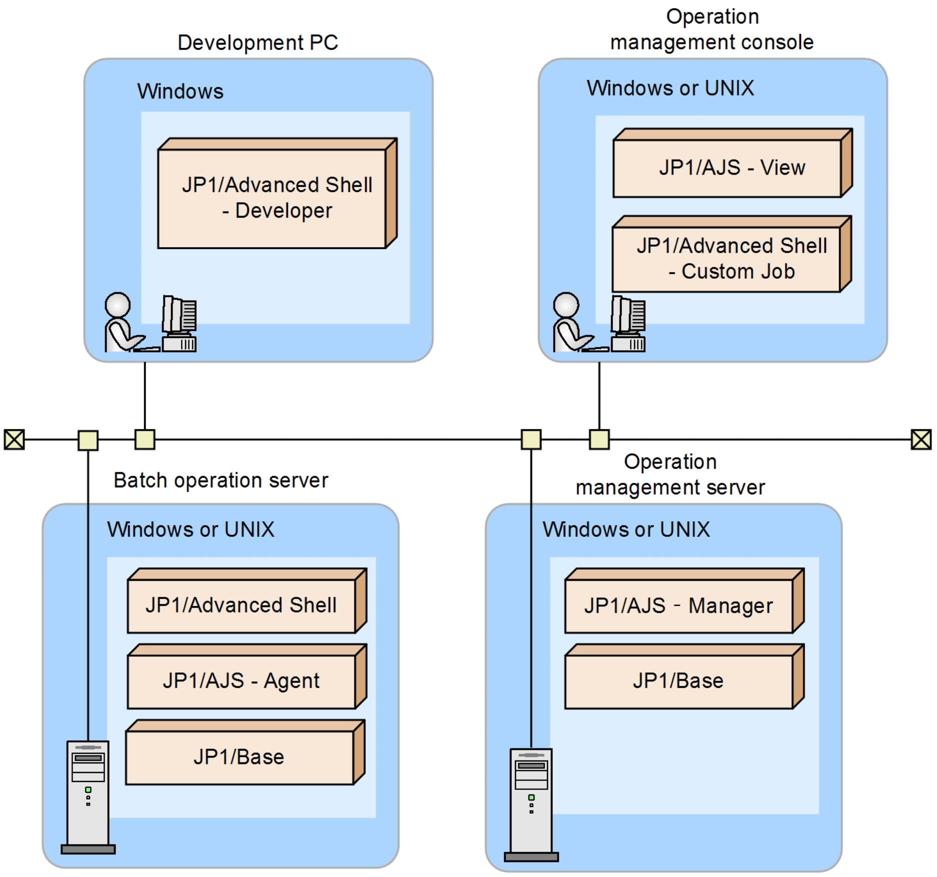
2.2.1 System configuration

This subsection explains the JP1/Advanced Shell system configuration for each execution mode.

(1) Executing batch jobs from JP1/AJS

The following shows the system configuration for executing batch jobs from JP1/AJS.

Figure 2–1: System configuration for executing batch jobs from JP1/AJS



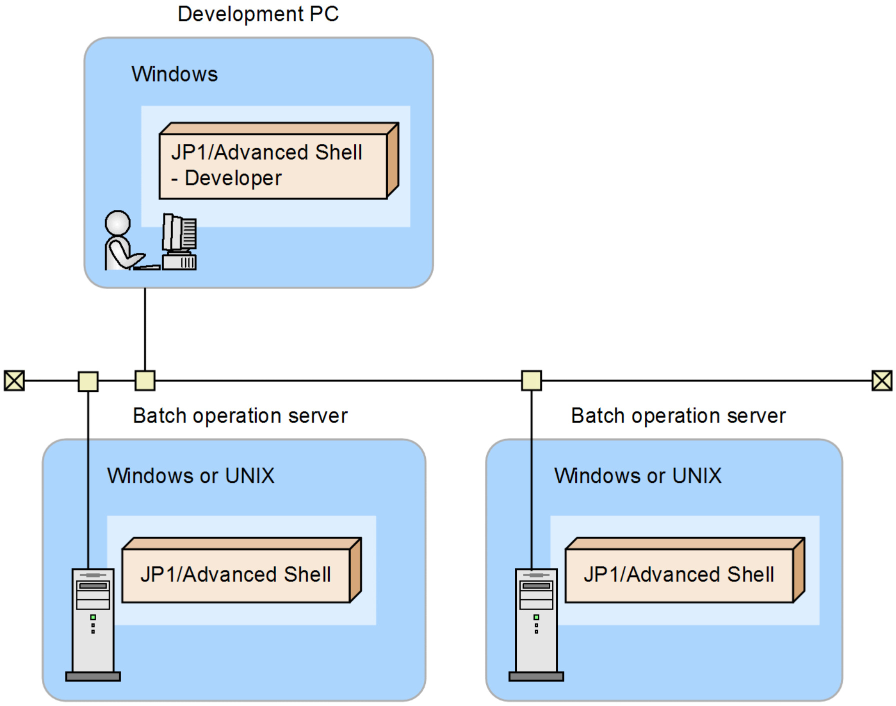
The following explains the role of each system component.

- Development PC: Enables use of JP1/Advanced Shell - Developer for creation of job definition scripts.
- Batch operation server: Enables manual or automatic execution of job definition scripts.
- Operation management server: Manages jobs that are executed.
- Operation management console: Enables use of JP1/AJS - View for displaying job execution results, and is used to define the job definition scripts that are to be executed automatically. To perform job definition in JP1/Advanced Shell, you must have JP1/Advanced Shell - Custom Job (custom job definition program).

(2) Executing batch jobs manually

The following shows the system configuration for executing batch jobs manually.

Figure 2–2: System configuration for executing batch jobs manually



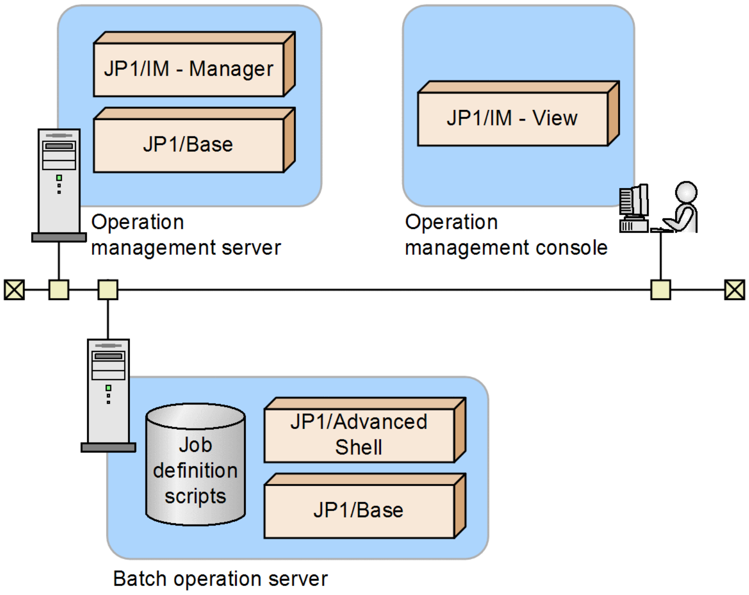
The following explains the role of each system component.

- Development PC: Enables use of JP1/Advanced Shell - Developer for creation of job definition scripts.
- Batch operation server: Enables manual execution of job definition scripts.

(3) Using the user-reply functionality

The following figure shows the system configuration for using the user-reply functionality.

Figure 2–3: System configuration for using the user-reply functionality



For details, see *3.7 Using the user-reply functionality*.

2.2.2 Programs required in each environment

This subsection explains the prerequisite programs for JP1/Advanced Shell and the related programs.

(1) Prerequisite and related programs for the execution environment

(a) Prerequisite programs for the execution environment

The following table lists the prerequisite programs for the execution environment.

Table 2–3: Prerequisite programs for the execution environment

Server type	OS
Same batch operation server as for JP1/Advanced Shell (Windows only)	Windows
Same batch operation server as for JP1/Advanced Shell (UNIX only)	AIX, HP-UX, Linux, or Solaris

(b) Related programs for the execution environment

The following tables list the related programs for each server in the execution environment.

Table 2–4: Related programs in the execution environment (when executing batch jobs from JP1/AJS)

Server type	Processing to be performed	Programs
Same batch operation server as for JP1/Advanced Shell	Executes job definition scripts from JP1/AJS	JP1/Base JP1/AJS - Agent#

Server type	Processing to be performed	Programs
Operation management server	Manages jobs	JP1/Base JP1/AJS - Manager#
Operation management console (Windows only)	Displays job execution results	JP1/AJS - View

#

JP1/AJS - Agent is not needed when JP1/AJS - Manager is installed on the same server as JP1/Advanced Shell, because JP1/AJS - Manager provides the JP1/AJS - Agent functions.

Table 2–5: Related programs in the execution environment (when using the user-reply functionality)

Server type	Programs
Same batch operation server as for JP1/Advanced Shell	JP1/Base
Operation management server	JP1/IM - Manager JP1/Base
Operation management console	JP1/IM - View

Note:

These programs are not required if you debug and run your coding for the user-reply functionality with the standard output specified as the output destination.

(2) Prerequisite and related programs for the execution environment (Windows only)

(a) Prerequisite program for the development environment

The following table shows the prerequisite program for the development environment.

Table 2–6: Prerequisite program for the development environment (Windows only)

Server type	OS
Same development PC as for JP1/Advanced Shell - Developer	Windows

(b) Related programs for the development environment

The following tables list the related programs for the servers in the development environment.

Table 2–7: Related programs in the development environment (when using the user-reply functionality)

Server type	Programs
Same development PC as for JP1/Advanced Shell - Developer	JP1/Base
Operation management server	JP1/IM - Manager JP1/Base
Operation management console	JP1/IM - View

Note:

These programs are not required if you debug and run your coding for the user-reply functionality with the standard output specified as the output destination.

(3) Prerequisite and related programs for the custom job definition program (Windows only)

(a) Prerequisite programs for the custom job definition program

The table below shows the prerequisite programs for the custom job definition program. Although the custom job definition program is supported only in Windows, you can use it to create both Windows and UNIX job definitions.

Table 2–8: Prerequisite programs for the custom job definition program (Windows only)

Server type	OS and Program
Same operation management console as for JP1/Advanced Shell - Custom Job	• OS Windows • Program JP1/AJS - View

(b) Related programs for the custom job definition program

There are no related programs for the custom job definition program.

2.2.3 Files used in JP1/Advanced Shell

(1) List of files used in JP1/Advanced Shell

The table below lists and describes the files that are used in JP1/Advanced Shell. To determine whether a file size can exceed 2 GB, see *(2) Handling of files that are larger than 2 GB (large files)*.

Table 2–9: Files used in JP1/Advanced Shell

File name (icon)	Extension	File location	File contents
Job definition script file ( )	`.ash`	Any location	A job definition script. The user can assign any file name.
Environment file#1	`.ase`	Any location	JP1/Advanced Shell environment settings.
System environment file	`.ase`	See *2.6.1(1) Specifying the system environment files*.	System environment settings.
Coverage information file	`.asc`	Any location	Coverage environment information for JP1/Advanced Shell.
Debugging information file	`.asd`	Same directory as for the job definition script files#2	Debugging information used by the editor (development environment)
System execution log#1	`.log`	Directory specified in the `LOG_DIR` parameter in the environment file	Log information that provides overall batch job execution logs for the system administrator.
Trace information#1	`.log`	• For the `adshexec` command, the directory specified in the `TRACE_DIR` parameter in the environment file • In all other cases, the directory specified by the program	JP1/Advanced Shell's internal trace logs.
Temporary file	`.tmp`	• For a temporary file specified in the `#-adsh_file_temp` command, the	Temporary file used internally by the system.

File name (icon)	Extension	File location	File contents
Temporary file	`.tmp`	directory specified in the `TEMP_FILE_DIR` parameter in the environment file • In all other cases, the directory specified by the program	Temporary file used internally by the system.
Coverage display temporary file	`.txt`	Temporary file directory specified by the system	Temporary file used in displaying coverage information. The format of file name is as follows: `adshexec_view_`*job-definition-script-file-name*`_`*year-month-date*`_`*hour-minute-second*`.txt`
Start log (UNIX only)	`.log`	`/opt/jp1as/system`	Log information that is collected when the user-reply functionality's management daemon is started and stopped.
`pid` file (UNIX only)	`.pid`	`/opt/jp1as/system`	File used by the user-reply functionality management daemon and `adshmdctl`.

#1

You can collect these files by using the `adshcollect` command. For details about how to collect the files, see *adshcollect command (collects information)*.

#2

Because a debugging information file cannot be saved, an error is displayed in the following cases:

- The job definition script file being edited is in a directory for which the user does not have write permissions.
- The job definition script file being edited is in a compressed folder.

Notes about specifying files and paths

- Do not use a file name that begins with a dot (`.`).
- The permitted maximum length for path names must comply with the specifications of the OS being used.
- The maximum file name length is 246 bytes (Windows only).
- Do not use reserved device names (such as `CON`, `AUX`, and `NUL`) for file names (Windows only).
- Do not use NTFS streams for file names (Windows only).
- Do not use hard links, symbolic links, or junctions (Windows only).
- You can use UNC names for file names and path names (example: `\\`*computer-name*`\`*shared-name*`\`*file-name*); however, make sure that a path name specified in this format does not end with *shared-name* (or *shared-name*`\`). The `cd` standard shell command does not support the UNC format. (Windows only)

  Examples of valid specification:

  ```
  \\server\share\dir
  \\10.111.222.33\share\dir
  ```

  Examples of invalid specification:

  ```
  \\server\share
  \\10.111.222.33\share
  ```

- Do not use UNC names for the folder path names for traces, system execution logs, spool, and temporary files (Windows only).

(2) Handling of files that are larger than 2 GB (large files)

JP1/Advanced Shell supports some of the large files (that are larger than 2 GB). Of the files supported by JP1/Advanced Shell, the files and commands that correspond to large files are as follows:

- Of the files created in the spool job directories, files STDOUT, STDERR, *step-number*_*step-name*_STDOUT, and *step-number*_*step-name*_STDERR to which user data is output

  Note that large files might adversely affect the overall system processing because the jobs executed from JP1/AJS3 transfer the contents of files STDERR and *step-number*_*step-name*_STDERR to JP1/AJS3's manager host. For details, see the *JP1/Automatic Job Management System 3 System Design (Configuration) Guide*.

  Note that when you use the spool job creation suppression functionality, no spool job is created when the job definition script is run. For details about the spool job creation suppression functionality, see *2.6.8(1)(a) Determining whether the spool job creation suppression functionality is to be used*.
- In redirect specifications, the files specified in >file, < file, >>file, >|file, <>file, n>file, and n<file
- Files specified in conditional expressions other than -t fd of the conditional expressions that are evaluated with the test or let command
- Files that are allocated by the extended script commands #-adsh_file, #-adsh_file_temp, and #-adsh_spoolfile
- Files handled by UNIX-compatible commands

  However, a file cannot be exceed 2 GB if it is used for the following operations:
  - Editing and displaying the number of bytes that exceeds 2 GB
  - Editing and displaying the number of lines that exceed 2 GB
  - Executing UNIX commands (such as diff and sort) that use a large amount of memory when large-sized files are specified

Whether large files are supported depends on the types of file systems and OS settings (example: ulimit setting). Before you design your operations, check if your environment supports large files.

2.2.4 Encoding used in JP1/Advanced Shell

You must code job definition script files and environment files used in JP1/Advanced Shell using the encoding that matches the value of the LANG environment variable in the environment in which JP1/Advanced Shell is run. If a different encoding is used, operation cannot be guaranteed. If you will execute the same job definition script file on different OSs, use an encoding that is supported by all the OSs to be used.

In UNIX, the language and encoding in which messages are output by JP1/Advanced Shell are determined by the value of the LANG environment variable.

The following table shows the values of the LANG environment variable and the encodings for job definition script files and environment files that are specified when JP1/Advanced Shell is used.

Table 2–10: Encodings corresponding to the LANG environment variable values

OS	Value of LANG environment variable	Encoding for job definition script files and environment files
Windows	--	Shift-JIS
Linux	ja_JP.UTF-8	UTF-8
AIX	Ja_JP ja_JP JA_JP JA_JP.UTF-8	Shift-JIS EUC UTF-8 UTF-8

OS	Value of LANG environment variable	Encoding for job definition script files and environment files

HP-UX	`ja_JP.SJIS`
	`ja_JP.eucJP`
	`ja_JP.utf8`
		Shift-JIS
		EUC
		UTF-8

Solaris	`ja_JP.PCK`
	`ja`
	`ja_JP.UTF-8`
		Shift-JIS
		EUC
		UTF-8

Legend:

--: Not applicable

2.2.5 Local time settings

JP1/Advanced Shell obtains and outputs local time information by referencing environment variables. You must specify the local time settings in the environment variables beforehand.

The commands provided by JP1/Advanced Shell output information according to the OS's time zone setting (Windows) or the `TZ` environment variable (UNIX). Use one of the methods listed below to specify the `TZ` environment variable. Note that no environment variable can be defined for custom job definition programs.

- JP1/AJS's job definition or environment variable definition
- System profile (`/etc/profile`)
- User profile (`$HOME/.profile`)

2.2.6 Notes about standard input

When the commands provided by JP1/Advanced Shell are used to input data from a terminal to the standard input, the permitted maximum length depends on the language specifications of the OS, terminal, shell, and programming language.

2.3 Installing and uninstalling (Windows only)

This section explains how to install and uninstall JP1/Advanced Shell in a Windows environment. You must first install prerequisite and related programs by referencing each program's documentation.

To install the product and related programs:

1. Install and set up the required products on the operation management server.
    - For running jobs in JP1/AJS
      JP1/AJS - Manager
    - For using the user-reply functionality
      JP1/IM - Manager
2. Install and set up the required products on the operation management console.
    - For running jobs in JP1/AJS
      JP1/AJS - View
    - For using the user-reply functionality
      JP1/IM - View
3. Install JP1/Advanced Shell - Custom Job on the operation management console.
   For details about installation of the custom job definition program, see *2.3.3 Installing JP1/Advanced Shell - Custom Job*.
4. Install and set up the required products on the batch operation server.
    - For running jobs in JP1/AJS
      JP1/AJS - Agent
    - For using the user-reply functionality
      JP1/Base
5. Install JP1/Advanced Shell on the batch operation server and specify settings such as environment information.
   For details about installation of JP1/Advanced Shell in a Windows environment, see *2.3.1 Installing JP1/Advanced Shell (Windows only)*.
   For details about the setup procedure for using the user-reply functionality, see *2.8.2 Setting up the user-reply functionality after JP1/Advanced Shell has been installed (Windows only)*.

2.3.1 Installing JP1/Advanced Shell (Windows only)

A user with an administrator role installs JP1/Advanced Shell. The two ways of installing JP1/Advanced Shell are remote installation using JP1/Software Distribution and installation from a CD-ROM. You install JP1/Advanced Shell - Developer in the same manner.

(1) Remote installation using JP1/Software Distribution

JP1/Advanced Shell supports use of JP1/Software Distribution for remote installation (software distribution).

For details about the remote installation method using JP1/Software Distribution, see the manuals *Job Management Partner 1/Software Distribution Description and Planning Guide* (for Windows systems) and *Job Management Partner 1/Software Distribution Administrator's Guide Volume 1* (for Windows systems).

(2) Installation from a CD-ROM

There are three types of installation of JP1/Advanced Shell from a CD-ROM:

- If you are newly deploying JP1/Advanced Shell, perform a new installation.
- If you are upgrading JP1/Advanced Shell, perform an overwrite installation.
- If you are re-installing the same version, perform a recovery installation.

The following subsections explain these three procedures.

(a) New installation

This subsection explains how to perform a new installation of JP1/Advanced Shell. Normally, the execution environment is installed on a server and the development environment is installed on a client PC. It is also possible to install both environments (execution and development) on the same PC.

To perform a new installation:

1. Log on as a user with an administrator role to the Windows machine on which JP1/Advanced Shell is to be installed.
2. Terminate all programs.
3. Place in the CD-ROM drive the CD-ROM that contains JP1/Advanced Shell.
4. Install JP1/Advanced Shell by entering required information as instructed by the installer.
   The following information will be requested during installation:
   - Product to be installed (JP1/Advanced Shell or JP1/Advanced Shell - Developer)
   - Customer Information
   - Destination Folder
5. When the Finish dialog box is displayed, click **Finish**.
   Installation is completed.

(b) Overwrite installation for upgrading

You perform an overwrite installation in the same manner as for a new installation.

You can upgrade JP1/Advanced Shell by performing an overwrite installation without having to uninstall the existing JP1/Advanced Shell.

(c) Recovery installation using the same version

To perform a recovery installation to correct problems in an already installed JP1/Advanced Shell:

1. Log on as a user with an administrator role to the Windows machine on which JP1/Advanced Shell is to be installed.
2. Terminate all programs.
3. Place in the CD-ROM drive the CD-ROM that contains JP1/Advanced Shell.
4. Enter required information as instructed by the installer, and then select **Program Maintenance**.
5. In **Program Maintenance**, select **Repair**.

6. When the Finish dialog box is displayed, click **Finish**.
   Recovery installation is completed.

2.3.2 Uninstalling JP1/Advanced Shell (Windows only)

(1) Uninstalling JP1/Advanced Shell manually

This subsection explains how to uninstall JP1/Advanced Shell. You uninstall JP1/Advanced Shell - Developer in the same manner.

To uninstall JP1/Advanced Shell:

1. Log on as a user with an administrator role to the Windows machine on which JP1/Advanced Shell has been installed.
2. Terminate all programs. If you are using the user-reply functionality, stop the services of JP1/Advanced Shell and then unregister them.
3. Place in the CD-ROM drive the CD-ROM that contains JP1/Advanced Shell.
4. Enter required information as instructed by the installer, and then select **Maintain Program**.
5. In **Program Maintenance**, select **Delete**.
6. When the Finish dialog box is displayed, click **Finish**.
   Uninstallation is completed.
7. If there are any unneeded files, such as spool, trace, and debugging information files, delete them.

If you were using the user-reply functionality, after JP1/Advanced Shell has been installed, delete the adapter command configuration file used for the user-reply functionality that has been set up for JP1/Base. For details about the storage folder for the adapter command configuration file used for the user-reply functionality, see *2.8.2(2) Setting up the adapter command (for the execution environment)* or *2.8.2(3) Setting up the adapter command (for the development environment)*.

(2) Using JP1/Software Distribution to uninstall JP1/Advanced Shell

For details about how to use JP1/Software Distribution to uninstall JP1/Advanced Shell, see the manuals *Job Management Partner 1/Software Distribution Description and Planning Guide* (for Windows systems) and *Job Management Partner 1/Software Distribution Administrator's Guide Volume 1* (for Windows systems).

2.3.3 Installing JP1/Advanced Shell - Custom Job

This subsection explains how to install the custom job definition program on the operation management console on which JP1/AJS - View is already installed. You install the custom job definition program by transferring it from JP1/Advanced Shell's installation directory to the operation management console.

Although the custom job definition program can be installed only in a Windows environment, it can be used to create definitions for both Windows and UNIX jobs. This subsection explains the new, overwrite, and recovery installation methods.

(1) New installation

To perform a new installation of JP1/Advanced Shell - Custom Job:

1. Log on as a user with an administrator role to the operation management console on which JP1/Advanced Shell - Custom Job is to be installed.
2. Obtain the installer for JP1/Advanced Shell - Custom Job.
   The installer is stored at the following location:
   - To use the Windows edition of JP1/Advanced Shell
     JP1/Advanced-Shell-installation-folder`\JP1ASE\util\setup.exe`
   - To use the UNIX edition of JP1/Advanced Shell
     `/opt/jp1as/util/setup.exe`
3. Transfer the installer (`setup.exe`) for JP1/Advanced Shell - Custom Job to the operation management console.
4. Start the command prompt and locate the folder to which the installer has been transferred, and then execute the following command:
   `setup.exe`
5. Install JP1/Advanced Shell - Custom Job by entering required information as instructed by the installer.
   Select the language of JP1/Advanced Shell - Custom Job to be installed, and then specify the following information:
   - Customer Information: Specify requested information, including a user name.
   - Destination Folder: Specify the folder in which JP1/Advanced Shell - Custom Job is to be installed.
6. When the Finish dialog box is displayed, click **Finish**.
   Installation is completed.
7. If you are installing the custom job definition program while a JP1/AJS3 - View whose version is earlier than 09-50 is installed, copy the custom job icons to the following folder:
   JP1/AJS-View-installation-folder`\image\custom`
   For details about the folder and the file names of the custom job icons to be copied, see *2.1.1 Installation folder (Windows only)*.

(2) Overwrite installation for upgrading

When you perform an overwrite installation of JP1/Advanced Shell - Custom Job for purposes of upgrading, there is no need to uninstall the existing JP1/Advanced Shell - Custom Job.

You perform the overwrite installation in the same manner as for a new installation.

If you copied the previous version's custom job icons to the JP1/AJS - View installation folder, there is no need to copy them again during upgrading.

(3) Recovery installation using the same version

To perform a recovery installation to correct problems in an already installed JP1/Advanced Shell - Custom Job:

1. Log on as a user with an administrator role to the Windows machine on which JP1/Advanced Shell - Custom Job is to be installed.

2. Obtain the installer for JP1/Advanced Shell - Custom Job.

   The installer is stored at the following location:

   - In Windows: *JP1/Advanced-Shell-installation-folder*`\JP1ASE\util\setup.exe`
   - In UNIX: `/opt/jp1as/util/setup.exe`

3. Transfer the installer (`setup.exe`) for JP1/Advanced Shell - Custom Job to the operation management console.

4. Start the command prompt and locate the folder to which the installer has been transferred, and then execute the following command:

   ```
   setup.exe
   ```

5. From **Program Maintenance**, select **Repair**.

6. When the Finish dialog box is displayed, click **Finish**.

   Recovery installation is completed.

2.3.4 Uninstalling JP1/Advanced Shell - Custom Job

To uninstall JP1/Advanced Shell - Custom Job:

1. Log on as a user with an administrator role to the Windows machine on which JP1/Advanced Shell - Custom Job has been installed.

2. Terminate all programs.

3. If a JP1/AJS - View whose version is earlier than 09-50 is installed, delete the custom job icons that were copied to the following folder:

   JP1/AJS-View-installation-folder`\image\custom`

4. On the **Control Panel**, select the product from **Add or Remove Programs**.

   If you are uninstalling JP1/Advanced Shell - Custom Job in an environment where user account control (UAC) is enabled, the User Account Control window is displayed. Select **Yes** in this window.

5. When the Finish dialog box is displayed, click **Finish**.

   Uninstallation is completed.

6. If there are any unneeded trace files, delete them.

2.4 Installing and uninstalling (UNIX only)

This section explains how to install and uninstall JP1/Advanced Shell in a UNIX environment. In a UNIX environment, you can install only the execution environment on the batch operation server. You must first install prerequisite and related programs by referencing each program's documentation.

To install the product and related programs:

1. Install and set up the required products on the operation management server.
   - For running jobs in JP1/AJS
     JP1/AJS - Manager
   - For using the user-reply functionality
     JP1/IM - Manager
2. Install and set up the required products on the operation management console in a Windows environment.
   - For running jobs in JP1/AJS
     JP1/AJS - View
   - For using the user-reply functionality
     JP1/IM - View
3. Install JP1/Advanced Shell - Custom Job on the operation management console.
   For details about installation of the custom job definition program, see *2.3.3 Installing JP1/Advanced Shell - Custom Job*.
4. Install and set up the required products on the batch operation server.
   - For running jobs in JP1/AJS
     JP1/AJS - Agent
   - For using the user-reply functionality
     JP1/Base
5. Install JP1/Advanced Shell on the batch operation server and specify settings such as environment information.
   For details about installation of JP1/Advanced Shell in a UNIX environment, see *2.4.1 Installing JP1/Advanced Shell (UNIX only)*.
   For details about the setup procedure for using the user-reply functionality, see *2.8.3 Setting up the user-reply functionality after JP1/Advanced Shell has been installed (UNIX only)*.

2.4.1 Installing JP1/Advanced Shell (UNIX only)

A user with an administrator role installs JP1/Advanced Shell. The two ways of installing JP1/Advanced Shell are remote installation using JP1/Software Distribution and installation from a CD-ROM.

(1) Remote installation using JP1/Software Distribution

JP1/Advanced Shell supports use of JP1/Software Distribution for remote installation (software distribution).

For details about the remote installation method using JP1/Software Distribution, see the manuals *Job Management Partner 1/Software Distribution Manager* and *Job Management Partner 1/Software Distribution SubManager* (for UNIX systems).

(2) Installation from a CD-ROM

This subsection explains how to install JP1/Advanced Shell from a CD-ROM.

Note that the directory and file names on the CD-ROM might be different from what is shown here, depending on the hardware environment. Use the `ls` command to check the file names and specify file names exactly as displayed.

To install JP1/Advanced Shell:

1. Specify the user permissions.
   Log on as a superuser to the server on which JP1/Advanced Shell is to be installed. Alternatively, use the `su` command to change the user permissions to superuser.

2. Terminate all programs.
   If any existing JP1-series programs and JP1/Advanced Shell program are running, terminate them.

3. Place the medium that contains JP1/Advanced Shell.

4. Mount the CD-ROM device by executing the following command:

   ```
   /bin/mount -r -o mode=0544 /dev/cdrom /cdrom
   ```

   `/cdrom` is the mount point of the CD-ROM device special file. If there is no mount point directory, create one. Note that the device special file name and mount point might differ depending on the environment.

5. Start the Hitachi Program Product Installer by executing the following command:

   In Linux

   `/cdrom/LINUX/setup /cdrom`#

   In AIX

   `/cdrom/AIX/setup /cdrom`#

   In HP-UX

   `/cdrom/IPFHPUX/setup /cdrom`#

   In Solaris

   `/cdrom/SOLARIS/setup /cdrom`#

   #: This example assumes `/cdrom` as the mount point.

   The Hitachi Program Product Installer starts and the initial window is displayed.

   The following is an example of the Hitachi Program Product Installer's initial window:

   ```
     Hitachi PP Installer 05-16

       L) List Installed Software.
       I) Install Software.
       D) Delete  Software.
       Q) Quit.

       Select Procedure ===>
   ```

6. In the Hitachi Program Product Installer's initial window, enter `I`.
   A list of programs that can be installed is displayed.

7. Select `JP1/Advanced Shell`, and then enter `I`.

JP1/Advanced Shell is installed. To select a program, move the cursor to the desired program, and then press the space bar to select it.

The following shows an example of the Hitachi Program Product Installer's installation window:

```
       PP-No.                VR     PP-NAME
<@>001 P-8112-B1AL           1010   Advanced Shell
:
:
F) Forward B) Backward J) Down K) Up Space) Select/Unselect I) Install Q)
Quit
```

`<@>` is displayed to the left of the selected program product. If you enter `I` following `<@>`, the following message is displayed on the last line:

```
Install PP? (y: install,  n: cancel)==>
```

If you enter `y` or `Y`, installation begins. If you enter `n` or `N`, installation is cancelled and the program product installation window is displayed again.

8. When installation is completed successfully, enter `Q`.

   The Hitachi Program Product Installer's initial window is displayed again.

Note that the following files are created during installation as installer's logs:

```
/opt/jp1as/instlog/ADSH_INST_LOG
/opt/jp1as/instlog/ADSH_INST_USERLOG
```

If the installer's log files are not created, possible causes are as follows:

- The installer's log files are not regular files.
- The user does not have write permission for the directory in which the installer's log files are to be created.
- A file with the same name already exists at the path of each log file of the installer.

  A file with the same name exists in the following cases:

  - "`/opt`" is not a directory.
  - "`/opt/jp1as`" is not a directory.
  - "`/opt/jp1as/instlog`" is not a directory.

When installation is completed, the default environment has been set up. To change the default settings, see the applicable subsections in *2.6 Specifying environment information for JP1/Advanced Shell*.

2.4.2 Uninstalling JP1/Advanced Shell (UNIX only)

(1) Uninstalling JP1/Advanced Shell manually

This subsection explains how to uninstall JP1/Advanced Shell. You uninstall JP1/Advanced Shell by following the Hitachi Program Product Installer's instructions.

Before you uninstall JP1/Advanced Shell, terminate all programs provided by JP1/Advanced Shell. If you are using the user-reply functionality, terminate the user-reply functionality's management daemon. The installer's log files and any newly created files are not deleted during uninstallation. To completely delete the environment, the user must delete these files.

To uninstall JP1/Advanced Shell:

1. Start the Hitachi Program Product Installer by executing the following command:

   ```
   /etc/hitachi_setup
   ```

   The Hitachi Program Product Installer starts and the initial window is displayed.

   The following shows an example of the Hitachi Program Product Installer's initial window:

   ```
     Hitachi PP Installer 05-16

       L) List Installed Software.
       I) Install Software.
       D) Delete  Software.
       Q) Quit.

       Select Procedure ===>
   ```

2. In the Hitachi Program Product Installer's initial window, enter `D`.

   A list of the software programs that can be uninstalled is displayed.

3. Select **JP1/Advanced Shell**, and then enter `D`.

   JP1/Advanced Shell is uninstalled. To select a program, move the cursor to the target program, and then press the space bar to select it.

   The following shows an example of the uninstallation window:

   ```
         PP-No.               VR     PP-NAME
      001 P-8112-B1AL          1010   Advanced Shell
   :
   :
   F) Forward B) Backward J) Down K) Up Space) Select/Unselect D) Delete Q)
   Quit
   ```

   `<@>` is displayed to the left of the selected program product. If you enter `D` following `<@>`, the following message is displayed on the last line:

   ```
   Delete PP? (y: delete, n: cancel) ==>
   ```

   If you enter `y` or `Y`, uninstallation begins. If you enter `n` or `N`, uninstallation is cancelled and the program product uninstallation window is displayed again.

4. When uninstallation is completed successfully, enter `Q`.

   The Hitachi Program Product Installer's initial window is displayed again.

5. If there are any unneeded files, such as execution log and trace files, delete them.

Note that the following files are created during uninstallation as installer's logs:

```
/opt/jp1as/instlog/ADSH_INST_LOG
/opt/jp1as/instlog/ADSH_INST_USERLOG
```

If the installer's log files are not created, possible causes are as follows:

- The installer's log files are not regular files.
- The user does not have write permission for the directory in which the installer's log files were created.
- A file with the same name already exists at the path of each log file of the installer.

  A file with the same name exists in the following cases:

  - "`/opt`" is not a directory.
  - "`/opt/jp1as`" is not a directory.
  - "`/opt/jp1as/instlog`" is not a directory.

Notes:

If the user-reply functionality's management daemon is running, the uninstallation process is cancelled and the following message is output to `/opt/jp1as/instlog/ADSH_INST_LOG`:

```
One or more /opt/jp1as/sbin/adshmd processes are running.
Please stop them.
```

When this message has been output, execute the `adshmdctl` command to terminate the user-reply functionality's management daemon, and then perform uninstallation again.

Uninstallation is also cancelled if the user-reply functionality's management daemon does not terminated normally. In such a case, start the user-reply functionality's management daemon and then terminate it, then perform uninstallation again.

(2) Using JP1/Software Distribution to uninstall JP1/Advanced Shell

For details about how to use JP1/Software Distribution to uninstall JP1/Advanced Shell, see the manuals *Job Management Partner 1/Software Distribution Manager* and *Job Management Partner 1/Software Distribution SubManager* (for UNIX systems).

(3) If you were using the user-reply functionality

If you were using the user-reply functionality, perform the following after uninstallation has been completed:

- Delete the adapter command configuration file used for the user-reply functionality that has been set up for JP1/Base. For details about the storage directory for the adapter command configuration file used for the user-reply functionality, see *2.8.3(2) Setting up JP1/Base*.
- If the user-reply functionality's management daemon has been set to start and terminate automatically, disable the automatic start and termination settings.

  In AIX:

  1. Disable the automatic start setting for the user-reply functionality's management daemon by executing the following command:

     `rmitab adshmd`

  2. If the user-reply functionality's management daemon for the logical host is set to start automatically, execute the `rmitab` command specifying the record of the user-reply functionality's management daemon for the logical host.
  3. To disable the automatic termination function at system shutdown, delete the following code from `/etc/rc.shutdown`:

     ```
     test -x /opt/jp1as/sbin/adshmdctl && /opt/jp1as/sbin/adshmdctl stop
     ```

4. If the user-reply functionality's management daemon for the logical host is set to terminate automatically, delete the specification for automatically terminating the user-reply functionality's management daemon for the logical host from `/etc/rc.shutdown`.

In Linux, HP-UX, and Solaris:

1. Delete from the target directory the `jp1_as_md` script file that was copied from the `/opt/jp1as/sample` directory.
2. If you have created automatic start and termination script files for the logical host, delete them from the target directory.
3. Delete the symbolic link that was created as a link to the `jp1_as_md` script file.
4. If you have created symbolic links to the automatic start and termination script files for the logical host, delete them.

For details about the target directory to which the automatic start and termination script files are to be copied, see *2.8.3(1) Starting and terminating the user-reply functionality's management daemon automatically*. For details about the target directory to which the automatic start and termination script files for the logical host are to be created and the target directory to which the symbolic links are to be created, see *2.9.5(2) Automatic startup and termination of the user-reply functionality's management daemon for the logical host in a non-cluster environment (UNIX only)*.

2.4.3 Using Hitachi Program Product Installer to display version information (UNIX only)

Because the Hitachi Program Product Installer installs the UNIX edition of JP1/Advanced Shell, you can display the JP1/Advanced Shell version information from Hitachi Program Product Installer.

To display the version information:

1. Start the Hitachi Program Product Installer by executing the following command:

```
/etc/hitachi_setup
```

2. In the initial window, enter **L**.

   A list of Hitachi products that have been installed is displayed. Check the displayed version information.

2.5 Specifying environment variables

The table below lists and describes the environment variables supported by JP1/Advanced Shell.

Important note

JP1/Advanced Shell sets and references shell and environment variables whose names begin with ADSH. Therefore, do not use a shell variable or an environment variable whose name begins with ADSH for any purpose other than those described in this manual.

Table 2–11: Environment variables supported by JP1/Advanced Shell

Environment variable name	Information to be specified	Timing of specification when the value is set automatically	Whether a value can be specified
ADSH_AJS_ENVF	Job environment file name for custom jobs	When the job starts as a custom job	Yes#1
ADSH_AJS_GCHE	Check option for custom jobs	When the job starts as a custom job	Yes#1
ADSH_AJS_LHOST	Logical host name for custom jobs	When the job starts as a custom job	Yes#1
ADSH_AJS_SCRF	Job definition script file name for custom jobs	When the job starts as a custom job	Yes#1
ADSH_ENV	Job environment file name	When the job starts as a custom job	Yes#2
ADSH_CMD_ARGORDER#3	Rule for determining the order of the command arguments specified on the command line. The only permitted value is seq. This environment variable has effect in the cut, date, diff, expand, ls, and stat commands. It also has effect in the analysis of user-defined options in the getopt command.	(Not specified automatically.)	Yes
ADSH_CMDEXPR_LENGTH	Character string length. This environment variable is specified when the length operator is used in the expr command. Specify b to acquire the character string length in bytes, and specify c to acquire the character string length in characters. If this environment variable is omitted or a value other than b or c is specified, length is not treated as an operator.	(Not specified automatically.)	Yes
ADSH_JOB_NAME	Job name	When the job starts	No
ADSH_JOBID	Job ID (fixed 6-digit decimal number with leading zeros added)	When the job starts	No

Environment variable name	Information to be specified	Timing of specification when the value is set automatically	Whether a value can be specified
ADSH_JOBRC_FATAL	Job return code in the event of a fatal error that interrupts job processing such as syntax errors. For details about how to specify the environment variable, see *2.6.15(2) ADSH_JOBRC_FATAL environment variable (specifies the return code in the event of an unresumable error in jobs*.	(Not specified automatically.)	Yes#2
ADSH_LANG (UNIX only)#4#6	The language and encoding in which messages are output by JP1/Advanced Shell. Set this environment variable if you want to temporarily change the messages that the adshexec command for a specific job outputs. For details about the values that you can specify, see *2.2.4 Encoding used in JP1/Advanced Shell*. If this environment variable is set within a job definition script or an environment file, the value of such an environment variable is valid only for a child job, root job, or shell operation command (other than the adshexec command) that starts from a job definition script.	(Not specified automatically.)	Yes
ADSH_LANG_JP1EVENT (UNIX only)#5	The language of messages output by JP1 events that are generated by the user-reply functionality. Set this environment variable if you want JP1 event messages to be output in a different language, in accordance with the settings of JP1/IM at the output destination, from the language in which messages are output by JP1/Advanced Shell. For details about the specifiable values, see *2.2.4 Encoding used in JP1/Advanced Shell*. If this environment variable is set within a job definition script, the value of such an environment variable is valid only for a child job, root job, or shell operation command (other than the adshexec command) that starts from a job definition script. If this environment variable is set within an environment file, JP1 event messages from a job definition script are output in the language that conforms to the value.	(Not specified automatically.)	Yes
ADSH_STEP_NAME	Job step name. When a command outside the job step is executed or	When the job step starts	No

Environment variable name	Information to be specified	Timing of specification when the value is set automatically	Whether a value can be specified
ADSH_STEP_NAME	when a job step name is omitted, the environment variable is not defined.	When the job step starts	No
AJS_BJEX_STOP	Interface used for forced termination from JP1/AJS. This environment variable must be defined when JP1/Advanced Shell batch jobs are defined in PC or UNIX jobs. Define the environment variable in PC or UNIX job definitions, not in OS settings.	When the job starts as a custom job	Yes (Only TERM is permitted.)
BLOCKSIZE	Number of bytes per block. This environment variable is used in the ls and stat commands. The default is 512.	(Not specified automatically.)	Yes
COLUMNS	Output width per line of command execution results. This environment variable is used in the -C option of the ls command and the l editing command of the sed command (edit command). This environment variable cannot be defined in job definition scripts.	(Not specified automatically.)	Yes (Cannot be specified in job definition scripts.)
GETOPT_COMPATIBLE	Parameter analysis method. This environment variable is used in the getopt command. There is no rule for the value to be set. If a value is set, JP1/Advanced Shell assumes that the getopt command is specified in format 1 for all arguments to be analyzed.	(Not specified automatically.)	Yes
POSIXLY_CORRECT[#3]	Rule for determining the order of the command arguments specified on the command line. There is no rule for the value to be set. If a value is set, the handling is the same as when seq is specified in the ADSH_CMD_ARGORDER environment variable. This environment variable cannot be defined in job definition scripts.	(Not specified automatically.)	Yes (Cannot be specified in job definition scripts.)
TMPDIR (UNIX only)	Directory to which temporary files are output. This environment variable is used in the diff and sort commands.	(Not specified automatically.)	Yes

#1

These environment variables can be used only in the unit definitions in JP1/AJS's ajsdefine command and in the job definitions in JP1/AJS - Definition Assistant. Do not use these environment variables in the JP1/Advanced Shell's job definition scripts or user environments, such as user profiles and system profiles.

#2

If an environment variable is set within a job definition script or an environment file, the value of such an environment variable is valid only for a child job or root job that is started from a job definition script.

#3

The POSIXLY_CORRECT environment variable has effect in the standard Linux OS commands as well as in the commands in which the ADSH_CMD_ARGORDER environment variable has effect, but the environment variable might have additional functionality than the rule for determining the specification order of the command arguments. Therefore, if the only thing you want to do is to set a rule for determining the specification order for command arguments for UNIX-compatible commands, use the ADSH_CMD_ARGORDER environment variable.

#4

The ADSH_LANG environment variable is prioritized over the LANG environment variable. If the ADSH_LANG environment variable is not specified, messages are output in the language and encoding specified in the LANG environment variable. If neither the ADSH_LANG environment variable nor the LANG environment variable is specified, the value C is assumed.

#5

The ADSH_LANG_JP1EVENT environment variable takes priority over both the ADSH_LANG and the LANG environment variables.

If the ADSH_LANG_JP1EVENT environment variable is not specified, messages are output in the language specified in the ADSH_LANG environment variable. If neither the ADSH_LANG_JP1EVENT nor the ADSH_LANG environment variable is specified, messages are output in the language specified in the LANG environment variable.

If the ADSH_LANG environment variable is set to a value other than C, or if the ADSH_LANG environment variable is not specified and the LANG environment variable is set to a value other than C, messages that are output by JP1 events are output in Japanese. In such a case, if you want JP1 event messages to be output in English, set the ADSH_LANG_JP1EVENT environment variable to C.

#6

If you execute the adshmdctl command with this environment variable specified, messages to the syslog are also output in the language and encoding specified in the ADSH_LANG environment variable.

Depending on the system, outputting the character codes of the language and encoding to the syslog might be impossible. In this case, do not use the adshmdctl command with this environment variable specified.

From job definition scripts, you can reference the default environment variables that are set by the OS and the environment variables that are specified in the export parameter in the environment files, in addition to the environment variables listed in the above table. For details about the export parameter, see *export parameter*.

2.6 Specifying environment information for JP1/Advanced Shell

Once you have installed JP1/Advanced Shell, you must specify environment information by performing the tasks listed below. After you have specified the environment information, you will be able to execute batch jobs based on the specified environment information.

- Specify the JP1/Advanced Shell environment files (environment information) and environment variables, as needed.
  For details about the environment files and environment variable settings, see *2.6.1 Specifying the environment files* through *2.6.15 Defining the return code in the event of an unresumable error in a job*.
  Also, read the following subsections as needed:
  - *2.6.16 Setting up the user-reply functionality*
  - *2.6.17 Checking the JP1 environment (UNIX only)*
  - *2.6.18 Setting up the shell (UNIX only)*
- If you want to change the directories and files used for JP1/Advanced Shell from their default settings, you must create new directories and files.
  For details, see *2.6.19 Creating the directories required for JP1/Advanced Shell*.
- Specify the definition files for collecting maintenance information.
  For details about collecting maintenance information, see *10.3 How to collect information*.

Important note

JP1/Advanced Shell sets and references shell and environment variables whose names begin with `ADSH`. Therefore, do not use a shell variable or an environment variable whose name begins with `ADSH` (in Windows, lower-case representations are included) for any purpose other than those described in this manual.

2.6.1 Specifying the environment files

The two types of environment files are system environment files and job environment files. The supported parameters are the same. The following table explains each type of file.

Table 2–12: Types of environment files

Type of environment file	Description
System environment file	An environment file of this type is common to all systems and is specified by the system administrator. Services and daemons use the settings in a system environment file. This environment file is used automatically when it is stored in the predefined directory.
Job environment file	This environment file is specified for each job by the developer. It includes the following: • Environment file specified in the `ADSH_ENV` environment variable • Job environment file specified in JP1/Advanced Shell Editor's Runtime Environment Settings dialog box • Job environment file specified when JP1/Advanced Shell custom jobs are defined

JP1/Advanced Shell services and daemons use the information defined in system environment files. The information defined in system environment files takes effect at the time a JP1/Advanced Shell service or daemon starts.

Job controllers use the information defined in system environment files and job environment files.

For details about the parameters that can be specified in the environment files, see *7. Parameters Specified in the Environment Files*.

All directories specified in parameters in a system environment file must exist. If you wish to change the default directories, you must create the new directories beforehand.

If you have edited a system environment file in a UNIX environment, check that there are no errors by executing the `adshmdctl` command with the `conftest` option specified.

The following subsections explain how to specify each environment file.

(1) Specifying the system environment files

The system administrator creates and specifies the system environment files. The created system environment files take effect when they are stored on the file paths specified in the following table.

Table 2–13: File names of system environment files

Environment	File name of system environment file
Windows (development environment)	*common-application-data-folder*`\HITACHI\JP1AS\JP1ASD\conf\adshrc.ase`
Windows (execution environment)	*common-application-data-folder*`\HITACHI\JP1AS\JP1ASE\conf\adshrc.ase`
UNIX	`/opt/jp1as/conf/adshrc.ase`

(2) Specifying the job environment files

To use a job environment file to execute batch jobs, specify the file path in the `ADSH_ENV` environment variable. Use the procedure described below to create and specify a job environment file.

In JP1/Advanced Shell Editor, you can specify the path for a job environment file in the Runtime Environment Settings dialog box. When you define a JP1/Advanced Shell custom job, you can specify the path for the job environment file that is to be used.

To create and specify a job environment file:

1. Copy the `sample.ase` environment file sample data from the following directory to a desired directory and file:#
   - Windows execution environment
     installation-folder`\JP1ASE\sample\sample.ase`
   - Windows development environment
     installation-folder`\JP1ASD\sample\sample.ase`
   - UNIX execution environment
     `/opt/jp1as/sample/sample.ase`
2. Specify the required parameters in the copy of the job environment file.
   For details about the parameters that are required in a job environment file, see *7. Parameters Specified in the Environment Files*. Make sure that the encoding of the job environment file matches the value of the `LANG` environment variable in the environment in which job definition scripts are to be run. For details about the encoding of job environment files and the `LANG` environment variable, see *2.2.4 Encoding used in JP1/Advanced Shell*.

3. Specify the path of the created job environment file in the ADSH_ENV environment variable so that the job environment file can be used during batch job execution.

   Use one of the following methods to specify the ADSH_ENV environment variable:

   - OS setting (Windows only)
   - System profile /etc/profile (UNIX only)
   - User profile ($HOME/.profile) (UNIX only)

#

Do not use any of the following characters in a job environment file directory or file name: & ( ) [ ] { } ^ = ; ! ' + , ` ~ # %. If any of these characters is used, JP1/Advanced Shell will not function normally.

2.6.2 Converting path names

Define the path conversion settings as parameters so that the path names used in job definition scripts can be used in both Windows and UNIX.

In JP1/Advanced Shell, you can specify paths in job definition scripts according to the platform as shown in the following.

Table 2–14: Rules for paths supported in Windows and UNIX environments

Item	Windows environment	UNIX environment
Directory separator	\\[#1]	/
Path separator	;	:
Capitalization of path name	Case sensitive[#2]	Case sensitive
Absolute path	A path name begins with *drive-letter*:\\[#1, #3]	A path name begins with /

#1

In a Windows environment, use \\ because \ is treated as an escape character. Alternatively, enclose the entire path name in single quotation marks.

#2

When path names are converted, they are case sensitive also in a Windows environment.

#3

UNC names are also supported. When you define path name conversion in job definition scripts, make sure that the path obtained after conversion will not end with a shared name (including a name ending with \). If a path name ends with a shared name, operation is not guaranteed.

To convert paths according to the above rules, the following definitions are required in the parameters:

- If you want to execute job definition scripts in a Windows environment:

  Define / and : so that the separators used in the UNIX environment can be interpreted correctly.

- If you want to execute job definition scripts in a UNIX environment:

  Define \\ and ; so that the separators used in the Windows environment can be interpreted correctly.

The following explains the parameters used to convert path names.

- PATH_CONV_ENABLE parameter

Enables the path conversion functionality. Specify the path separator and directory separator before conversion. In a Windows environment, define / and :. In a UNIX environment, define \\ and ;.

- PATH_CONV_RULE parameter (Windows only)

  As the path name conversion target, define one of the following:

  - The range enclosed in double quotation marks is the conversion target (path conversion setting 1)
  - All text excluding the range enclosed in single quotation marks is the conversion target (path conversion setting 2)

  If the parameter is omitted or in UNIX, the path conversion setting 1 is applied, in which case only the range enclosed in double quotation marks is converted.

- PATH_CONV parameter

  Defines path name character strings before and after conversion. When job definition scripts are executed, path names are converted according to the rules defined in the parameter. Only the range in a path name that is defined by the PATH_CONV_RULE parameter is converted.

  If a path name matches the conversion character string defined in the PATH_CONV parameter, path separators and directory separators are also converted.

(1) Example of file path conversion (path conversion setting 1)

This example shows how job definition scripts before execution are converted according to the information in the environment file.

- Information in the environment file

  The following shows an example environment file for Windows:

```
#-adsh_conf PATH_CONV_ENABLE / :
#-adsh_conf PATH_CONV_RULE  1
#-adsh_conf PATH_CONV /home/hitachi/bin "C:\\Program Files"  <-1.
#-adsh_conf PATH_CONV /tmp "C:\\temp"                        <-2.
```

- Job definition script before execution

```
#-adsh_path_var DIR,DIR2
"/home/hitachi/bin/myprog1" "/tmp/file"  <-1., 2.

DIR="/home/hitachi/bin"                  <-1.
"$DIR/myprog1" "/tmp/file"               <-2.
DIR2=$DIR
"$DIR2/myprog2" "/tmp/file"              <-2.
```

- Job definition script after execution

  The following results when paths have been converted:

```
"C:\\Program Files\\myprog1" "C:\\temp\\file"  <-1., 2.

DIR="C:\\Program Files"                        <-1.
"$DIR\\myprog1" "C:\\temp\\file"               <-2.
DIR2=$DIR
"$DIR2\\myprog2" "C:\\temp\\file"              <-2.
```

1. Path /home/hitachi/bin is converted to C:\\Program Files according to the PATH_CON parameter definition.

Also, the directory separator is converted from `/` to `\\` according to the `PATH_CONV_ENABLE` parameter definition.

2. Path `/tmp` is converted to `C:\\temp` according to the `PATH_CONV` parameter definition.

   Also, the directory separator is converted from `/` to `\\` according to the `PATH_CONV_ENABLE` parameter definition.

(2) Example of file path conversion (path conversion setting 2)

This example shows how job definition scripts before execution are converted according to the information in the environment file.

- Information in the environment file

  The following shows an example environment file for Windows:

```
#-adsh_conf  PATH_CONV_ENABLE  / :
#-adsh_conf  PATH_CONV_RULE  2
#-adsh_conf  PATH_CONV /home/user01  d:\\home\\user01
#-adsh_conf  PATH_CONV BB/AA BB\\AA
```

- Job definition script before execution

```
#-adsh_job  JOB001
#-adsh_path_var DIR01
echo -E "/home/user01/file"
cat /home/user01/file
DIR01=/home/user01
cat $DIR01/file02               <-1.
PATH=/home/user01/prog:$PATH    <-2.
uap01
DIR02=/home/user01
PATH="$DIR02:/home/user01/prog:$PATH"
AA=10
BB=200
let ANS=BB/AA                   <-3.
echo $ANS
cat BB/AA
```

- Job definition script after execution

  The following results when paths have been converted:

```
#-adsh_job  JOB001
#-adsh_path_var DIR01
echo -E "d:\\home\\user01\\file"
cat "d:\\home\\user01"\\file
DIR01="d:\\home\\user01"
cat "$DIR01"\\file02                       <-1.
PATH="d:\\home\\user01"\\prog";""$PATH"   <-2.
uap01
DIR02="d:\\home\\user01"
PATH="$DIR02;d:\\home\\user01\\prog;$PATH"
AA=10
BB=200
let ANS="BB\\AA"                         <-3.
echo $ANS
cat "BB\\AA"
```

1. Because shell variable `DIR01` that handles path names is defined in the job definition script before conversion, shell variable `DIR01` is enclosed in double quotation marks (") and then a directory separator is added after the shell variable.
2. Because the top of the character string matches the conversion rule, it is enclosed in double quotation marks ("). The path separator is converted to `";"`. Furthermore, because the character string was converted to the path, variable name `PATH` is also enclosed in double quotation marks.
3. Conversion takes place because the arithmetic expression matches the path conversion setting. To suppress this conversion, the job definition script must be corrected with either of the following methods:
    - Enclose the variable in single quotation marks such as `let ANS='BB/AA'`.
    - Add `$` at the beginning of the variable name to be referenced such as `let ANS=$BB/$AA`.

(3) Notes

- If this definition is used to convert path names to obtain Windows path names, the directory separator in path names becomes \. Therefore, if a path name is displayed by the `echo` command unconditionally in a job definition script, \ and the immediately following character are replaced with the escape character.

  If you do not want to replace these characters with the escape character, execute the `echo` command with the `-E` option specified. For details, see *echo command (outputs what is specified in arguments to the standard output)* in *9.3 Standard shell commands*.
- The metacharacters ~, ~+, and ~- cannot be replaced if they are enclosed in quotation marks, or if they are specified immediately before an escape character (\) or immediately before a character string enclosed in quotations marks. For the metacharacters, use the corresponding shell variables by referencing *5.1.6 Metacharacters*.

2.6.3 Converting file paths when files are input and output

The file paths specified in job definition scripts are converted to the file paths that are subject to input and output operations according to the rules defined for file input and output operations. The specified file paths must correspond exactly to the file paths to be input and output.

(1) File path conversion conditions during file input and output operations

When a file input or output operation occurs, the file path is converted by using the redirect characters (<, >, <>, >>).

Input operations also occur in job definition scripts that are run by using the `.` (dot) or `#-adsh_script` command. However, in these job definition scripts, file paths are not converted during file input and output operations. If you want to convert such file paths, define conversion rules in the `COMMAND_CONV_ARG` parameter that converts arguments during command execution.

File path conversion for file input and output operations applies to job definition scripts that are subject to path conversion, as described in *2.6.2 Converting path names*.

You can perform file path conversion during file input and output operations between the different platforms (UNIX ➜ Windows or Windows ➜ UNIX) as well as between the same platforms (UNIX ➜ UNIX or Windows ➜ Windows).

(2) Example of file path conversion during file input and output operations

The following subsections explain how job definition scripts are converted when files are input and output according to the information (PATH_CONV_ACCESS parameter) in the environment file defined for file input and output operations.

(a) Information in the environment file

The following shows an example environment file:

```
#-adsh_conf PATH_CONV_ENABLE / :
#-adsh_conf PATH_CONV /tmp "D:\\tmp"
#-adsh_conf PATH_CONV_ACCESS /dev/null nul
```

(b) Job definition script before execution

The following shows an example job definition script before execution:

```
while read LOG
do
  echo $LOG > /dev/null
done < "/tmp/input.txt"
```

(c) Job definition script during execution

The job definition script is interpreted during execution as follows:

```
while read LOG
do
  echo $LOG > nul
done < "D:\tmp\input.txt"
```

(3) Example of combining the PATH_CONV and PATH_CONV_ACCESS parameters

This subsection presents an example in which the PATH_CONV and PATH_CONV_ACCESS parameters are combined in the Windows edition. If both parameters are specified, the PATH_CONV parameter takes precedence. When the same parameter is specified more than once, the parameters are processed sequentially in the order they are specified.

(a) Contents of environment file

The following example shows the contents of an environment file (with a number assigned to each line):

```
1.   #-adsh_conf PATH_CONV_ENABLE / :
2.   #-adsh_conf PATH_CONV /tmp "C:\\temp"
3.   #-adsh_conf PATH_CONV_ACCESS /tmp/result.log "C:\\jp1as_tmp\
\result3.log"
4.   #-adsh_conf PATH_CONV_ACCESS "C:\temp\result.log" "C:\\jp1as_tmp\
\result4.log"
5.   #-adsh_conf PATH_CONV_RULE 1
```

(b) Contents of job definition script and the conversion method

If the following job definition scripts are executed on the environment file shown in (a), they produce different results.

```
cat data.txt > "/tmp/result.log"
```

In this example, the range enclosed in double quotation marks (") is converted by the PATH_CONV parameter because 1 is specified in the PATH_CONV_RULE parameter.

Because "/tmp/result.log" specified in the cat command is enclosed in double quotation marks ("), it is converted to "C:\\temp\\result.log" by the rule on line 2 in the environment file. Therefore, the rule on line 3 does not apply. The rule on line 4 applies; as a result, this path is further converted to "C:\\jp1as_tmp\\result4.log".

```
cat data2.txt > /tmp/result.log
```

Because /tmp/result.log specified in the cat command is not enclosed in double quotation marks ("), it is not subject to conversion by the PATH_CONV parameter on line 2. The rule on line 3 applies and the path is converted to "C:\\jp1as_tmp\\result3.log".

2.6.4 Converting arguments during command execution

JP1/Advanced Shell converts arguments when commands are executed; this applies to standard shell commands, extended shell commands, extended script commands, reserved script commands, functions, external commands, and user programs. You can perform this conversion between the following platforms:

- Between the same platforms: UNIX → UNIX or Windows → Windows
- Between different platforms: UNIX → Windows or Windows → UNIX

JP1/Advanced Shell analyzes each line of a job definition script according to the defined rules. If a character string in a specified argument exactly matches the character string in an argument of a command to be executed, the character string in the argument is converted to the specified character string. You use the COMMAND_CONV_ARG parameter to specify conversion rules.

(1) Example of combining the PATH_CONV and COMMAND_CONV_ARG parameters

This subsection presents an example in which the PATH_CONV and COMMAND_CONV_ARG parameters are combined in the Windows edition. If both parameters are specified, the PATH_CONV parameter takes precedence. When the same parameter is specified more than once, the parameters are processed sequentially in the order they are specified.

(a) Contents of environment file

The following example shows the contents of an environment file (with a number assigned to each line):

```
1.  #-adsh_conf PATH_CONV_ENABLE / :
2.  #-adsh_conf PATH_CONV /tmp "C:\\temp"
3.  #-adsh_conf COMMAND_CONV_ARG /tmp/data.txt "C:\\jp1as_tmp\\data3.txt"
4.  #-adsh_conf COMMAND_CONV_ARG "C:\temp\data.txt" "C:\\jp1as_tmp\
\data4.txt"
5.  #-adsh_conf PATH_CONV_RULE 1
```

(b) Contents of job definition script and the conversion method

If the following job definition scripts are executed on the environment file shown in (a), they produce different results.

```
cat "/tmp/data.txt" > ./result.log
```

This example specifies `1` in the `PATH_CONV_RULE` parameter; therefore, the range enclosed in double quotation marks (`"`) is converted by the `PATH_CONV` parameter.

Because `"/tmp/data.txt"` specified in the `cat` command is enclosed in double quotation marks (`"`), it is converted to `"C:\\temp\\data.txt"` by the rule on line 2 in the environment file. Therefore, the rule on line 3 does not apply. The rule on line 4 applies; as a result, this path is further converted to `"C:\\jp1as_tmp\\data4.txt"`.

```
cat /tmp/data.txt > ./result.log
```

Because `/tmp/result.log` specified in the `cat` command is not enclosed in double quotation marks (`"`), it is not subject to conversion by the `PATH_CONV` parameter specified on line 2. The rule on line 3 applies and the path is converted to `"C:\\jp1as_tmp\\data3.txt"`.

2.6.5 Defining files to be started as child jobs

You can specify a job definition script as a command name in another job definition script. This enables you to run a job definition script specified in the `adshexec` command as a JP1/Advanced Shell job. This feature is useful in the following cases:

- Migrating a user's existing asset shell scripts from a UNIX environment to a Windows environment
- Executing an existing shell script that is run in the OS's shell in a UNIX environment as a JP1/Advanced Shell job without rewriting its contents

Of the job definition scripts that are executed as descendant processes, those jobs that are executed by using specific environment setting parameters are called child jobs. For details about root jobs and child jobs, see *3.1.2(1) Root jobs and child jobs*. For details about how to execute job definition scripts as child jobs, see *3.2.3 Executing job definition scripts as child jobs*.

If you will be starting job definition script files as child jobs, you must specify in an environment file the conditions for the files to be used. The following provides an overview of the environment setting parameters.

- `CHILDJOB_EXT` parameter

  Defines the extension for a job definition script file that is to be executed as a child job.
- `CHILDJOB_PGM` parameter

  Defines the path to be replaced so that a job definition script file is executed as a child job.
- `CHILDJOB_SHEBANG` parameter

  Defines the path of the executable program of the job definition script file that is to be executed as a child job.

A job definition script file that you create that satisfies the default definition for the `CHILDJOB_SHEBANG` parameter is run as a child job.

For details about the individual parameters, see *7. Parameters Specified in the Environment Files*.

> **Important note**
>
> If you want to run both root and child jobs by using the same environment file parameters, do not change the `ADSH_ENV` environment variable values or the contents of the environment files during job execution.

2.6.6 Specifying definitions for using UNIX-compatible commands

(1) Definitions for using executable UNIX-compatible commands in existing job definition scripts

If you will be using executable UNIX-compatible commands in existing job definition scripts, set the path to the directory in which the UNIX-compatible commands are installed in the `PATH` environment variable. This method eliminates the need for correcting the existing job definition scripts. If there is a command having the same name as a UNIX-compatible command, you can always run the UNIX-compatible command in JP1/Advanced Shell's job definition scripts by specifying the path at the beginning of the `PATH` environment variable value by using the `export` parameter in the environment file. For details about the `export` parameter, see *export parameter (defines an environment variable)* in *7. Parameters Specified in the Environment Files*. Before you run your job definition scripts, make sure that the correct paths have been set in each environment in which the job definition scripts are to be run.

(2) Preparations for using the script-format UNIX-compatible commands (Windows only)

The script-format UNIX-compatible commands use sample script files provided by JP1/Advanced Shell.

Execute the script-format UNIX-compatible commands (such as `chmod` and `su`) according to the sample script file provided by JP1/Advanced Shell.

To execute script-format UNIX-compatible commands:

1. Copy to a desired folder the files that you will be using of the sample script files stored at the following location:
   - Windows execution environment
     installation-folder`\JP1ASE\sample`
   - Windows development environment
     installation-folder`\JP1ASD\sample`

   For the types of sample script files, see *8.5 UNIX-compatible commands (script format) (Windows only)*.

2. Rename the copied files to applicable command names.
   For example, rename sample script files `script_chmod1` and `script_su1` as `chmod` and `su`, respectively. If you want to define a command that does nothing, copy sample script file `script_0` and then rename it.

3. To specify only the file name of the sample script, not its absolute or relative path, do either of the following:
   - Store the sample script to be run in the folder defined in the `PATH` environment variable.
   - Add to the `PATH` environment variable the path of the folder containing the sample script that is to be run.

4. If necessary, define `KNAX6831-I` message output suppression.

If you do not want the KNAX6831-I message to be output after the sample script has run, specify the following coding in the job environment file:

```
#-adsh_conf JOBLOG_SUPPRESS_MSG     KNAX6831-I
```

If you want to suppress output of the KNAX6831-I message for all job definition scripts in the system, specify the above coding in the system environment file.

5. Run the job definition scripts.

   Run the job definition scripts by using the job environment file created in step 4. If you specified the definition in the system environment file, the information specified in step 3 is imported automatically.

2.6.7 Defining the handling of unsupported conditional expressions (Windows only)

The following parameter is used for defining the handling when unsupported conditional expressions are executed in the test command.

- UNSUPPORT_TEST parameter

In a Windows environment, conditional expressions for evaluating the file attributes that are listed below are not supported. If any of these conditional expressions is specified, an error will result. However, by specifying the UNSUPPORT_TEST parameter, you can display a message and handle a specified conditional expression either as an error or as a normal event. The unsupported conditional expressions are as follows:

- -h *file*: Uses a symbolic link for the file.
- -G *file*: Checks if the group to which the file belongs matches the group executing the calling process.
- -L *file*: Uses a symbolic link for the file (same as -h).
- -O *file*: Checks if the owner of the file has a valid user ID for the process.
- *file1* -ef *file2*: Checks if *file1* and *file2* exist and if the entities of *file1* and *file2* are the same (symbolic link targets or hard link targets are the same).

2.6.8 Defining job execution results and log output information

Job execution results are output to the spool directory. You can reference some of the output information as job execution logs. In the event of a problem, you can collect logs and investigate the cause of the problem. In the environment file, define the output destination and contents of these logs.

The following table lists the types of log information that are output while JP1/Advanced Shell is running, and where each type is stored.

Table 2–15: Log information output while running JP1/Advanced Shell and its storage locations

Log information	Information that is output	Storage location
Job execution log	Log of batch jobs	Under the spool root directory
System execution log	Comprehensive JP1/Advanced Shell execution log	Directory specified by the LOG_DIR parameter[#1] in the environment file.

Log information	Information that is output	Storage location
Trace log	JP1/Advanced Shell internal trace log	Directory specified by the `TRACE_DIR` parameter#1 in the environment file.
(UNIX only) Start log#2	Start and end log of the user-reply functionality's management daemon	Under `/opt/jp1as/system`

#1

If the parameter is omitted, the default value is used.

#2

This log information is collected when the user-reply functionality's management daemon is started and stopped.

This log information is stored under the `/opt/jp1as/system` directory with the following file names:

- User-reply functionality's management daemon on a physical host: `adshmd.log`
- User-reply functionality's management daemon on a logical host: `adshmd_`*logical-host-name*`.log`

The following subsections explain the spool output information and how to define output information for each log.

(1) Defining the spool output information

This subsection explains the spool-related parameters for each output information to be defined.

(a) Determining whether the spool job creation suppression functionality is to be used

The spool job creation suppression functionality enables you to prevent the spool directory's disk space usage from increasing continually. It also eliminates the need to delete unneeded directories and files from the spool directory.

Use the `SPOOLJOB_CREATE` parameter to enable the spool job creation suppression functionality. For details, see *SPOOLJOB_CREATE parameter*.

JP1/Advanced Shell always runs with the following settings while the spool job creation suppression functionality is in effect:

- `#-adsh_conf EVENT_COLLECT NO`
  Disables the operation information acquisition functionality.
- `#-adsh_conf OUTPUT_MODE_CHILD MINIMUM`
  Executes child jobs in the minimum output mode.
- `#-adsh_conf OUTPUT_MODE_ROOT MINIMUM`
  Executes root jobs in the minimum output mode.
- `#-adsh_conf SPOOLJOB_CHILDJOB DELETE`
  Deletes the spool jobs for child jobs when the child jobs are terminated.
- `adshexec -m MINIMUM`
  Executes the specified job in the minimum output mode regardless of the specified `-m` option.
- `adshscripttool -exec -m MINIMUM`
  Executes the specified child job in the minimum output mode regardless of the specified `-m` option.

Note that when the spool job creation suppression functionality is used, none of the following functions that use spool job directories can be used:

- `#-adsh_spoolfile` command

If this command is used, the KNAX6385-E message is issued and the script is terminated.

- adshfile command
  If this command is used, the KNAX1880-E message is issued and the command is terminated.
- Collection of job execution logs
  Job execution logs are not collected.
- Operation results collection functionality
  Operation results are not collected.

Even when the spool job creation suppression functionality is in effect, spool directories are still needed.

When the spool job creation suppression functionality is in effect, the following environment setting parameters are ignored:

- EVENT_COLLECT
- JOBEXECLOG_PRINT
- JOBLOG_SUPPRESS_MSG
- OUTPUT_MODE_CHILD
- OUTPUT_MODE_ROOT
- OUTPUT_STDOUT
- PERMISSION_SPOOLJOB_DIR
- PERMISSION_SPOOLJOB_FILE
- SPOOLJOB_CHILDJOB

During CUI debugging, a DBG file with the name shown below is created. This file is deleted automatically when debugging is finished. If deletion of this file fails, an error message is output to the standard error output and to the system execution log.

```
temporary-file-directory/ADSH_DBG_process-ID_job-ID
```

- *temporary-file-directory*
  Temporary file directory defined in the TEMP_FILE_DIR parameter
- *process-ID*
  Process ID consisting of five or more digits
- *job-ID*
  Job ID consisting of six digits

(b) Defining the path name of the spool root directory

The following parameter is used for defining the path name of the spool root directory:

- SPOOL_DIR parameter: Defines the path name of the spool root directory.

If you will be using the user-reply functionality, define the SPOOL_DIR parameter only in the system environment file.

(c) Changing the spool job directory or file permissions (UNIX only)

When a job is terminated, its execution results are output to the spool job directory created for that job. You can use the following parameters to change the permissions for the directory or the files under that directory:

- `PERMISSION_SPOOLJOB_DIR` parameter

  Specify this parameter to change permissions for the spool job directory.

  The default is `700`.

- `PERMISSION_SPOOLJOB_FILE` parameter

  Specify this parameter to change permissions for the files under the spool job directory.

  The default is `600` (in `.DBG` files, `666`).

(d) Defining the standard output and standard error for spool jobs

When a job is executed, JP1/Advanced Shell's information messages, warning messages, and job execution logs are output in addition to the job execution results. The standard output and the standard error output are output to files under the spool job directory.

You specify the simple output mode or the minimum output mode to suppress output of the standard output and the standard error output to files under the spool job directory. You use the following parameters and command options for this specification:

- `OUTPUT_MODE_ROOT` parameter

  Specifies the expansion output mode, simple output mode, or minimum output mode for root jobs.

- `OUTPUT_MODE_CHILD` parameter

  Specifies the expansion output mode, simple output mode, or minimum output mode for child jobs.

- `-m` option of the `adshexec` command

  Specifies the expansion output mode, simple output mode, or minimum output mode for jobs.

- `-m` option of the `adshscripttool` command

  Specifies the simple output mode or minimum output mode for child jobs.

If these parameters and option are omitted, expansion output mode is assumed, in which case the standard output and the standard error are output to files under the spool job directory.

In the simple and minimum output modes, JP1/Advanced Shell's information and warning messages are not output to the standard output or the standard error output. Also, when jobs are terminated, job execution logs are not output to the standard error output. In addition, in the minimum output mode, messages whose output is suppressed are not output to the job execution logs under spool job directories.

For details about the difference in output information among the simple output mode, expansion output mode, and minimum output mode, see *3.3.4 Suppressing output of information and warning messages to job execution logs*.

(2) Defining the information to be output to the job execution log

This subsection explains information related to the job execution log that is to be specified during the environment setup. For details about the information that is output to the job execution log, see *3.4 Job execution log*.

(a) Defining the types of job execution logs to be output to the standard error

When a job is terminated, the information listed below is output as job execution logs to the standard error. The output job execution logs are displayed on the terminal screen used when the `adshexec` command is executed, and in JP1/AJS - View's Execution Results Details dialog box.

- `JOBLOG` file (Messages indicating the job's execution status, including command execution results and file allocation results)
- Job definition script
- Contents of the standard error during job execution

To output only the contents of the standard error during job execution to the standard error, specify the parameter shown in the following to limit the contents of job execution logs to be output:

- `JOBEXECLOG_PRINT` parameter

If a job is executed in JP1/Advanced Shell - Developer or the job controller is started in the debugger mode, `JOBLOG`, standard output, and standard error are output to the console regardless of the specification of this parameter. When the job is terminated, the contents of the job execution log are not output to the standard error.

When a job is executed in the simple output mode or the minimum output mode, job execution logs are not output to the standard error output when the job is terminated regardless of the specification of the `JOBEXECLOG_PRINT` parameter.

(b) Merging the job execution logs for a root job and child jobs

The following parameter enables you to choose whether child jobs' spool jobs are to be deleted when the child jobs are terminated, or to be merged into the root job's spool jobs:

- `SPOOLJOB_CHILDJOB` parameter

If child jobs' spool jobs are merged into the root jobs spool jobs, the child jobs are output in the order they are terminated in such a format that the root job can be identified from the child jobs.

(3) Defining the information to be output to the system execution log

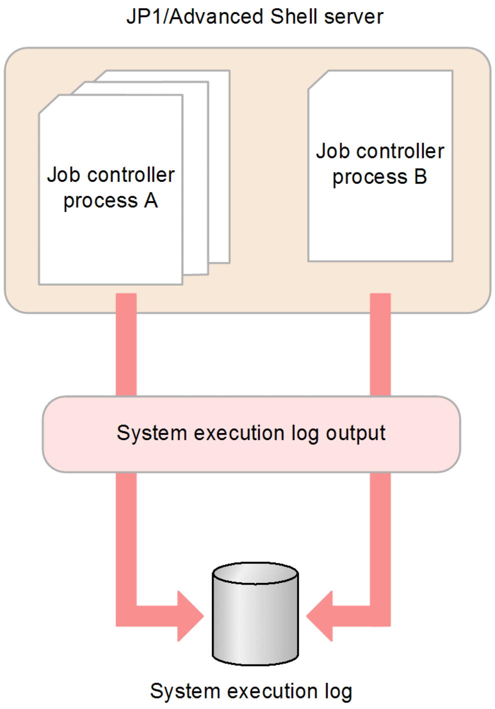
The system execution log provides system administrators with a comprehensive execution history of batch jobs.

The log information is output to `AdshLog.log` under the directory specified by the `LOG_DIR` parameter in the environment file. The files are swapped (`AdshLog_1.log`, `AdshLog_2.log`, ..., `AdshLog_`*`N`*`.log`) according to conditions (such as maximum file size) specified in parameter settings. Because a new system execution log file is created when log files are swapped, the owner of each file will be the user at the time swapping occurs.

(a) Flow of output to the system execution log

The system execution log is the destination for log information about the batch jobs running in each job controller process. You can specify in the environment files the output destination for the system execution log, as well as parameters that control log file swapping (such as maximum file size and number of files). The following figure shows the flow of output to the system execution log.

Figure 2–4: Flow of output to the system execution log



The system execution log is created as follows.

- Messages to be output to the system execution log are collected and output in CSV format.
  For details about the messages that are output, see *11.2 Message output destinations*.
- In time, log file swapping is performed and a backup is created.
  - Just before it exceeds the file size specified in the `LOG_FILE_SIZE` parameter in the environment file, the current system execution log file is renamed so that it becomes a backup file, and a new system execution log is created and message output continues to it.
  - The file name of the backup will be `AdshLog_`*`N`*`.log` (where *N* is an integer). *N* is assigned a number in ascending order from the newest backup, starting from 1.
  - The maximum number of backups to be created is specified in the `LOG_FILE_CNT` parameter in the environment file. When the number of backup files exceeds this value, the oldest backup file is deleted.

(b) Parameters required to output the system execution log

The following parameters are used for outputting system execution logs:

- `LOG_DIR` parameter: Defines the path name of the directory to which system execution logs are to be output.
- `LOG_FILE_CNT` parameter: Defines the number of files used for backing up system execution logs.
- `LOG_FILE_SIZE` parameter: Defines the file size for output of system execution logs.

If multiple users output system execution logs to the same file, the `LOG_FILE_CNT` and `LOG_FILE_SIZE` parameter values specified by the last user who starts output of system execution logs take effect. Therefore, we recommend that you use the same value for `LOG_FILE_CNT` and `LOG_FILE_SIZE`.

(c) Contents of the system execution log

The following shows an example of a message output to the system execution log:

```
seqnum=1, date=2013-12-06T10:41:19.242+09:00, pgmid=adshexec, jobid=6,
pid=2720, msgid=KNAX0004-I, msg="Job ID=000006, JP1NBQSQueueName=,
scheduler job ID="
seqnum=2, date=2013-12-06T10:41:19.250+09:00, pgmid=adshexec, jobid=6,
pid=2720, msgid=KNAX0091-I, msg="JOB1 The job started."
seqnum=3, date=2013-12-06T10:41:19.251+09:00, pgmid=adshexec, jobid=6,
pid=2720, msgid=KNAX7901-I, msg="The adshexec command will wait for all
asynchronous processes at the end of the job."
seqnum=4, date=2013-12-06T10:41:19.251+09:00, pgmid=adshexec, jobid=6,
pid=2720, msgid=KNAX7902-I, msg="The adshexec command will run in tty stdin
mode."
seqnum=5, date=2013-12-06T10:41:19.252+09:00, pgmid=adshexec, jobid=6,
pid=2720, msgid=KNAX0092-I, msg="JOB1.STEP1 step started."
seqnum=6, date=2013-12-06T10:41:19.263+09:00, pgmid=adshexec, jobid=6,
pid=2720, msgid=KNAX6116-I, msg="Execution of the command D:\bin\uap01.exe
(line=5) finished successfully. exit status=0 execution time=0.008s CPU
time=0.000s"
seqnum=7, date=2013-12-06T10:41:19.274+09:00, pgmid=adshexec, jobid=6,
pid=2720, msgid=KNAX6597-I, msg="JOB1.STEP1 step succeeded. exit status=0
execution time=0.022s CPU time=0.015s"
seqnum=8, date=2013-12-06T10:41:19.274+09:00, pgmid=adshexec, jobid=6,
pid=2720, msgid=KNAX0092-I, msg="JOB1.STEP2 step started."
seqnum=9, date=2013-12-06T10:41:19.289+09:00, pgmid=adshexec, jobid=6,
pid=2720, msgid=KNAX6116-I, msg="Execution of the command D:\bin\uap02.exe
(line=10) finished successfully. exit status=0 execution time=0.008s CPU
time=0.015s"
seqnum=10, date=2013-12-06T10:41:19.290+09:00, pgmid=adshexec, jobid=6,
pid=2720, msgid=KNAX6597-I, msg="JOB1.STEP2 step succeeded. exit status=0
execution time=0.016s CPU time=0.015s"
seqnum=11, date=2013-12-06T10:41:19.290+09:00, pgmid=adshexec, jobid=6,
pid=2720, msgid=KNAX0092-I, msg="JOB1.STEP2 step started."
seqnum=12, date=2013-12-06T10:41:19.299+09:00, pgmid=adshexec, jobid=6,
pid=2720, msgid=KNAX6116-I, msg="Execution of the command D:\bin\uap03.exe
(line=15) finished successfully. exit status=0 execution time=0.008s CPU
time=0.000s"
seqnum=13, date=2013-12-06T10:41:19.299+09:00, pgmid=adshexec, jobid=6,
pid=2720, msgid=KNAX6597-I, msg="JOB1.STEP2 step succeeded. exit status=0
execution time=0.009s CPU time=0.000s"
seqnum=14, date=2013-12-06T10:41:19.300+09:00, pgmid=adshexec, jobid=6,
pid=2720, msgid=KNAX0098-I, msg="JOB1 The job ended. exit status=0
execution time=0.053s CPU time=0.030s"
```

The following table lists and explains the data items that are added in front of the message texts in the system execution log:

Data items output to the system execution log	Meaning
`seqnum`	Message's serial number
`date`	Output date and time (in the format *yyyy*-*mm*-*dd*`T`*hh*`:`*mm*`:`*ss*`.`*sssTZD*)
`pgmid`	Program ID In a job controller, `adshexec` is output.
`jobid`	Job ID
`pid`	Process ID

Data items output to the system execution log	Meaning
`msgid`	Message ID of the output message
`msg`	Message text of the output message

(4) Defining the information to be output to trace logs

Trace logs are JP1/Advanced Shell's internal trace logs. In the event of a problem in JP1/Advanced Shell, traces are collected to resolve the problem.

The following table shows the types of trace logs.

Table 2–16: Types of trace logs

No.	Type of trace log	Trace log output destination (default value)	Log file count	File size
1	JP1/Advanced Shell execution environment trace log (Windows and UNIX)# and JP1/Advanced Shell's job definition script operation information output command (`adshevtout` command) trace log#	• Windows *common-application-data-folder* `\Hitachi\JP1AS\JP1ASE\trace` • UNIX `/opt/jp1as/trace`	<<4>> ((1 to 64))	<<2>> ((1 to 16))
2	JP1/Advanced Shell - Developer's non-editor trace log#	*common-application-data-folder*`\Hitachi\JP1AS\JP1ASD\trace`	<<4>> ((1 to 64))	<<2>> ((1 to 16))
3	JP1/Advanced Shell custom job trace log	*common-application-data-folder*`\Hitachi\JP1AS\JP1ASV\trace`	1	1
4	JP1/Advanced Shell - Developer's editor trace log	*common-application-data-folder*`\Hitachi\JP1AS\JP1ASD\adshedit\trace`	1	1
5	JP1/Advanced Shell and JP1/Advanced Shell - Developer's shared commands trace log	*common-application-data-folder*`\Hitachi\JP1AS\misc\trace`	4	2

#
Environment setting parameters can be used to change the trace log output destination, number of log files, and file size.

The following parameters are used for defining trace files:

- `TRACE_DIR` parameter
  Defines the path name of the directory to which traces are to be output. The trace log file names are `AdshTrace_`*`n`*`.log` (*n*: number of files).
- `TRACE_FILE_CNT` parameter
  Defines the number of files to which traces are to be output. The specified number of trace log files are used sequentially and then overwritten in wraparound fashion once all of the files become full.
- `TRACE_FILE_SIZE` parameter
  Defines the file size for output of traces.
- `TRACE_LEVEL` parameter
  Defines the trace output level.

If multiple users output traces to the same file, the `TRACE_FILE_CNT` and `TRACE_FILE_SIZE` parameters are handled as follows:

- The largest user values specified for the `TRACE_FILE_CNT` and `TRACE_FILE_SIZE` parameters take effect.

- If the TRACE_FILE_CNT and TRACE_FILE_SIZE parameter values are changed, the new values are compared with the existing number of trace files and file size, and the largest user values specified for these parameters take effect.

To reduce the number of trace files or the file size, you must delete all the files in the trace folder. Before you delete all files from the trace folder, make sure that no job is outputting traces to the corresponding trace files.

2.6.9 Defining the return codes of extended script commands

The following parameters are used to change the default values for the return codes that indicate whether extended script commands failed or were successful:

- ADSHCMD_RC_ERROR parameter: Defines the return code to be used to indicate that an extended script command failed.
- ADSHCMD_RC_SUCCESS parameter: Defines the return code to be used to indicate that an extended script command was successful.

For details, see *5.8.7 Return codes of extended script commands and handling of errors*.

2.6.10 Sharing among multiple environments

You can run multiple environments on the same host by using the following environment setting parameters to specify different directories:

- LOG_DIR parameter
- SPOOL_DIR parameter
- TEMP_FILE_DIR parameter
- TRACE_DIR parameter

To inherit information to a standby server during cluster operation, any directory to be inherited must be shared among the multiple hosts. In such a case, you must share among the multiple hosts at least the directory specified in the following parameter:

- SPOOL_DIR parameter

2.6.11 Enabling coverage information collection without having to specify the option during batch job execution

By specifying the environment setting parameters listed below for collecting coverage information, you eliminate the need to specify the coverage information collection option (-t option) during batch job execution:

- BATCH_CVR parameter: Uses the coverage auto-acquisition functionality.
- ASC_FILE parameter: Defines the accumulation file naming rules to be used by the coverage auto-acquisition functionality.

The following subsections show example settings in the environment file and the command to be executed.

(1) Example settings in the environment file

This example specifies the following parameters in the environment file:

```
#-adsh_conf BATCH_CVR YES
#-adsh_conf ASC_FILE ./cvrg/ver001-*
```

The following explains each line of the settings:

1. `BATCH_CVR`: Uses the coverage auto-acquisition functionality.
2. `ASC_FILE`: Defines the accumulation file naming rules used by the coverage auto-acquisition functionality.

(2) Command to be executed

This example executes the following command using the above settings in the environment file:

```
adshexec sample.ash
```

This command produces the same results as if the `adshexec` command with `adshexec -t -o ./cvrg/ver001-sample sample.ash` specified were executed. However, if the `adshexec` command with `adshexec -t sample.ash` specified (the `-t` option specified) were executed, the return code would be `1`, resulting in an error.

2.6.12 Migrating job definition scripts from UNIX to Windows

This subsection explains how to migrate UNIX job definition scripts to Windows job definition scripts. Before you perform the procedure, make sure that the encoding of the job definition scripts and environment files matches the value of the `LANG` environment variable that is used on the target platform.

To migrate from UNIX job definition scripts to Windows job definition scripts:

1. Enable the path conversion functionality.
   To convert the separators used in UNIX job definition scripts to those supported by Windows platforms, specify the following parameter in the environment file:
   ```
   #-adsh_conf PATH_CONV_ENABLE / :
   ```
2. Specify the setting needed for converting the specified paths.
   If program paths are specified explicitly in job definition scripts, specify the parameter shown below to convert the paths to paths used in the Windows environment. This example converts the paths of UNIX-compatible commands.
   ```
   #-adsh_conf PATH_CONV /opt/jp1as " "C:\\Program Files\\HITACHI\\JP1AS\
   \JP1ASE"
   ```
3. Specify the setting needed for converting the separators around the shell variables that handle paths.
   If program paths are specified by using shell variables in job definition scripts, add the following command for each job definition script to convert the separators in the paths using shell variables:
   ```
   #-adsh_path_var VAR
   ```
4. Select the path conversion setting. (Windows only)

Specify the path conversion setting by using the PATH_CONV_RULE parameter.

If you want to convert a part enclosed in double quotation marks ("), specify path conversion setting 1 as follows:

```
#-adsh_conf PATH_CONV_RULE 1
```

If you want to convert a part that is not enclosed in double quotation marks (") as well as a part enclosed in double quotation marks, specify path conversion setting 2 as follows:

```
#-adsh_conf PATH_CONV_RULE 2
```

5. Verify that conversion is enabled.

   In path conversion setting 1, verify that the paths to be converted are enclosed in double quotation marks (") as shown in the following example:

   ```
   "$VAR/cmd/ls" -l "/opt/jp1as/sample"
   ```

   In path conversion setting 2, verify that the path that you do not wish to convert is enclosed in single quotation marks (').

6. Check the path conversion results.

   Perform a syntax check on the job definition script (adshexec -c command) and check the path conversion results in the generated script image. If the conversion results are not correct, either change the path conversion setting or edit the job definition script, and then perform a syntax check again.

The following shows examples of path conversion settings 1 and 2.

- Example of conversion using path conversion setting 1

  Environment setting parameters:

  ```
  #-adsh_conf PATH_CONV_ENABLE / :
  #-adsh_conf PATH_CONV_RULE 1
  #-adsh_conf PATH_CONV /opt/jp1as " "c:\\Program Files\\HITACHI\\JP1AS\
  \JP1ASE"
  ```

  Example of conversion:

  ```
  D:\home\user001>"C:\Program Files\Hitachi\JP1AS\JP1ASE\bin\adshexec.exe" -
  c sample1.ash

  ***** D:\home\user001\sample1.ash *****
  0001 : #-adsh_job SAMPLE
  0002 : #-adsh_path_var VAR
  0003 :
  0004 : VAR=/opt/jp1as
  0005 : "$VAR/cmd/ls" -l "/opt/jp1as/sample"
  0006 :

  ***** Converted lines in "C:\home\user001\sample1.ash" *****
  0005 : "$VAR\\cmd\\ls" -l "c:\\Program Files\\HITACHI\\JP1AS\\JP1ASE\
  \sample"
  KNAX7999-I Advanced Shell ended. exit status=0

  D:\home\user001>
  ```

- Example of conversion using path conversion setting 2

  Environment setting parameters:

```
#-adsh_conf PATH_CONV_ENABLE / :
#-adsh_conf PATH_CONV_RULE 2
#-adsh_conf PATH_CONV /opt/jp1as " "c:\\Program Files\\HITACHI\\JP1AS\
\JP1ASE"
```

Example of conversion:

```
D:\home\user001>"C:\Program Files\Hitachi\JP1AS\JP1ASE\bin\adshexec.exe" -
c sample1.ash

***** D:\home\user001\sample1.ash *****
0001 : #-adsh_job SAMPLE
0002 : #-adsh_path_var VAR
0003 :
0004 : VAR=/opt/jp1as
0005 : "$VAR/cmd/ls" -l "/opt/jp1as/sample"
0006 :

***** Converted lines in "D:\home\user001\sample1.ash" *****
0004 : VAR="c:\\Program Files\\HITACHI\\JP1AS\\JP1ASE"
0005 : "$VAR\\cmd\\ls" -l "c:\\Program Files\\HITACHI\\JP1AS\\JP1ASE\
\sample"
KNAX7999-I Advanced Shell ended. exit status=0

D:\home\user001>
```

2.6.13 Loading the files specified in the ENV shell variable

You can use the `KSH_ENV_READ` environment setting parameter to specify whether the `.env` files specified in the `ENV` environment variable are to be loaded when the job controller starts. If this parameter is omitted, the default value depends on the OS, as shown in the following:

- Linux and Windows: `YES` (loads `ENV` files)
- AIX, HP-UX, and Solaris: `NO` (does not load `ENV` files)

2.6.14 Defining the process that will be executing the last command in a pipe (UNIX only)

To define whether the last command in a pipe is to be executed by the current process or another process, specify the `PIPE_CMD_LAST` environment setting parameter as follows:

- `CURRENT`: Executes in the current process.
- `OTHER`: Executes in another process.

If the `PIPE_CMD_LAST` parameter is omitted, `CURRENT` is assumed.

2.6.15 Defining the return code in the event of an unresumable error in a job

If a job is terminated due to an error, such as a memory shortage or a job definition script parsing error, the job controller's return code is set to `1`. You can change this return code to any value from `1` to `255` by setting a value in the `ADSH_JOBRC_FATAL` environment variable.

For details about how to specify the `ADSH_JOBRC_FATAL` environment variable, see *(2) ADSH_JOBRC_FATAL environment variable (specifies the return code in the event of an unresumable error in jobs)*.

(1) Whether the ADSH_JOBRC_FATAL environment variable can be applied

The following table lists the types of errors that might occur while the job controller is running and whether the return code defined in the `ADSH_JOBRC_FATAL` environment variable can be applied.

Table 2–17: Whether the ADSH_JOBRC_FATAL environment variable setting is applied by error type

No.	Timing of error	Error type	Applicability#1
1	When a job is started by the `adshexec` command or a job definition script is debugged from the editor	Errors that prevent the OS from starting execution of the `adshexec` command. For example, this type of error occurs when the load module that is used by the `adshexec` command does not exist.	N
2	Before a job definition script is run	Parsing errors in the `ADSH_JOBRC_FATAL` environment variable	N#2
3		Event file initialization errors	N
4		(UNIX only) Errors that occur during signal reception	N#3
5		(Windows only) Errors that occur if any of the following processes is terminated immediately by a function such as `TerminateProcess`: • `adshexec.exe` • `adshexecsub.exe` • `adshesub.exe` • `adshedit.exe`	N
6		All errors other than those listed in 2 to 5 above. The typical errors are as follows: • Command line parsing errors in the `adshexec` command • Invalid status errors in a job definition script file specified in the `adshexec` command's argument • Environment file parsing errors • Job definition script parsing errors • Initialization errors in the job execution log, system execution log, and trace log • Initialization errors in `asc` files • (Windows only) License check errors • (Windows only) Errors resulting from a reception of control signal (**CTRL** + **C**, **CTRL** + **BREAK**, `CTRL_CLOSE_EVENT`)	Y
7	While a job definition script is running	Errors that do not stop job definition script processing#4	N
8		Errors caused when a job is terminated by using the debugger's command, menu, or button listed below during debugging: • (UNIX only) Executing of the `kill` or `quit` command or re-execution of the `run` command	N

No.	Timing of error	Error type	Applicability[#1]
8	While a job definition script is running	- (Windows only) Selecting the **Quit Debugging** menu, clicking the **Quit Debugging** button, or closing the editor window to cancel debugging	N
9		(UNIX only) Errors that occur during signal reception	N[#3]
10		(Windows only) Errors that occur if any of the following processes is terminated immediately by a function such as `TerminateProcess`: - `adshexec.exe` - `adshexecsub.exe` - `adshesub.exe` - `adshedit.exe`	N
11		This includes all errors other than those listed in 7 to 10 above. The typical errors are as follows: - Special built-in command errors (excluding the `typeset` errors and the errors in the `return` command executed within a function or an external script) - Assignment operation errors[#5] - Errors in variable substitution in a job termination format[#6] - Errors caused by the specification of an out-of-range array element (outside the range from `0` to `65535`) - Resource allocation errors caused by a shortage of memory and disk capacity - Input/output errors - Internal conflict errors - (Windows only) Errors resulting from a reception of control signal (**CTRL** + **C**, **CTRL** + **BREAK**, `CTRL_CLOSE_EVENT`) - (Windows only) Errors resulting from the execution of a conditional expression containing the operator `-h`, `-G`, `-L`, `-O`, or `-ef` (except when a value other than `ERR` is specified in the `UNSUPPORT_TEST` parameter)	Y
12	After execution of a job definition script	(UNIX only) Errors that occur during signal reception	N[#3]
13		The following errors related to files and directories: - Parsing errors in the files allocated by an extended script command - Postprocessing errors in spool job management files - Postprocessing errors in the root job's spool job directory - Post processing errors in event files	N
14		(Windows only) Errors that occur if any of the following processes is terminated immediately by a function such as `TerminateProcess`: - `adshexec.exe` - `adshexecsub.exe` - `adshesub.exe` - `adshedit.exe`	N
15		All errors other than those listed in 12 to 14 above. The typical errors are as follows: - Postprocessing errors in the `asc` files - Child jobs' spool job directory deletion errors - (UNIX only) `DBG` file parsing errors - Internal conflict errors - (Windows only) Errors resulting from a reception of control signal (**CTRL** + **C**, **CTRL** + **BREAK**, `CTRL_CLOSE_EVENT`)	Y

Legend:

Y: The setting of the `ADSH_JOBRC_FATAL` environment variable takes effect.

N: The setting of the `ADSH_JOBRC_FATAL` environment variable does not take effect.

#1

The options specified in the `adshexec` command do not affect whether the setting of the `ADSH_JOBRC_FATAL` environment variable takes effect. For example, if the `adshexec` command with the `-c` option specified is executed and a syntax error occurs, the `ADSH_JOBRC_FATAL` environment variable setting still takes effect.

This applicability depends on the OS as follows:

- In UNIX, the `-d` option is specified in the `adshexec` command

  For the debugger and the jobs subject to debugging that are executed by the `run` command, the following items in the table are checked to determine the applicability:

  Debugger: Items 1 to 6 and 12 to 15

  Jobs subject to debugging: Items 7 to 15

- In Windows, debugging is executed from the editor

  For the jobs to be debugged, items 1 to 15 in the table are checked to determine the applicability.

#2

The return code is `255`.

#3

If the job controller receives a signal and the job terminates with an error, the job's return code is 128 + *the signal number*.

#4

If an extended script command results in an error while a job definition script is running, the subsequent job steps and commands listed below are not executed; however, this is not treated as an unresumable error. Therefore, the setting of the `ADSH_JOBRC_FATAL` environment variable does not take effect.

- Job steps whose `run` attribute is omitted or `normal`
- Instructions outside job steps

#5

This does not apply when an assignment operation is specified in the argument of a regular built-in command.

The table below shows the classification of assignment operation specifications and whether the setting of the `ADSH_JOBRC_FATAL` environment variable is applied. The error indicated in this table occurs if an attempt is made to assign a value to the `NUM` variable that has been defined as being read-only in `readonly NUM`.

Classification of assignment operation specification	Example of error	Applicability
An assignment operation is specified on its own.	`NUM=100`	Y
An assignment operation is specified in the argument of a reserved script command.	`time NUM=100`	Y
An assignment operation is specified in the argument of a special built-in command.	`export NUM=100`	Y
An assignment operation is specified in the argument of a regular built-in command.	`let NUM=100`	N

Legend:

Y: The setting of the `ADSH_JOBRC_FATAL` environment variable takes effect.

N: The setting of the `ADSH_JOBRC_FATAL` environment variable does not take effect.

#6

As a result of variable substitution, the job might become unresumable and be terminated with an error depending on the status of variable. The following shows the status of variable that results in an error for each format:

`${variable:?[word]}`

If `variable` has been defined and its value is null (empty character string) or is undefined, an error results.

`${variable?[word]}`

If `variable` is undefined, an error results.

(2) ADSH_JOBRC_FATAL environment variable (specifies the return code in the event of an unresumable error in jobs)

The `ADSH_JOBRC_FATAL` environment variable is used to specify the job controller's return code in the event a job becomes unresumable and is terminated with an error. The specified return code is applied to jobs that are executed by using the `adshexec` command and to jobs that are executed from JP1/Advanced Shell - Developer's editor.

The following shows how to apply the value of this environment variable globally in the entire system:

- Windows
  Define `ADSH_JOBRC_FATAL` as a system environment variable.
- UNIX
  Specify the `ADSH_JOBRC_FATAL` environment variable setting in `/etc/profile`.

If this environment variable is not specified and a job terminates with an unresumable error, the controller's return code is set to `1`.

(a) Values permitted in the environment variable

termination-code ~<unsigned integer> ((1 to 255))

Specifies the return code to be set when a job cannot be resumed. If the value is padded with leading zeros such as `001`, the leading zeros are deleted and the value is treated as being `1`.

(b) Notes

- If the `ADSH_JOBRC_FATAL` environment variable is defined by using the `export` parameter in the environment file or the `ADSH_JOBRC_FATAL` environment variable is defined or changed within a file or a job definition script specified in the `ENV` shell variable, this functionality does not take effect within that job. The functionality takes effect on another job that is started from that job.
- The `ADSH_JOBRC_FATAL` environment variable defines the final return codes for jobs. It does not affect the return codes of individual commands and job steps.
- If any of the following values is set, the job terminates, without being executed, with an error with return code `255`:
  - Value consisting of four or more characters (example: `1234`)
  - Value outside the permitted range (example: `500`)
  - Non-numeric characters (example: `1A4`, `+8`, `8.0`)
  - Value consisting of no character (null character string)
- Whether the `ADSH_JOBRC_FATAL` environment variable is applied in the event of an error depends on each job. If an unresumable error occurs only within a root job or a child job, the `ADSH_JOBRC_FATAL` environment variable will not be applied to any other root job or child job to change its return code.

(c) Examples

The following shows an example of a UNIX job that was started with `8` set in the `ADSH_JOBRC_FATAL` environment variable and terminated with an unresumable error.

The job could not be resumed because the directory specified in the SPOOL_DIR parameter in the environment file was not found:

Contents of `/etc/profile`:

```
ADSH_JOBRC_FATAL=8
export ADSH_JOBRC_FATAL
```

Command specification at the job start:

```
$ /opt/jp1as/bin/adshexec test.sh
```

The following shows the execution results:

```
KNAX0441-E The directory specified for the parameter "SPOOL_DIR" does not
exist. line=1
KNAX0410-E An error occurred when parsing the environment file
"sample.ase". For details, see the message output before this one.
KNAX0240-I The setting specified for the environment variable
ADSH_JOBRC_FATAL was applied. value=8      ...1.
KNAX7999-I Advanced Shell ended. exit status=8             ...2.
```

The following explains execution results 1 and 2:

1. This is a message indicating that the `ADSH_JOBRC_FATAL` environment variable was applied.
2. The setting of the `ADSH_JOBRC_FATAL` environment variable was applied as the job controller's return code.

A child job was terminated due to an error in the special built-in command (unset command).

Contents of `/etc/profile`:

```
ADSH_JOBRC_FATAL=8
export ADSH_JOBRC_FATAL
```

Contents of the environment file:

```
#-adsh_conf CHILDJOB_SHEBANG /bin/sh
```

Contents of the root job's job definition script (`prt.sh`):

```
./cld.sh
./cmdA
```

Contents of the child job's job definition script (`cld.sh`):

```
#!/bin/sh
val=10
unset
./cmdX $val
```

Command specification at the job start:

```
$ /opt/jp1as/bin/adshexec prt.sh
```

The following shows the execution results:

```
********  JOB CONTROLLER MESSAGE  ********
14:14:22 010467 KNAX0091-I ADSH010467 The job started.
14:14:22 010467 KNAX7901-I The adshexec command will wait for all
asynchronous processes at the end of the job.
14:14:22 010467 KNAX7902-I The adshexec command will run in tty stdin
```

```
mode.
14:14:22 010467 KNAX6831-I The command definition matched the rule
specified by the environment settings parameter CHILDJOB_SHEBANG.
script="./cld.sh" shebang="/bin/sh"
14:14:22 010467 KNAX6521-E The command ./cld.sh (line=1) failed. exit
status=8 execution time=0.008s CPU time=0.000s
14:14:22 010467 KNAX6116-I Execution of the command ./cmdA (line=2)
finished successfully. exit status=0 execution time=0.001s CPU time=0.000s
14:14:22 010467 KNAX0101-E ADSH010467 An error occurred during execution
of the job.
14:14:22 010467 KNAX0098-I ADSH010467 The job ended. exit status=0
execution time=0.012s CPU time=0.000s

********   Script IMAGE    ********

***** /home/usr/work/prt.sh *****
0001 : ./cld.sh
0002 : ./cmdA

***** CONVERSION INFORMATION *****

********   JOB SCOPE STDERR    ********
KNAX7901-I The adshexec command will wait for all asynchronous processes
at the end of the job.
KNAX0724-I The job ID was assigned. job ID=010468
KNAX0101-E ADSH010468 An error occurred during execution of the job.
14:14:22 010468 KNAX6571-I The child job ADSH010468 started. parent
job=ADSH010467 parent job ID=010467
14:14:22 010468 KNAX7901-I The adshexec command will wait for all
asynchronous processes at the end of the job.
14:14:22 010468 KNAX7902-I The adshexec command will run in tty stdin
mode.
14:14:22 010468 KNAX6110-I Execution of the command val=10 (line=2)
finished successfully. exit status=0 execution time=0.000s CPU time=0.000s
14:14:22 010468 KNAX6015-E No argument is specified. filename="/home/usr/
work/cld.sh" line=3
14:14:22 010468 KNAX6521-E The command unset (line=3) failed. exit
status=1 execution time=0.000s CPU time=0.000s
14:14:22 010468 KNAX6584-I A job stopped because a command that
terminates execution of the script was executed.
14:14:22 010468 KNAX0101-E ADSH010468 An error occurred during execution
of the job.
14:14:22 010468 KNAX6578-I The child job ADSH010468 ended. exit status=8
execution time=0.001s CPU time=0.000s
KNAX0240-I The setting specified for the environment variable
ADSH_JOBRC_FATAL was applied. value=8
KNAX0101-E ADSH010467 An error occurred during execution of the job.
KNAX0098-I ADSH010467 The job ended. exit status=0 execution time=0.012s
CPU time=0.000s

******** JOBSTEP OUTPUT ********
KNAX6380-I A job name will be added to the spool job directory of the
root job. spool job directory="/var/opt/jp1as/spool/010467-ADSH010467/"
KNAX7999-I Advanced Shell ended. exit status=0
```

The following explains execution results 1 and 2:

1. The setting of the `ADSH_JOBRC_FATAL` environment variable was applied as the child job's return code.

2. This is a message indicating that the ADSH_JOBRC_FATAL environment variable was applied to the child job.

The following shows an example of a Windows job that was started with 16 set in the ADSH_JOBRC_FATAL system environment variable and terminated with an unresumable error.

The job could not be resumed because the directory specified in the LOG_DIR parameter in the environment file was not found.

This example specifies the following system environment variable and value:

- Variable: ADSH_JOBRC_FATAL
- Value: 16

Command specification at the job start:

```
adshexec test.ash
```

The following shows the execution results:

```
KNAX0441-E The directory specified for the parameter "LOG_DIR" does not
exist. line=1
KNAX0410-E An error occurred when parsing the environment file
"sample.ase". For details, see the message output before this one.
KNAX0240-I The setting specified for the environment variable
ADSH_JOBRC_FATAL was applied. value=16      ...1.
KNAX7999-I Advanced Shell ended. exit status=16             ...2.
```

The following explains execution results 1 and 2:

1. This is a message indicating that the ADSH_JOBRC_FATAL system environment variable was applied.
2. The setting of the ADSH_JOBRC_FATAL environment variable was applied as the job's return code.

A child job was terminated due to an error in the special built-in command (unset command)

This example specifies the following system environment variable and value:

- Variable: ADSH_JOBRC_FATAL
- Value: 16

Contents of the environment file:

```
#-adsh_conf CHILDJOB_SHEBANG /bin/sh
```

Contents of the root job's job definition script (prt.sh):

```
./cld.sh
./cmdA
```

Contents of the child job's job definition script (cld.sh):

```
#!/bin/sh
val=10
unset
./cmdX $val
```

Command specification at the job start:

```
adshexec prt.sh
```

The following shows the execution results:

```
********  JOB CONTROLLER MESSAGE  ********
15:01:30 000004 KNAX0091-I ADSH000004 The job started.
15:01:30 000004 KNAX7901-I The adshexec command will wait for all
asynchronous processes at the end of the job.
15:01:30 000004 KNAX7902-I The adshexec command will run in tty stdin
mode.
15:01:30 000004 KNAX6831-I The command definition matched the rule
specified by the environment settings parameter CHILDJOB_SHEBANG.
script=".\cld.sh" shebang="/bin/sh"
15:01:30 000004 KNAX6521-E The command .\cld.sh (line=1) failed. exit
status=16 execution time=0.197s CPU time=0.077s
15:01:30 000004 KNAX6116-I Execution of the command .\cmdA.exe (line=2)
finished successfully. exit status=0 execution time=0.030s CPU time=0.016s
15:01:30 000004 KNAX0101-E ADSH000004 An error occurred during execution
of the job.
15:01:30 000004 KNAX0098-I ADSH000004 The job ended. exit status=0
execution time=0.233s CPU time=0.093s

********   Script IMAGE    ********

***** D:\work\prt.sh *****
0001 : .\\cld.sh
0002 : .\\cmdA

***** CONVERSION INFORMATION *****

********   JOB SCOPE STDERR    ********
KNAX7901-I The adshexec command will wait for all asynchronous processes
at the end of the job.
KNAX0724-I The job ID was assigned. job ID=000005
KNAX0101-E ADSH000005 An error occurred during execution of the job.
15:01:30 000005 KNAX6571-I The child job ADSH000005 started. parent
job=ADSH000004 parent job ID=000004
15:01:30 000005 KNAX7901-I The adshexec command will wait for all
asynchronous processes at the end of the job.
15:01:30 000005 KNAX7902-I The adshexec command will run in tty stdin
mode.
15:01:30 000005 KNAX6110-I Execution of the command val=10 (line=2)
finished successfully. exit status=0 execution time=0.000s CPU time=0.000s
15:01:30 000005 KNAX6015-E No argument is specified. filename="D:\work
\cld.sh" line=3
15:01:30 000005 KNAX6521-E The command unset (line=3) failed. exit
status=1 execution time=0.000s CPU time=0.000s
15:01:30 000005 KNAX6584-I A job stopped because a command that
terminates execution of the script was executed.
15:01:30 000005 KNAX0101-E ADSH000005 An error occurred during execution
of the job.
15:01:30 000005 KNAX6578-I The child job ADSH000005 ended. exit status=16
execution time=0.012s CPU time=0.016s
KNAX0240-I The setting specified for the environment variable
ADSH_JOBRC_FATAL was applied. value=16
KNAX0101-E ADSH000004 An error occurred during execution of the job.
KNAX0098-I ADSH000004 The job ended. exit status=0 execution time=0.233s
CPU time=0.093s

******** JOBSTEP OUTPUT ********
KNAX6380-I A job name will be added to the spool job directory of the
```

```
root job. spool job directory="C:\Documents and Settings\All Users
\Documents\Hitachi\JP1AS\JP1ASE\spool\003115-ADSH003115\"
KNAX7999-I Advanced Shell ended. exit status=0
```

The following explains execution results 1 and 2:

1. The setting of the ADSH_JOBRC_FATAL environment variable was applied as the child job's return code.
2. This is a message indicating that the ADSH_JOBRC_FATAL system environment variable was applied to the child job.

2.6.16 Setting up the user-reply functionality

To use the user-reply functionality, you must set up an appropriate environment in both JP1/Advanced Shell and JP1/IM. For details about this environment setup, see *2.8 Setting up the user-reply functionality*.

2.6.17 Checking the JP1 environment (UNIX only)

The JP1 environment is determined by /etc/opt/jp1base/conf/jp1bs_param.conf. Specify the character encoding according to the environment being used. For details, see the *Job Management Partner 1/Base User's Guide*.

2.6.18 Setting up the shell (UNIX only)

The table below shows the login shell used when jobs are started from JP1/AJS. Specify the settings so that the correct login shell can be used.

OS type	Login shell
Linux	Bash
AIX	Korn (ksh)
HP-UX	
Solaris	

Notes:

If the adshexec command is run as a child process of the login shell when a job is started from JP1/AJS and then forced termination occurs, the login shell's processing might terminate before the adshexec command's job execution results are passed to JP1/AJS. If this happens, the job execution results might not be applied to JP1/AJS - View.

To avoid this, first (before starting) check the definitions in the login script file to verify that the login shell's process is overwritten (such as by deleting the trap command specification).

For details about the definitions, see the manual *Job Management Partner 1/Automatic Job Management System 3 Configuration Guide 1* or *Job Management Partner 1/Automatic Job Management System 3 Troubleshooting*.

2.6.19 Creating the directories required for JP1/Advanced Shell

If you want to change the default settings for the directories required for execution after you have installed JP1/Advanced Shell, you must create new directories, and then specify them in the environment files.

The directories required for JP1/Advanced Shell and the information to be specified are described below. The user who will be running JP1/Advanced Shell must grant the required permissions to these directories.

- Directory for temporary files
  Specify the directory in which the files to be used only within batch jobs are to be created temporarily.
- Directory for the spool
  Specify the directory used to store job execution logs and program output data files.
- Directory for system execution logs
  Specify the directory used to store batch job logs as system execution logs that are used by the system administrator for monitoring execution of batch jobs.
- Directory for traces
  Specify the directory used to store the statuses for troubleshooting purposes in the event of system failure.

The table below lists the directories required in JP1/Advanced Shell. For details about how to specify the environment setting parameters, see *7. Parameters Specified in the Environment Files*.

Table 2–18: Directories required in JP1/Advanced Shell

Directory	Environment setting parameter	Default directory or path	Default permissions
Directory for temporary files	`TEMP_FILE_DIR`	• Execution environment (Windows only) *shared-documents-folder*`\Hitachi\JP1AS\JP1ASE\temp` • Development environment (Windows only) *shared-documents-folder*`\Hitachi\JP1AS\JP1ASD\temp` • Execution environment (UNIX only) `/var/opt/jp1as/temp`	`CRWD` (Windows) `1777` (UNIX)
Directory for the spool	`SPOOL_DIR`	• Execution environment (Windows only) *shared-documents-folder*`\Hitachi\JP1AS\JP1ASE\spool` • Development environment (Windows only) *shared-documents-folder*`\Hitachi\JP1AS\JP1ASD\spool` • Execution environment (UNIX only) `/var/opt/jp1as/spool`	`CRWD` (Windows) `1777` (UNIX)

Directory for system execution logs	`LOG_DIR`
	`LOG_FILE_CNT`
	`LOG_FILE_SIZE`
		• Execution environment (Windows only)
		shared-documents-folder`\Hitachi\JP1AS\JP1ASE\log`
		• Development environment (Windows only)
		shared-documents-folder`\Hitachi\JP1AS\JP1ASD\log`
		• Execution environment (UNIX only)
		`/opt/jp1as/log`
			`CRWD` (Windows)
			`0777` (UNIX)

Directory for traces	`TRACE_DIR`
	`TRACE_FILE_CNT`
	`TRACE_FILE_SIZE`
	`TRACE_LEVEL`
		• Execution environment (Windows only)
		common-application-data-folder`\Hitachi\JP1AS\JP1ASE\trace`
		• Development environment (Windows only)
		common-application-data-folder`\Hitachi\JP1AS\JP1ASD\trace`
			`CRWD` (Windows)
			`1777` (UNIX)

Directory	Environment setting parameter	Default directory or path	Default permissions

Directory for traces	`TRACE_DIR`
	`TRACE_FILE_CNT`
	`TRACE_FILE_SIZE`
	`TRACE_LEVEL`
		• Custom job definition program (Windows only)
		common-application-data-folder`\Hitachi\JP1AS\JP1ASV\trace`
		• Execution environment (UNIX only)
		`/opt/jp1as/trace`
			`CRWD` (Windows)
			`1777` (UNIX)

Legend:

The letters shown in the *Default permissions* column indicate the following Windows permissions:

`C`: Create, `R`: Read, `W`: Write, `D`: Delete

(1) Required permissions

This subsection describes the permissions required for the users who execute batch jobs.

(a) In Windows

Grant full control to the users who will be executing batch jobs.

(b) In UNIX

Grant to the users who will be executing batch jobs the file permissions shown below for each type of directory.

Table 2–19: File permissions for directories

Directory type	Read permission (r)	Write permission (w)	Execution permission (x)	Sticky bit (t)
Directory for temporary files	R	R	R	S
Directory for the spool	R	R	R	S
Directory for system execution logs	R	R	R	N
Directory for traces	R	R	R	S

Legend:

R: Specification is required.

S: Specify according to system operation guidelines.

N: Do not specify.

Specify the sticky bit for directories according to the system operation guidelines.

If no sticky bit is specified for a directory for which a user has write permission, that user can delete any file directly under that directory.

If a sticky bit is specified for a directory, only the owner of the directory or files can delete any file directly under that directory. No other user can delete these files even if the user has write permission for the directory.

(2) File systems

Because the size of the spool might become large depending on the applications, we recommend that you create and use a dedicated file system. Note that JP1/Advanced Shell does not support NFS or Hitachi Striping File System (HSFS).

2.6.20 Setting up a JP1/AJS environment

To run JP1/Advanced Shell from JP1/AJS, you must set up an execution environment beforehand. This subsection discusses various aspects of the JP1/AJS environment setup.

(1) Estimating the capacity of JP1/AJS logs

When JP1/Advanced Shell is run from JP1/AJS, the amount of internal job execution log data increase by about 350 bytes per job.

In Windows Server 2008 and Windows Server 2012

```
%ALLUSERSPROFILE%\Hitachi\JP1\JP1_DEFAULT\JP1AJS2\log\jpqagent\jpqagt_{00|
01|02|03|04|05|06|07}.log
```

The default value of `%ALLUSERSPROFILE%` is *system-drive*`\Program Data`.

In Windows Server 2003

```
JP1/AJS3-installation-folder\log\jpqagent\jpqagt_{00|01|02|03|04|05|06|
07}.log
```

In UNIX

```
/var/opt/jp1ajs2/log/jpqagent/jpqagt_{00|01|02|03|04|05|06|07}.log
```

To estimate the total size of logs that are output when JP1/Advanced Shell is run from JP1/AJS, use the formula provided in the applicable JP1/AJS manual.

2.6.21 Performing user-specific postprocessing when a job is terminated forcibly

JP1/Advanced Shell enables you to perform user-specific postprocessing when a forced termination request is received from JP1/AJS by means of the UNIX `SIGTERM` signal or the Windows `taskkill` command. This feature enables the user to enhance operational flexibility by performing user-specific termination processing when a forced termination request is received. You must define the `TRAP_ACTION_SIGTERM` environment setting parameter in order to perform user-specific postprocessing when a forced termination request is received.

Note that the operand supported by the `TRAP_ACTION_SIGTERM` environment setting parameter is different between the UNIX edition and the Windows edition. For details about the `TRAP_ACTION_SIGTERM` environment setting parameter, see *TRAP_ACTION_SIGTERM parameter (defines the job controller's action when a forced termination request is received)* in *7. Parameters Specified in the Environment Files*.

The following shows an example:

Contents of the environment variable

```
#-adsh_conf TRAP_ACTION_SIGTERM TERM
```

Contents of the job definition script

```
#-adsh_job JOB01
trap "SIGTERM_received" TERM
UAP01
```

➔ If a forced termination request is received while `UAP01` is running, the job controller executes `SIGTERM_received`, performs postprocessing (such as deleting allocated files and forcibly terminating descendant processes), and then terminates the job.

2.7 Specifying environment information for JP1/AJS (applicable when JP1/AJS is used)

This section explains the specification of environment information when JP1/AJS is used.

You automate job execution in JP1/AJS by registering jobs into JP1/AJS - View. In JP1/AJS - View, the commands and batch files that are used for operations are defined as jobs, and system operations are automated by associating the execution order of those jobs.

JP1/AJS - View supports definitions for the following types of jobs:

- Custom jobs
- PC jobs (for Windows)
- UNIX jobs (for UNIX)

For details about JP1/AJS - View, see the *Job Management Partner 1/Automatic Job Management System 3 Operator's Guide*.

2.7.1 Registering custom jobs in JP1/AJS - View

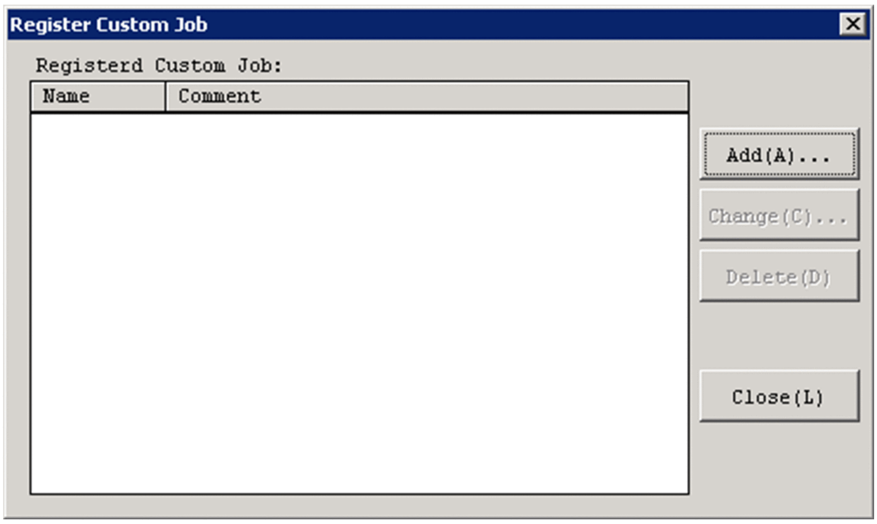
Use of custom jobs makes job definition easier and more accurate compared with directly specifying commands and batch files in jobs.

Custom jobs are templates for jobs that make job creation easy when jobs that link JP1/AJS - View and other programs are defined.

With custom jobs, you can use GUI to define jobs for JP1/Advanced Shell.

To register custom jobs into JP1/AJS - View:

1. From the Windows **Start** menu, select **All Programs**, **JP1_Automatic Job Management System 3 - View**, and then **Register Custom Jobs**.
   The Register Custom Job dialog box is displayed.

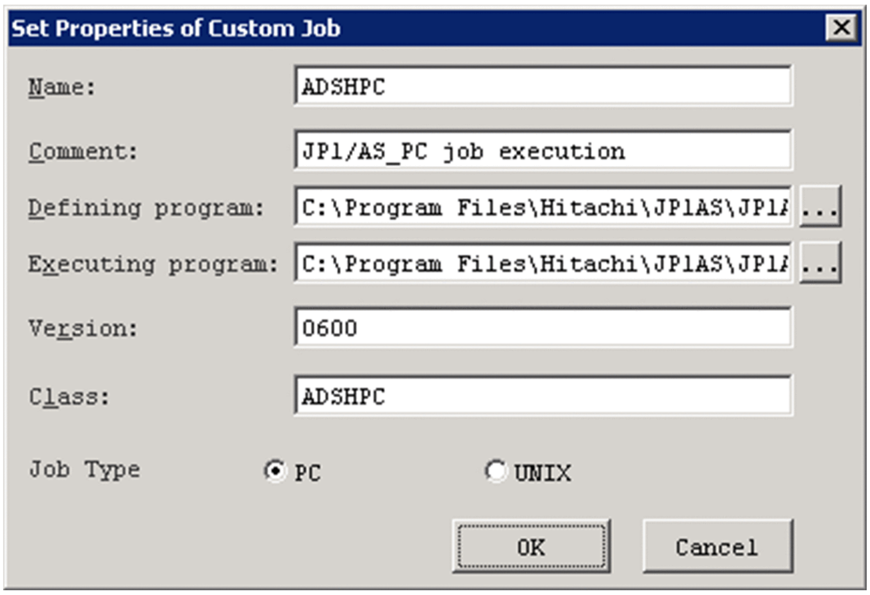


2. Click the **Add** button.

The Set Properties of Custom Job dialog box is displayed.

3. Register a custom job for JP1/Advanced Shell. You register a custom job for a Windows execution environment and for a UNIX execution environment as shown in the following.

- In Windows



Set Properties of Custom Job

Name: ADSHPC

Comment: JP1/AS_PC job execution

Defining program: C:\Program Files\Hitachi\JP1AS\JP1

Executing program: C:\Program Files\Hitachi\JP1AS\JP1

Version: 0600

Class: ADSHPC

Job Type ● PC ○ UNIX

Name: Specify `ADSHPC`.

Comment: We recommend that you specify the fixed character string `JP1/AS_PC job execution`. You can specify any character string up to 40 bytes in length or you can omit this field.

Defining program: *installation-folder*`\JP1ASV\bin\adshctmjpc.bat`. This is the folder on the PC on which the custom job is installed.

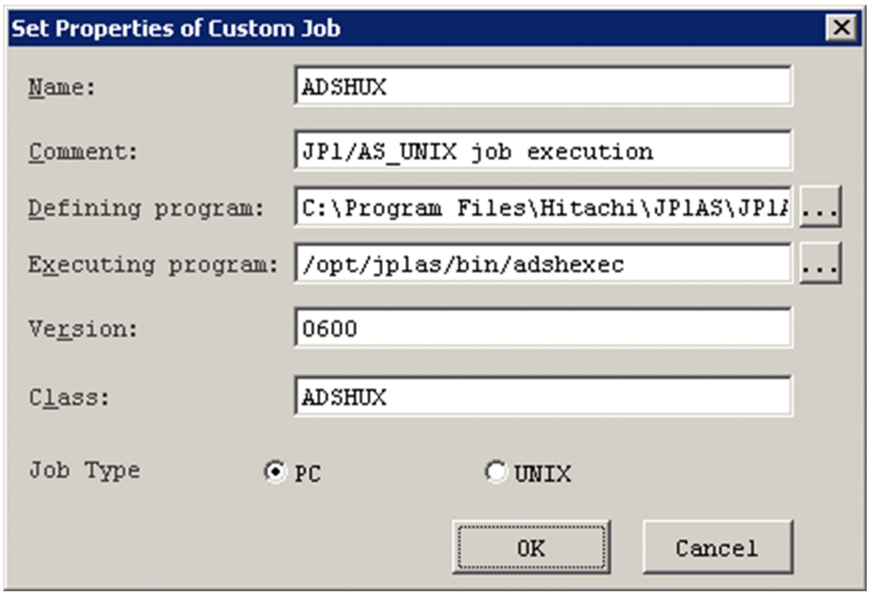
Executing program: *installation-folder*`\JP1ASE\bin\adshexec.exe`. This is the folder on the PC on which the execution environment is installed.

Version: `0600`. This is the interface version of JP1/AJS - View.

Class: Specify `ADSHPC`.

Job Type: For a job for JP1/Advanced Shell, always select **PC**.

- In UNIX



Set Properties of Custom Job

Name: ADSHUX

Comment: JP1/AS_UNIX job execution

Defining program: C:\Program Files\Hitachi\JP1AS\JP1

Executing program: /opt/jp1as/bin/adshexec

Version: 0600

Class: ADSHUX

Job Type ● PC ○ UNIX

Name: Specify `ADSHUX`.

Comment: We recommend that you specify the fixed character string `JP1/AS_UNIX job execution`. You can specify any character string up to 40 bytes in length or you can omit this field.

Defining program: *installation-directory*`\JP1ASV\bin\adshctmjunix.bat`. This is the folder on the PC on which the custom job is installed.

Executing program: `/opt/jp1as/bin/adshexec`

Version: `0600`. This is the interface version of JP1/AJS - View.

Class: Specify `ADSHUX`.

Job Type: For a job for JP1/Advanced Shell, always select **PC**.

4. Click the **OK** button.

   The custom job is registered into JP1/AJS - View.

If you are installing the custom job definition program while a JP1/AJS3 - View whose version is earlier than 09-50 is installed, copy the custom job icons to the following folder:

```
JP1/AJS-View-installation-folder\image\custom
```

The following table explains the icons used in JP1/AJS.

Table 2–20: Explanation of icons used in JP1/AJS

Icon	name	File name	Description
	Job icon for execution in Windows	`CUSTOM_PC_ADSHPC.gif`	This is the Windows custom job icon used in JP1/AJS - View's jobnet editor. It is displayed on the **Custom Jobs** page in the Jobnet Editor window.
	Job icon for execution in UNIX	`CUSTOM_PC_ADSHUX.gif`	This is the UNIX custom job icon used in JP1/AJS - View's jobnet editor. It is displayed on the **Custom Jobs** page in the Jobnet Editor window.
	JP1/Advanced Shell (execution definition) icon	`adshctmj.exe`	This is the definition program that links JP1/AJS in custom jobs.

Notes:

If JP1/Advanced Shell is installed on multiple machines and their paths differ from one machine to another, specify the JP1/Advanced Shell installation path in a variable and then specify the variable's name, instead of the full path, in **Executing program** during custom job registration. For details about the specification method, see *Defining the work path used during job execution as a variable* in the *Job Management Partner 1/Automatic Job Management System 3 Configuration Guide 1*.

The following shows an example specification:

```
jajs_config -k "[JP1_DEFAULT\JP1NBQAGENT\Variable]" "jp1asebin"="C:\Program Files\Hitachi\JP1AS\JP1ASE\bin"
```

You can share multiple paths in JP1/AJS - View by specifying the path of JP1/Advanced Shell as a variable in JP1/AJS on each machine on which JP1/Advanced Shell is installed and then specifying `$jp1asebin$\adshexec.exe` in **Executing program** during custom job registration in JP1/AJS - View.

2.7.2 Defining and executing a jobnet

To automate job execution in JP1/AJS, you can define registered custom jobs, PC jobs (for Windows), or UNIX jobs (for UNIX) into a jobnet in JP1/AJS - View and then execute the jobnet. For details about JP1/AJS - View, see the description of job definition in the *Job Management Partner 1/Automatic Job Management System 3 Operator's Guide*.

To define and execute a jobnet in JP1/AJS3 - View:

1. From the Windows **Start** menu, select **All Programs, JP1_Automatic Job Management System 3 - View**, and then **Job Management System**.

   The JP1/AJS3 - View - Login window is displayed.

2. To log in, specify your user name, password, and the host to connect.

   The JP1/AJS3 - View window is displayed.

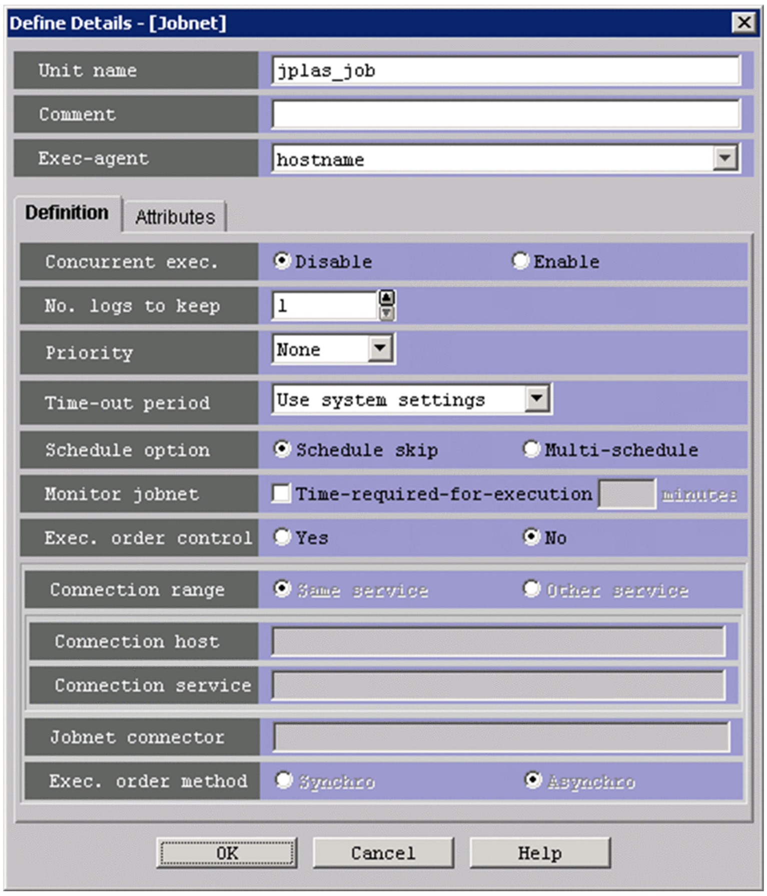
3. Select **Edit**, **New**, then **Jobnet**.

   The Define Details - [Jobnet] dialog box is displayed.

4. Specify information including attributes of the jobnet, and then click the **OK** button.

   Specify the appropriate information in **Exec-agent** according to the operating environment. This information can be omitted. For details about the JP1/AJS items, see the applicable JP1/AJS manual.

   The jobnet is created and displayed in the list area.



Define Details - [Jobnet]

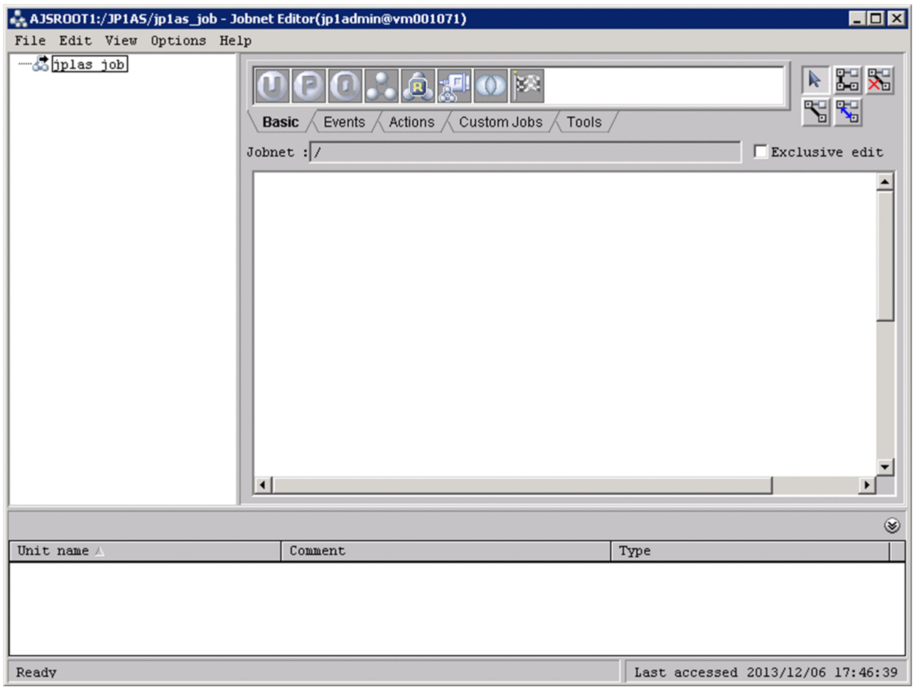
Unit name	jplas_job
Comment	
Exec-agent	hostname

Definition | Attributes

Concurrent exec.	● Disable	○ Enable
No. logs to keep	1	
Priority	None	
Time-out period	Use system settings	
Schedule option	● Schedule skip	○ Multi-schedule
Monitor jobnet	□ Time-required-for-execution minutes	
Exec. order control	○ Yes	● No
Connection range	● Same service	○ Other service
Connection host		
Connection service		
Jobnet connector		
Exec. order method	○ Synchro	● Asynchro

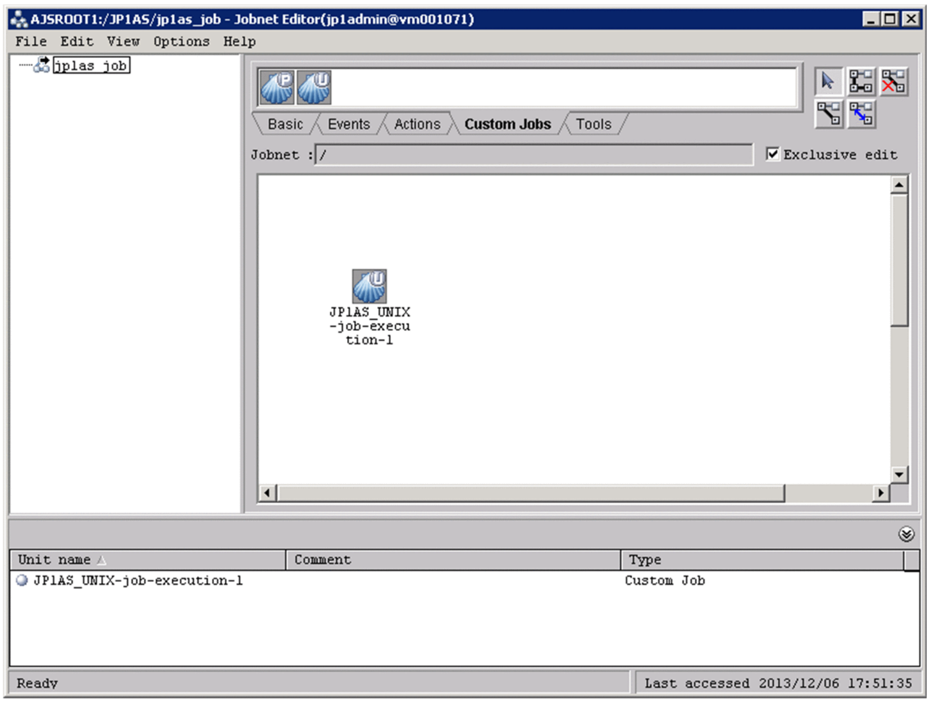
5. Double-click the created jobnet.

   The Jobnet Editor window is displayed.


AJSROOT1:/JP1AS/jp1as_job - Jobnet Editor(jp1admin@vm001071)
File Edit View Options Help
jp1as_job
Basic
Events
Actions
Custom Jobs
Tools
Jobnet :
/
□ Exclusive edit
Unit name
Comment
Type
Ready
Last accessed 2013/12/06 17:46:39

6. Select **Exclusive edit** so that no other user can access the job while you are defining and associating it.

7. Drag the required custom job, PC job, or UNIX job icon from the icon list to the map area.

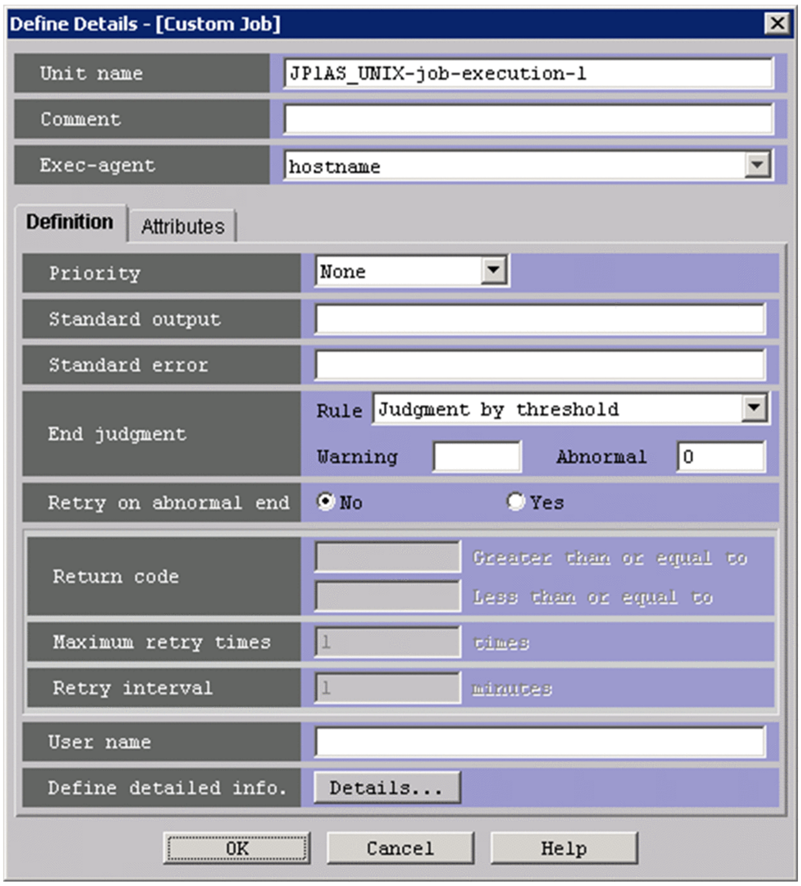

AJSROOT1:/JP1AS/jp1as_job - Jobnet Editor(jp1admin@vm001071)
File Edit View Options Help
jp1as_job
Basic
Events
Actions
Custom Jobs
Tools
Jobnet :
/
☑ Exclusive edit
JP1AS_UNIX
-job-execu
tion-1
Unit name
Comment
Type
Ready
Last accessed 2013/12/06 17:51:35

Unit name	Comment	Type
JP1AS_UNIX-job-execution-1		Custom Job

The Define Details - [Custom Job], Define Details - [PC Job], or Define Details - [UNIX Job] dialog box is displayed. The steps below explain the definition method for custom jobs. For details about the settings required to use PC or UNIX jobs in JP1/Advanced Shell, see *2.7.3 Defining jobs as PC or UNIX jobs*.

8. In the Define Details - [Custom Job] dialog box, define information including job attributes.

   Specify the appropriate information on the **Definition** and **Attributes** pages as described in the applicable JP1/AJS manual. Also specify the appropriate information in **Exec-agent** according to the operating environment. This information can be omitted.



Define Details - [Custom Job]

Unit name	JP1AS_UNIX-job-execution-1
Comment	
Exec-agent	hostname

Priority	None
Standard output	
Standard error	
End judgment	Rule: Judgment by threshold Warning: Abnormal: 0
Retry on abnormal end	● No ○ Yes
Return code	Greater than or equal to Less than or equal to
Maximum retry times	1 times
Retry interval	1 minutes
User name	
Define detailed info.	Details...

If you specify **Standard output** on the **Definition** page while one of the following is specified, an empty file is output:

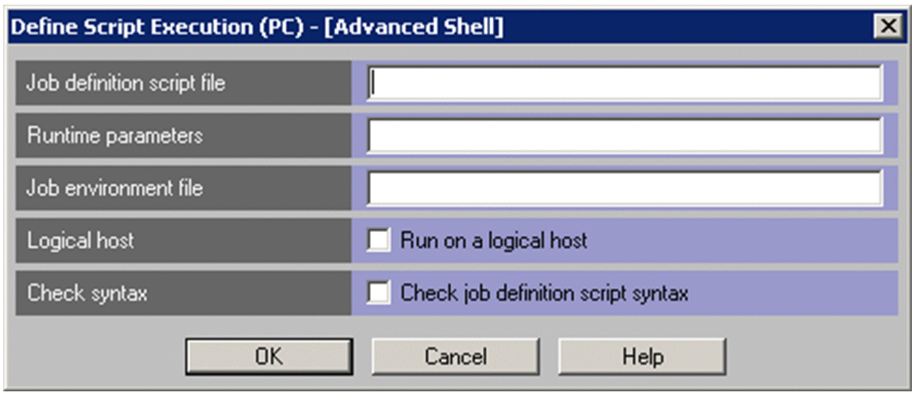
- The `-s` option is specified in the `adshexec` command or `SPOOL` is specified in the `OUTPUT_STDOUT` parameter in the environment file (the standard output is redirected to spool files).
- `EXTENDED` is specified in the `OUTPUT_MODE_ROOT` parameter (the expansion output mode is selected).

This is because the contents of the standard output are output to a separate file by JP1/Advanced Shell's job controller and nothing is output to the standard output that is returned to JP1/AJS.

9. Select the **Definition** tab, and then click the **Details** button.

   The Define Script Execution dialog box for the current type of custom job is displayed. The following shows the dialog box for Windows and for UNIX.

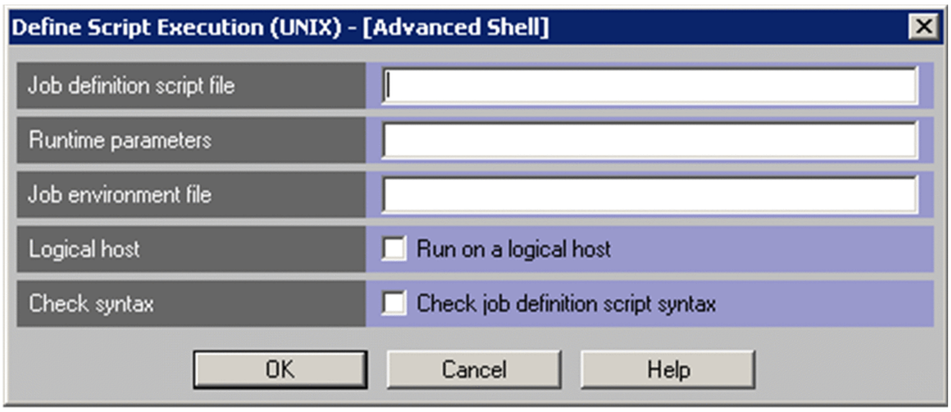
   - In Windows



Define Script Execution (PC) - [Advanced Shell]

Job definition script file	
Runtime parameters	
Job environment file	
Logical host	□ Run on a logical host
Check syntax	□ Check job definition script syntax

   - In UNIX



Define Script Execution (UNIX) - [Advanced Shell]

Job definition script file

Runtime parameters

Job environment file

Logical host: □ Run on a logical host

Check syntax: □ Check job definition script syntax

For more information about steps 9 and 10, also see *(2) Supplementary information about the jobnet definition procedure for custom jobs*.

10. In the Define Script Execution dialog box, specify the information required for JP1/Advanced Shell execution, and then click the **OK** button.

    - **Job definition script file** ~<path name> Windows: ((1 to 247 bytes)), UNIX: ((1 to 1,023 bytes))

      Specifies the name of the job definition script file. This item cannot be omitted.

    - **Runtime parameters** ~<ASCII character string>((0 to 1,022 bytes))

      Defines the parameters that are to be passed when the job definition script file is executed. If you specify multiple parameters, use the space as the delimiter.

    - **Job environment file** ~<path name> Windows: ((0 to 247 bytes)), UNIX: ((0 to 1,023 bytes))

      Specifies the name of the job environment file. When this item is specified, the system uses the specified job environment file, even if the `ADSH_ENV` environment variable is specified in JP1/Advanced Shell's job controller environment. If this item is omitted but the `ADSH_ENV` environment variable is defined in JP1/Advanced Shell's job controller environment, the system uses the specified `ADSH_ENV` environment variable value during execution. If this item is omitted and the `ADSH_ENV` environment variable is not defined in JP1/Advanced Shell's job controller environment, the system assumes that no job environment file is specified. JP1/Advanced Shell's job controller that was started by JP1/AJS sets the path of the job environment file that it used in the `ADSH_ENV` environment variable and then starts job execution. If another job controller is started as a descendant process, that job controller uses the value of the `ADSH_ENV` environment variable. Therefore, if the value of the `ADSH_ENV` environment variable is not set during job execution, the job controller uses the same job environment file that was used by JP1/Advanced Shell's job controller that was started by JP1/AJS. If the value of the `ADSH_ENV` environment variable is set during job execution, the job controller uses the job environment file containing the new value.

    - **Logical host**

      Specifies whether the job is to be executed on a logical host. If the **Run on a logical host** check box is selected, the job is executed on the logical host specified in JP1/AJS (value of the `JP1_HOSTNAME` environment variable).

    - **Check syntax**

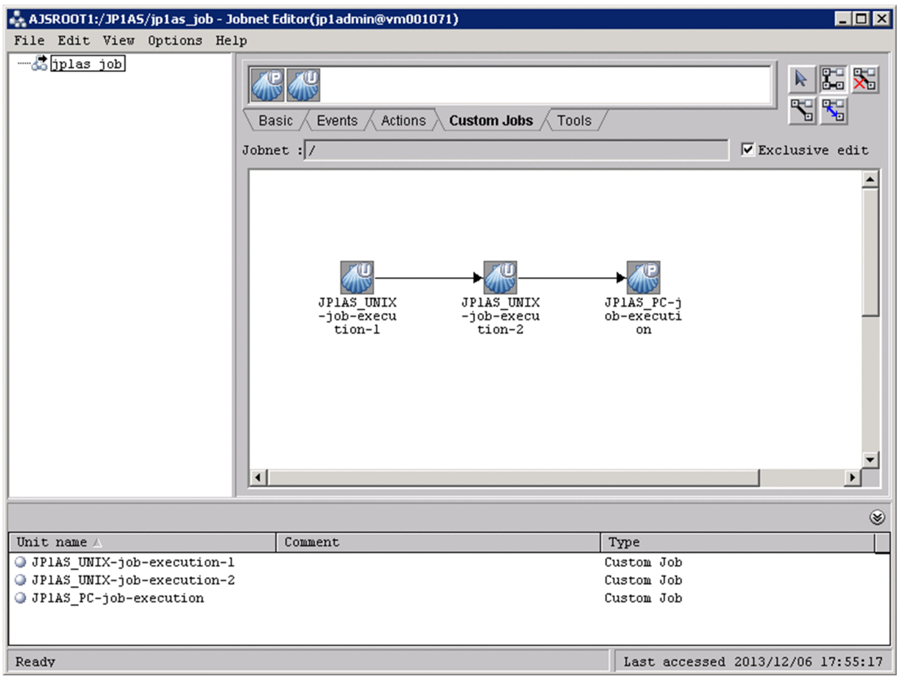
      Specifies whether the job's contents are to be checked. When the **Check job definition script syntax** check box is selected, the contents of the job definition script file are checked but the job definition script file is not executed.

11. Display the Define Details - [Custom Job] dialog box again, and then click the **OK** button.

    The job is defined in the jobnet. If necessary, define another job in the same manner.

12. Associate the jobs according to their execution order.

    The jobnet is defined. The following shows an example job definition in JP1/AJS - View.



AJSROOT1:/JP1AS/jp1as_job - Jobnet Editor(jp1admin@vm001071)

Jobnet :/ ☑ Exclusive edit

Unit name	Comment	Type
JP1AS_UNIX-job-execution-1		Custom Job
JP1AS_UNIX-job-execution-2		Custom Job
JP1AS_PC-job-execution		Custom Job

Ready Last accessed 2013/12/06 17:55:17

13. Execute the jobnet by using JP1/AJS.

    JP1/Advanced Shell's job controller returns its return code as the job return code to JP1/AJS.

(1) Notes about jobnet definitions

- Specification of coverage

  If you want to enable coverage from custom jobs, specify the coverage auto-acquisition functionality in the environment file.

- Run-time directory to be used when JP1/Advanced Shell's job controller is started from JP1/AJS

  When JP1/Advanced Shell's job controller is started from JP1/AJS, the run-time directory is set to the one that is used when JP1/AJS - Agent (or JP1/AJS - Manager) starts JP1/Advanced Shell's job controller. For details about the run-time directory that is used when JP1/AJS - Agent (or JP1/AJS - Manager) starts JP1/Advanced Shell's job controller, see the manual *Job Management Partner 1/Automatic Job Management System 3 Configuration Guide 1*. In the JP1/AJS manuals, the run-time directory is referred to as a work path (work directory).

- Environment variables to be used when JP1/Advanced Shell's job controller is started from JP1/AJS

  Normally, when JP1/Advanced Shell's job controller for Windows is started from JP1/AJS, the system environment variable settings are enabled when the JP1/AJS services are started and no user environment variables are loaded. For details, see the applicable JP1/AJS manual.

- Connecting to an overseas version of JP1/AJS - Manager whose language is set to English

  When connecting to an overseas version of JP1/AJS - Manager whose language is set to English, in the Define Script Execution dialog box, in the definition information, use only ASCII alphanumeric characters.

(2) Supplementary information about the jobnet definition procedure for custom jobs

With respect to the job execution settings in steps 9 and 10 in *2.7.2 Defining and executing a jobnet*, you can also specify the unit definition in the `ajsdefine` command in JP1/AJS and the job definition in JP1/AJS - Definition Assistant.

For details about the unit definition, see the manual *Job Management Partner 1/Automatic Job Management System 3 Command Reference 2*.

For details about JP1/AJS - Definition Assistant, see the manual *JP1/Automatic Job Management System 3 - Definition Assistant*.

The following provides the details of the environment variables and parameters that are specified in the unit definition and JP1/AJS - Definition Assistant when custom jobs are used.

- Environment variables specified in JP1/Advanced Shell's Define Script Execution dialog box:
  `ADSH_AJS_SCRF`: Job definition script file name
  `ADSH_AJS_ENVF`: Job environment file name
  `ADSH_AJS_LHOST`: Logical host
  `ADSH_AJS_GCHE`: Check syntax
- Environment variables passed to JP1/Advanced Shell's job controller
  Define the environment variables as `env` parameters in the unit definition file.
  The following table describes the environment variable items that are passed to JP1/Advanced Shell's job controller.

Table 2–21: Environment variable items that are passed to JP1/Advanced Shell's job controller

Item name	Description
Input range	Length (in bytes) of a character string that can be specified as an environment variable value (character string following =) (Shift-JIS)
Setting	Type of characters that can be defined • Character string: Characters other than control characters (`0x00` through `0x1f`, `0x7f`) • Symbolic name: Single-byte alphanumeric characters and `@`, `#`, `_` • Numeric characters
Initial value	Value to be loaded when the custom job is started as a new job
Omission	If a required value is omitted, an error occurs in JP1/Advanced Shell's job controller.

The following table shows the input range and settings of environment variable items that are passed to JP1/Advanced Shell's job controller.

Table 2–22: Input range and settings of environment variable items that are passed to JP1/Advanced Shell's job controller

Environment variable	Input range	Setting	Initial value	Omission
`ADSH_AJS_SCRF`	PC job: 1 to 247 bytes UNIX job: 1 to 1,023 bytes	Character string	Null character string	Not permitted
`ADSH_AJS_ENVF`	PC job: 0 to 247 bytes UNIX job: 0 to 1,023 bytes	Character string	Null character string	Permitted
`ADSH_AJS_LHOST`	0 to 2 bytes	• If syntax is checked: `-h` • If syntax is not checked: Null character string	Null character string	Permitted
`ADSH_AJS_GCHE`	0 to 2 bytes	• If syntax is checked: `-c` • If syntax is not checked:	Null character string	Permitted

Environment variable	Input range	Setting	Initial value	Omission
ADSH_AJS_GCHE	0 to 2 bytes	Null character string	Null character string	Permitted
AJS_BJEX_STOP	4 bytes	"TERM"	"TERM"	Not permitted

- Details of the parameters that are passed to JP1/Advanced Shell's job controller

  The field **Runtime parameters** is defined as the prm parameter in the unit definition file and passed as a parameter for JP1/Advanced Shell's job controller. The permitted values are as follows:

  - Input range: 1 to 1,023 bytes#
  - Setting: Character string
  - Initial value: Null character string
  - Omission: Permitted

  To define for run-time parameters in the Define Script Execution dialog box a value equivalent to the null character string, specify a one-byte space in the prm parameter in the unit definition file.

  #

  The maximum length of 1,023 bytes applies only when a character string consisting only of spaces is specified in the prm parameter. If any non-space characters are specified, the permitted maximum length is 1,022 bytes. If a character string consisting only of spaces is specified in the prm parameter, the specified character string minus a one-byte space is displayed in the **Runtime parameters** text box in the Define Script Execution dialog box.

  ### Important note

  Note the following about deleting definition information from the unit definition file:

  - For env parameters, delete the environment variable value (character string following =) or delete the entire env parameter.
  - For prm parameters, specify a one-byte space as the value of prm or delete the prm parameter specification.

The following shows an example unit definition:

```
unit=unit-name,,executing-user,;
{
    ty=cpj;
    cty="ADSHUX";
    sc="/opt/jp1as/bin/adshexec";
    env="AJS_BJEX_STOP=TERM";                                -->1.
    env="ADSH_AJS_SCRF=/tmp/JP1AS/scr/samplescrfile.ash";   -->2.
    prm="param1 param2";                                     -->3.
    env="ADSH_AJS_ENVF=/tmp/JP1AS/env/sampleenvfile";       -->4.
    env="ADSH_AJS_LHOST=-h";                                 -->5.
    env="ADSH_AJS_GCHE=-c";                                  -->6.
    tho=0;
}
```

The following explains the lines of code based on the numbers shown in the right margin:

1. AJS_BJEX_STOP: Because the AJS_BJEX_STOP environment variable is used by the system, make sure that you define this environment variable and specify TERM as its value.

2. `ADSH_AJS_SCRF`: Specifies the name of the job definition script file.
   Make sure that you define the `ADSH_AJS_SCRF` environment variable.

3. Specifies the `prm` parameter in the unit definition file.

4. `ADSH_AJS_ENVF`: Specifies the name of the job environment file.

5. `ADSH_AJS_LHOST`: If the job is to be executed on a logical host, specify `-h`; if not, specify nothing.

6. `ADSH_AJS_GCHE`: If only syntax checking is to be performed, specify `-c`; otherwise, specify nothing.

After you have created a unit definition file, you can define the job by using JP1/AJS's `ajsdefine` command and JP1/AJS3 - Definition Assistant.

2.7.3 Defining jobs as PC or UNIX jobs

(1) Defining jobs as PC jobs

This subsection explains the items required to define JP1/Advanced Shell batch jobs as PC jobs.

- File name
  Specify the path of the `adshexec` command in **File name** on the **Definition** tab in the Define Details - [PC Job] dialog box or in `sc="`*script-file-name*`"` in the unit definition file:

  ```
  "installation-folder\JP1ASE\bin\adshexec.exe"
  ```

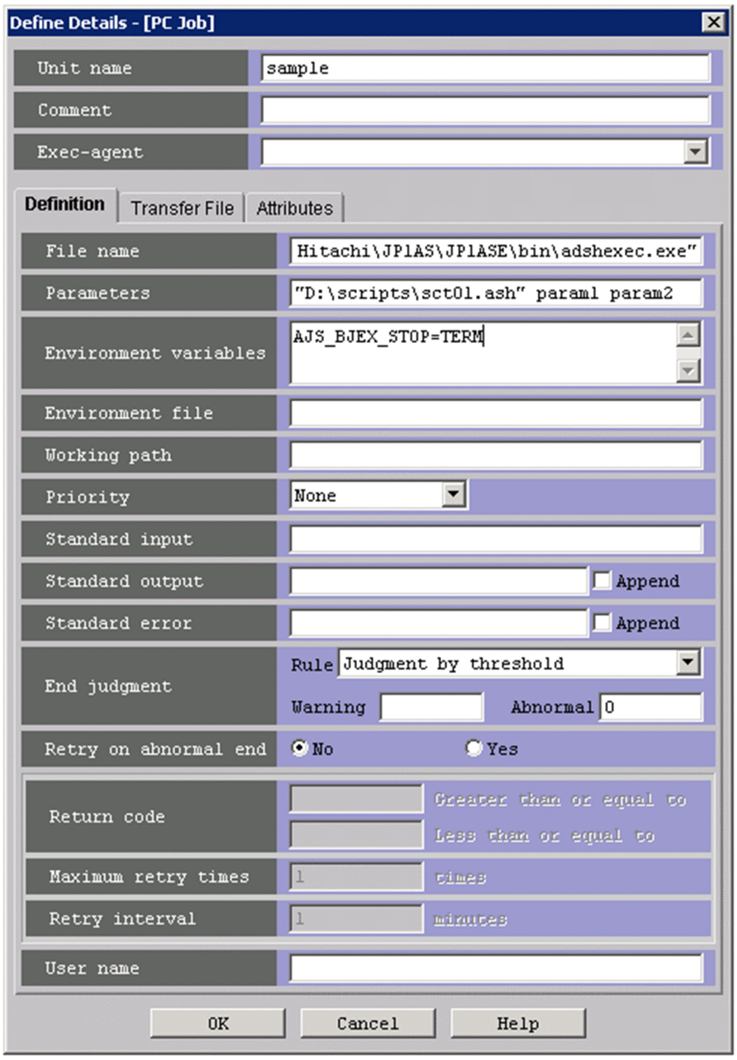
- Parameters
  Specify the options, job definition script file name, and runtime parameters for the `adshexec` command in **Parameters** on the **Definition** tab in the Define Details - [PC Job] dialog box or in `prm="`*parameter*`"` in the unit definition file.

- Environment variables
  Specify the following value in **Environment variables** on the **Definition** tab in the Define Details - [PC Job] dialog box or in `env="`*environment-variable*`"` in the unit definition file:

  ```
  AJS_BJEX_STOP=TERM
  ```

The following figure shows a specification example of a JP1/Advanced Shell batch job.

Figure 2–5: Specification example of the Definition tab in the Define Details - [PC Job] dialog box



Define Details - [PC Job]

Unit name	sample
Comment	
Exec-agent	

Definition | Transfer File | Attributes

File name	Hitachi\JP1AS\JP1ASE\bin\adshexec.exe"
Parameters	"D:\scripts\sct01.ash" param1 param2
Environment variables	AJS_BJEX_STOP=TERM
Environment file	
Working path	
Priority	None
Standard input	
Standard output	□ Append
Standard error	□ Append
End judgment	Rule Judgment by threshold Warning Abnormal 0
Retry on abnormal end	● No ○ Yes
Return code	Greater than or equal to Less than or equal to
Maximum retry times	1 times
Retry interval	1 minutes
User name	

(2) Defining jobs as UNIX jobs

This subsection explains the items required to define JP1/Advanced Shell batch jobs as UNIX jobs.

- Script file name

  Specify the path of the `adshexec` command in **Script file name** on the **Definition** tab in the Define Details - [UNIX Job] dialog box or in `sc="`*script-file-name*`"` in the unit definition file:

  ```
  /opt/jp1as/bin/adshexec
  ```

  Alternatively, you can specify the path of the `adshexec` command following `#!` on the first line (example: `#!/opt/jp1as/bin/adshexec`), and then specify the path of the job definition script file with execution permissions granted:

  ```
  path-of-job-definition-script-file
  ```

  Do not specify this information in **Command statement** on the **Definition** tab in the Define Details - [UNIX Job] dialog box or in `te="`*command-text*`"` in the unit definition file.

- Parameters

Specify the options, job definition script file name, and runtime parameters for the `adshexec` command in **Parameters** on the **Definition** tab in the Define Details - [UNIX Job] dialog box or in `prm="`*parameter*`"` in the unit definition file.

If you specified a job definition script file name for the script file name, specify only the runtime parameters.

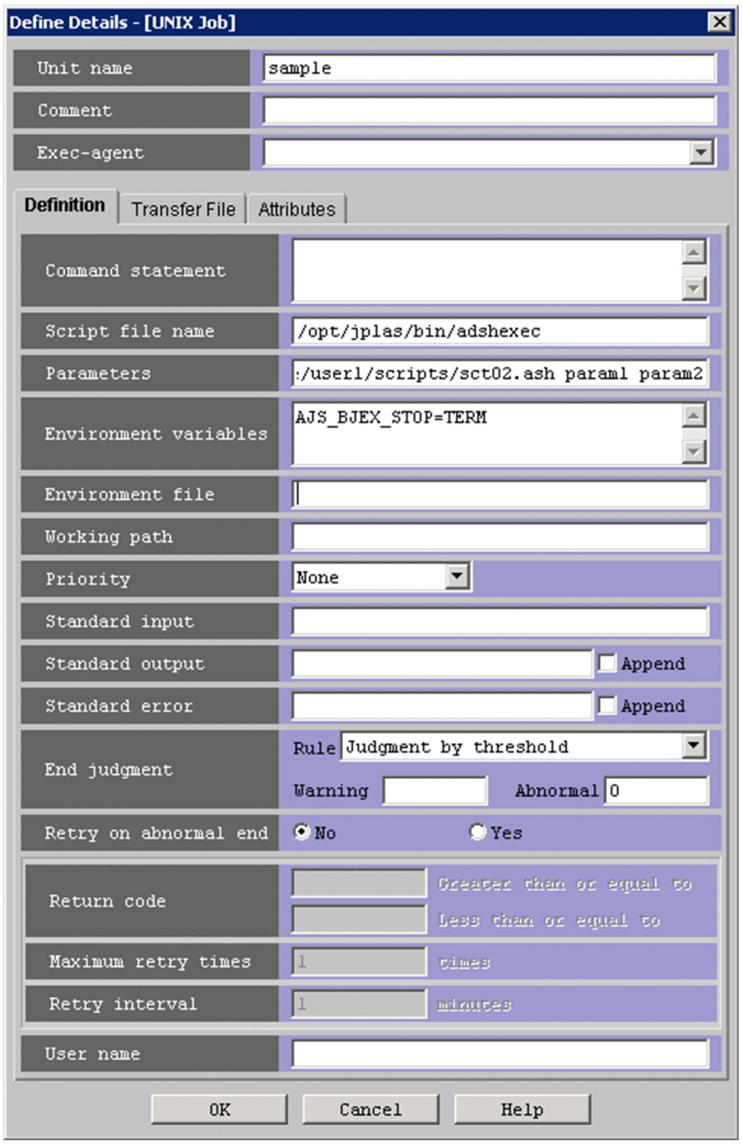
- Environment variables

  Specify the following value in **Environment variables** on the **Definition** tab in the Define Details - [UNIX Job] dialog box or in `env="`*environment-variable*`"` in the unit definition file:

```
AJS_BJEX_STOP=TERM
```

The following figure shows a specification example of a JP1/Advanced Shell batch job.

Figure 2–6: Specification example of the Definition tab in the Define Details - [UNIX Job] dialog box (when specifying the adshexec command)



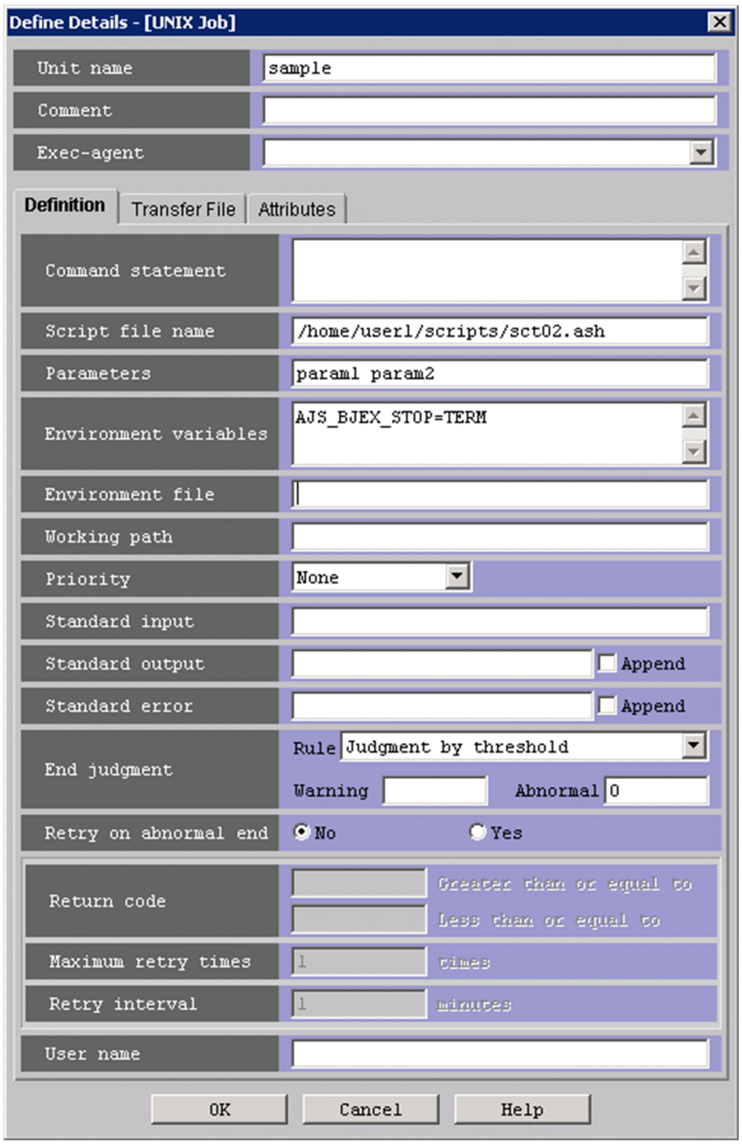
Define Details - [UNIX Job]

Unit name	sample
Comment	
Exec-agent	

Definition | Transfer File | Attributes

Command statement	
Script file name	/opt/jplas/bin/adshexec
Parameters	:/user1/scripts/sct02.ash param1 param2
Environment variables	AJS_BJEX_STOP=TERM
Environment file	
Working path	
Priority	None
Standard input	
Standard output	□ Append
Standard error	□ Append
End judgment	Rule Judgment by threshold Warning Abnormal 0
Retry on abnormal end	● No ○ Yes
Return code	Greater than or equal to Less than or equal to
Maximum retry times	1 times
Retry interval	1 minutes
User name	

Figure 2–7: Specification example of the Definition tab in the Define Details - [UNIX Job] dialog box (when specifying a job definition script file path)



Define Details - [UNIX Job]

Unit name	sample
Comment	
Exec-agent	

Definition | Transfer File | Attributes

Command statement	
Script file name	/home/user1/scripts/sct02.ash
Parameters	param1 param2
Environment variables	AJS_BJEX_STOP=TERM
Environment file	
Working path	
Priority	None
Standard input	
Standard output	□ Append
Standard error	□ Append
End judgment	Rule Judgment by threshold Warning Abnormal 0
Retry on abnormal end	● No ○ Yes
Return code	Greater than or equal to Less than or equal to
Maximum retry times	1 times
Retry interval	1 minutes
User name	

2.8 Setting up the user-reply functionality

2.8.1 Specifying the environment files to use the user-reply functionality

In JP1/Advanced Shell, you can specify the operating environment by using two types of environment files, the system environment file for applying the same operating environment to multiple jobs, and a job environment file for applying a different operating environment to each job. If you will be using the user-reply functionality, you must edit the system environment file as appropriate for the system's environment.

Once you have edited the system environment file, you must restart the daemon or service.

(1) Specifying the spool root directory

Use the `SPOOL_DIR` parameter to specify the spool root directory to which job execution results are to be output. When you will be using the user-reply functionality, you must define the `SPOOL_DIR` parameter only in the system environment file. Path names cannot contain multibyte characters.

For details about the `SPOOL_DIR` parameter, see *SPOOL_DIR parameter (defines the spool root directory path name)* in *7. Parameters Specified in the Environment Files*.

(2) Specifying the JP1 event destination host

You must use the `HOSTNAME_JP1IM_MANAGER` parameter in the system environment file to specify the operation management server on which the JP1/IM - Manager that is the destination of JP1 events is running. If this parameter is omitted, the host name displayed when the `hostname` command is executed on the server on which JP1/Advanced Shell is running is assumed as the destination of JP1 events. Define the `HOSTNAME_JP1IM_MANAGER` parameter only in the system environment file. Also, verify that the name of the host on which JP1/Advanced Shell is run can be resolved on the host specified in this parameter.

For details about the `HOSTNAME_JP1IM_MANAGER` parameter, see *HOSTNAME_JP1IM_MANAGER parameter (specifies the operation management server on which JP1/IM - Manager is running that is to be the destination of JP1 events)* in *7. Parameters Specified in the Environment Files*.

(3) Specifying JP1 event flow control

When you use the user-reply functionality, the conditions shown in the table below concerning output of JP1 events must be satisfied. These conditions apply to all JP1 events that are received by the same JP1/IM.

No.	Description	Output condition
1	JP1 events	The number of JP1 events output per second must be less than 2.
2	Reply-waiting events	The number of reply-waiting events output per minute must be less than 1.

In JP1/Advanced Shell, you use the `USERREPLY_JP1EVENT_INTERVAL` parameter to specify the minimum interval at which JP1 events are to be issued by the `adshecho` or `adshread` command. If multiple jobs issue events, specify the parameter in such a manner that the frequency of JP1 event issuance by the `adshecho` and `adshread` commands from all jobs satisfies the above conditions.

For details about the `USERREPLY_JP1EVENT_INTERVAL` parameter, see *USERREPLY_JP1EVENT_INTERVAL parameter (specifies the minimum interval at which JP1 events are to be issued)* in *7. Parameters Specified in the Environment Files*.

(4) Specifying the maximum number of concurrent reply-request messages that can be output

You use the `USERREPLY_WAIT_MAXCOUNT` parameter to specify the maximum number of concurrent reply-request messages that can be output by the user-reply functionality for each physical or logical host. Specify in this parameter a value that is at least equal to the number of jobs that will execute the `adshread` command concurrently.

For details about the `USERREPLY_WAIT_MAXCOUNT` parameter, see *USERREPLY_WAIT_MAXCOUNT parameter (specifies the maximum number of concurrent reply-request messages that can be output for a physical or logical host)* in *7. Parameters Specified in the Environment Files*.

2.8.2 Setting up the user-reply functionality after JP1/Advanced Shell has been installed (Windows only)

This subsection explains the settings to be specified after JP1/Advanced Shell has been installed to use the user-reply functionality. These settings must be specified by a user with an administrator role.

(1) Setting up the service

(a) Specifying how to start the JP1/Advanced Shell service

To have the JP1/Advanced Shell service start automatically:

1. From the Windows **Control Panel**, select **Administrative Tools**, and then **Services**.
2. From the displayed list of service names, display the properties of the following service:
   - To use the user-reply functionality from the execution environment, service name beginning with **AdshmSvcE**
   - To use the user-reply functionality from the development environment, service name beginning with **AdshmSvcD**
3. On the **General** page, change **Startup Type** as follows:
   Immediately after JP1/Advanced Shell has been installed, the initial setting is **Manual**. If you want to have the service start automatically whenever Windows starts, change this setting to **Automatic**.

(b) Starting the JP1/Advanced Shell service

This subsection explains how to start the JP1/Advanced Shell service manually. You can skip this procedure if you set **Startup Type** to **Automatic** in *(a) Specifying how to start the JP1/Advanced Shell service*, and then started Windows.

To start the JP1/Advanced Shell service

1. From the Windows **Control Panel**, select **Administrative Tools**, and then **Services**.
2. From the displayed list of service names, display the properties of the following service:
   - To use the user-reply functionality from the execution environment, service name beginning with **AdshmSvcE**
   - To use the user-reply functionality from the development environment, service name beginning with **AdshmSvc**
3. On the **General** page, click the **Start** button.

If the service does not start, check the error information output to the event logs.

If the KNAX7552-E message is issued as error information, check and, if necessary, revise the system environment file, and then restart the service.

(c) Registering a service

If a service (AdshmSvcD or AdshmSvcE) to be registered automatically during installation is deleted, you must re-register the service to use the user-reply functionality. You can use the adshmsvcd and adshmsvce commands to register services.

The following explains how to register a service.

- How to re-register a service

  If **AdshmSvcD** or **AdshmSvcE** are not displayed in **Services** in Administrative Tools, registration of the service might have been deleted. You can re-register the service by executing the following command:

  - To register the AdshmSvcD service

    ```
    adshmsvcd -install
    ```

  - To register the AdshmSvcE service

    ```
    adshmsvce -install
    ```

  If the command terminates normally, the registered service is displayed in **Services** in Administrative Tools.

For details about how to start a registered services, see *(b) Starting the JP1/Advanced Shell service*.

(2) Setting up the adapter command (for the execution environment)

To use the user-reply functionality in the execution environment, you must specify the adapter command shown below in JP1/Base. Specify this setting only once after you have installed JP1/Advanced Shell. However, if you have re-installed JP1/Base, you must perform this setup again.

To set up the adapter command:

1. Check the path specified in cmdpath in the adapter command configuration file used for the user-reply functionality. If it differs from the JP1/Advanced Shell installation folder, correct it to the installation folder.

   The adapter command configuration file used for the user-reply functionality is stored at the following location:

   ```
   Installation-folder\JP1ASE\sample
   \Adapter_HITACHI_JP1_AS_ASE_USERREPLY.conf
   ```

   When JP1/Advanced Shell is installed, the adapter command configuration file used for the user-reply functionality contains the following information:

   - In a 64-bit edition of Windows

     ```
     fileversion 07000000
     upperpp /HITACHI/JP1/IM/CC
     componenttype JP1_AS_ASE_USERREPLY
     cmdpath C:\Program Files (x86)\Hitachi\JP1AS\JP1ASE\bin\adshuserreply.exe
     ```

   - In non-64-bit editions of Windows

     ```
     fileversion 07000000
     upperpp /HITACHI/JP1/IM/CC
     componenttype JP1_AS_ASE_USERREPLY
     cmdpath C:\Program Files\Hitachi\JP1AS\JP1ASE\bin\adshuserreply.exe
     ```

2. Copy the adapter command configuration file used for the user-reply functionality shown in step 1 to the JP1/Base installation target.

   The target folder is as follows:

   ```
   JP1/Base-installation-folder\plugin\conf
   ```

Responses can now be entered for reply-waiting events that are displayed in JP1/IM - View.

(3) Setting up the adapter command (for the development environment)

To use the user-reply functionality in the development environment of JP1/Advanced Shell, you must specify the settings described here. However, if you specify the standard output as the output destination of the user-reply functionality, there is no need to perform this setup.

Specify this setting only once after you have installed JP1/Advanced Shell. However, if you have re-installed JP1/Base, you must perform this setup again.

To set up the adapter command:

1. Check the path specified in `cmdpath` in the adapter command configuration file used for the user-reply functionality. If it differs from the JP1/Advanced Shell installation folder, correct it to the installation folder.

   The adapter command configuration file used for the user-reply functionality is stored at the following location:

   ```
   JP1/Advanced-Shell-installation-folder\JP1ASD\sample
   \Adapter_HITACHI_JP1_AS_ASD_USERREPLY.conf
   ```

   When JP1/Advanced Shell is installed, the adapter command configuration file used for the user-reply functionality contains the following information:

   - In a 64-bit edition of Windows

     ```
     fileversion 07000000
     upperpp /HITACHI/JP1/IM/CC
     componenttype JP1_AS_ASD_USERREPLY
     cmdpath C:\Program Files (x86)\Hitachi\JP1AS\JP1ASD\bin\adshuserreply.exe
     ```

   - In non-64-bit editions of Windows

     ```
     fileversion 07000000
     upperpp /HITACHI/JP1/IM/CC
     componenttype JP1_AS_ASD_USERREPLY
     cmdpath C:\Program Files\Hitachi\JP1AS\JP1ASD\bin\adshuserreply.exe
     ```

2. Copy the file shown in step 1 to the JP1/Base installation target.

   The target folder in JP1/Base is as follows:

   ```
   JP1/Base-installation-folder\plugin\conf
   ```

Responses can now be entered for reply-waiting events that are displayed in JP1/IM - View.

2.8.3 Setting up the user-reply functionality after JP1/Advanced Shell has been installed (UNIX only)

This subsection explains the settings to be specified after JP1/Advanced Shell has been installed to use the user-reply functionality.

After you have set up JP1/Advanced Shell, you must enable the function for reply-waiting events in JP1/IM - Manager. For details, see *2.8.4 Specifying environment information in JP1/IM - Manager*.

(1) Starting and terminating the user-reply functionality's management daemon automatically

This subsection explains how to have the user-reply functionality's management daemon start and terminate automatically when the system starts up and shuts down.

(a) In AIX

- Setting up the automatic start function at system startup

  You use the `mkitab` command to have the user-reply functionality's management daemon start automatically at the time of system startup. The specified setting takes effect the next time the system starts.

  The following shows an example of the `mkitab` command:

```
mkitab "adshmd:2:wait:/opt/jp1as/sbin/adshmdctl start"
```

  Set the user-reply functionality's management daemon to start after the services of linked JP1-series products have started. For example, to have JP1/Base, JP1/IM - Manager, and JP1/Advanced Shell start automatically in this order, execute the `mkitab` commands as follows:

```
mkitab -i hntr2mon "jp1base:2:wait:/etc/opt/jp1base/jbs_start"
mkitab -i jp1base "jp1cons:2:wait:/etc/opt/jp1cons/jco_start"
mkitab -i jp1cons "adshmd:2:wait:/opt/jp1as/sbin/adshmdctl start"
```

  After you have specified the settings, execute the following `lsitab` command to check the settings:

```
lsitab -a
```

  The following shows an example of the output after command execution:

```
init:2:initdefault:
brc::sysinit:/sbin/rc.boot 3 >/dev/console 2>&1 # Phase 3 of
system boot
    :
hntr2mon:2:once:/opt/hitachi/HNTRLib2/etc/D002start
jp1base:2:wait:/etc/opt/jp1base/jbs_start
jp1cons:2:wait:/etc/opt/jp1cons/jco_start
adshmd:2:wait:/opt/jp1as/sbin/adshmdctl start
```

- Setting up the automatic termination function at system shutdown

  To have the user-reply functionality's management daemon terminate automatically at the time of system shutdown, you must edit `/etc/rc.shutdown` to add the following code so that the user-reply functionality's management daemon is terminated before the services of linked JP1-series products are stopped:

```
test -x /opt/jp1as/sbin/adshmdctl && /opt/jp1as/sbin/adshmdctl stop
    :
```

```
termination processing for services of linked JP1-series products
    :
```

(b) In Linux

In JP1/Advanced Shell, the script file `jp1_as_md` for automatically starting and terminating the user-reply functionality's management daemon is stored in the `/opt/jp1as/sample` directory. The procedure for using this script file to set the user-reply functionality's management daemon to start and terminate automatically is described below.

- Adding to the `/etc/rc.d/init.d` directory

  Add `jp1_as_md` stored in the `/opt/jp1as/sample` directory to `/etc/rc.d/init.d`. To add `jp1_as_md`, specify as follows:

```
cp /opt/jp1as/sample/jp1_as_md /etc/rc.d/init.d
chmod u=rwx,go=rx /etc/rc.d/init.d/jp1_as_md
chown root:root /etc/rc.d/init.d/jp1_as_md
```

- Creating a symbolic link for automatic startup

  Create a symbolic link to `/etc/rc.d/init.d/jp1_as_md` in the `/etc/rc.d/rc`*N*`.d` directory (*N* indicates the execution level at startup). You must name the symbolic link in such a manner that the user-reply functionality's management daemon is started after the services of linked JP1-series products have been started.

  For example, if the name of the symbolic link for JP1/Base's automatic start script provided in the `/etc/rc.d/rc3.d` and `/etc/rc.d/rc5.d` directories is `S99_JP1_10_BASE` and the name of the symbolic link for JP1/IM - Manager's automatic start script is `S99_JP1_20_CONS`, the user-reply functionality's management daemon will be started after JP1/Base and JP1/IM - Manager if you specify as follows, using `S99_JP1_70_AS` as the name of the symbolic link so that it falls after `S99_JP1_20_CONS`:

```
ln -s /etc/rc.d/init.d/jp1_as_md /etc/rc.d/rc3.d/S99_JP1_70_AS
ln -s /etc/rc.d/init.d/jp1_as_md /etc/rc.d/rc5.d/S99_JP1_70_AS
chown -h root:root /etc/rc.d/rc3.d/S99_JP1_70_AS
chown -h root:root /etc/rc.d/rc5.d/S99_JP1_70_AS
```

- Creating a symbolic link for automatic termination

  Create a symbolic link to `/etc/rc.d/init.d/jp1_as_md` in the `/etc/rc.d/rc`*N*`.d` directory (*N* indicates the execution level at termination). You must name the symbolic link in such a manner that the user-reply functionality's management daemon is terminated before the services of the linked JP1-series products are stopped.

  For example, if the name of the symbolic link for JP1/Base's automatic termination script provided in the `/etc/rc.d/rc0.d` and `/etc/rc.d/rc6.d` directories is `K01_JP1_90_BASE` and the name of the symbolic link for JP1/IM - Manager's automatic termination script is `K01_JP1_80_CONS`, the user-reply functionality's management daemon will be terminated before JP1/Base and JP1/IM - Manager if you specify as follows, using `K01_JP1_30_AS` as the name of symbolic link so that it falls before `K01_JP1_80_CONS`:

```
ln -s /etc/rc.d/init.d/jp1_as_md /etc/rc.d/rc0.d/K01_JP1_30_AS
ln -s /etc/rc.d/init.d/jp1_as_md /etc/rc.d/rc6.d/K01_JP1_30_AS
chown -h root:root /etc/rc.d/rc0.d/K01_JP1_30_AS
chown -h root:root /etc/rc.d/rc6.d/K01_JP1_30_AS
```

  If you change the `_JP1_30_AS` part in the name of symbolic link `K01_JP1_30_AS`, you must also change `_JP1_30_AS` on the following lines in the `jp1_as_md` script file:

```
touch /var/lock/subsys/_JP1_30_AS
rm -f /var/lock/subsys/_JP1_30_AS
```

(c) In HP-UX

In JP1/Advanced Shell, the script file jp1_as_md for automatically starting and terminating the user-reply functionality's management daemon is stored in the /opt/jp1as/sample directory. The procedure for using this script file to set the user-reply functionality's management daemon to start and terminate automatically is described below.

- Adding to the /sbin/init.d directory

  Add jp1_as_md stored in the /opt/jp1as/sample directory to /sbin/init.d. To add jp1_as_md, specify as follows:

  ```
  cp /opt/jp1as/sample/jp1_as_md /sbin/init.d
  chmod u=rx,go=r /sbin/init.d/jp1_as_md
  chown root:sys /sbin/init.d/jp1_as_md
  ```

- Creating a symbolic link for automatic startup

  Create a symbolic link to /sbin/init.d/jp1_as_md in the /sbin/rc2.d directory. You must name the symbolic link in such a manner that the user-reply functionality's management daemon is started after the services of the linked JP1-series products have been started.

  For example, if the name of the symbolic link for JP1/Base's automatic start script provided in the /sbin/rc2.d directory is S900jp1_base and the name of the symbolic link for JP1/IM - Manager's automatic start script is S901jp1_cons, the user-reply functionality's management daemon will be started after JP1/Base and JP1/IM - Manager if you specify as follows, using S905jp1_as_md as the name of the symbolic link so that 905 falls after 901:

  ```
  ln -s /sbin/init.d/jp1_as_md /sbin/rc2.d/S905jp1_as_md
  chown -h root:sys /sbin/rc2.d/S905jp1_as_md
  ```

- Creating a symbolic link for automatic termination

  Create a symbolic link to /sbin/init.d/jp1_as_md in the /sbin/rc1.d directory. You must name the symbolic link in such a manner that the user-reply functionality's management daemon is terminated before the services of linked JP1-series products are stopped.

  For example, if the name of the symbolic link for JP1/Base's automatic termination script provided in the /sbin/rc1.d directory is K100jp1_base and the name of the symbolic link for JP1/IM - Manager's automatic termination script is K099jp1_cons, the user-reply functionality's management daemon will be terminated before JP1/Base and JP1/IM - Manager if you specify as follows, using K095jp1_as_md as the name of symbolic link so that 095 falls before 099:

  ```
  ln -s /sbin/init.d/jp1_as_md /sbin/rc1.d/K095jp1_as_md
  chown -h root:sys /sbin/rc1.d/K095jp1_as_md
  ```

(d) In Solaris

In JP1/Advanced Shell, the script file jp1_as_md for automatically starting and terminating the user-reply functionality's management daemon is stored in the /opt/jp1as/sample directory. The procedure for using this script file to set the user-reply functionality's management daemon to start and terminate automatically is described below.

- Adding to the /etc/init.d directory

  Add jp1_as_md stored in the /opt/jp1as/sample directory to /etc/init.d. To add jp1_as_md, specify as follows:

```
cp /opt/jp1as/sample/jp1_as_md /etc/init.d
chmod u=rwx,go=r /etc/init.d/jp1_as_md
chown root:sys /etc/init.d/jp1_as_md
```

- Creating a symbolic link for automatic startup

  Create a symbolic link to `/etc/init.d/jp1_as_md` in the `/etc/rc2.d` directory (*N* indicates the execution level at startup). You must name the symbolic link in such a manner that the user-reply functionality's management daemon is started after the services of the linked JP1-series products have been started.

  For example, if the name of the symbolic link for JP1/Base's automatic start script provided in the `/etc/rc2.d` directory is `S99_JP1_10_BASE` and the name of the symbolic link for JP1/IM - Manager's automatic start script is `S99_JP1_20_CONS`, the user-reply functionality's management daemon will be started after JP1/Base and JP1/IM - Manager if you specify as follows, using `S99_JP1_70_AS` as the name of the symbolic link so that it falls after `S99_JP1_20_CONS`:

  ```
  ln -s /etc/init.d/jp1_as_md /etc/rc2.d/S99_JP1_70_AS
  chown -h root:sys /etc/rc2.d/S99_JP1_70_AS
  ```

- Creating a symbolic link for automatic termination

  Create a symbolic link to `/etc/init.d/jp1_as_md` in the `/etc/rc0.d` directory. You must name the symbolic link in such a manner that the user-reply functionality's management daemon is terminated before the services of linked JP1-series products are stopped.

  For example, if the name of the symbolic link for JP1/Base's automatic termination script provided in the `/etc/rc0.d` directory is `K01_JP1_90_BASE` and the name of the symbolic link for JP1/IM - Manager's automatic termination script is `K01_JP1_80_CONS`, the user-reply functionality's management daemon will be terminated before JP1/Base and JP1/IM - Manager if you specify as follows, using `K01_JP1_30_AS` as the name of symbolic link so that it falls before `K01_JP1_80_CONS`:

  ```
  ln -s /etc/init.d/jp1_as_md /etc/rc0.d/K01_JP1_30_AS
  chown -h root:sys /etc/rc0.d/K01_JP1_30_AS
  ```

(2) Setting up JP1/Base

To use the user-reply functionality, you must have first copied the adapter command configuration file that is provided by JP1/Advanced Shell and used for the user-reply functionality to the corresponding directory in JP1/Base. Copy the following adapter command configuration file used for the user-reply functionality to the corresponding directory in JP1/Base:

- Source directory (directory containing the adapter command configuration file that is provided by JP1/Advanced Shell and used for the user-reply functionality)

  ```
  /opt/jp1as/sample
  ```

  Name of the adapter command configuration file that is provided by JP1/Advanced Shell and used for the user-reply functionality

  `Adapter_HITACHI_JP1_AS_USERREPLY.conf`

- Target directory (corresponding directory in JP1/Base)

  ```
  /opt/jp1base/plugin/conf
  ```

Perform this setup only once after you have installed JP1/Advanced Shell. However, if you have re-installed JP1/Base, you must perform this setup again.

2.8.4 Specifying environment information in JP1/IM - Manager

This subsection explains how to set up JP1/IM - Manager to use the user-reply functionality. You perform this setup in JP1/IM - Manager on the host that was specified in the `HOSTNAME_JP1IM_MANAGER` parameter in *2.8.1(2) Specifying the JP1 event destination host*.

(1) Copying the definition file for extended event attributes to JP1/IM - Manager

Copy the definition file for extended event attributes from the `sample` directory under the JP1/Advanced Shell installation directory to JP1/IM - Manager. Perform the following procedure as a user with superuser permissions.

To copy the definition file for extended event attributes and enable it:

1. Copy the definition file for extended event attributes to the machine on which JP1/IM - Manager is installed.
    - Source definition file for extended event attributes
      The definition file for extended event attributes that is to be copied depends on the character encoding used in the target OS, as shown in the following:

Target OS	Definition file for extended event attributes to be copied

Windows	`hitachi_jp1_as_SJIS_attr_ja.conf`
	`hitachi_jp1_as_C_attr_cn.conf`
	`hitachi_jp1_as_C_attr_en.conf`

AIX	`hitachi_jp1_as_SJIS_attr_ja.conf`
	`hitachi_jp1_as_EUC_attr_ja.conf`
	`hitachi_jp1_as_UTF8_attr_ja.conf`
	`hitachi_jp1_as_C_attr_cn.conf`
	`hitachi_jp1_as_C_attr_en.conf`

Linux	`hitachi_jp1_as_UTF8_attr_ja.conf`
	`hitachi_jp1_as_C_attr_cn.conf`
	`hitachi_jp1_as_C_attr_en.conf`

HP-UX	`hitachi_jp1_as_SJIS_attr_ja.conf`
	`hitachi_jp1_as_EUC_attr_ja.conf`
	`hitachi_jp1_as_UTF8_attr_ja.conf`
	`hitachi_jp1_as_C_attr_cn.conf`
	`hitachi_jp1_as_C_attr_en.conf`

Solaris	`hitachi_jp1_as_SJIS_attr_ja.conf`
	`hitachi_jp1_as_EUC_attr_ja.conf`
	`hitachi_jp1_as_UTF8_attr_ja.conf`
	`hitachi_jp1_as_C_attr_cn.conf`
	`hitachi_jp1_as_C_attr_en.conf`

    - Target directory
      The following table shows the target directory to which the definition file for extended event attributes is to be copied on the machine on which JP1/IM - Manager is installed:

Target OS	Target directory
Windows	*JP1/IM-Manager-console-path*`\conf\console\attribute\` Or the following if JP1/IM - Manager is operating in a cluster configuration: *shared-folder*`\jp1cons\conf\console\attribute\`

Target OS	Target directory
UNIX	`/etc/opt/jp1cons/conf/console/attribute/` Or the following if JP1/IM - Manager is operating in a cluster configuration: *shared-directory*`/jp1cons/conf/console/attribute/`

For details about JP1/IM - Manager's console path, see the *Job Management Partner 1/Integrated Management - Manager Configuration Guide*.

For details about the shared folder and directory, see the description of cluster systems in the *Job Management Partner 1/Integrated Management - Manager Configuration Guide*.

2. Restart JP1/IM - Manager.

(2) Setting up JP1/IM - Manager and JP1/IM - View

The function related to reply-waiting events must be enabled in JP1/IM - Manager and JP1/IM - View. If this function is not enabled, replies cannot be entered in JP1/IM - View.

For details about how to set up JP1/IM - Manager and JP1/IM - View and the settings for communication between JP1/Advanced Shell and JP1/IM - Manager, see the description related to linking with JP1/Advanced Shell in the *Job Management Partner 1/Integrated Management - Manager Configuration Guide*.

2.8.5 Specifying environment information in JP1/Base

If you will be using the user-reply functionality, the character encoding of JP1/Base that is run on the same host as JP1/Advanced Shell must match the character encoding used for event notification messages and reply-request messages that are specified in the `adshecho` and `adshread` commands.

For details about the character encoding settings in JP1/Base, see the sections on installation and setup in the *Job Management Partner 1/Base User's Guide*.

2.9 Running in a cluster configuration

2.9.1 Prerequisites and scope of support for cluster operations

In a cluster system, JP1/Advanced Shell is run in a logical host environment and can inherit the job execution environment in the event of system switchover. However, execution of a job that was underway at the time system switchover occurred cannot continue. If necessary, you must re-execute such a job manually after system switchover is completed.

When JP1/Advanced Shell is run on a logical host, the cluster software program must manage the logical IP address as well as allocation, deletion, and operation monitoring of the shared disk. In addition, you must configure the system and set up the environment so that the following prerequisites are satisfied.

(1) Prerequisites for a logical host environment

When JP1/Advanced Shell is run in a logical host environment, the following prerequisites apply to the logical IP address and the shared disk.

Table 2–23: Prerequisites for logical host environment

Logical host component	Prerequisites
Shared disk	• A shared disk that can be inherited from the active server to the standby server is available. • The shared disk is allocated before the JP1/Advanced Shell program starts. • The shared disk allocation is not released while the user-reply functionality's management daemon or service are running. • The shared disk allocation is released after the user-reply functionality's management daemon or service have stopped. • The shared disk is locked to prevent illegal access from multiple nodes. • Files are protected from unexpected events, such as system shutdown, by using file systems with journal functions. • When a planned termination is performed on a running program during system switchover, the file contents are guaranteed and inherited. • Forced system switchover is available even if a process is using the shared disk during system switchover. • In the event of a failure of the shared disk, the cluster software controls the recovery processing. If it is necessary to start and stop the user-reply functionality's management daemon or service as an extension of the recovery processing, the cluster software will issue requests to start and stop the user-reply functionality's management daemon or service.
Logical IP address	• A logical IP address that can be inherited is available for communication. • A unique logical IP address can be obtained from the logical host name. • The logical IP address is allocated before the user-reply functionality's management daemon or service start. • The logical IP address is not deleted while the user-reply functionality's management daemon or service is running. • The correspondence between logical host name and logical IP address remains unchanged while the user-reply functionality's management daemon or service is running. • The logical IP address is deleted after the user-reply functionality's management daemon or service has stopped. • In the event of a network failure, the cluster software controls the recovery processing without having to involve the user-reply functionality's

Logical host component	Prerequisites
Logical IP address	management daemon or service in the recovery processing. If it is necessary to start and stop the user-reply functionality's management daemon or service as an extension of the recovery processing, the cluster software is to issue requests to start and stop the user-reply functionality's management daemon or service. • If multiple logical hosts are started on the same physical host, one IP address is allocated to each logical host.

(2) Prerequisites for a physical host environment

In a cluster system that runs JP1/Advanced Shell on a logical host, each server's physical host environment must satisfy the following prerequisites.

Table 2–24: Prerequisites for physical host environment

Physical host component	Prerequisites
Server machine	• The cluster configuration consists of at least two server machines. • Sufficient CPU performance is available for the processing to be performed (for example, if multiple logical hosts will run concurrently, there will be sufficient CPU performance). • There is enough real memory for the processing to be performed (for example, if multiple logical hosts will run concurrently, there will be enough real memory capacity).
Disk	• Files are protected from unexpected events, such as system shutdown, by using file systems with journal functions.
Network	• If the user-reply functionality's management daemon or service are used in a physical host environment, the IP address corresponding to the physical host name (obtained by the `hostname` command) is supported for communications (communications will not be disabled by a program such as the cluster software).# • The correspondence between host name and IP address remains unchanged while the user-reply functionality's management daemon or service is running (the correspondence will not be changed by a program such as the cluster software or a name server). • In Windows, a LAN board corresponding to the host name is given priority by the network binding settings (no other LAN board, such as a heartbeat LAN, is granted priority).
OS and cluster software	• The environment settings of the individual servers are identical so that the same processing can be performed after system switchover. • The cluster software and its version are supported by JP1/Advanced Shell. • Patches and service packs required by JP1/Advanced Shell and cluster software have been applied.

#

Depending on the cluster software, the IP address corresponding to the physical host name (host name displayed by the `hostname` command) might not be supported for communications. In such a case, the user-reply functionality's management daemon or service cannot run in the physical host environment. Use the user-reply functionality's management daemon or service in the logical host environment only.

(3) Scope of support by JP1/Advanced Shell

When JP1/Advanced Shell is run in a cluster system, it supports only its own operation. The logical host environment (shared disk and logical IP address) is controlled by the cluster software.

If the prerequisites for the logical and physical host environments are not satisfied or there is a problem in the control of the logical host environment, JP1/Advanced Shell might not function normally. In such a case, check the physical and logical host environments or the cluster software settings and revise the prerequisites as necessary.

(4) Conditions for logical host names

The following conditions apply to logical host names:

- Permitted number of characters

  Windows: 1 to 196 bytes (a maximum of 63 bytes is recommended)

  UNIX: 1 to 255 bytes (a maximum of 63 bytes is recommended)

  If the cluster software or JP1 products being used impose limitations, you must observe those limitations in naming the logical hosts.
- Permitted characters

  Alphanumeric characters and the hyphen (-)

2.9.2 Specifying environment information for cluster operation

This subsection explains the environment settings for JP1/Advanced Shell to support cluster operation.

(1) Installing and setting up the JP1-series products to be linked

Install and set up on the active and standby servers the JP1-series products to be linked. For details about the installation and setup of the JP1-series products to be linked, see the applicable manuals.

(2) Installing JP1/Advanced Shell

Install JP1/Advanced Shell on the local disk of both the active and standby servers.

Do not install JP1/Advanced Shell on the shared disk.

(3) Specifying environment information for JP1/Advanced Shell

To use JP1/Advanced Shell in a cluster system, perform the tasks described below.

(a) Evaluating the configuration of directories and files

Evaluate according to the system operation guidelines the directory and file configuration items shown in the following table.

Table 2–25: Directory and file configuration evaluation items

Type of directory or file	Creation criterion
Directory for temporary files	S or L
Directory for the spool	S
Directory for system execution logs	S or L
Directory for traces	L
System environment file	L

Type of directory or file	Creation criterion
Job environment file	S or L
Job definition script	S or L
File referenced from job definition scripts	S or L
Other	S or L

Legend:

S: Create on the shared disk.

S or L: Create on the shared disk or logical disk according to the system operation guidelines.

L: Create on the local disk.

(b) Specifying environment information on the physical hosts

Specify the environment information for JP1/Advanced Shell on the physical hosts of the active and standby servers. For details about the specification of environment information, see *2.6 Specifying environment information for JP1/Advanced Shell*.

(c) Specifying environment information on the logical host

Specify the environment information for JP1/Advanced Shell on the logical host of the active server.

To specify the environment information on the logical host:

1. Create the directories needed for execution.

   According to the directory configuration evaluated in *(a) Evaluating the configuration of directories and files*, create the following directories required for running JP1/Advanced Shell on the shared disk or logical disk:

   - Directory for temporary files
   - Directory for the spool
   - Directory for system execution logs
   - Directory for traces

   The following explains how to create directories:

   - When creating the directories on the shared disk

     Create the directories on the shared disk in such a manner that the active server can access the shared disk.

   - When creating the directories on the local disk

     Create the directories on the local disks of both the active and standby servers.

2. Specify the environment file.

   You must specify the settings required for each logical host in the system environment file of JP1/Advanced Shell. To do this, specify the environment setting parameters for each logical host between the `lhost_start` and `lhost_end` conditional parameters. For details about the `lhost_start` and `lhost_end` conditional parameters, see *lhost_start and lhost_end parameters (define a set of parameters applicable only to a specified logical host)* in *7.4 Conditional parameters*.

   To run JP1/Advanced Shell in a logical host environment, specify at least the following parameters in the system environment file:

   - Parameters for the directories required for running JP1/Advanced Shell

     According to the directory configuration evaluated in *(a) Evaluating the configuration of directories and files*, specify in the system environment file the directories created in step 1. Specify these directories in the system environment file only, so that they are swapped when system switchover occurs. Do not specify them in job environment files. For details about the parameters for directories required for running JP1/Advanced Shell, see *2.6.19 Creating the directories required for JP1/Advanced Shell*.

- Parameters for the user-reply functionality

  If you will be using the user-reply functionality in a logical host environment, specify in the system environment file the appropriate parameters for the user-reply functionality for the logical host.

  For details about the parameters for the user-reply functionality, see *2.8.1 Specifying the environment files to use the user-reply functionality*.

An example of the settings in the system environment file with physical and logical hosts specified are shown below. In this example, `/shdsk1/lhost001` and `/shdsk2/lhost002` are directories on the shared disk, and `/lhost001`, `/lhost002`, and `/phost` are directories on the local host.

```
###
### Settings common to both physical and logical hosts
###

#-adsh_conf USERREPLY_JP1EVENT_INTERVAL   500

###
### Settings for each of the physical and logical hosts
###

#### specify parameter for only logical host (lhost001).
#-adsh_conf lhost_start                   lhost001
#-adsh_conf LOG_DIR                       "/shdsk1/lhost001/log"
#-adsh_conf SPOOL_DIR                     "/shdsk1/lhost001/spool"
#-adsh_conf TEMP_FILE_DIR                 "/shdsk1/lhost001/temp"
#-adsh_conf TRACE_DIR                     "/lhost001/trace"
#-adsh_conf HOSTNAME_JP1IM_MANAGER        IMlhost001
#-adsh_conf USERREPLY_WAIT_MAXCOUNT       5
#-adsh_conf lhost_end

#### specify parameter for only logical host (lhost002).
#-adsh_conf lhost_start                   lhost002
#-adsh_conf LOG_DIR                       "/shdsk2/lhost002/log"
#-adsh_conf SPOOL_DIR                     "/shdsk2/lhost002/spool"
#-adsh_conf TEMP_FILE_DIR                 "/shdsk2/lhost002/temp"
#-adsh_conf TRACE_DIR                     "/lhost002/trace"
#-adsh_conf HOSTNAME_JP1IM_MANAGER        IMlhost002
#-adsh_conf USERREPLY_WAIT_MAXCOUNT       5
#-adsh_conf lhost_end

#### specify parameter for physical host.
#-adsh_conf phost_start
#-adsh_conf LOG_DIR                       "/phost/log"
#-adsh_conf SPOOL_DIR                     "/phost/spool"
#-adsh_conf TEMP_FILE_DIR                 "/phost/temp"
#-adsh_conf TRACE_DIR                     "/phost/trace"
#-adsh_conf HOSTNAME_JP1IM_MANAGER        IMphost001
#-adsh_conf USERREPLY_WAIT_MAXCOUNT       10
#-adsh_conf phost_end
```

3. Register the user-reply functionality's management service for the logical hosts (Windows only).

   To use the user-reply functionality in a logical host environment, you must register the user-reply functionality's management service for the logical hosts on both the active and standby servers. You can use the `adshmsvcd` and `adshmsvce` commands to register the user-reply functionality's management service. To register the service for a logical host, execute the command with the `-install` and `-lhostname` options specified.

When the command terminates normally, the registered service is displayed in the **Services** administrative tool.

For example, to use `lhost001` as a logical host in the execution environment of the active server, execute the following command:

```
adshmsvce -install -lhostname lhost001
```

If the command terminates normally, **AdshmSvcE_ lhost001** is displayed in the **Services** administrative tool.

4. Check the file configuration on the active and standby servers.

   According to the file configuration evaluated in *(a) Evaluating the configuration of directories and files*, perform the following tasks:

   - For the files to be created on the local disk

     The configuration of the files to be referenced and the file contents must be identical between the active and standby servers. Copy the files created on the local disk of the active server, such as the system environment file created in *step 2. Specify the environment file*, to the same path on the standby server.

   - For the files to be created on the shared disk

     Create the files so that the active server can access them on the shared disk.

     Important note

     For the created files, set the permissions so that they can be accessed from both the active and standby servers. If access permissions are granted to a specific user or group, you must specify the same user name and user ID (UID) or the same group name and group ID (GID) on both the active and standby servers.

(4) Registering into the cluster software (Windows)

Register the user-reply functionality's management service on the logical host into the cluster software so that it can be started and stopped from the cluster software. If you do not use the user-reply functionality in a logical host environment, skip this task.

(a) Registering into the cluster software

In Windows, register into the cluster software the service with the name shown in the following that has been registered as a service for the logical host:

Name	Service name
`AdshmSvcE_`*logical-host-name*	User-reply functionality's management service

For details about how to register the service, see the applicable cluster software's documentation. After you have registered the user-reply functionality's management service into the cluster software, use the cluster software to start and stop the service.

(b) Specifying the start and stop sequence

To run the user-reply functionality's management service on the logical host, the shared disk and logical IP address must be available. The start and stop sequence depends on the linked JP1-series products.

- When the logical host is started

1. Allocate the shared disk and logical IP address and enable them.
2. Start the services of the linked JP1-series products (except JP1/AJS).#
3. Start the user-reply functionality's management service.
4. Start the services of JP1/AJS.

- When the logical host is terminated
  1. Stop the services of JP1/AJS.
  2. Stop the user-reply functionality's management service.
  3. Stop the services of the linked JP1-series products (except JP1/AJS).#
  4. Release the allocation of the shared disk and the logical IP address.

#

For details about the start and stop sequence among the services of the linked JP1-series products, see the individual product manuals.

(5) Registering into the cluster software (UNIX)

Register into the cluster software the user-reply functionality's management daemon on the logical host so that it can be started and stopped from the cluster software. If you do not use the user-reply functionality in a logical host environment, skip this task.

(a) Registering into the cluster software

The following table provides the information needed for registering the user-reply functionality's management daemon into the cluster software.

Table 2–26: Functions to be registered in to the cluster software and the commands used by each function

Function to be registered	Description

Start	Starts the user-reply functionality's management daemon.
	Command to be used
	`adshmdctl`
	Example
	`adshmdctl -h` *logical-host-name* `start`
	Checking the result of starting the daemon
	Use the operation monitoring described below to check the result of starting the user-reply functionality's management daemon; do not use the return code.

Stop	Stops the user-reply functionality's management daemon.
	Command to be used
	`adshmdctl`
	Example
	`adshmdctl -h` *logical-host-name* `stop`
	Checking the result of stopping the daemon
	Use the operation monitoring described below to check the result of stopping the user-reply functionality's management daemon; do not use the return code.

Operation monitoring	Monitors the user-reply functionality's management daemon to check if it is running normally. **Command to be used** `adshmdctl`

Function to be registered	Description
Operation monitoring	**Example** `adshmdctl -h` *logical-host-name* `status` **Checking the result of operation monitoring** The following explains the return code: Return code = `0` (running) The user-reply functionality's management daemon is running normally. Return code = `1` (stopped) The user-reply functionality's management daemon is stopped for some reason. Treat this status as an error.

For details about the `adshmdctl` command, see *adshmdctl command (starts and stops the user-reply functionality management daemon) (UNIX only)* in *8. Commands Used During Operations*.

Important note

If the user-reply functionality's management daemon is terminated without releasing the shared memory due to some error, the next startup will fail. If this occurs, take appropriate action according to the information provided in *Function* in *adshmdctl command (starts and stops the user-reply functionality management daemon) (UNIX only)* in *8. Commands Used During Operations*.

(b) Specifying the start and stop sequence

To run the user-reply functionality's management daemon on the logical host, the shared disk and logical IP address must be available. The start and stop sequence depends on the linked JP1-series products.

- When the logical host is started
  1. Allocate the shared disk and logical IP address and enable them.
  2. Start the daemons and services of the linked JP1-series products (except JP1/AJS).#
  3. Start the user-reply functionality's management daemon.
  4. Start the services of JP1/AJS.
- When the logical host is terminated
  1. Stop the services of JP1/AJS.
  2. Stop the user-reply functionality's management daemon.
  3. Stop the daemons and services of the linked JP1-series products (except JP1/AJS).#
  4. Release the allocation of the shared disk and the logical IP address.

#

For details about the start and stop sequence among the services of the linked JP1-series products, see the individual product manuals.

2.9.3 How to specify commands during cluster operation

To execute a command on a logical host, you must specify the logical host name in the command. A command in which no logical host name is specified is executed on the physical host. Specify the logical host in commands as shown in the following.

(1) adshexec command (executes a batch job)

(a) Executing from a JP1/AJS custom job

To execute the command from a custom job, select the **Logical host** check box in **Define detailed info** in the Define Details - [Custom Job] dialog box to execute the command on the logical host.

Note that the JP1/AJS Exec-agent that executes the custom job must be running on the logical host. If the JP1/AJS Exec-agent is running on the physical host, the command will not function normally.

For details about custom jobs, see *2.7 Specifying environment information for JP1/AJS (applicable when JP1/AJS is used)*.

(b) Executing from a program other than a JP1/AJS custom job

To execute the command from a program other than a custom job, specify the command as follows:

```
adshexec -h "" path-name-of-job-definition-script-file
```

If the command is executed from JP1/AJS - Agent running on the logical host, the logical host name is set in the `JP1_HOSTNAME` environment variable. If the null character string (which is two double-quotation marks, `""`) is specified in the `-h` option, the `adshexec` command acquires the logical host name from the `JP1_HOSTNAME` environment variable. For details about the `JP1_HOSTNAME` environment variable, see the *Job Management Partner 1/Base User's Guide*.

Note that the executing JP1/AJS Exec-agent must be running on the logical host. If the JP1/AJS Exec-agent is running on the physical host, see *Executing from a program other than JP1/AJS* below.

(c) Executing from a program other than JP1/AJS

Execute the command with the logical host name specified in the `-h` option, as shown in the following:

```
adshexec -h logical-host-name path-name-of-job-definition-script-file
```

(2) Commands other than adshexec

Execute the command with the logical host name specified in the `-h` option, as shown in the following:

```
command -h logical-host-name
```

This example executes adshlsmsg as a logical host:

```
adshlsmsg -h logical-host-name
```

If the user-reply functionality is used, the logical host name specified in the user-reply functionality's management daemon or service must the same as the logical host name specified in the `adshexec` command. For details about the individual commands that can be executed as a logical host, see *8.3 Shell operation commands*.

2.9.4 Notes about cluster operation

This subsection provides additional information about use of JP1/Advanced Shell in a cluster operation.

- (Windows only) UNC names can be used as file and path names. However, the path names ending with a shared name (including names ending with \) are not supported.
- JP1/Advanced Shell does not support some file systems. For details, see *2.6.19(2) File systems*.
- When multiple logical hosts are configured and multiple copies of JP1/Advanced Shell are run, the user-reply functionality is still executed for each logical host. Information cannot be referenced from the user-reply functionality running on one logical host by the user-reply functionality running on another logical host.
- If system switchover occurs while a job using reply-request messages of the user-reply functionality is running, reply-request events might remain in JP1/IM - View. If this occurs, use JP1/IM - View to release the reply-request events manually.
- (UNIX only) If the shared disk is disconnected by the cluster software while a JP1/Advanced Shell job is running, the job terminates with an error by signal.
- (Windows only) If the shared disk is disconnected by the cluster software while a JP1/Advanced Shell job is running, the job terminates with an error when it attempts to access a file on the shared disk.
- If you run JP1/Advanced Shell in a cluster environment, do not collect coverage information.

2.9.5 Settings for running a logical host in a non-cluster environment

This subsection provides an overview of the configuration and operations of logical hosts in a non-cluster environment. In the case of logical hosts that run in a non-cluster environment, you specify the same environment information as for logical hosts that run in a normal cluster system.

(1) Specifying environment information to run logical hosts in a non-cluster system

This subsection explains how to run JP1/Advanced Shell on a logical host in a non-cluster environment without linking JP1/Advanced Shell to cluster software.

(a) Preparing the logical host environment

Provide a disk area and IP address for the logical host to create a logical host environment.

- Disk area for the logical host

  Create storage directories on the local disk for the files that will be used by JP1/Advanced Shell separately from the directories used by the physical host and the JP1-series products on other logical hosts.
- IP address for the logical host

  Allocate in the OS the IP address that will be used by JP1/Advanced Shell for the logical host. It can be a real IP address or an alias IP address. Make sure that the IP address can be uniquely identified from the logical host.

  The prerequisites for these tasks are the same as for operation in a cluster system. However, the conditions related to cluster software do not apply because JP1/Advanced Shell is not run in a cluster environment.

In *2.9 Running in a cluster configuration*, replace the information about shared disk and logical IP address with the disk area and IP address allocated above for the logical host.

- Estimating performance

When you estimate performance, determine whether JP1/Advanced Shell can be run as a system in terms of the following:

- Determine whether there are enough resources to run multiple JP1-series products in the system. If there are not enough resources, the system might not function correctly or adequate performance might not be realized.

(b) Specifying environment information for the logical host

Specify environment information for the logical host using the same procedure as for the active server in a cluster system. For details about the specification of environment information for cluster operation, see *2.9.2 Specifying environment information for cluster operation*. Note that in a cluster system, the environment information must be specified on both servers involved in system switchover, but for a logical host that is run in a non-cluster environment, specify the environment information only on the server on which JP1/Advanced Shell will be run.

(2) Automatic startup and termination of the user-reply functionality's management daemon for the logical host in a non-cluster environment (UNIX only)

This subsection explains how to have the user-reply functionality's management daemon start and terminate automatically when the system starts and shuts down.

(a) In AIX

- Setting up the automatic start function at system startup

  You use the following mkitab command to have the user-reply functionality's management daemon start automatically at the time of system startup:

  ```
  mkitab "record-of-user-reply-functionality-management-daemon-for-logical
  host:2:wait:/opt/jp1as/sbin/adshmdctl -h logical-host-name start"
  ```

  Set the user-reply functionality's management daemon for the logical host to start after the logical host services of linked JP1-series products have started. For example, to have JP1/Base and JP1/IM - Manager on the logical host start automatically in this order, execute mkitab commands as follows:

  ```
  mkitab -i record-of-JP1-Base-for-logical-host "record-of-JP1/IM-Manager-
  for-logical-host:2:wait:/etc/opt/jp1cons/jco_start.cluster logical-host-
  name"
  mkitab -i record-of-JP1/IM-Manager-for-logical-host "record-of-user-reply-
  functionality-management-daemon-for-logical host:2:wait:/opt/jp1as/sbin/
  adshmdctl -h logical-host-name start"
  ```

- Setting up the automatic termination function at system shutdown

  To have the user-reply functionality's management daemon terminate automatically at the time of system shutdown, you must edit /etc/rc.shutdown to add the following code so that the user-reply functionality's management daemon for the logical host stops before any of the logical host services of linked JP1-series products are stopped:

  ```
  test -x /opt/jp1as/sbin/adshmdctl && /opt/jp1as/sbin/adshmdctl -h logical-
  host-name stop
       :
  termination processing for services of linked JP1-series products
       :
  ```

(b) In Linux

- Creating automatic start and stop scripts

Create automatic start and stop scripts for the logical host in the /etc/rc.d/init.d directory. The following shows an example:

```
#!/bin/sh

JP1_HOSTNAME=logical-host-name

case $1 in
'start')
      if [ -x /opt/jp1as/sbin/adshmdctl ]
      then
              /opt/jp1as/sbin/adshmdctl -h $JP1_HOSTNAME start
              touch /var/lock/subsys/lock-file-name
      fi
      ;;
'stop')
      if [ -x /opt/jp1as/sbin/adshmdctl ]
      then
              /opt/jp1as/sbin/adshmdctl -h $JP1_HOSTNAME stop
              rm -f /var/lock/subsys/lock-file-name
      fi
      ;;
esac

exit 0
```

Specify for the lock file name the symbolic link name created for automatic stop without the leading numeric part (K*XX* part). For example, if the symbolic link name for automatic stop is K01_JP1_AS_CLUSTER, specify _JP1_AS_CLUSTER.

- Creating a symbolic link for automatic startup

  Create a symbolic link to the created automatic start and stop scripts in the /etc/rc.d/rc*N*.d directory (*N* indicates the execution level at startup). You must name the symbolic link in such a manner that the user-reply functionality's management daemon is started after the logical host services of the linked JP1-series products have been started. For details about how to create symbolic links, see *2.8.3(1) Starting and terminating the user-reply functionality's management daemon automatically*.

- Creating a symbolic link for automatic termination

  Create a symbolic link to the created automatic start and stop scripts in the /etc/rc.d/rc*N*.d directory (*N* indicates the execution level at termination). You must name the symbolic link in such a manner that the user-reply functionality's management daemon is terminated before the logical host services of the linked JP1-series products are stopped. For details about how to create symbolic links, see *2.8.3(1) Starting and terminating the user-reply functionality's management daemon automatically*.

Note that the user is responsible for creating the automatic start and stop scripts for the logical host services of the linked JP1-series products and the symbolic links to the created automatic start and stop scripts. For details about the names of automatic start and stop scripts for the logical host services of linked JP1-series products and the names of symbolic links, see the manuals for the linked JP1-series products.

(c) In HP-UX

- Creating automatic start and stop scripts

  Create the automatic start and stop scripts for the logical host in the /sbin/init.d directory. The following shows an example:

```
#!/bin/sh

```

```
## Set Environment-variables
PATH=/sbin:/bin:/usr/bin:/opt/jp1as/sbin
export PATH
JP1_HOSTNAME=logical-host-name

case $1 in
start_msg)
      echo "Start Advanced Shell - adshmd $JP1_HOSTNAME"
      ;;
stop_msg)
      echo "Stop Advanced Shell - adshmd $JP1_HOSTNAME"
      ;;
'start')
      if [ -x /opt/jp1as/sbin/adshmdctl ]
      then
              /opt/jp1as/sbin/adshmdctl -h $JP1_HOSTNAME start
      fi
      ;;
'stop')
      if [ -x /opt/jp1as/sbin/adshmdctl ]
      then
              /opt/jp1as/sbin/adshmdctl -h $JP1_HOSTNAME stop
      fi
      ;;
esac

exit 0
```

- Creating a symbolic link for automatic startup

  Create a symbolic link to the created automatic start and stop scripts in the `/sbin/rc2.d` directory. You must name the symbolic link in such a manner that the user-reply functionality's management daemon is started after the logical host services of the linked JP1-series products have been started. For details about how to create symbolic links, see *2.8.3(1) Starting and terminating the user-reply functionality's management daemon automatically*.

- Creating a symbolic link for automatic termination

  Create a symbolic link to the created automatic start and stop scripts in the `/sbin/rc1.d` directory. You must name the symbolic link in such a manner that the user-reply functionality's management daemon is terminated before the logical host services of the linked JP1-series products are stopped. For details about how to create symbolic links, see *2.8.3(1) Starting and terminating the user-reply functionality's management daemon automatically*.

Note that the user is responsible for creating the automatic start and stop scripts for the logical host services of the linked JP1-series products and the symbolic links to the created automatic start and stop scripts. For details about the names of automatic start and stop scripts for the logical host services of linked JP1-series products and the names of symbolic links, see the manuals for the linked JP1-series products.

(d) In Solaris

- Creating automatic start and stop scripts

  Create the automatic start and stop scripts for the logical host in the `/etc/init.d` directory. The following shows an example:

```
#!/bin/sh

JP1_HOSTNAME=logical-host-name

case $1 in
'start')
```

```
        if [ -x /opt/jp1as/sbin/adshmdctl ]
        then
                /opt/jp1as/sbin/adshmdctl -h $JP1_HOSTNAME start
        fi
        ;;
'stop')
        if [ -x /opt/jp1as/sbin/adshmdctl ]
        then
                /opt/jp1as/sbin/adshmdctl -h $JP1_HOSTNAME stop
        fi
        ;;
esac

exit 0
```

- Creating a symbolic link for automatic startup

  Create a symbolic link to the created automatic start and stop scripts in the `/etc/rc2.d/` directory. You must name the symbolic link in such a manner that the user-reply functionality's management daemon is started after the services of the linked JP1-series products have been started. For details about how to create symbolic links, see *2.8.3(1) Starting and terminating the user-reply functionality's management daemon automatically*.

- Creating a symbolic link for automatic termination

  Create a symbolic link to the created automatic start and stop scripts in the `/etc/rc0.d/` directory. You must name the symbolic link in such a manner that the user-reply functionality's management daemon is started after the services of the linked JP1-series products have been started. For details about how to create symbolic links, see *2.8.3(1) Starting and terminating the user-reply functionality's management daemon automatically*.

Note that the user is responsible for creating the automatic start and stop scripts for the logical host services of linked JP1-series products and the symbolic links to the created automatic start and stop scripts. For details about the names of automatic start and stop scripts for the logical host services of linked JP1-series products and the names of symbolic links, see the manuals for the linked JP1-series products.

(3) How to specify logical hosts

You specify commands that are to execute on a logical host in the same manner as for commands that are used on the logical host in a cluster system. For details about how to specify commands for the logical host in a cluster system, see *2.9.3 How to specify commands during cluster operation*.

2.10 Installing the HTML manual

You can use the HTML manual for JP1/Advanced Shell's custom job programs and JP1/Advanced Shell Editor by copying the HTML manual to a specified folder.

To install the HTML manual:

1. Locate the manual CD-ROM supplied as a standard with the program product.
2. From the manual CD-ROM, copy all of the JP1/Advanced Shell HTML and CSS files and the `GRAPHICS` folder to the following folders:
    - To view Help from JP1/Advanced Shell:
      installation-directory`\JP1ASE\doc\en\help`
    - To view Help from JP1/Advanced Shell Editor:
      installation-directory`\JP1ASD\doc\en\help`
    - To view Help from JP1/Advanced Shell's custom job definition programs:
      installation-directory`\JP1ASV\doc\en\help`

Part 3: Operation

3 Executing Batch Jobs

This chapter explains how to execute batch jobs and the batch job processing in JP1/Advanced Shell (execution environment).

3.1 Structure of jobs

This section explains the structure of jobs.

3.1.1 Operator's tasks in JP1/AJS jobs

This subsection explains the general procedure for the operator's tasks when JP1/AJS is used to execute jobs.

(1) Defining jobs

To use JP1/AJS to execute jobs, you must define the jobs according to the procedure explained in *2.7.2 Defining and executing a jobnet*.

(2) Executing jobs

The three methods for using JP1/AJS to execute jobs are planned execution, fixed execution, and immediate execution. For details about these three execution methods, see the *Job Management Partner 1/Automatic Job Management System 3 Operator's Guide*.

If you do not use JP1/AJS, you can execute jobs (job definition scripts) by entering commands from the command prompt or shell.

(3) Monitoring a jobnet

In JP1/AJS, you start the jobnet monitor to check job execution status.

(4) Re-executing jobs

If you need to re-execute jobs, re-execute them from the JP1/AJS - View window.

3.1.2 Jobs

Any request to start the job controller from a JP1/AJS or Windows command prompt or a UNIX shell is accepted as a job. A general user who will be using the job passes the job definition script containing a collection of instructions to the job controller.

The job controller analyzes the instructions to determine what is being requested by the user and executes the job in a manner that makes efficient use of system resources.

(1) Root jobs and child jobs

In general, a job is the unit in which the system is requested to perform a single integrated task prepared by a general user. Individual tasks that are requested are treated as being mutually independent.

A job consists of a series of processing programs. To execute these processing programs, you must define their execution order, execution conditions, and the files that will be required for them to execute.

Jobs are classified into root jobs and child jobs.

- Root job

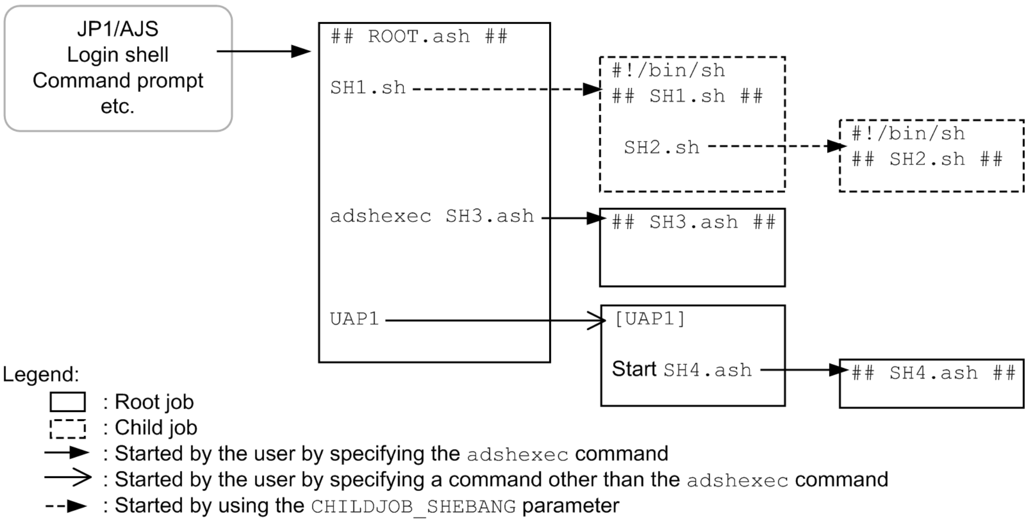
Of the jobs to be executed from programs, such as JP1/AJS and login shell, all jobs other than child jobs are root jobs.

- Child job

  In job definition scripts that are run as descendant processes of root jobs, the jobs that are executed by using one of the parameters listed below or its default definition are child jobs:

  - `CHILDJOB_EXT` parameter
  - `CHILDJOB_PGM` parameter
  - `CHILDJOB_SHEBANG` parameter

The following figure shows an example procedure for starting root jobs and child jobs.



Legend:

□ : Root job

⬚ : Child job

─► : Started by the user by specifying the `adshexec` command

→ : Started by the user by specifying a command other than the `adshexec` command

--► : Started by using the `CHILDJOB_SHEBANG` parameter

Explanation:

- Jobs `SH1.sh` and `SH2.sh` that were started by using the `CHILDJOB_SHEBANG` parameter become child jobs.
- Job `ROOT.ash` that was started from a program such as JP1/AJS becomes a root job.
- Job `SH3.ash` that was started by specifying the `adshexec` command becomes a root job.
- Job `SH4.ash` that is a descendant process of the job definition script but was started via a process that is not `adshexec` becomes a root job.

(2) Job input modes

Jobs are executed in one of the following modes according to the status of the standard input:

- Terminal input mode

  If the standard input is associated with the terminal when a job starts, the job is executed in this mode. The following is an example of starting a job in this mode:

  - In the login shell, executing the `adshexec` command with the standard input associated with the terminal

- Non-terminal input mode

  If the standard input is not associated with the terminal when a job starts, the job is executed in this mode.

  The following are examples of starting a job in this mode:

  - Starting a job from JP1/AJS
  - In the login shell, executing the `adshexec` command by redirecting the standard input from a file
  - In the login shell, executing the `adshexec` command in a middle of or at the end of a pipe

When a job starts, the `KNAX7902-I` message indicating the mode used to execute the job is displayed.

In UNIX, how a job is forcibly terminated depends on the job input mode. For details, see *3.10 Forcibly terminating jobs*.

In Windows, the job input mode has no effect on the job processing.

(3) Relationship between jobs

Root jobs are mutually independent. This means that jobs processed concurrently do not influence each other, nor can an executed root job influence root jobs that are executed later. Also, information cannot be inherited from one root job to another (except for the information in files).

However, there might be relationships between jobs, such as the following:

- Depending on the scheduling by JP1/AJS, root jobs are associated with each other in terms of their execution order.
- If multiple jobs use the same regular file concurrently, those jobs are associated with each other. A schedule must be designed in JP1/AJS so that the jobs are executed in the appropriate order.
- There might be a relationship between a root job and child jobs and between child jobs.

(4) Relationship between jobs and environment files

Both root and child jobs load the system environment file and a job environment file when the jobs start. Therefore, a root job and its child jobs use parameters in different environment files in the following cases:

- After a root job loaded the environment files at the start, the value of the ADSH_ENV environment variable is changed to a different job environment file path before a child job loads the environment files when it starts.
- After a root job loaded the environment files at the start, the contents of the system environment file or job environment file is changed before a child job loads the environment files when it starts.

If you want to run a root job and its child jobs using the same environment file parameters, do not change the value of the ADSH_ENV environment variable or the contents of the environment files while the jobs are executing.

If the export parameter is defined in the environment files, its value takes precedence over the value of the environment variable that the job inherits from its parent process. The following shows an example.

- Root job root.ash

```
export ENV1=SCRIPTFILE
childjob.ash  #start the child job
```

- Child job childjob.ash

```
echo $ENV1
```

- Contents of environment file adshrc.ase that are loaded by root job root.ash and child job childjob.ash

```
export ENV1=ENVFILE
```

- echo output results of child job childjob.ash

```
export ENV1=ENVFILE
```

In this example, the jobs are run in the following procedure:

1. Root job root.ash starts and ENVFILE is set in ENV1 by the export parameter in environment file adshrc.ase.

2. Root job `root.ash` sets `SCRIPTFILE` in environment variable `ENV1` before its child job `childjob.ash` starts.
3. Child job `childjob.ash` starts and inherits environment variable `ENV1` from root job `root.ash`. Immediately after the process of child job `childjob.ash` started, the value of `ENV1` was `SCRIPTFILE`.
4. Child job `childjob.ash` loads environment file `adshrc.ase` at the start. `ENVFILE` is set in `ENV1` by the `export` parameter.
5. The result of `echo` by child job `childjob.ash` is `ENVFILE`, which was set in step 4.

(5) Temporary files and regular files

Batch jobs perform processing by referencing data, such as the information provided by open base products, as temporary files or regular files.

(a) Temporary files

Temporary files are used temporarily during job execution. They are created automatically by jobs and job steps and deleted automatically when the jobs terminate. Temporary files are created in a directory defined in the environment files.

We recommend that you manage the temporary files for batch jobs separately from the temporary files for applications. The directory for storing temporary files in JP1/Advanced Shell is specified in the `TEMP_FILE_DIR` parameter. Normally, the temporary files are deleted, but they might remain if a failure occurs. For this reason, make sure that you delete temporary files periodically.

(b) Regular files

Regular files are used to input and output job definition scripts and can be placed in any directory. These files are retained as job results after jobs have terminated, but you can delete them during job execution.

(6) Asynchronously executed processes

In JP1/Advanced Shell, a job is not terminated until all the related root jobs, child jobs, and commands have terminated.

(a) Asynchronous execution by using & and |& (UNIX only)

A job is not terminated until all the processes executed with `&` and `|&` specified are completed.

Note that if an asynchronously executed process is terminated due to receipt of a signal such as `SIGSTOP` at the time the job terminates, the job might be terminated without waiting for that process to terminate. If you want to terminate a job without waiting for termination of asynchronously executed processes, create the job definition script in such a manner that the OS shell is used only for the parts that you want to execute asynchronously.

The following shows an example.

Example of job definition script that waits for termination of asynchronously executed processes

```
#!/opt/jp1as/bin/adshexec
mycommand A &
```

If this job definition script is executed in JP1/Advanced Shell, the job is terminated after `mycommand` has completed. If you want to terminate this job without waiting for completion of `mycommand`, create the job definition script as shown below. This example uses `/bin/ksh` as the OS shell.

Example of job definition script that does not wait for termination of asynchronously executed processes

```
#!/opt/jp1as/bin/adshexec
/bin/ksh exec_cmdA.sh
```

Contents of exec_cmdA.sh

```
mycommand A &
```

(b) Using the exec command to execute external commands

If an external command is specified in the argument of the exec command, the adshexec command executes the external command as a child process and waits for its completion. When the external command is completed, no commands following the exec command will be executed. In such a case, the return code of the completed external command becomes the return code of the job definition script.

(c) Message notifying that the command is to wait for completion of asynchronously executed process

When the adshexec command starts, it outputs the KNAX7901-I message notifying that the command is to wait for completion of asynchronously executed process when the job terminates. This message is normally output to the job execution logs, system execution logs, and standard error. During debugging, this message is output to the standard error output.

(d) Job processing when an asynchronously executed process is stopped (UNIX only)

If an asynchronously executed process is stopped due to receipt of a signal such as SIGSTOP, JP1/Advanced Shell sends SIGHUP and SIGCONT to child processes or descendant processes when the job terminates. When this transmission is completed, JP1/Advanced Shell waits for one second and then performs job postprocessing.

How SIGHUP and SIGCONT are transmitted depends on the job input mode, as described in the following:

- Terminal input mode
  SIGHUP and SIGCONT are sent only to the child processes of the adshexec command. SIGHUP and SIGCONT are not sent to any of the descendant processes of the adshexec command, including grandchild processes. If grandchild processes remain, use the ps command to obtain the process IDs of the remaining processes, and then manually terminate them with the kill command.
- Non-terminal input mode
  SIGHUP and SIGCONT are sent to the descendant processes of the adshexec command.

3.1.3 Job steps

The job step is a unit of job execution and is a part of a job definition script consisting of a group of commands. When you allocate regular files and temporary files, you can define the files that are valid only within one job step. Such files are postprocessed when the job step in which the files were allocated terminates.

Several job steps are mutually related. If one job step is not processed correctly, execution of the next job step might be meaningless. In such a case, you can specify job step execution conditions so that subsequent processing is skipped.

Jobs steps are used for the following purposes:

- Automating command and program error processing
- Controlling shell script execution in units of job steps

(1) Automating termination processing and log output in the event of a command error

Conventional scripts require the return code to be checked, error messages output, temporary files deleted, and other error handling procedures to be performed for each command that is executed.

A job step enables you to monitor the return code of a command executed within the job step, output an error message, delete temporary files, and perform predefined processing (such as an error handling procedure).

The following shows an example script that uses job steps to automate termination processing and output a log in the event of an error.

Comparison based on whether job steps are used

When job steps are not used

```
progA
ret=$?
if [[ $ret != 0 ]]; then    --- (1)
  echo "progA error"        --- (1)
  exit $ret                 --- (1)
fi                          --- (1)

TEMP="/tmp/tempfile"

progB ${INFILE_B} ${TEMP}
ret=$?
if [[ $ret != 0 ]]; then
  echo "progB error"
  rm ${TEMP}                --- (2)
  exit $ret
fi

progC ${TEMP} ${OUTFILE_C}
ret=$?
if [[ $ret != 0 ]]; then
  echo "progC error"
fi

rm ${TEMP}                  --- (2)
exit $ret
```

When job steps are used

```
#-adsh_job J01

#-adsh_step_start S01        --- (1)
  progA
#-adsh_step_end

#-adsh_step_start S02
  #-adsh_file_temp TEMP      --- (2)
  progB ${INFILE_B} ${TEMP}
```

```
10    progC ${TEMP} ${OUTFILE_C}
11  #-adsh_step_end
```

The following explains (1) and (2) in the example script.

About (1)

If you do not use job steps, you must check for an error each time a command is executed, and then code error message output processing and script cancellation processing.

On the other hand, if you define a group of commands as a job step, you can output an error message automatically in the event of an error, and then terminate the job step without executing the subsequent commands.

About (2)

If no job step is used, the user must code each and every process for deleting the created temporary files.

If a job step is used, the temporary files allocated for the job step by JP1/Advanced Shell's temporary file function are deleted automatically when the job step terminates.

When you use job steps, processes such as in (1) and (2) above can be automated. The following shows the log output results in the case where `progB` results in an error.

Job execution logs (excerpt)

```
********  JOB CONTROLLER MESSAGE  ********
17:05:25 000515 KNAX0091-I J01 The job started.
17:05:25 000515 KNAX7901-I The adshexec command will wait for all
asynchronous processes at the end of the job.
17:05:25 000515 KNAX7902-I The adshexec command will run in tty stdin
mode.
17:05:25 000515 KNAX0092-I J01.S01 step started.
17:05:25 000515 KNAX6116-I Execution of the command ./progA (line=4)
finished successfully. exit status=0 execution time=0.000s CPU time=0.000s
17:05:25 000515 KNAX6597-I J01.S01 step succeeded. exit status=0
execution time=0.001s CPU time=0.000s
17:05:25 000515 KNAX0092-I J01.S02 step started.
17:05:25 000515 KNAX1601-I J01.S02 Allocation of file(s) for a step
started.
17:05:25 000515 KNAX6409-I The file TEMP was allocated as "create".
path=/var/opt/jp1as/temp/TEMP_000515_J01_CGNCtb
17:05:25 000515 KNAX6521-E The command ./progB (line=9) failed. exit
status=1 execution time=0.000s CPU time=0.000s  ... (1)
17:05:25 000515 KNAX6410-I The file TEMP was deallocated as "del".
path=/var/opt/jp1as/temp/TEMP_000515_J01_CGNCtb
17:05:25 000515 KNAX1604-I The file /var/opt/jp1as/temp/
TEMP_000515_J01_CGNCtb was deleted.                          ... (2)
17:05:25 000515 KNAX6596-E J01.S02 step failed. exit status=1 execution
time=0.002s CPU time=0.000s
17:05:25 000515 KNAX0101-E J01 An error occurred during execution of the
job.
17:05:25 000515 KNAX0098-I J01 The job ended. exit status=1 execution
time=0.006s CPU time=0.000s                      ... (3)
```

The following explains job execution logs (1) through (3):

About (1)

The command resulted in an error, so the subsequent commands were not executed.

About (2)

A temporary file allocated by JP1/Advanced Shell's temporary file function was deleted automatically.

About (3)

The job terminated with the return code specified in the error handling procedure.

(2) Controlling execution in units of job steps

You can define a group of related commands as a job step and control job execution according to the results of the job step's processing. JP1/Advanced Shell provides various functions for controlling execution in units of job steps.

The following shows an example script that controls execution in units of job steps.

Comparison based on whether job steps are used

When job steps are not used

```
retmax=0
VAR=`progA`
export VAR
progB

tempVAR=$VAR                       --- (1)
VAR=`progC`
progD
retD=$?                            --- (2)
if [[ $retmax -lt $retD ]]; then
  retmax=$retD                     --- (3)
fi
VAR=$tempVAR                       --- (1)

if [[ $retD -ge 16 ]]; then        --- (4)
  exit $retD
fi

if [[ $retD -ne 0 ]]; then         --- (5)
  if [[ $retD -eq 4 ]]; then       --- (2)
    result="progD: warning"
  else
    result="progD: error"
  fi
  progE $result
  retE=$?
  if [[ $retmax -lt $retE ]]; then
    retmax=$retE                   --- (3)
  fi
  if [[ $retE -ge 16 ]]; then      --- (4)
    exit $retE
  fi
fi

progF                              --- (6)
exit $retmax                       --- (3)
```

When job steps are used

```
#-adsh_job_stop 16:                     --- (4)
VAR=`progA`
export VAR
progB

```

```
#-adsh_step_start S01 -stepVar VAR      --- (1)
  VAR=`progC`
  export VAR
  progD
#-adsh_step_end

#-adsh_step_start S02 -run abnormal     --- (5)
  if [[ $ADSH_STEPRC_S01 -eq 4 ]]; then --- (2)
    result="STEP01: warning"
  else
    result="STEP01: error"
  fi
  progE $result
#-adsh_step_end

#-adsh_step_start S03 -run always       --- (6)
  progF
  exit $ADSH_RC_STEPMAX                 --- (3)
#-adsh_step_end
```

The following explains (1) through (6) in the example script.

About (1)

If you do not use job steps, using a shell variable with a duplicated name for another purpose requires a separate process (such as saving its value temporarily in another variable).

If you use a job step, you can use shell variables that are valid only within the job step by using the `stepVar` attribute of the `#-adsh_step_start` extended script command to declare the variable names.

About (2)

If you do not use job steps, to branch processing based on the return code of a specific command, you must store the command's return code in a separate shell variable.

If you use job steps, you can reference each job step's return code, which is set automatically by JP1/Advanced Shell.

About (3)

If you do not use job steps, to reference the maximum value of a command's return code, you must update the maximum value each time the command is executed.

If you use job steps, you can reference the maximum value of each job step's return code, which is set automatically by JP1/Advanced Shell.

About (4)

If you do not use job steps, to terminate a script when a command's return code exceeds a threshold value, you must check the threshold value each time a command is executed.

If you use job steps, you can monitor each job step's return code automatically by using the `#-adsh_job_stop` extended script command to declare the threshold value.

About (5)

If you do not use job steps, you must control execution by determining whether the subsequent processing is to be executed based on a command's return code.

If you use job steps, you can define that a job step is to be executed only when the preceding process results in an error. You do this by specifying `abnormal` in the `run` attribute of the `#-adsh_step_start` extended script command.

About (6)

If you use job steps, you can define a job step to always execute, regardless of whether the preceding process was successful. You do this by specifying `always` in the `run` attribute of the `#-adsh_step_start` extended script command.

3.2 Starting batch jobs

This section explains for each execution method how to start batch jobs. It also explains the job controller processing after batch jobs have started.

3.2.1 Starting jobs by using JP1/AJS from the execution environment

This subsection explains how to start JP1/Advanced Shell's batch job applications by using JP1/AJS from the execution environment.

For details about using JP1/AJS for automation of batch job applications, see the applicable JP1/AJS manual. For details about how to define and execute JP1/Advanced Shell jobs in jobnets, see *2.7.2 Defining and executing a jobnet*.

When you automate batch job applications, you can reduce costs as well as run your system more securely with a smaller staff. JP1/AJS is a product for automating standard batch job applications. JP1/AJS can also automate a combination of complex batch job applications. Using JP1/Advanced Shell together with JP1/AJS operations provides the following advantages:

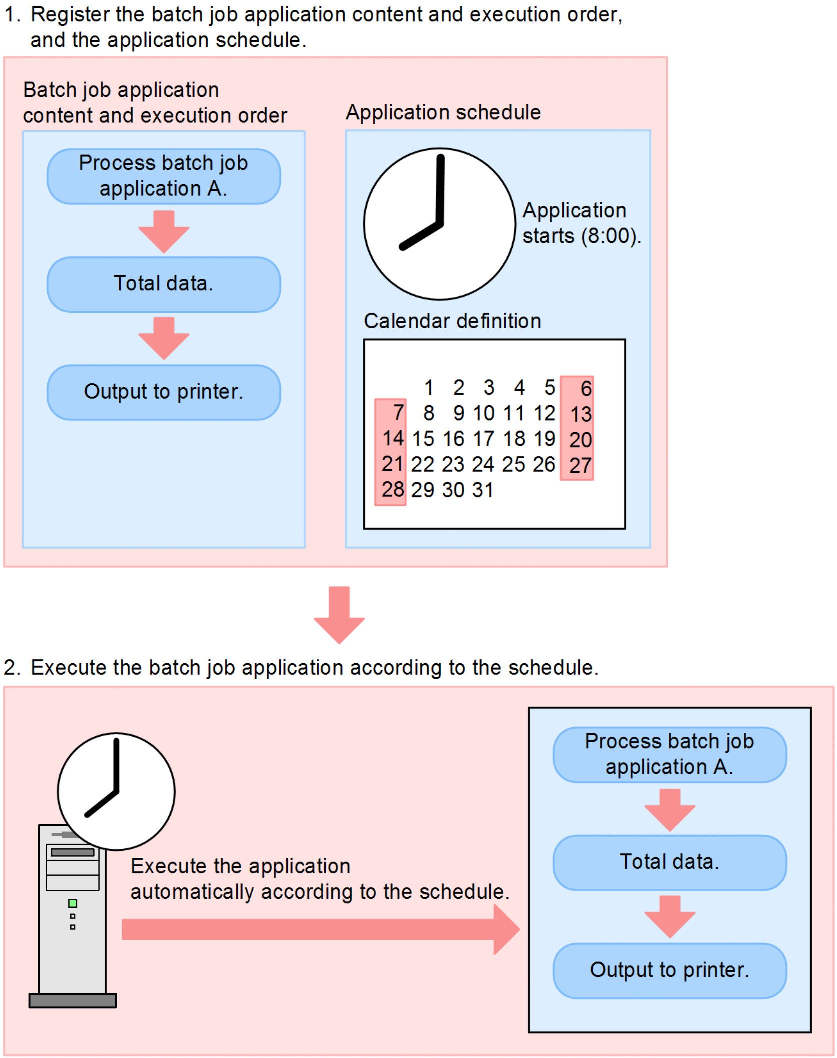
- You can use the temporary file function to allocate files that are used temporarily and delete them when the job or job step terminates.
- You can share job definitions among multiple applications by calling external scripts.
- You can achieve flexible job definitions by changing, adding, and deleting coding in job definition scripts.

To use JP1/AJS to execute batch job applications automatically, you must define the following:

- Content and processing order of the batch job applications
- Schedule for executing the batch job applications or registration of events that trigger execution of the batch job applications

The following figure provides an overview of using JP1/AJS to automate batch job applications. The numbers in the figure correspond to the numbers in the explanation that follows.

Figure 3–1: Overview of using JP1/AJS to automate batch job applications


1. Register the batch job application content and execution order, and the application schedule.
Batch job application content and execution order
Process batch job application A.
Total data.
Output to printer.
Application schedule
Application starts (8:00).
Calendar definition
1 2 3 4 5 6
7 8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
2. Execute the batch job application according to the schedule.
Execute the application automatically according to the schedule.
Process batch job application A.
Total data.
Output to printer.

1. Registers the batch job application content and execution order, and the application schedule.
2. The batch application is executed automatically according to the registered schedule.

(1) Defining batch job applications and their execution order

Many applications are executed at a specified time in a specified order.

For example, totaling of sales slips is executed in the following order:

1. Extract data from the database.
2. Sort data.
3. Output to printer.

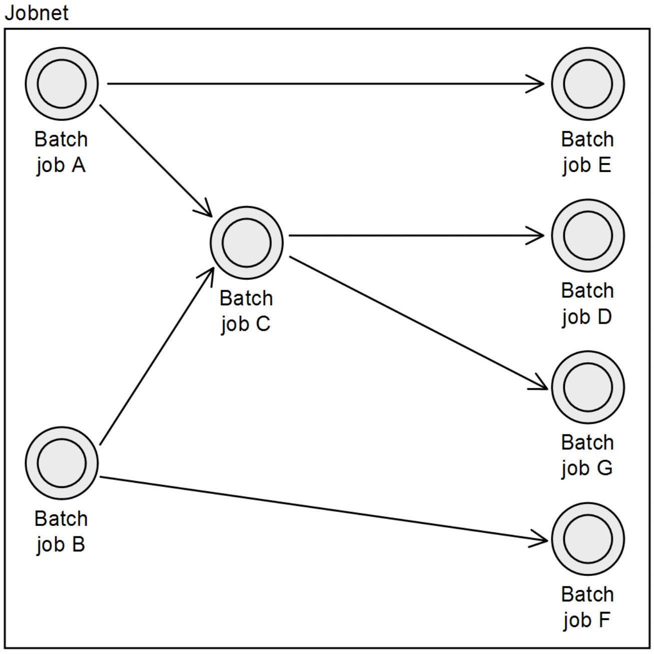
Steps 1 through 3 can be automated as a job controller's job step by defining these steps in a job definition script file, but the task of extracting data from the database at 12:00 cannot be automated. To define batch job applications and their execution order in JP1/Advanced Shell and JP1/AJS, define in the job controller the series of steps that make up the applications and then define the relationships among the definitions of the individual batch job applications and their execution order as the JP1/AJS execution order or execution time.

If batch job applications are broken up into task units, such as commands, application programs, or job definition scripts, JP1/AJS alone can achieve jobs equivalent to those that can be achieved by JP1/Advanced Shell. They are also called jobs in JP1/AJS.

When batch job applications and execution orders are defined in JP1/Advanced Shell and JP1/AJS, the batch job execution orders are defined by using jobnets in JP1/AJS.

The following figure shows a jobnet used when batch job applications and their execution order are defined in JP1/Advanced Shell and JP1/AJS.

Figure 3–2: Jobnet used to define batch job applications and their execution order in JP1/Advanced Shell and JP1/AJS



Explanation

The following explains the execution order of the batch jobs that are defined by using JP1/AJS jobnet.

- When batch job A terminates, batch job E is executed.
- When batch jobs A and B terminate, batch job C is executed.
- When batch job C terminates, batch jobs D and G are executed.
- When batch job B terminates, batch job F is executed.

(2) Defining the definition schedule of batch job applications and their execution order

To automatically define a definition schedule for multiple batch job applications and their execution order, you need a schedule definition that determines when this definition is to be executed.

JP1/AJS's schedule definition contains such information as a calendar that specifies the company's business days and holidays, the date and time execution is to begin, and an execution interval. Based on this definition, JP1/AJS determines the execution schedule and automatically starts JP1/Advanced Shell's job execution on the specified date and time.

(3) Registering the timing of starting batch job applications

You can register an event, such as when a file is created or when some specific event occurs, as the timing for starting a batch job application. If you have registered the required information, you can start a batch job application at a specified time as well as whenever some specified event (such as creation of a file) occurs.

3.2.2 Starting batch jobs by using commands from the execution environment

(1) Specifying job definition scripts in the argument of the adshexec command

To start batch jobs by using commands from the execution environment, you use the `adshexec` command shown below. In Windows, enter the command from the command prompt; in UNIX, enter the command from the shell.

```
adshexec batchjob1.ash
```

You can also use the `-r` option of the `adshexec` command to directly specify the contents of a job definition script. To specify multiple commands, use the `adshexec` command as follows:

```
adshexec -r "export DATA=file01 ; pgm001"
```

In UNIX, you can also debug batch jobs by specifying the `-d` option in the `adshexec` command. For details about the `adshexec` command, see *adshexec command (executes a batch job)* in *8.3 Shell operation commands*.

(2) Specifying job definition scripts as commands

In UNIX, you can start a batch job by simply entering the name of the job definition script (assuming that execution permissions have been granted to that job definition script) by specifying the path of the `adshexec` command beginning with `#!` on the first line (example: `#! /opt/jp1as/bin/adshexec`).

Job definition script file (file name: `/home/user1/scripts/batchjob2.ash`):

```
#! /opt/jp1as/bin/adshexec
#-adsh_job SAMPLE
(followed by the body of the job definition script)
```

Execution example of batch job start:

```
/home/user1/scripts/batchjob2.ash
```

Notes

In Windows, a batch job cannot be started by a method such as specifying from the command prompt the path of the `adshexec` command beginning with `#!` on the first line, and then entering the file name of the job definition script.

However, if you provide a job definition script in which `#!` followed by `/opt/jp1as/bin/adshexec` or `/opt/jp1as/bin/adshexec -m MINIMUM` is specified on the first line, and then enter its file name from another job definition script, you can start child jobs in Window as well as in UNIX. Therefore, we recommend that you specify `#!` followed by `/opt/jp1as/bin/adshexec` or `/opt/jp1as/bin/adshexec -m MINIMUM` on the first line of new job definition scripts even in Windows.

If the first line already contains `#!/bin/sh`, such as when existing shell scripts have been migrated, you can also run the shell scripts as child jobs without editing the scripts.

For details about child jobs, see *3.2.3 Running job definition scripts as child jobs*.

3.2.3 Running job definition scripts as child jobs

This subsection explains how to run job definition scripts as child jobs and the operation of child jobs. For details about priority, see *5.1.11(3) Priority of command execution methods* and *5.1.11(4) Priority of child jobs or external commands that have the same name as the function*.

(1) How to execute child jobs

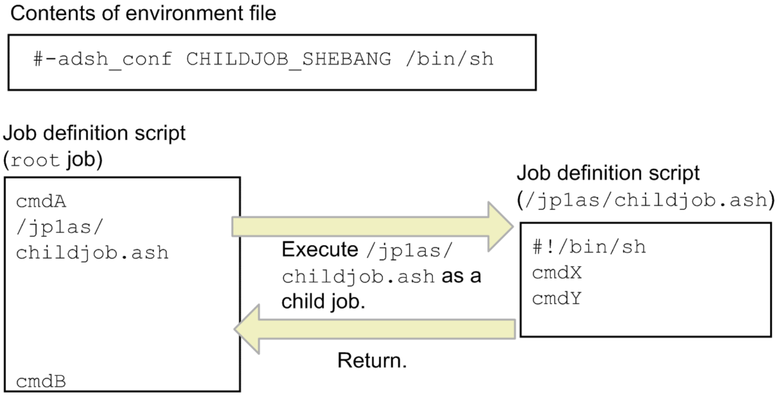
(a) Executing child jobs by specifying parameters in environment files

In job definition scripts that are run as descendant processes of root jobs, the jobs that are executed by using one of the parameters listed below or its default definition are called child jobs:

- `CHILDJOB_EXT` parameter
- `CHILDJOB_PGM` parameter
- `CHILDJOB_SHEBANG` parameter

The following figure shows an example operation for starting a child job by specifying the `CHILDJOB_SHEBANG` parameter.

Figure 3–3: Example operation for starting a child job



Contents of environment file

```
#-adsh_conf CHILDJOB_SHEBANG /bin/sh
```

Job definition script (`root` job)

```
cmdA
/jp1as/
childjob.ash

cmdB
```

Job definition script (`/jp1as/childjob.ash`)

```
#!/bin/sh
cmdX
cmdY
```

This example specifies job definition script `childjob.ash` in the root job's job definition script. Because `childjob.ash` satisfies the `CHILDJOB_SHEBANG` parameter definition, JP1/Advanced Shell starts JP1/Advanced Shell's job as a child process and executes `childjob.ash` as a child job.

If the root job that starts the child job is executed on a logical host, the child job is also executed on the logical host.

(b) Executing child jobs by using a default definition for the parameter

By using a default definition for the `CHILDJOB_SHEBANG` parameter, you can start child jobs without having to specify the parameter in environment files.

The following two values have been defined as defaults for the `CHILDJOB_SHEBANG` parameter:

Default definition	Output mode when the child job is started
/opt/jp1as/bin/adshexec	Operation is performed according to the specified OUTPUT_MODE_CHILD parameter.
/opt/jp1as/bin/adshexec -mMINIMUM	Operation is performed in the minimum output mode.

If you specify a job definition script whose first line is #! /opt/jp1as/bin/adshexec in another job definition script, the former can be run as a child job. If you want to execute only a specific child job in the minimum output mode, specify #! /opt/jp1as/bin/adshexec -mMINIMUM on the first line of that job definition script.

The operation of a child job executed by a default definition is the same as for a child job started by the method described in *(a) Executing child jobs by specifying parameters in environment files*.

Note that the CHILDJOB_SHEBANG parameter specified in the environment file takes precedence over the CHILDJOB_SHEBANG parameter's default definitions. If a value that is the same as a default definition is specified in the CHILDJOB_SHEBANG parameter in the environment file, the following takes effect:

- Contents of the environment variable

```
#-adsh_conf CHILDJOB_SHEBANG "/opt/jp1as/bin/adshexec -mMINIMUM"
```

- Job definition script that is started by the child job

```
#! /opt/jp1as/bin/adshexec -mMINIMUM
                  :
```

In this example, /opt/jp1as/bin/adshexec -mMINIMUM specified in the job definition script satisfies the definition of the CHILDJOB_SHEBANG parameter in the environment file. Therefore, the output mode for the child job depends on the specified OUTPUT_MODE_CHILD parameter.

(2) Functional comparison with root jobs and external scripts

The following table compares the functions of root jobs, child jobs, and external scripts:

Function	Type of job		External script	
	Root job	Child job	External script of . (dot) command	External script of #-adsh_script
Relationship of processes with the calling job	Runs in the child process of the calling job.	Runs in the child process of the calling job.	Runs in the same process as the calling job.	Runs in the same process as the calling job.
Job controller to be started	• In UNIX adshexec command • In Windows adshexec.exe command + adshexecsub.exe command	• In UNIX adshexec command • In Windows adshexecsub.exe command	None	None
Spool job directory	Created	Created in the root job's spool job directory and deleted when the job terminates. The user selects one of the following about the job execution log:	Not created. (outputs command execution results to the calling job's JOBLOG and does not output script images).	Not created. (outputs command execution results to the calling job's JOBLOG and outputs script images to

Function	Type of job		External script	
	Root job	Child job	External script of . (dot) command	External script of #-adsh_script
Spool job directory	Created	• Output only JOBLOG to stderr. • Merge into the root job's job execution log.	Not created. (outputs command execution results to the calling job's JOBLOG and does not output script images).	the calling job's SCRIPT).
Job start and termination messages	Provided (KNAX0091-I and KNAX0098-I)	Provided (KNAX6571-I and KNAX6578-I)	None	None
Loading of environment files	Loaded	Loaded	Not loaded. (depends on the calling job's processing).	Not loaded. (depends on the calling job's processing).
Whether standard input can be used	Can be used.	Can be used.	Can be used.	Can be used.
Destination of standard output	Depends on the specification of the -s option, the -m option, the OUTPUT_STDOUT parameter, and the OUTPUT_MODE_ROOT parameter.	Output destination inherited from its parent process.	Depends on the calling job's processing.	Depends on the calling job's processing.

(3) Behavior of child jobs when signals are received

This subsection explains the behavior of descendant jobs when signals are received.

The following example jobs are used to explain the behavior of descendant jobs when termination request signals are sent to the root job, descendant jobs, and an external command:

```
adshexec(1)-adshexec(2)-adshexec(3)-sleep
```

The following table describes the behaviors when forced termination is performed from JP1/AJS (by sending SIGTERM from JP1/AJS to adshexec(1)) and SIGTERM is sent from the login shell to adshexec(1), adshexec(2), adshexec(3), and sleep.

Timing	adshexec(1)	adshexec(2)	adshexec(3)	sleep
Forced termination from JP1/AJS	Terminates with error rc=143.	Terminates with error rc=143 due to signal.	Terminates with error rc=143 due to signal.	Terminates with error rc=143 due to signal.
Sending SIGTERM from the login shell to adshexec(1)	Terminates with error rc=143 due to signal.	Terminates with error rc=143 due to signal.	Terminates with error rc=143 due to signal.	Terminates with error rc=143 due to signal.
Sending SIGTERM from the login shell to adshexec(2)	• If a job step is specified The job step terminates with error rc=143 and executes a step error block and the step	Terminates with error rc=143 due to signal.	Terminates with error rc=143 due to signal.	Terminates with error rc=143 due to signal.

Timing	adshexec(1)	adshexec(2)	adshexec(3)	sleep
Sending `SIGTERM` from the login shell to `adshexec(2)`	following `run abnormal/always`. • If a job step is not specified Performs the next processing.	Terminates with error `rc=143` due to signal.	Terminates with error `rc=143` due to signal.	Terminates with error `rc=143` due to signal.
Sending `SIGTERM` from the login shell to `adshexec(3)`	Depends on the result of `adshexec(2)`	• If a job step is specified The job step terminates with error `rc=143` and executes a step error block and the step following `run abnormal/always`. • If a job step is not specified Performs the next processing.	Terminates with error `rc=143` due to signal.	Terminates with error `rc=143` due to signal.
Sending `SIGTERM` from the login shell to `sleep`	Depends on the result of `adshexec(2)`	Depends on the result of `adshexec(3)`	• If a job step is specified The job step terminates with error `rc=143` and executes a step error block and the step following `run abnormal/always`. • If a job step is not specified Performs the next processing.	Terminates with error `rc=143` due to signal.

(4) Notes about child jobs that are executed from another child job

If a descendant job that was executed from another descendant job is terminated abruptly through the intermediate job by a means such as `SIGKILL` in UNIX or `TerminateProcess` in Windows, the root job might terminate without waiting for all its descendant jobs to finish. To prevent such an outcome, do not execute an abrupt termination of this type. However, should such occur, check the execution results of the relevant root job and its descendant jobs.

For the descendant jobs other than the abruptly terminated job, the spool job directory might have been deleted or it might remain after a failed attempt to delete it. Even if it has been deleted, the logs will still be preserved, because the contents of `JOBLOG` will have been output to the standard error output.

Example:

This example illustrates the case where a descendant job is executed from another descendant job (the chain of one job calling the next is indicated by ➜):

```
[root job] ➜ [descendant job (child)] ➜ [descendant job (grandchild)]
```

In this case, if [descendant job (child)] terminates abruptly, [root job] might terminate earlier than [descendant job (grandchild)]. In such a case, the behavior of each job and the status of the spool job directory are as follows:

Item	Type of job		
	Root job	Descendant job (child)	Descendant job (grandchild)
Behavior of the job	The job behaves as though the descendant job (child) had terminated with an error. The behavior is the same as when a user program terminates abruptly due to an error.	The job is terminated abruptly.	The job terminates normally. However, in Windows, it might behave as if it was forcibly terminated, depending on the status of other, related jobs.
Status of the spool job directory	In Windows, if the descendant job (grandchild) has already opened the job execution log, the renaming of the spool job directory will fail. Otherwise, or in the case of UNIX, the spool job directory will be renamed as per normal.	Directory remains without being deleted.	If the root job has successfully renamed its spool job directory, the renaming of this spool job directory will fail. Otherwise, the contents of `JOBLOG` will be output to `stderr` and will be deleted as per normal.

3.2.4 Specifying what is to be executed by a job from the command line

If you use the `-r` option of the `adshexec` command to specify on the command line any commands that can be described in a job definition script file, such as standard shell commands and UNIX-compatible commands, you can execute the commands without having to create a job definition script file. To specify the `pwd` command, which is a standard shell command, on the command line, execute the following `adshexec` command:

```
adshexec -r pwd
```

You can specify on the command line any contents that can be described in a job definition script file, such as multiple commands delimited by the command separator. The following `adshexec` command specifies multiple commands on the command line:

```
adshexec -r "export DATA=file01 ; pgm001"
```

If you specify any spaces on the command line, you must enclose the command line specification in single or double quotation marks (`'` or `"`). Because metacharacters, such as `$`, `*`, and the semicolon (`;`), are expanded, depending on the shell used to execute the `adshexec` command, you must enclose them in single or double quotation marks (`'` or `"`) or use an escape character (\). To specify metacharacters, execute the `adshexec` command as follows:

In UNIX:

■ When an escape character is specified

- Entered command:

```
adshexec -m MINIMUM -r "A=(1 2 3); echo \${A[@]}"
```

- Output results:

```
1 2 3
```

- When no escape character is specified
- Entered command:

```
adshexec -m MINIMUM -r "A=(1 2 3); echo ${A[@]}"
```

- Output results:

  Nothing is output.

■ Output of positional parameter `$0` (when an escape character is specified)

- Entered command:

```
adshexec -m SIMPLE -r "echo \$0"
```

- Output results:

```
adshexec
```

- Output of positional parameter `$0` (when no escape character is specified)
- Entered command:

```
adshexec -m SIMPLE -r "echo $0"
```

- Output results:

```
-bash
```

`adshexec` receives the contents obtained by converting positional parameter `$0` by the login shell. If the login shell is `bash`, `-bash` is output.

In Windows:

■ When an escape character is specified

- Entered command:

```
adshexec -m MINIMUM -r "A=(1 2 3); echo \${A[@]}"
```

- Output results:

```
${A[@]}
```

■ When no escape character is specified

- Entered command:

```
adshexec -m MINIMUM -r "A=(1 2 3); echo ${A[@]}"
```

- Output results:

```
1 2 3
```

■ Output of positional parameter `$0` (when an escape character is specified)

- Entered command:

```
adshexec -m SIMPLE -r "echo \$0"
```

- Output results:

```
$0
```

- Output of positional parameter `$0` (when no escape character is specified)
- Entered command:

```
adshexec -m SIMPLE -r "echo $0"
```

- Output results:

```
adshexec
```

Note the following about executing the adshexec command with the -r option specified:

- If you want to use the execution results of the command line in other programs or output the execution results of the command line to the console or files, also specify -m SIMPLE or -m MINIMUM at the same time.
- Collection of coverage information by the -t or BATCH_CVR parameter is not supported.
- -r CMDLINE is output for the following part of a path that indicates the path name of the job definition script file:
  - Path name of the job definition script file that is output to the script image file
  - Path name of the job definition script file that is output to the operation information for the job definition script
  - Path name of the job definition script file that is displayed in message texts output by JP1/Advanced Shell
- The $0 positional parameter stores executable program name adshexec.
- Spool directories are created when a command is executed with the -r option specified. Spool job directories are created only when the -r option is specified as the root job. Note that frequent execution of a command with the -r option specified increases the number of spool jobs in the spool.

3.2.5 Job controller processing after batch jobs have started

Batch jobs are executed as job controller processes. The job controller is started in the following manner:

- In the execution environment, the job controller is started from JP1/AJS - Agent according to JP1/AJS's schedule.
- In the execution environment, the user enters a command to start a process called a job controller.
- On the development PC, the user runs a text while editing the development environment.

To process a job after starting it:

1. The job controller analyzes the options for starting a batch job and JP1/Advanced Shell's environment files.
2. The job controller analyzes the entered job definition script file at the initial stage. During this analysis process, the job controller analyzes syntax and creates a table for storing job information without executing commands.
3. The job controller's job execution control analyzes and executes the job definition script file.
4. The file management function used in extended script commands allocates and releases regular files, temporary files, and program output data files.
5. The shell variables and environment variables in extended script commands store job step return codes in shell variables and set job information in environment variables so that this information can be referenced by user programs.

In both the Windows and the UNIX execution environment, the job controller analyzes and executes job definition scripts. For details about creating job definition scripts, see *4. Using JP1/Advanced Shell - Developer (Windows Only)* and *5. Creating Job Definition Scripts*.

3.3 Outputting job execution results

Job execution results are output as spool job directories under the spool root directory. The contents of some directories in the spool job directory are output as job execution logs that can be used to check messages during job execution.

You can suppress output to the job execution log of some types of messages.

3.3.1 Specifying the destinations of the standard output and the standard error output

The destinations of the standard output and the standard error output for jobs executed in JP1/Advanced Shell depend on specified options and the job execution mode, as described in the following table:

Item		Standard error output		Standard output		
		Root job	Child job	Root job		Child job
				SPOOL specified in the OUTPUT_STDOUT parameter[#1]	PARENT specified in the OUTPUT_STDOUT parameter[#1]	
Normal execution	Expansion output mode[#2]	Spool files	Output destination at the time the process starts	Spool files	Output destination at the time the process starts	Output destination at the time the process starts
	Simple output mode or minimum output mode[#2]	Output destination at the time the process starts	Output destination at the time the process starts	Output destination at the time the process starts	Output destination at the time the process starts	Output destination at the time the process starts
Debug execution		Output destination at the time the process starts	Output destination at the time the process starts	Output destination at the time the process starts	Output destination at the time the process starts	Output destination at the time the process starts

#1

In addition to the `OUTPUT_STDOUT` parameter, the `adshexec` command's `-s` option can be used to specify the value.

#2

Use the following commands or parameters to specify the expansion, simple, or minimum output mode:

- `-m` option in the `adshexec` command
- `-m` option in the `adshscripttool` command
- `OUTPUT_MODE_ROOT` parameter (for root jobs)
- `OUTPUT_MODE_CHILD` parameter (for child jobs)

3.3.2 Outputting job execution results to spool

In the spool root directory specified in the environment file, create a directory for each job and output job execution results to that directory. Job execution logs and the files output by programs in job steps are output to the directory for each job.

The following shows the structure of the spool directory:

```
spool-root-directory
|-lock-file
+-spool-job-directory
    +-.adshallocfile or adshallocfile#1
    +-.joborder or adsh.joborder#1
    +-.sysout or sysout.ini#1
    +-EVENTFILE_ROOT_INF_000000_000000_000001
    +-EVENTFILE_execution-start-date-and-time_job-ID
      :
    +-EVENTFILE_execution-start-date-and-time_job-ID
    +-JOBLOG#2
    +-JOBLOG_job-ID_sequence-number
    +-JOBLOG_number-giving-the-order-in-which-a-child-job-starts#1
    +-SCRIPT#2
    +-SCRIPT_number-giving-the-order-in-which-a-child-job-starts#1
    +-STDERR#2
    +-STDOUT#2
    +-step-number_step-name_STDOUT#2
    +-step-number_step-name_STDERR#2
    +-0000_job-name_sequence-number-of-file-environment-variable-definition-
name_file-environment-variable-definition-name#3
    +-Cnumber-giving-the-order-in-which-a-child-job-starts_0000_job-
name_sequence-number-of-file-environment-variable-definition-name_file-
environment-variable-definition-name#3
    +-step-number_step-name_sequence-number-of-file-environment-variable-
definition-name_file-environment-variable-definition-name#3
      :
    +-step-number_step-name_sequence-number-of-file-environment-variable-
definition-name_file-environment-variable-definition-name
    |-Cnumber-giving-the-order-in-which-a-child-job-starts_step-number_step-
name_sequence-number-of-file-environment-variable-definition-name_file-
environment-variable-definition-name#3
```

#1

This is a temporary file created during job execution. The following explains the contents of such temporary files:

File name	Description
`.adshallocfile` or `adshallocfile`	Allocation management file
`.joborder` or `adsh.joborder`	File that manages the start order of child jobs
`.sysout` or `sysout.ini`	Spool job management file
`JOBLOG_`*number-giving-the-order-in-which-a-child-job-starts*	Job execution log for a child job for merging that is output when `MERGE` (merging the child job's spool job into
`SCRIPT_`*number-giving-the-order-in-which-a-child-job-starts*	

File name	Description
SCRIPT_*number-giving-the-order-in-which-a-child-job-starts*	the root job's spool job) is specified in the SPOOLJOB_CHILDJOB parameter in the environment file

If a job is terminated immediately by SIGKILL in UNIX or TerminateProcess in Windows, these files might remain in the spool job directory. When you delete spool directories, also delete these files.

#2

The contents of this file are also output to the job execution log. For details about what is output to the job execution log, see *3.4 Job execution log*.

#3

This is a program output data file that is allocated by the #-adsh_spoolfile command. For details about program output data files, see *5.9.3 Allocating program output data files and performing postprocessing*.

Important note

- In Windows, all output files except EVENTFILE are appended with the extension .sysout.
- Do not place under the spool directory any user-specific file that is not a file or directory created by JP1/Advanced Shell.

The following subsections explain the files and directories that are not temporary files.

(1) spool-root-directory

The directory name is specified in the SPOOL_DIR parameter in the environment file.

(2) lock-file

This file is used to lock each spool directory so that the same event file is not used by multiple commands at the same time. This file is created when the adshevtout and adshhk commands are executed.

The name of the lock file is as shown below. Do not delete a created lock file.

- In UNIX: .spool.lck
- In Windows: spool.lck

(3) spool-job-directory

This directory has the job sequence number as its name and is created for each job. When the job terminates, the directory is renamed to *job-ID*−*job-name*.

You can use the adshhk command to delete accumulated spool jobs. For details about the adshhk command, see *3.8 Deleting spool jobs*.

(4) EVENTFILE_ROOT_INF_000000_000000_000001

This is a root job search event file. It contains the information used to determine whether the condition specified in the adshevtout command (output job definition script operation information) is satisfied. This file is created for each root job. This file is not created in the following cases:

- JP1/Advanced Shell - Developer is used
- The job is executed in the debugger mode.
- The `adshexec` command terminated before the `KNAX0091-I` message was issued.

(5) EVENTFILE_execution-start-date-and-time_job-ID

This is an event file. While the event file is being created, `_making` is appended to the file name. This file is created for each root job and each child job.

Example of event file name:

`EVENTFILE_20120422_193502_123456`

This file is not created in the following cases:

- JP1/Advanced Shell - Developer is used
- The job is executed in the debugger mode.
- The `adshexec` command terminated before the `KNAX0091-I` message was issued.

execution-start-date-and-time

The root or child job's execution start date and time (UTC) are output in the following format:

```
YYYYMMDD_hhmmss_dddddd
```

YYYY: Four-digit decimal number indicating the calendar year (`1970` through `2038`)
MM: Two-digit decimal number indicating the month (`01` through `12`)
DD: Two-digit decimal number indicating the date (`01` through `31`)
hh: Two-digit decimal number indicating the hour (`00` through `23`)
mm: Two-digit decimal number indicating the minute (`00` through `59`)
ss: Two-digit decimal number indicating the second (`00` through `59`)
dddddd: Six-digit decimal number indicating the microsecond (`000000` through `999999`)

job-ID

A six-digit decimal number is assigned to each root job or child job.

(6) JOBLOG

This is for job execution messages. Messages indicating the job's execution status, including command execution results and file allocation results, are output to this directory.

(7) JOBLOG_job-ID_sequence-number

This is the job execution log for a child job.

This file is created when the child job starts, but only if one of the following methods was used to specify that the child job is to be run in the simple output mode or the minimum output mode:

- Specifying `SIMPLE` or `MINIMUM` in the `-m` option of the `adshexec` command
- Specifying `SIMPLE` or `MINIMUM` in the `-m` option of the `adshscripttool` command
- Specifying `SIMPLE` or `MINIMUM` in the `OUTPUT_MODE_CHILD` parameter

This file is not created when MERGE (merging the child job's spool job into the root job's spool job) is specified in the SPOOLJOB_CHILDJOB parameter.

(8) SCRIPT

This is for script image files. The contents of the first job definition script started and the contents of external job definition script files specified in the #-adsh_script command are output to this directory. External job definition script files specified using other methods, such as the . (dot) command, are not output to this directory. When you want to output the contents of job definition scripts as logs, you must use the #-adsh_script command.

If MERGE is specified in the SPOOLJOB_CHILDJOB parameter when the root job is run in the expansion output mode and the child job is run in the minimum output mode, the child job's SCRIPT is not merged into the root job's SCRIPT. For details, see *3.4.1(3)(c) When the output mode of the root job differs from that of the child job*.

(9) STDERR

This is the standard error output for the job. This file is not created when either of the following methods was used when the root job started to specify that the root job is to be run in the simple output mode or the minimum output mode:

- Specifying SIMPLE or MINIMUM in the -m option of the adshexec command
- Specifying SIMPLE or MINIMUM in the OUTPUT_MODE_ROOT parameter

The following header is output at the beginning of the file:

```
********    JOB SCOPE STDERR    ********
```

(10) STDOUT

This is the standard output for the job. It is created when the -s option is specified in the adshexec command or SPOOL is specified in the OUTPUT_STDOUT parameter in the environment file. This file is not created when either of the following methods was used when the root job started to specify that the root job is to be run in the simple output mode or the minimum output mode:

- Specifying SIMPLE or MINIMUM in the -m option of the adshexec command
- Specifying SIMPLE or MINIMUM in the OUTPUT_MODE_ROOT parameter

The following header is output at the beginning of the file:

```
********    JOB SCOPE STDOUT    ********
```

(11) step-number_step-name_STDOUT

If job steps are defined, this is the standard output within the corresponding job step. If the job step name consists of more than eight bytes, only the first eight bytes of the job step name are used for *step-name*.

This standard output is created when the -s option is specified in the adshexec command or SPOOL is specified in the OUTPUT_STDOUT parameter in the environment file. This file is not created when either of the following methods was used when the root job started to specify that the root job is to be run in the simple output mode or the minimum output mode:

- Specifying SIMPLE or MINIMUM in the -m option of the adshexec command

- Specifying `SIMPLE` or `MINIMUM` in the `OUTPUT_MODE_ROOT` parameter

(12) step-number_step-name_STDERR

If job steps are defined, this is the standard error output within the corresponding job step. If the job step name consists of more than eight bytes, only the first eight bytes of the job step name are used for *step-name*.

This file is not created when either of the following methods was used when the root job started to specify that the root job is to be run in the simple output mode or the minimum output mode:

- Specifying `SIMPLE` in the `-m` option of the `adshexec` command
- Specifying `SIMPLE` in the `OUTPUT_MODE_ROOT` parameter

(13) 0000_job-name_sequence-number-of-file-environment-variable-definition-name_file-environment-variable-definition-name

This is a program output data file allocated by the `#-adsh_spoolfile` command outside the job step.

(14) Cnumber-giving-the-order-in-which-a-child-job-starts_0000_job-name_sequence-number-of-file-environment-variable-definition-name_file-environment-variable-definition-name

This is a program output data file allocated by the `#-adsh_spoolfile` command outside the child job's job step.

This file is created only when `MERGE` (merging a child job's spool job into the root job's spool job) is specified in the `SPOOLJOB_CHILDJOB` parameter in the environment file.

(15) step-number_step-name_sequence-number-of-file-environment-variable-definition-name_file-environment-variable-definition-name

This is a program output data file allocated by the `#-adsh_spoolfile` command inside the job step.

(16) Cnumber-giving-the-order-in-which-a-child-job-starts_step-number_step-name_sequence-number-of-file-environment-variable-definition-name_file-environment-variable-definition-name

This is a program output data file allocated by the `#-adsh_spoolfile` command inside the child job's job step.

This file is created only when `MERGE` (merging a child job's spool job to the root job's spool job) is specified in the `SPOOLJOB_CHILDJOB` parameter in the environment file.

3.3.3 Suppressing output of specific information messages to job execution logs

You can limit the size of the job execution log file by suppressing output to the job execution log file of specific information messages. You specify the `JOBLOG_SUPPRESS_MSG` parameter in the environment file to use this function.

For the messages whose output can be suppressed and the details of the JOBLOG_SUPPRESS_MSG parameter, see *JOBLOG_SUPPRESS_MSG parameter (defines a message that is not to be output to job execution logs)* in *7. Parameters Specified in the Environment Files*.

3.3.4 Suppressing output of information and warning messages to job execution logs

If you intend to use job execution results with other programs, you might want to suppress the following types of output:

- Output of files under the spool job directory to the standard output and the standard error output
- Output of information and warning messages (excluding some exception messages) to the standard output and the standard error output
- Output to the standard error output of job execution logs at the time of job termination

To suppress these outputs, use one the following methods to specify the simple output mode or the minimum output mode:

- Specifying the OUTPUT_MODE_ROOT parameter (for a root job) or the OUTPUT_MODE_CHILD parameter (for a child job) when you specify environment settings

  For details about the OUTPUT_MODE_ROOT parameter, see *OUTPUT_MODE_ROOT parameter (specifies the method for outputting the execution results of a root job)*. For details about the OUTPUT_MODE_CHILD parameter, see *OUTPUT_MODE_CHILD parameter (defines the method for outputting the execution results of a child job)*.
- Specifying the -m option in the adshexec command during job execution

  For details about the adshexec command, see *adshexec command (executes a batch job)*.
- Specifying the -m option in the adshscripttool command

  For details about the adshscripttool command, see *adshscripttool command (supports creation of job definition scripts) (Windows only)*.

If both are specified, the adshexec command specification takes effect. If neither is specified, the expansion output mode is assumed.

(1) Differences in output contents among the expansion output mode, the simple output mode, and the minimum output mode

The following table describes the differences in the output contents among the expansion output mode, the simple output mode, and the minimum output mode.

Table 3–1: Differences in output contents among the expansion, the simple, and the minimum output modes

Output timing	Expansion output mode	Simple output mode	Minimum output mode
Job execution	The contents of the standard output and the standard error output depend on the type of job: • Root jobs The contents of the standard output and the	The contents of the standard output and the standard error output are output to their output destinations at the time the process started. Of the JP1/Advanced Shell messages that are to be output to the standard output and the	Same as at the left

Output timing	Expansion output mode	Simple output mode	Minimum output mode
Job execution	standard error output are output to the spool job directory. - Child jobs Output to their output destinations at the time the process started.	standard error output, only error messages are output.[#1]	Same as at the left
Job termination	Job execution logs are output to the standard error output (for child jobs, the output is to the root-job's standard error output[#2]).	Job execution logs are not output to the standard error output. However, error messages to be output only to `JOBLOG` are also output to the standard error output during job execution in order to report errors. A child job's `JOBLOG` is created under the root job's spool job directory and is retained even after the job has terminated.[#2] This handling is in effect regardless of the specification of the `JOBEXECLOG_PRINT` parameter.	Same as at the left
Debug execution	`JOBLOG` is output to the standard error output at suitable times. JP1/Advanced Shell messages that are to be output to the standard output and the standard error output are output to the standard output and the standard error output in effect at the time debug execution started.	`JOBLOG` is not output to the standard error output. JP1/Advanced Shell messages that are to be output to the standard output and the standard error output are output to the standard output and the standard error output in effect at the time debug execution started. When debugging is terminated, only error messages are output. However, a child job that is not subject to debugging is run in the same manner as in normal execution.	`JOBLOG` is not output to the standard error output. JP1/Advanced Shell messages that are to be output to the standard output and the standard error output are output to the standard output and the standard error output in effect at the time debug execution started. When debugging terminates, only messages that are not subject to output suppression are output. However, a child job that is not subject to debugging is run in the same manner as in normal execution.

#1

In addition to error messages, some messages are output as exceptions. For details about such exception messages and the destinations by message type, see *11.2 Message output destinations*.

#2

Not output when a child job's job execution log is merged into the root job's job execution log (`MERGE` is specified in the `SPOOLJOB_CHILDJOB` parameter).

If you start a job definition script from another job definition script in the simple output mode or the minimum output mode, use child jobs. If a root job is started in the simple output mode or the minimum output mode, error messages are displayed in the standard error output.

(2) How to locate the spool job directory of a job that was executed in the simple output mode or the minimum output mode

When the simple output mode or the minimum output mode is selected, messages that display job IDs and spool job directory names are no longer output. The following describes how to locate the spool job directory of a job that was executed in the simple output mode or the minimum output mode:

- Specify unique job names with the `#-adsh_job` command (declares a name for a job).
- When the job starts, output the values of the following environment variables to the standard error output or to a specific file so that they can be referenced later:
  - `ADSH_JOBID` environment variable (stores a job ID)
  - `ADSH_JOB_NAME` (stores a job name)
- Locate the spool job directory based on the job execution date and time.

3.4 Job execution log

The purpose of a job execution log is to notify users of the results of executing batch jobs. This log information, excluding the contents for the standard output for user programs, is output to the files under the spool job directory and to the standard error output when a job terminates. You can use JP1/AJS - View, among other methods, to view job execution logs.

The following information is output to the job execution log:

- Start and end messages for batch jobs
- Start and end messages for job steps
- Contents of job definition scripts
- Results of executing commands
- Status and postprocessing results of files that have been prepared
- Standard output from user programs (`stdout`)#1
- Standard error output from user programs (`stderr`)#2
- Messages related to acquiring coverage information

#1

Output while the job is running to the standard output in effect at the time the job started if either of the conditions listed below is satisfied. This means that after the job has terminated, the execution results are lost and cannot be verified with JP1/AJS - View.

- The `-s` option is specified in the `adshexec` command or `PARENT` is specified in the `OUTPUT_STDOUT` parameter in the environment file.
- The root job is in the simple output mode or the minimum output mode.

#2

Not output to the files under the spool job directory if the root job is in the simple output mode or the minimum output mode. However, while the job is running, the user program's standard error output is output to the standard error output in effect at the time the job started.

If you are not using JP1/AJS, refer to the `JOBLOG` file in the batch job's directory under the spool root directory specified in the `SPOOL_DIR` parameter in the environment file.

The `JOBLOG_SUPPRESS_MSG` parameter can be set to suppress output to the `JOBLOG` file of some information messages. For details, such as the messages whose output can be suppressed, see *JOBLOG_SUPPRESS_MSG parameter (defines a message that is not to be output to job execution logs)* in *7. Parameters Specified in the Environment Files*.

3.4.1 Outputting the contents of the job execution log by job type

What is output to the job execution log depends on the type of job that is executed, as described in the following subsections.

(1) Destination and output contents of the job execution log when root jobs are executed

This subsection explains the destination and output contents of the job execution log when root jobs are executed in the expansion output mode, the simple output mode, or the minimum output mode (as specified in the `OUTPUT_MODE_ROOT` parameter).

(a) When the expansion output mode is selected

The following table describes the output contents of the job execution log when the expansion output mode is selected:

Message output destination	Description
`JOBLOG`	Output to spool files. During debug execution, `JOBLOG` is also output to the standard error output during job execution.
Script image	Output to spool files.
Destination of the standard output	Output to the destination specified by either of the following methods: • `-s` option in the `adshexec` command • `OUTPUT_STDOUT` parameter in the environment file During debug execution, this information is also output to the standard error output in effect at the time debug execution started.
Destination of the standard error output	Output to spool files. During debug execution, this information is output to the standard error output in effect at the time debug execution started.

A spool job directory is created for each job.

After job execution, the contents of the job execution log, excluding the contents for the standard output, are output to the standard error output.

During debug execution, the contents of the job execution log after job execution are not output to the standard error output.

(b) When the simple output mode is selected

The following table describes the output contents of the job execution log when the simple output mode is selected:

Message output destination	Description
`JOBLOG`	Output to spool files.
Script image	
Destination of the standard output	Not output to spool files. This information is output to the destination in effect when the process started. Error messages for the standard error output and the standard output are output (during debug execution, messages other than error messages are also output). Also, error messages for `JOBLOG` are output to the standard error output. During normal execution, output of any message whose type is `W` or `I` (excluding signal reception and event reception messages) is suppressed.
Destination of the standard error output	

Note that the job execution log is not output to the standard error output when the job terminates.

(c) When the minimum output mode is selected

The following table describes the output contents of the job execution log when the minimum output mode is selected:

Message output destination	Description
JOBLOG	Messages that are not subject to output suppression are output to spool files.
Script image	
Destination of the standard output	Not output to spool files. This information is output to the destination in effect when the process started. Messages for the standard error output and the standard output that are not subject to output suppression are output (during debug execution, messages that are not subject to output suppression are also output). Also, messages for JOBLOG that are not subject to output suppression are output to the standard error output.
Destination of the standard error output	

Note that the job execution log is not output to the standard error output when the job terminates.

Output of the following messages is suppressed:

Timing	Message whose output is suppressed
During normal execution	• Messages whose type is W or I (excluding signal reception and event reception messages) • The following messages whose type is E: KNAX0101-E, KNAX2201-E, KNAX6521-E, KNAX6522-E, KNAX6541-E, KNAX6542-E, KNAX6551-E, KNAX6552-E, KNAX6561-E, KNAX6562-E, KNAX6586-E, KNAX6591-E, KNAX6592-E, KNAX6593-E, KNAX6594-E, KNAX6596-E • The following messages whose type is I: KNAX7893-I, KNAX7896-I
During debug execution	• Messages whose type is W or I (excluding signal reception and event reception messages) • The following messages whose type is E: KNAX0101-E, KNAX2201-E, KNAX6521-E, KNAX6522-E, KNAX6541-E, KNAX6542-E, KNAX6551-E, KNAX6552-E, KNAX6561-E, KNAX6562-E, KNAX6586-E, KNAX6591-E, KNAX6592-E, KNAX6593-E, KNAX6594-E, KNAX6596-E

(2) Destination and output contents of the job execution log when child jobs are executed

This subsection explains the destination and output contents of the job execution log when child jobs are executed in the expansion output mode, the simple output mode, or the minimum output mode (as specified in the OUTPUT_MODE_CHILD parameter). For the output contents when a child job's spool job is to be merged into the root job's spool job, see *(3) Merging a child job's spool job into the root job's spool job*.

(a) When the expansion output mode is selected

The following table describes the output contents of the job execution log when the expansion output mode is selected:

Message output destination	Description
JOBLOG	Temporarily output to spool files. When the child job terminates, this file is deleted from the spool after JOBLOG has been output to the standard error output in effect when the process started.
Script image	Temporarily output to spool files, but is deleted when the child job terminates.
Destination of the standard output	Output to the destination in effect when the process started.
Destination of the standard error output	

A spool job directory is created during job execution, but this directory is deleted after the job has executed. Because the program output data file allocated in the spool job directory is also deleted, the KNAX6380-I message indicating successful renaming of the spool job directory is not output.

After job execution, the contents of JOBLOG are output to the standard error output. However, the following header lines are not output:

```
---------------------------------------------------------------
 Advanced Shell version-number

 [Information]
   Job ID          : job-ID
   Spool directory : spool-job-directory-path
   Date            : execution-date
   EnvFile(system) : environment-file-path (System environment file)
   EnvFile(job)    : environment-file-path (Job environment file)
   Host name       : host-name
 [Environment variable from Automatic Job Management System]
   environment-variables-passed-from-JP1/AJS
---------------------------------------------------------------
********  JOB CONTROLLER MESSAGE  ********
```

Because no job execution log other than JOBLOG is output, the job execution log cannot be referenced from other programs, such as JP1/AJS or the login shell.

Because the job return code is output to JOBLOG for the parent process's job, the KNAX7999-I message indicating termination of adshexec command execution is not output to the standard error output.

(b) When the simple output mode is selected

The following table describes the output contents of the job execution log when the simple output mode is selected:

Message output destination	Description
JOBLOG	Output to spool files for each child job in the root job's spool.
Script image	Temporarily output to spool files, but is deleted when the child job terminates.
Destination of the standard output	Output to the destination in effect when the process started. Error messages for the standard error output and the standard output are output. Error messages for JOBLOG are also output to the standard error output. Only error messages are output. Output of messages whose type is W or I (excluding signal reception and event reception messages) is suppressed.
Destination of the standard error output	

Messages are output to JOBLOG during job execution. Error messages among these messages are also output to the standard error output.

The job execution log is output to spool files for each child job under the root job's spool job directory. The job execution log is not output to the standard error output even after the job has executed.

(c) When the minimum output mode is selected

The following table describes the output destination of the job execution log when the minimum output mode is selected:

Message output destination	Description
JOBLOG	Messages that are not subject to output suppression are output to files for each child job in the root job's spool.
Script image	Temporarily output to spool files, but is deleted when the child job terminates.
Destination of the standard output	Output to the destination in effect when the process started. Messages for the standard error output and the standard output that are not subject to output suppression are output. Also, messages for JOBLOG that are not subject to output suppression are output to the standard error output. Messages whose output is suppressed are not output.
Destination of the standard error output	

Output of the following messages is suppressed:

- Messages whose type is W or I (excluding signal reception and event reception messages)
- The following messages whose type is E:

  KNAX0101-E, KNAX2201-E, KNAX6521-E, KNAX6522-E, KNAX6541-E, KNAX6542-E, KNAX6551-E, KNAX6552-E, KNAX6561-E, KNAX6562-E, KNAX6586-E, KNAX6591-E, KNAX6592-E, KNAX6593-E, KNAX6594-E, KNAX6596-E
- The following messages whose type is I:

  KNAX7893-I, KNAX7896-I

(3) Merging a child job's spool job into the root job's spool job

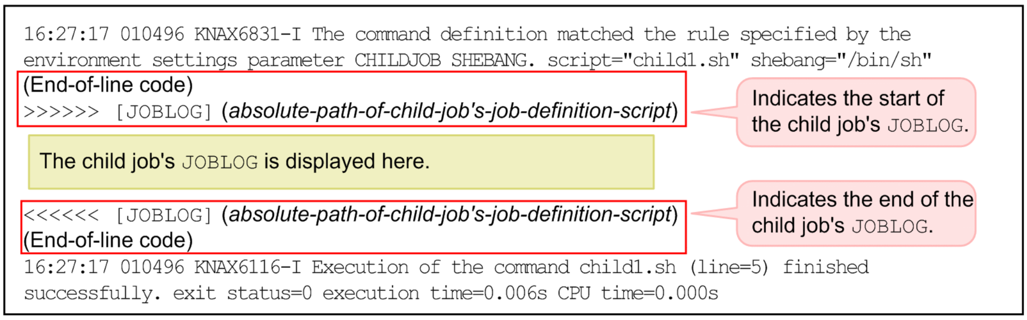
If MERGE is specified in the SPOOLJOB_CHILDJOB parameter, the child job's spool job is merged into the root job's spool job. An overview of the job execution log contents is provided below. For an example of job execution log output, see *3.4.3 Example of job execution log output (when a child job's spool job is merged into the root job's spool job)*.

(a) During normal operation

- JOBLOG

  JOBLOG for a child job is output between a message indicating that the rule for replacing the child job's execution was satisfied and a message indicating that the command used to execute the child job has terminated.

  The symbols indicating the start and end of the child job's JOBLOG are displayed before and after JOBLOG, as shown in the following figure:



```
16:27:17 010496 KNAX6831-I The command definition matched the rule specified by the
environment settings parameter CHILDJOB SHEBANG. script="child1.sh" shebang="/bin/sh"
(End-of-line code)
>>>>>> [JOBLOG] (absolute-path-of-child-job's-job-definition-script)

The child job's JOBLOG is displayed here.

<<<<<< [JOBLOG] (absolute-path-of-child-job's-job-definition-script)
(End-of-line code)
16:27:17 010496 KNAX6116-I Execution of the command child1.sh (line=5) finished
successfully. exit status=0 execution time=0.006s CPU time=0.000s
```

Indicates the start of the child job's JOBLOG.

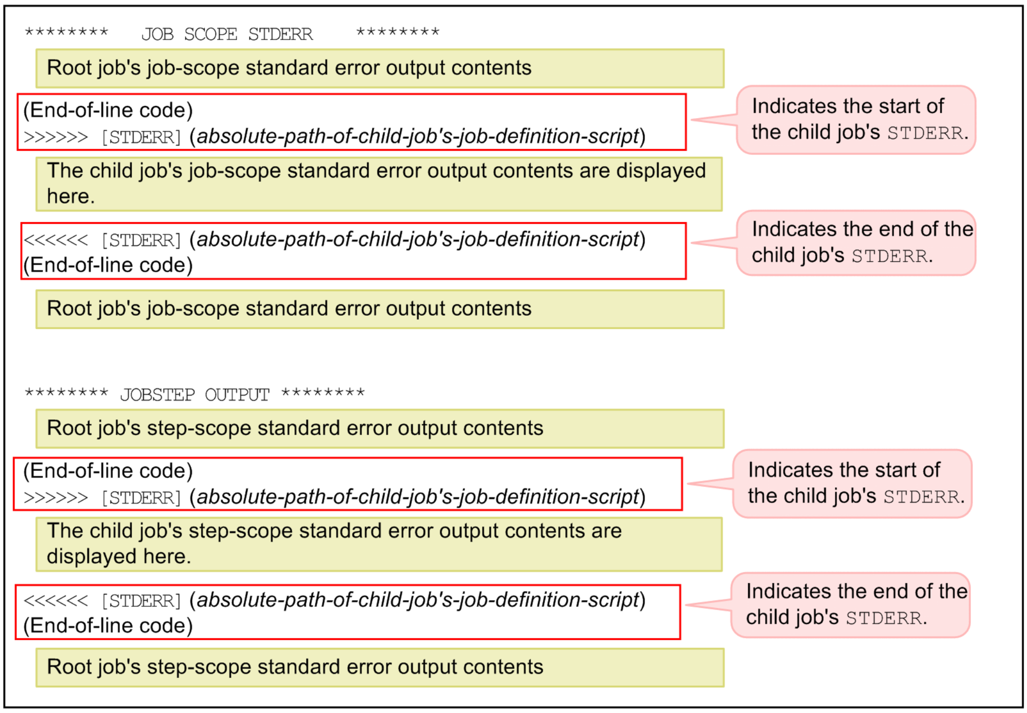
Indicates the end of the child job's JOBLOG.

- SCRIPT

  A child job's SCRIPT is displayed before its root job's SCRIPT. The header (******** Script IMAGE ********) is not displayed in the child job output section.
- STDERR

  The symbols indicating the start and end of the child job's STDERR are displayed before and after the child job's standard error output as shown below.

These symbols are not displayed when the child job is in the simple output mode or the minimum output mode.



```
********   JOB SCOPE STDERR    ********
Root job's job-scope standard error output contents
(End-of-line code)
>>>>>> [STDERR] (absolute-path-of-child-job's-job-definition-script)
The child job's job-scope standard error output contents are displayed here.
<<<<<< [STDERR] (absolute-path-of-child-job's-job-definition-script)
(End-of-line code)
Root job's job-scope standard error output contents

******** JOBSTEP OUTPUT ********
Root job's step-scope standard error output contents
(End-of-line code)
>>>>>> [STDERR] (absolute-path-of-child-job's-job-definition-script)
The child job's step-scope standard error output contents are displayed here.
<<<<<< [STDERR] (absolute-path-of-child-job's-job-definition-script)
(End-of-line code)
Root job's step-scope standard error output contents
```

Indicates the start of the child job's STDERR.
Indicates the end of the child job's STDERR.
Indicates the start of the child job's STDERR.
Indicates the end of the child job's STDERR.

- STDOUT

  STDOUT is not merged.

(b) During debug execution

- JOBLOG

  JOBLOG is merged in the same format as during normal execution. For the standard error output, the same contents are also displayed immediately after termination of the child job.

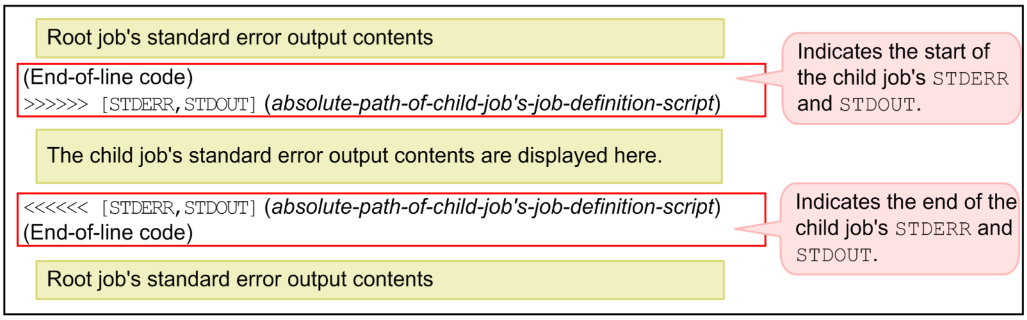
- SCRIPT

  SCRIPT is merged in the same format as during normal execution. The contents of SCRIPT are not output to the standard error output.

- STDERR, STDOUT

  The symbols indicating the start and end of a set of a child job's STDERR and STDOUT are displayed before and after the child job's standard error output, as shown below. The text [STDERR,STDOUT] is displayed in the standard error output even when the child job's standard error output and standard output are redirected.

  These symbols are not displayed when the child job is in the simple output mode or the minimum output mode.



```
Root job's standard error output contents
(End-of-line code)
>>>>>> [STDERR,STDOUT] (absolute-path-of-child-job's-job-definition-script)
The child job's standard error output contents are displayed here.
<<<<<< [STDERR,STDOUT] (absolute-path-of-child-job's-job-definition-script)
(End-of-line code)
Root job's standard error output contents
```

Indicates the start of the child job's STDERR and STDOUT.
Indicates the end of the child job's STDERR and STDOUT.

(c) When the output mode of the root job differs from that of the child job

When MERGE is specified in the SPOOLJOB_CHILDJOB environment setting parameter and the root job is run in the expansion output mode and the child job is run in the minimum output mode, the results of merging JOBLOG and SCRIPT are as follows:

- JOBLOG
    - If no message has been output to JOBLOG of the child job running in the minimum output mode

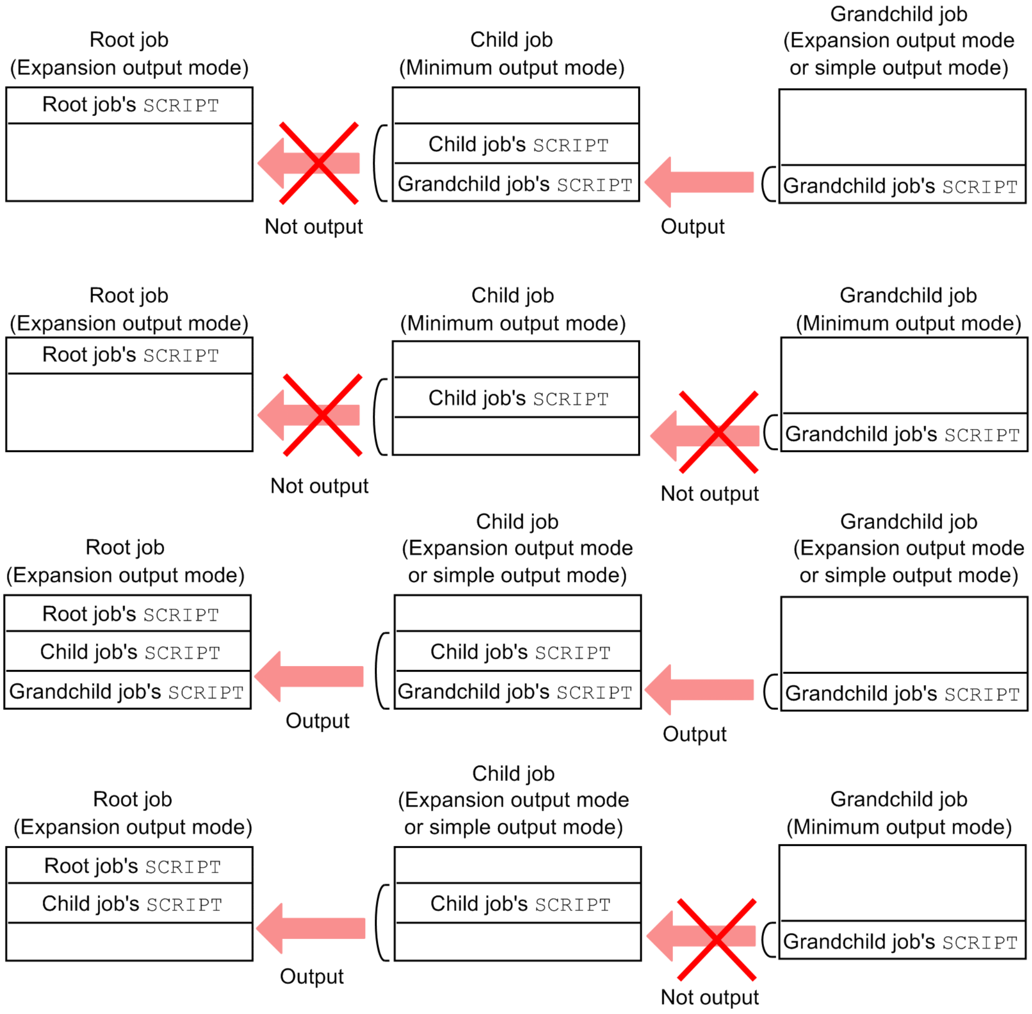
      This child job's JOBLOG is not merged into the root job's JOBLOG. Therefore, the strings indicating the beginning and end of output of the child job's JOBLOG are also not output.
    - If messages have been output to JOBLOG of the child job running in the minimum output mode

      This child job's JOBLOG is merged into the root job's JOBLOG. The strings indicating the beginning and end of output of the child job's JOBLOG are also output.

  During both normal execution and debug execution, the strings that indicate the beginning and end of output of the child job's JOBLOG are >>>>>> [JOBLOG] *path-name* and <<<<<< [JOBLOG] *path-name*, respectively.
- SCRIPT

  The child job's SCRIPT is not merged into the root job's SCRIPT.

  When the child job is run in the minimum output mode and its grandchild job is run in the expansion output mode or the simple output mode, the child job's SCRIPT is not merged either. Therefore, the grandchild job's SCRIPT merged into the child job's SCRIPT is not output.



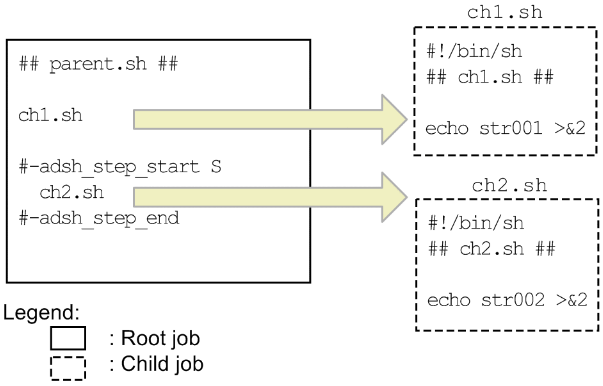
The same applies when the root job is run in the simple output mode or the minimum output mode. However, in this case, SCRIPT is not output to the standard error output.

3.4.2 Examples of job execution log output

The following examples illustrate the job execution log information that is output when a root job and child jobs are executed.

(1) Example 1 (defining ch1.sh and ch2.sh)

As shown below, this example defines child job ch1.sh that is started from the root job and child job ch2.sh that is started from within the root job's job step.



```
## parent.sh ##

ch1.sh

#-adsh_step_start S
  ch2.sh
#-adsh_step_end
```

ch1.sh

```
#!/bin/sh
## ch1.sh ##

echo str001 >&2
```

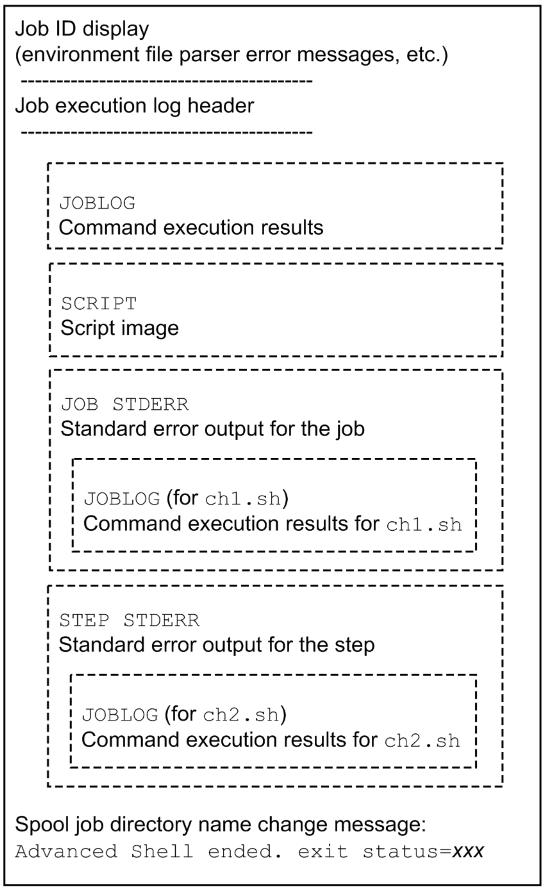
ch2.sh

```
#!/bin/sh
## ch2.sh ##

echo str002 >&2
```

The following subsections explain the configuration and output examples of the root job's job execution log that is output to the standard error output.

(a) Configuration of job execution log

The figure below shows the configuration of the job execution log and the locations where the execution results of child jobs ch1.sh and ch2.sh are output. The execution results of ch1.sh are output within the job scope, and the execution results of ch2.sh are output within the step scope.



Job ID display
(environment file parser error messages, etc.)

Job execution log header

JOBLOG
Command execution results

SCRIPT
Script image

JOB STDERR
Standard error output for the job

JOBLOG (for ch1.sh)
Command execution results for ch1.sh

STEP STDERR
Standard error output for the step

JOBLOG (for ch2.sh)
Command execution results for ch2.sh

Spool job directory name change message:
Advanced Shell ended. exit status=*xxx*

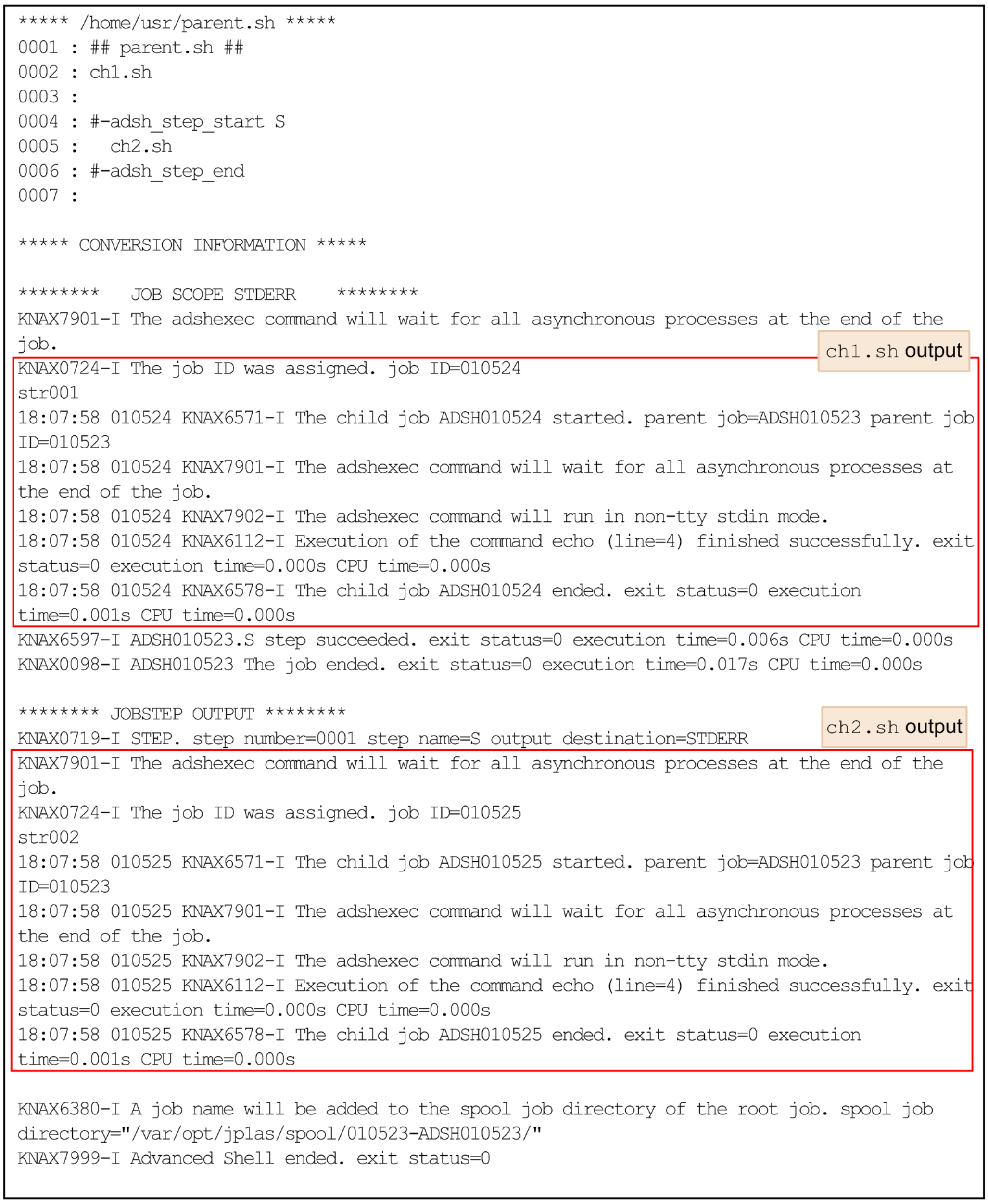
(b) Example of job execution log output

The following shows an example of job execution log output:

```
KNAX7901-I The adshexec command will wait for all asynchronous processes at the end of the
job.
KNAX0724-I The job ID was assigned. job ID=010523
---------------------------------------------------------------
 Advanced Shell 10-10

 [Information]
   Job ID          : 010523
   Spool directory : /var/opt/jp1as/spool/010523/
   Date            : 2013/12/06
   EnvFile(system) :
   EnvFile(job)    : /home/usr/adsh_job.ase
   Host name       : HOST01
 [Environment variable from Automatic Job Management System]
   JP1JobName      : adshexec
   JP1JobID        : 1043
   JP1_USERNAME    : usr
   JP1UNCName      : HOST01
   JP1NBQSQueueName: \\HOST01\@SYSTEM
   JP1Priority     : 1
   AJSEXECID       : @A335
---------------------------------------------------------------
********  JOB CONTROLLER MESSAGE  ********
18:07:58 010523 KNAX0091-I ADSH010523 The job started.
18:07:58 010523 KNAX7901-I The adshexec command will wait for all asynchronous processes at
the end of the job.
18:07:58 010523 KNAX7902-I The adshexec command will run in non-tty stdin mode.
18:07:58 010523 KNAX6831-I The command definition matched the rule specified by the
environment settings parameter CHILDJOB_SHEBANG. script="ch1.sh" shebang="/bin/sh"
18:07:58 010523 KNAX6116-I Execution of the command ch1.sh (line=2) finished successfully.
exit status=0 execution time=0.007s CPU time=0.000s
18:07:58 010523 KNAX0092-I ADSH010523.S step started.
18:07:58 010523 KNAX6831-I The command definition matched the rule specified by the
environment settings parameter CHILDJOB_SHEBANG. script="ch2.sh" shebang="/bin/sh"
18:07:58 010523 KNAX6116-I Execution of the command ch2.sh (line=5) finished successfully.
exit status=0 execution time=0.006s CPU time=0.000s
18:07:58 010523 KNAX6597-I ADSH010523.S step succeeded. exit status=0 execution time=0.006s
CPU time=0.000s
18:07:58 010523 KNAX0098-I ADSH010523 The job ended. exit status=0 execution time=0.017s CPU
time=0.000s

********   Script IMAGE    ********
```

(Continued)

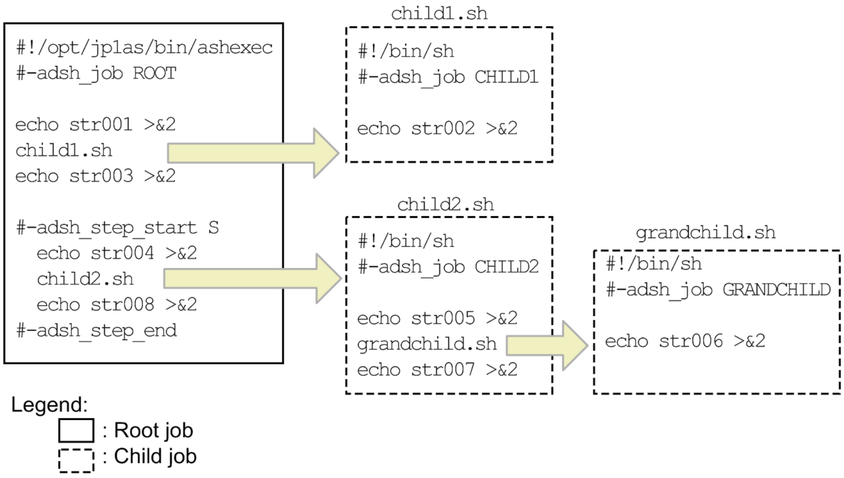
(Continued)



```
***** /home/usr/parent.sh *****
0001 : ## parent.sh ##
0002 : ch1.sh
0003 :
0004 : #-adsh_step_start S
0005 :   ch2.sh
0006 : #-adsh_step_end
0007 :

***** CONVERSION INFORMATION *****

********   JOB SCOPE STDERR    ********
KNAX7901-I The adshexec command will wait for all asynchronous processes at the end of the
job.
KNAX0724-I The job ID was assigned. job ID=010524
str001
18:07:58 010524 KNAX6571-I The child job ADSH010524 started. parent job=ADSH010523 parent job
ID=010523
18:07:58 010524 KNAX7901-I The adshexec command will wait for all asynchronous processes at
the end of the job.
18:07:58 010524 KNAX7902-I The adshexec command will run in non-tty stdin mode.
18:07:58 010524 KNAX6112-I Execution of the command echo (line=4) finished successfully. exit
status=0 execution time=0.000s CPU time=0.000s
18:07:58 010524 KNAX6578-I The child job ADSH010524 ended. exit status=0 execution
time=0.001s CPU time=0.000s
KNAX6597-I ADSH010523.S step succeeded. exit status=0 execution time=0.006s CPU time=0.000s
KNAX0098-I ADSH010523 The job ended. exit status=0 execution time=0.017s CPU time=0.000s

******** JOBSTEP OUTPUT ********
KNAX0719-I STEP. step number=0001 step name=S output destination=STDERR
KNAX7901-I The adshexec command will wait for all asynchronous processes at the end of the
job.
KNAX0724-I The job ID was assigned. job ID=010525
str002
18:07:58 010525 KNAX6571-I The child job ADSH010525 started. parent job=ADSH010523 parent job
ID=010523
18:07:58 010525 KNAX7901-I The adshexec command will wait for all asynchronous processes at
the end of the job.
18:07:58 010525 KNAX7902-I The adshexec command will run in non-tty stdin mode.
18:07:58 010525 KNAX6112-I Execution of the command echo (line=4) finished successfully. exit
status=0 execution time=0.000s CPU time=0.000s
18:07:58 010525 KNAX6578-I The child job ADSH010525 ended. exit status=0 execution
time=0.001s CPU time=0.000s

KNAX6380-I A job name will be added to the spool job directory of the root job. spool job
directory="/var/opt/jp1as/spool/010523-ADSH010523/"
KNAX7999-I Advanced Shell ended. exit status=0
```

(2) Example 2 (defining child1.sh, child2.sh, and grandchild.sh)

As shown below, this example defines child job `child1.sh`, child job `child2.sh`, and `grandchild.sh` that is started from child job `child2.sh`.



```
#!/opt/jp1as/bin/ashexec
#-adsh_job ROOT

echo str001 >&2
child1.sh
echo str003 >&2

#-adsh_step_start S
  echo str004 >&2
  child2.sh
  echo str008 >&2
#-adsh_step_end
```

child1.sh

```
#!/bin/sh
#-adsh_job CHILD1

echo str002 >&2
```

child2.sh

```
#!/bin/sh
#-adsh_job CHILD2

echo str005 >&2
grandchild.sh
echo str007 >&2
```

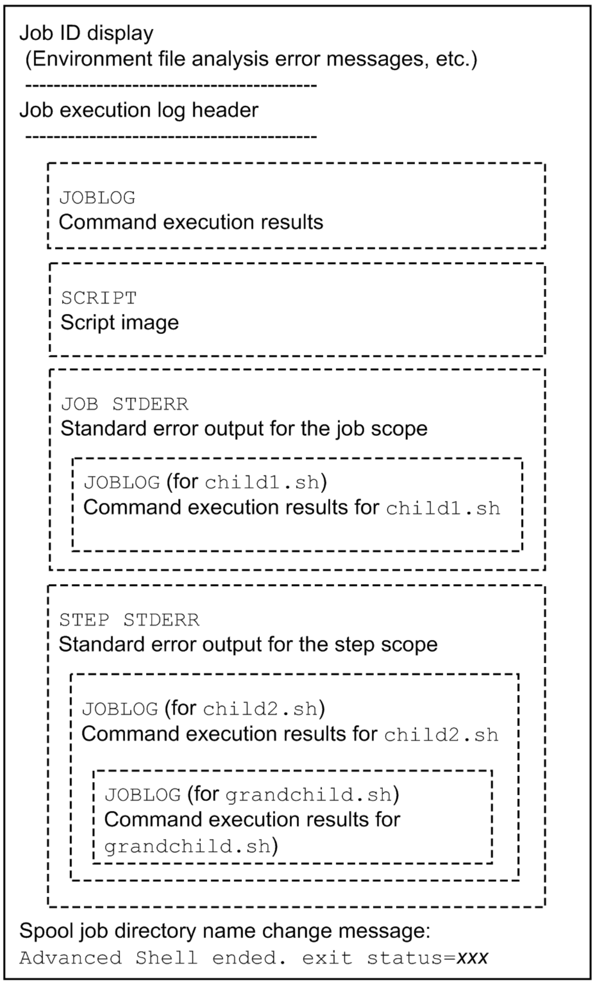
grandchild.sh

```
#!/bin/sh
#-adsh_job GRANDCHILD

echo str006 >&2
```

The following subsections explain the configuration and output examples of the root job's job execution log that is output to the standard error output.

(a) Configuration of job execution log

The figure below shows the configuration of the job execution log and the locations where the execution results of child job `child1.sh`, child job `child2.sh`, and `grandchild.sh` are output. The execution results of `grandchild.sh` are output within the execution results of child job `child2.sh`.



Job ID display
(Environment file analysis error messages, etc.)

Job execution log header

`JOBLOG`
Command execution results

`SCRIPT`
Script image

`JOB STDERR`
Standard error output for the job scope

`JOBLOG` (for `child1.sh`)
Command execution results for `child1.sh`

`STEP STDERR`
Standard error output for the step scope

`JOBLOG` (for `child2.sh`)
Command execution results for `child2.sh`

`JOBLOG` (for `grandchild.sh`)
Command execution results for `grandchild.sh`)

Spool job directory name change message:
`Advanced Shell ended. exit status=`*`xxx`*

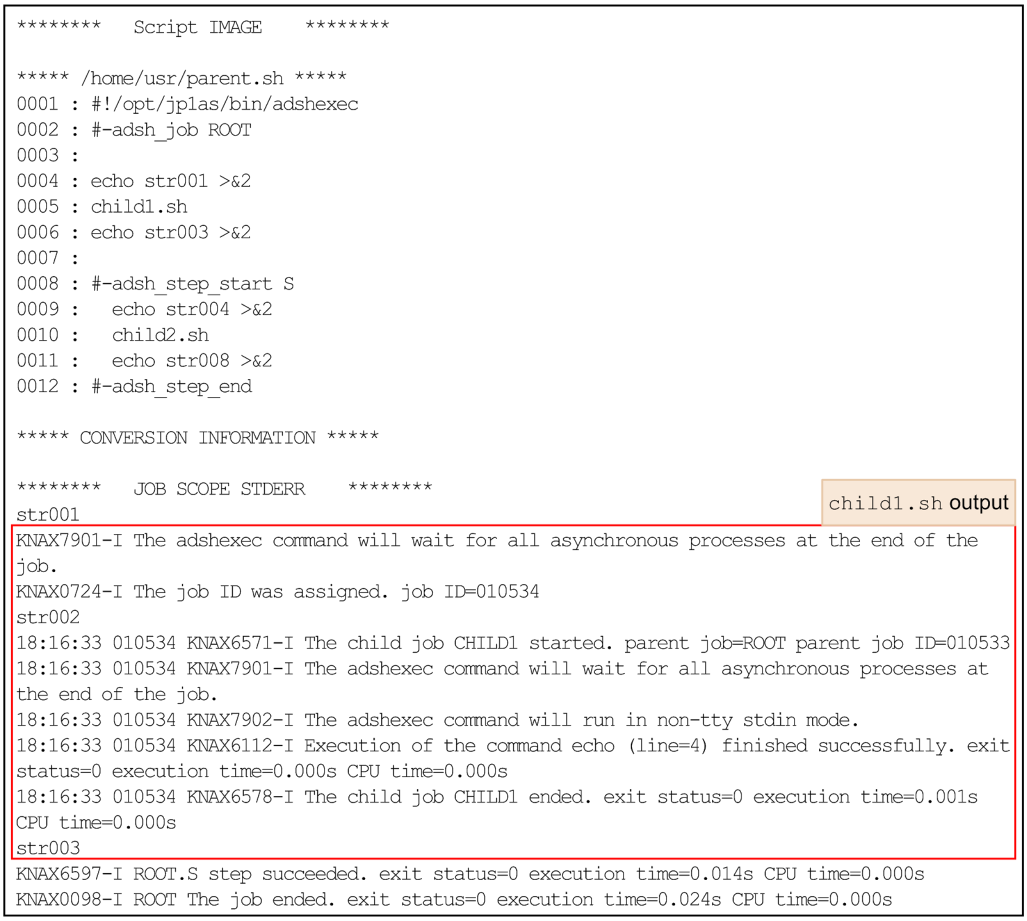
(b) Example of job execution log output

The following shows an example of job execution log output:

```
KNAX7901-I The adshexec command will wait for all asynchronous processes at the end of the
job.
KNAX0724-I The job ID was assigned. job ID=010533
---------------------------------------------------------------
 Advanced Shell 10-10

 [Information]
   Job ID          : 010533
   Spool directory : /var/opt/jp1as/spool/010533/
   Date            : 2013/12/06
   EnvFile(system) :
   EnvFile(job)    : /home/usr/adsh_job.ase
   Host name       : HOST01
 [Environment variable from Automatic Job Management System]
   JP1JobName      : adshexec
   JP1JobID        : 1047
   JP1_USERNAME    : usr
   JP1UNCName      : HOST01
   JP1NBQSQueueName: \\HOST01\@SYSTEM
   JP1Priority     : 1
   AJSEXECID       : @A339
---------------------------------------------------------------
********  JOB CONTROLLER MESSAGE  ********
18:16:33 010533 KNAX0091-I ROOT The job started.
18:16:33 010533 KNAX7901-I The adshexec command will wait for all asynchronous processes at
the end of the job.
18:16:33 010533 KNAX7902-I The adshexec command will run in non-tty stdin mode.
18:16:33 010533 KNAX6112-I Execution of the command echo (line=4) finished successfully. exit
status=0 execution time=0.000s CPU time=0.000s
18:16:33 010533 KNAX6831-I The command definition matched the rule specified by the
environment settings parameter CHILDJOB_SHEBANG. script="child1.sh" shebang="/bin/sh"
18:16:33 010533 KNAX6116-I Execution of the command child1.sh (line=5) finished successfully.
exit status=0 execution time=0.006s CPU time=0.000s
18:16:33 010533 KNAX6112-I Execution of the command echo (line=6) finished successfully. exit
status=0 execution time=0.000s CPU time=0.000s
18:16:33 010533 KNAX0092-I ROOT.S step started.
18:16:33 010533 KNAX6112-I Execution of the command echo (line=9) finished successfully. exit
status=0 execution time=0.000s CPU time=0.000s
18:16:33 010533 KNAX6831-I The command definition matched the rule specified by the
environment settings parameter CHILDJOB_SHEBANG. script="child2.sh" shebang="/bin/sh"
18:16:33 010533 KNAX6116-I Execution of the command child2.sh (line=10) finished
successfully. exit status=0 execution time=0.013s CPU time=0.000s
18:16:33 010533 KNAX6112-I Execution of the command echo (line=11) finished successfully.
exit status=0 execution time=0.000s CPU time=0.000s
18:16:33 010533 KNAX6597-I ROOT.S step succeeded. exit status=0 execution time=0.014s CPU
time=0.000s
18:16:33 010533 KNAX0098-I ROOT The job ended. exit status=0 execution time=0.024s CPU
time=0.000s
```

(Continued)

(Continued)



```
********   Script IMAGE    ********

***** /home/usr/parent.sh *****
0001 : #!/opt/jp1as/bin/adshexec
0002 : #-adsh_job ROOT
0003 :
0004 : echo str001 >&2
0005 : child1.sh
0006 : echo str003 >&2
0007 :
0008 : #-adsh_step_start S
0009 :   echo str004 >&2
0010 :   child2.sh
0011 :   echo str008 >&2
0012 : #-adsh_step_end

***** CONVERSION INFORMATION *****

********   JOB SCOPE STDERR    ********
str001
KNAX7901-I The adshexec command will wait for all asynchronous processes at the end of the
job.
KNAX0724-I The job ID was assigned. job ID=010534
str002
18:16:33 010534 KNAX6571-I The child job CHILD1 started. parent job=ROOT parent job ID=010533
18:16:33 010534 KNAX7901-I The adshexec command will wait for all asynchronous processes at
the end of the job.
18:16:33 010534 KNAX7902-I The adshexec command will run in non-tty stdin mode.
18:16:33 010534 KNAX6112-I Execution of the command echo (line=4) finished successfully. exit
status=0 execution time=0.000s CPU time=0.000s
18:16:33 010534 KNAX6578-I The child job CHILD1 ended. exit status=0 execution time=0.001s
CPU time=0.000s
str003
KNAX6597-I ROOT.S step succeeded. exit status=0 execution time=0.014s CPU time=0.000s
KNAX0098-I ROOT The job ended. exit status=0 execution time=0.024s CPU time=0.000s
```

(Continued)

(Continued)

child2.sh output

grandchild.sh output

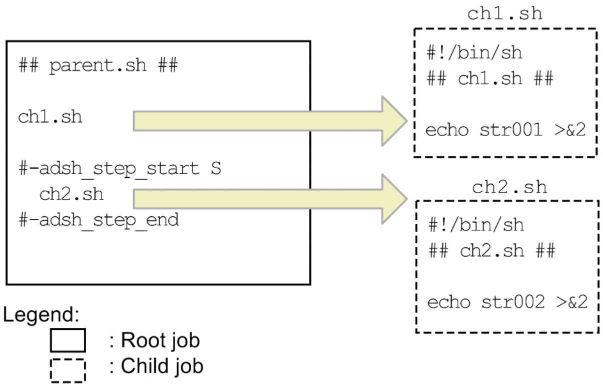
```
******** JOBSTEP OUTPUT ********
KNAX0719-I STEP. step number=0001 step name=S output destination=STDERR
str004
KNAX7901-I The adshexec command will wait for all asynchronous processes at the end of the
job.
KNAX0724-I The job ID was assigned. job ID=010535
str005
KNAX7901-I The adshexec command will wait for all asynchronous processes at the end of the
job.
KNAX0724-I The job ID was assigned. job ID=010536
str006
18:16:33 010536 KNAX6571-I The child job GRANDCHILD started. parent job=CHILD2 parent job
ID=010535
18:16:33 010536 KNAX7901-I The adshexec command will wait for all asynchronous processes at
the end of the job.
18:16:33 010536 KNAX7902-I The adshexec command will run in non-tty stdin mode.
18:16:33 010536 KNAX6112-I Execution of the command echo (line=4) finished successfully. exit
status=0 execution time=0.000s CPU time=0.000s
18:16:33 010536 KNAX6578-I The child job GRANDCHILD ended. exit status=0 execution
time=0.001s CPU time=0.000s
str007
18:16:33 010535 KNAX6571-I The child job CHILD2 started. parent job=ROOT parent job ID=010533
18:16:33 010535 KNAX7901-I The adshexec command will wait for all asynchronous processes at
the end of the job.
18:16:33 010535 KNAX7902-I The adshexec command will run in non-tty stdin mode.
18:16:33 010535 KNAX6112-I Execution of the command echo (line=4) finished successfully. exit
status=0 execution time=0.000s CPU time=0.000s
18:16:33 010535 KNAX6831-I The command definition matched the rule specified by the
environment settings parameter CHILDJOB_SHEBANG. script="grandchild.sh" shebang="/bin/sh"
18:16:33 010535 KNAX6116-I Execution of the command grandchild.sh (line=5) finished
successfully. exit status=0 execution time=0.005s CPU time=0.000s
18:16:33 010535 KNAX6112-I Execution of the command echo (line=6) finished successfully. exit
status=0 execution time=0.000s CPU time=0.000s
18:16:33 010535 KNAX6578-I The child job CHILD2 ended. exit status=0 execution time=0.008s
CPU time=0.000s
str008

KNAX6380-I A job name will be added to the spool job directory of the root job. spool job
directory="/var/opt/jp1as/spool/010533-ROOT/"
KNAX7999-I Advanced Shell ended. exit status=0
```

3.4.3 Example of job execution log output (when a child job's spool job is merged into the root job's spool job)

This subsection presents examples of job execution log output when MERGE is selected with the SPOOLJOB_CHILDJOB environment setting parameter (to merge a child job's spool job into the root job's spool job).

(1) Example 1 (defining ch1.sh and ch2.sh)

As shown below, this example defines child job ch1.sh that is started from the root job and child job ch2.sh that is started from within the root job's job step.



```
## parent.sh ##

ch1.sh

#-adsh_step_start S
  ch2.sh
#-adsh_step_end
```

ch1.sh

```
#!/bin/sh
## ch1.sh ##

echo str001 >&2
```

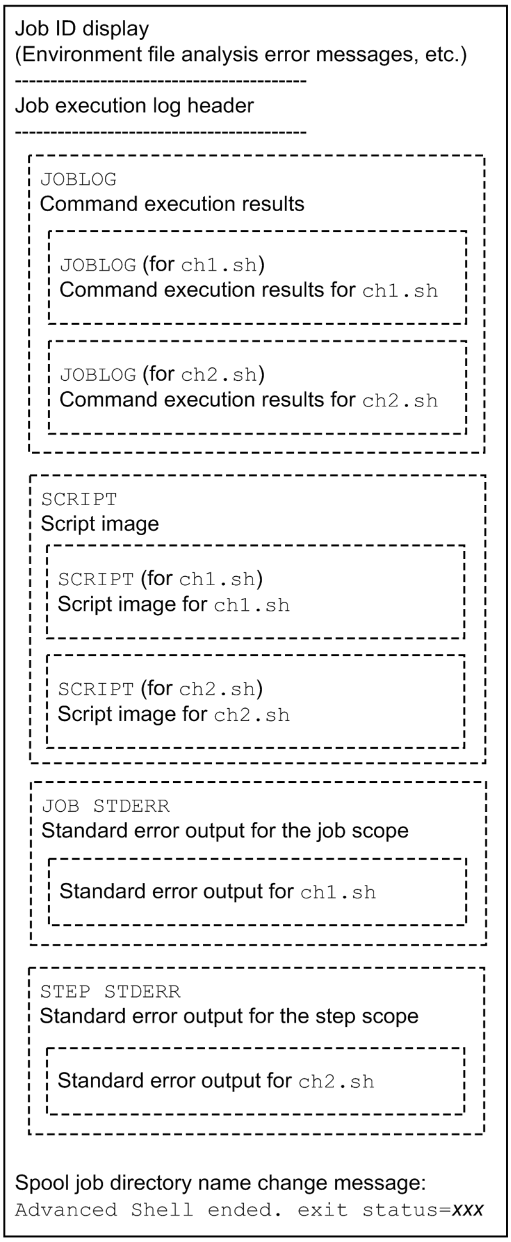
ch2.sh

```
#!/bin/sh
## ch2.sh ##

echo str002 >&2
```

The following subsections explain the configuration and output examples of the root job's job execution log that is output to the standard error.

(a) Configuration of job execution log

The figure below shows the configuration of the job execution log and the locations where the execution results of child jobs `ch1.sh` and `ch2.sh` are output. The child jobs' execution results are also output to `JOBLOG` and `SCRIPT`.



Job ID display
(Environment file analysis error messages, etc.)

Job execution log header

`JOBLOG`
Command execution results

`JOBLOG` (for `ch1.sh`)
Command execution results for `ch1.sh`

`JOBLOG` (for `ch2.sh`)
Command execution results for `ch2.sh`

`SCRIPT`
Script image

`SCRIPT` (for `ch1.sh`)
Script image for `ch1.sh`

`SCRIPT` (for `ch2.sh`)
Script image for `ch2.sh`

`JOB STDERR`
Standard error output for the job scope

Standard error output for `ch1.sh`

`STEP STDERR`
Standard error output for the step scope

Standard error output for `ch2.sh`

Spool job directory name change message:
`Advanced Shell ended. exit status=`*xxx*

(b) Example of job execution log output

The following shows an example of job execution log output:

```
KNAX7901-I The adshexec command will wait for all asynchronous processes at the end of the
job.
KNAX0724-I The job ID was assigned. job ID=010517
---------------------------------------------------------------
 Advanced Shell 10-10

 [Information]
   Job ID          : 010517
   Spool directory : /var/opt/jp1as/spool/010517/
   Date            : 2013/12/06
   EnvFile(system) :
   EnvFile(job)    : /home/usr/adsh_job.ase
   Host name       : HOST01
 [Environment variable from Automatic Job Management System]
   JP1JobName      : adshexec
   JP1JobID        : 1042
   JP1_USERNAME    : usr
   JP1UNCName      : HOST01
   JP1NBQSQueueName: \\HOST01\@SYSTEM
   JP1Priority     : 1
   AJSEXECID       : @A334
---------------------------------------------------------------
********  JOB CONTROLLER MESSAGE  ********
17:18:11 010517 KNAX0091-I ADSH010517 The job started.
17:18:11 010517 KNAX7901-I The adshexec command will wait for all asynchronous processes at
the end of the job.
17:18:11 010517 KNAX7902-I The adshexec command will run in non-tty stdin mode.
17:18:11 010517 KNAX6831-I The command definition matched the rule specified by the
environment settings parameter CHILDJOB_SHEBANG. script="ch1.sh" shebang="/bin/sh"
```

ch1.sh output

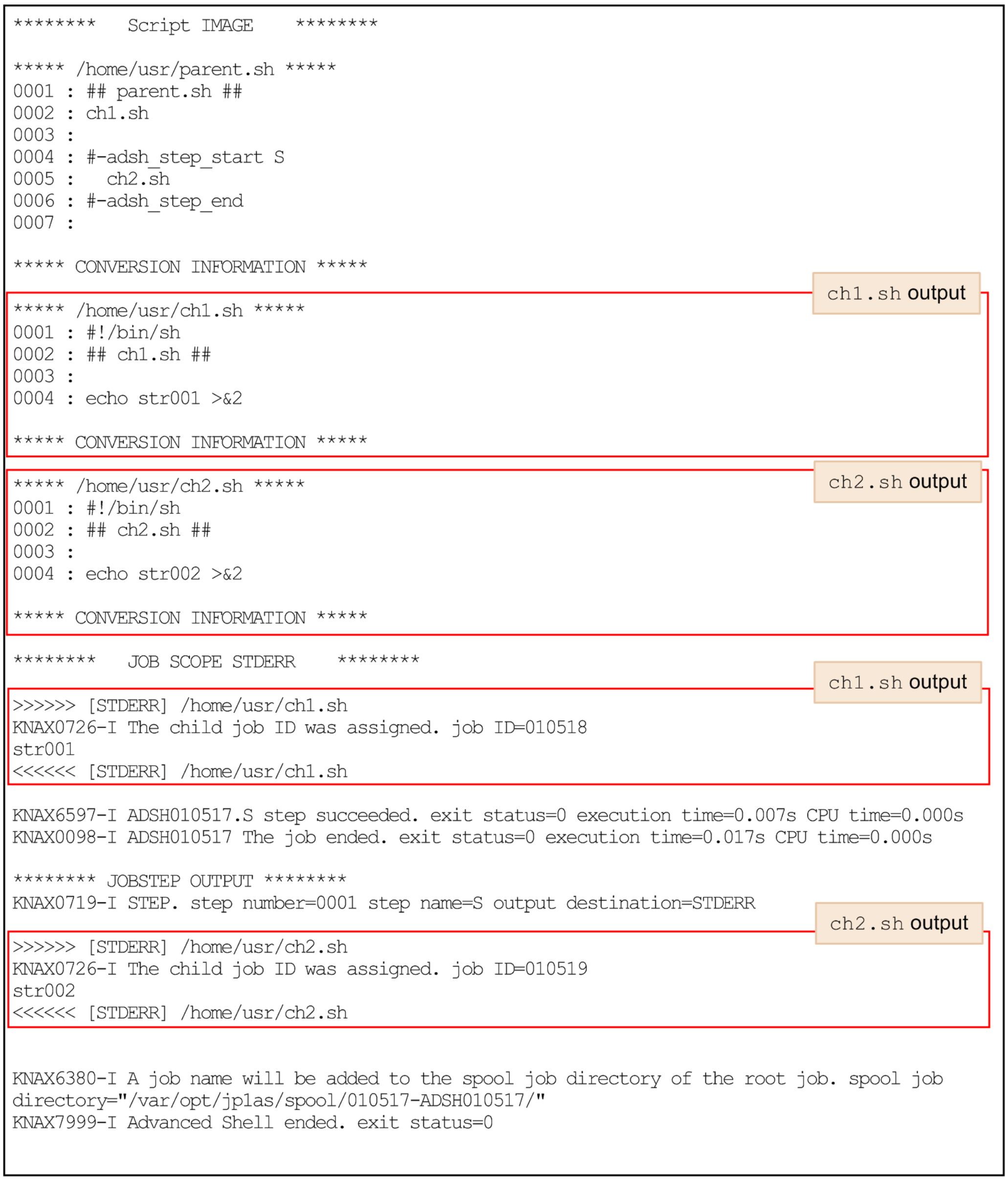
```
>>>>>> [JOBLOG] /home/usr/ch1.sh
17:18:11 010518 KNAX6571-I The child job ADSH010518 started. parent job=ADSH010517 parent job
ID=010517
17:18:11 010518 KNAX6572-I The child job ADSH010518 will use the job environment file "/home/
usr/adsh_job.ase".
17:18:11 010518 KNAX7902-I The adshexec command will run in non-tty stdin mode.
17:18:11 010518 KNAX6112-I Execution of the command echo (line=4) finished successfully. exit
status=0 execution time=0.000s CPU time=0.000s
17:18:11 010518 KNAX6578-I The child job ADSH010518 ended. exit status=0 execution
time=0.001s CPU time=0.000s
<<<<<< [JOBLOG] /home/usr/ch1.sh

17:18:11 010517 KNAX6116-I Execution of the command ch1.sh (line=2) finished successfully.
exit status=0 execution time=0.006s CPU time=0.000s
17:18:11 010517 KNAX0092-I ADSH010517.S step started.
17:18:11 010517 KNAX6831-I The command definition matched the rule specified by the
environment settings parameter CHILDJOB_SHEBANG. script="ch2.sh" shebang="/bin/sh"
```

ch2.sh output

```
>>>>>> [JOBLOG] /home/usr/ch2.sh
17:18:11 010519 KNAX6571-I The child job ADSH010519 started. parent job=ADSH010517 parent job
ID=010517
17:18:11 010519 KNAX6572-I The child job ADSH010519 will use the job environment file "/home/
usr/adsh_job.ase".
17:18:11 010519 KNAX7902-I The adshexec command will run in non-tty stdin mode.
17:18:11 010519 KNAX6112-I Execution of the command echo (line=4) finished successfully. exit
status=0 execution time=0.000s CPU time=0.000s
17:18:11 010519 KNAX6578-I The child job ADSH010519 ended. exit status=0 execution
time=0.001s CPU time=0.000s
<<<<<< [JOBLOG] /home/usr/ch2.sh

17:18:11 010517 KNAX6116-I Execution of the command ch2.sh (line=5) finished successfully.
exit status=0 execution time=0.007s CPU time=0.000s
17:18:11 010517 KNAX6597-I ADSH010517.S step succeeded. exit status=0 execution time=0.007s
CPU time=0.000s
17:18:11 010517 KNAX0098-I ADSH010517 The job ended. exit status=0 execution time=0.017s CPU
time=0.000s
```

(Continued)

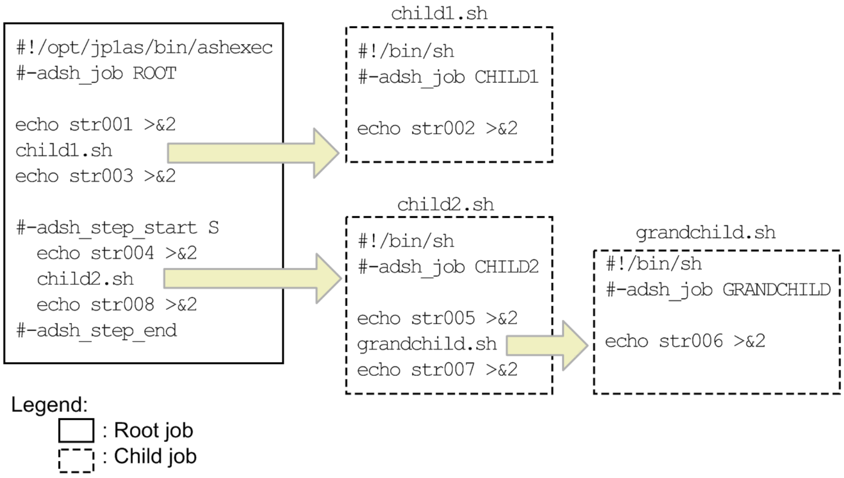
(Continued)



```
********   Script IMAGE    ********

***** /home/usr/parent.sh *****
0001 : ## parent.sh ##
0002 : ch1.sh
0003 :
0004 : #-adsh_step_start S
0005 :   ch2.sh
0006 : #-adsh_step_end
0007 :

***** CONVERSION INFORMATION *****

***** /home/usr/ch1.sh *****
0001 : #!/bin/sh
0002 : ## ch1.sh ##
0003 :
0004 : echo str001 >&2

***** CONVERSION INFORMATION *****

***** /home/usr/ch2.sh *****
0001 : #!/bin/sh
0002 : ## ch2.sh ##
0003 :
0004 : echo str002 >&2

***** CONVERSION INFORMATION *****

********   JOB SCOPE STDERR    ********

>>>>>> [STDERR] /home/usr/ch1.sh
KNAX0726-I The child job ID was assigned. job ID=010518
str001
<<<<<< [STDERR] /home/usr/ch1.sh

KNAX6597-I ADSH010517.S step succeeded. exit status=0 execution time=0.007s CPU time=0.000s
KNAX0098-I ADSH010517 The job ended. exit status=0 execution time=0.017s CPU time=0.000s

******** JOBSTEP OUTPUT ********
KNAX0719-I STEP. step number=0001 step name=S output destination=STDERR

>>>>>> [STDERR] /home/usr/ch2.sh
KNAX0726-I The child job ID was assigned. job ID=010519
str002
<<<<<< [STDERR] /home/usr/ch2.sh


KNAX6380-I A job name will be added to the spool job directory of the root job. spool job
directory="/var/opt/jp1as/spool/010517-ADSH010517/"
KNAX7999-I Advanced Shell ended. exit status=0
```

(2) Example 2 (defining child1.sh, child2.sh, and grandchild.sh)

As shown below, this example defines child job `child1.sh`, child job `child2.sh`, and `grandchild.sh` that is started from child job `child2.sh`.



```
#!/opt/jp1as/bin/ashexec
#-adsh_job ROOT

echo str001 >&2
child1.sh
echo str003 >&2

#-adsh_step_start S
  echo str004 >&2
  child2.sh
  echo str008 >&2
#-adsh_step_end
```

child1.sh

```
#!/bin/sh
#-adsh_job CHILD1

echo str002 >&2
```

child2.sh

```
#!/bin/sh
#-adsh_job CHILD2

echo str005 >&2
grandchild.sh
echo str007 >&2
```

grandchild.sh

```
#!/bin/sh
#-adsh_job GRANDCHILD

echo str006 >&2
```

Legend:

□ : Root job

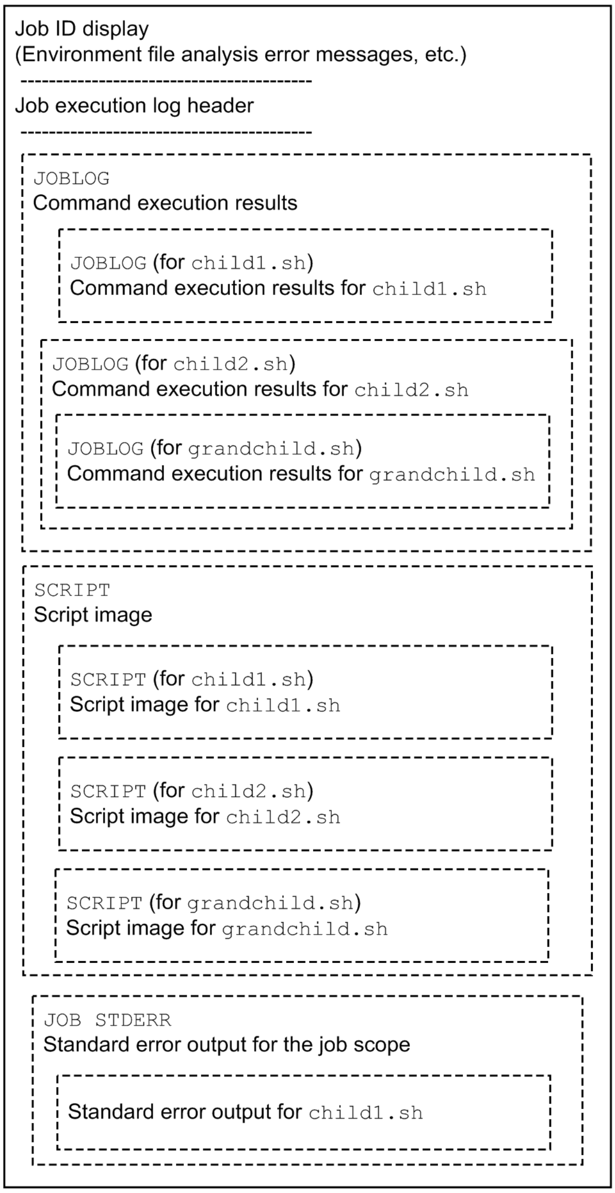
⬚ : Child job

The following subsections explain the configuration and output examples of the root job's job execution log that is output to the standard error output.

(a) Configuration of job execution log

The figure below shows the configuration of the job execution log and the locations where the execution results of child job `child1.sh`, child job `child2.sh`, and `grandchild.sh` are output. The child jobs' execution results are also output to `JOBLOG` and `SCRIPT`.

- Job execution log that can be referenced from other programs (such as JP1/AJS)



Job ID display
(Environment file analysis error messages, etc.)

Job execution log header

`JOBLOG`
Command execution results

`JOBLOG` (for `child1.sh`)
Command execution results for `child1.sh`

`JOBLOG` (for `child2.sh`)
Command execution results for `child2.sh`

`JOBLOG` (for `grandchild.sh`)
Command execution results for `grandchild.sh`

`SCRIPT`
Script image

`SCRIPT` (for `child1.sh`)
Script image for `child1.sh`

`SCRIPT` (for `child2.sh`)
Script image for `child2.sh`

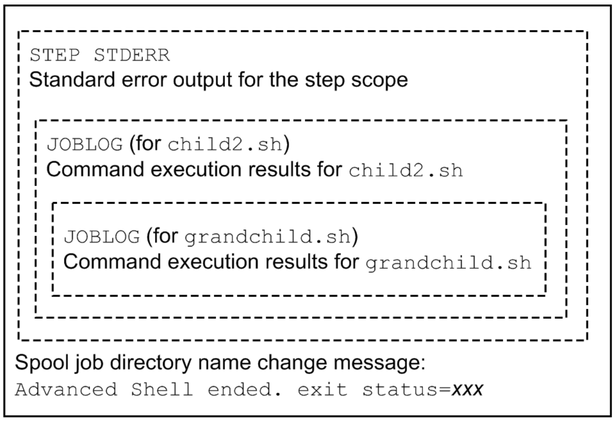
`SCRIPT` (for `grandchild.sh`)
Script image for `grandchild.sh`

`JOB STDERR`
Standard error output for the job scope

Standard error output for `child1.sh`

(Continued)

(Continued)



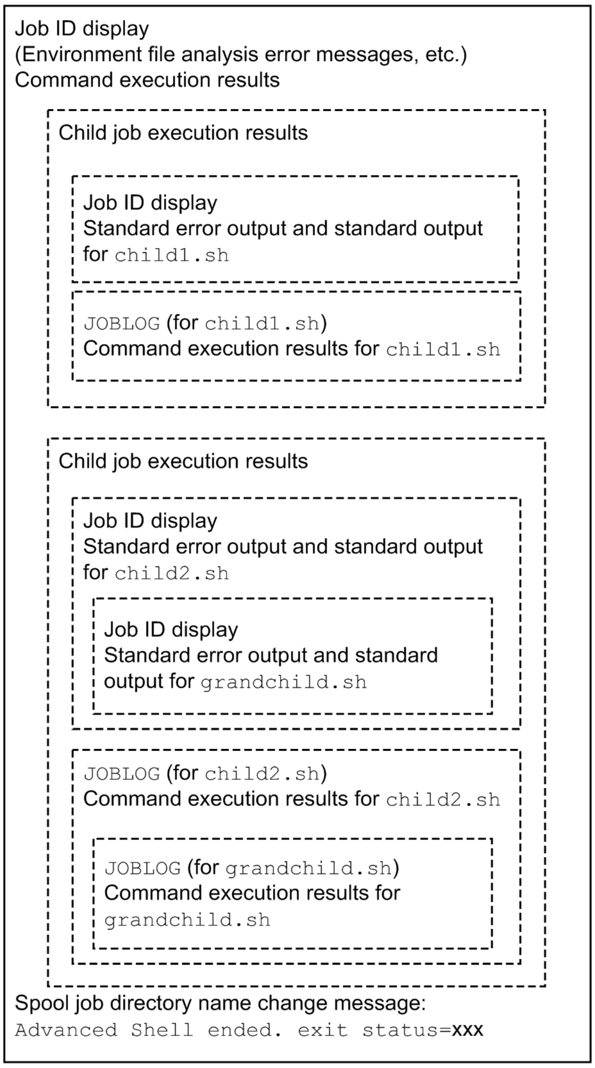
`STEP STDERR`
Standard error output for the step scope

`JOBLOG` (for `child2.sh`)
Command execution results for `child2.sh`

`JOBLOG` (for `grandchild.sh`)
Command execution results for `grandchild.sh`

Spool job directory name change message:
`Advanced Shell ended. exit status=`*`xxx`*

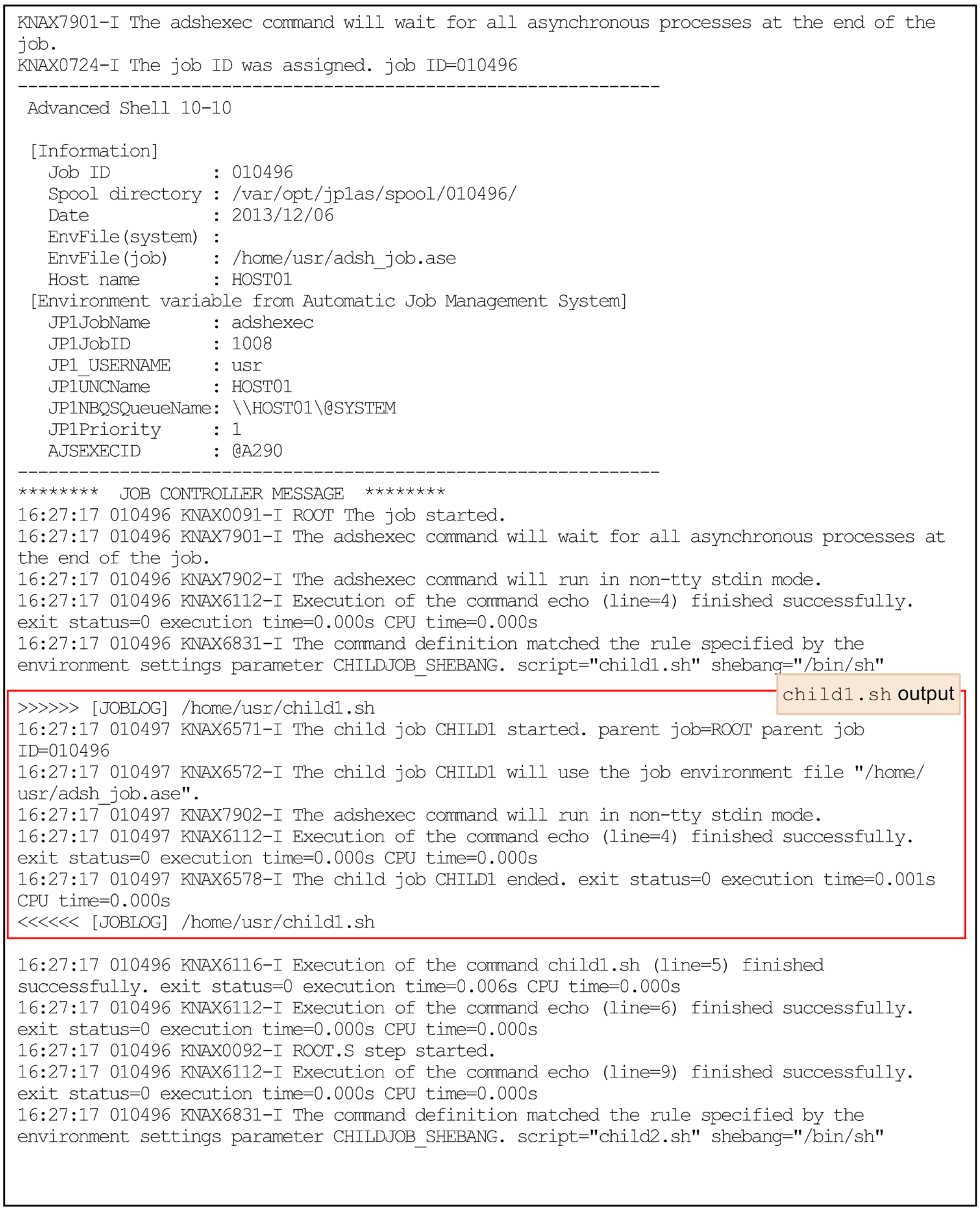
- Job execution log during debug execution


Job ID display
(Environment file analysis error messages, etc.)
Command execution results
Child job execution results
Job ID display
Standard error output and standard output for child1.sh
JOBLOG (for child1.sh)
Command execution results for child1.sh
Child job execution results
Job ID display
Standard error output and standard output for child2.sh
Job ID display
Standard error output and standard output for grandchild.sh
JOBLOG (for child2.sh)
Command execution results for child2.sh
JOBLOG (for grandchild.sh)
Command execution results for grandchild.sh
Spool job directory name change message:
Advanced Shell ended. exit status=xxx

(b) Example of job execution log output

The following are examples of job execution log output.

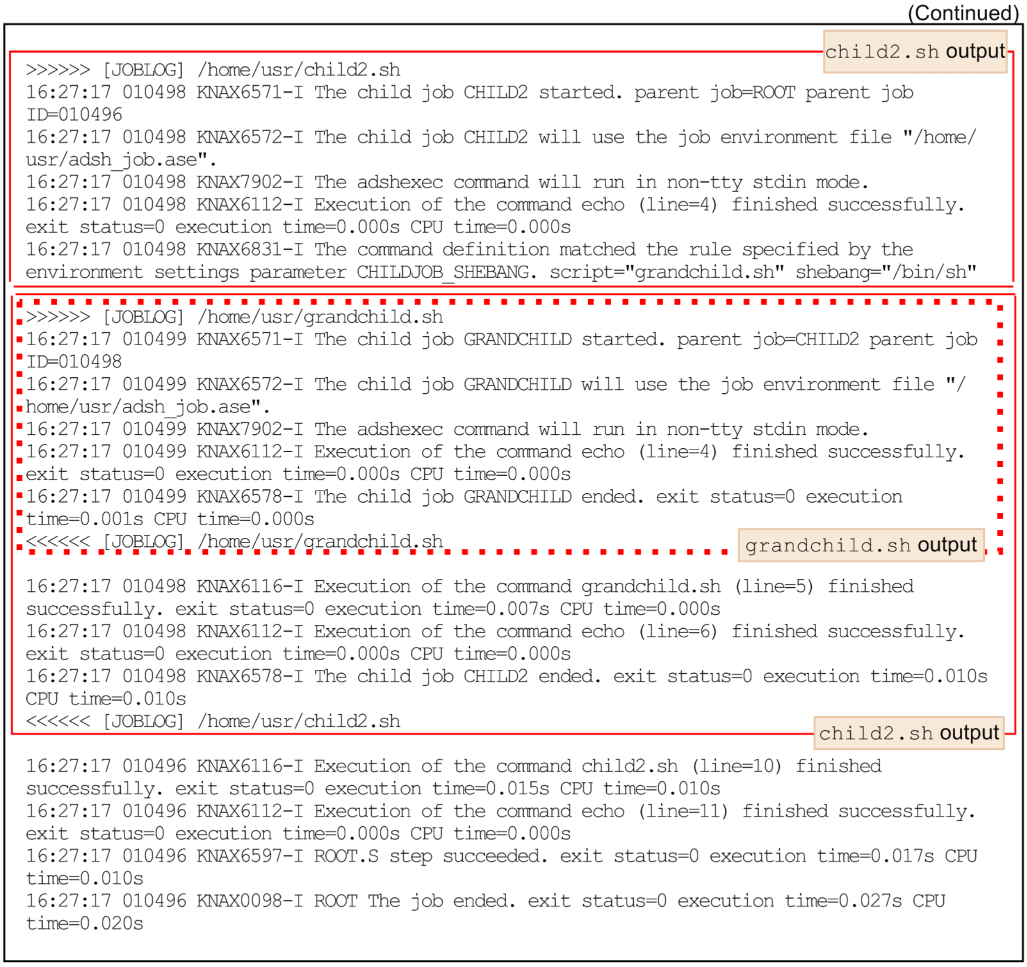
- Job execution log that can be referenced from other programs (such as JP1/AJS)



```
KNAX7901-I The adshexec command will wait for all asynchronous processes at the end of the
job.
KNAX0724-I The job ID was assigned. job ID=010496
---------------------------------------------------------------
 Advanced Shell 10-10

 [Information]
   Job ID          : 010496
   Spool directory : /var/opt/jp1as/spool/010496/
   Date            : 2013/12/06
   EnvFile(system) :
   EnvFile(job)    : /home/usr/adsh_job.ase
   Host name       : HOST01
 [Environment variable from Automatic Job Management System]
   JP1JobName      : adshexec
   JP1JobID        : 1008
   JP1_USERNAME    : usr
   JP1UNCName      : HOST01
   JP1NBQSQueueName: \\HOST01\@SYSTEM
   JP1Priority     : 1
   AJSEXECID       : @A290
---------------------------------------------------------------
********  JOB CONTROLLER MESSAGE  ********
16:27:17 010496 KNAX0091-I ROOT The job started.
16:27:17 010496 KNAX7901-I The adshexec command will wait for all asynchronous processes at
the end of the job.
16:27:17 010496 KNAX7902-I The adshexec command will run in non-tty stdin mode.
16:27:17 010496 KNAX6112-I Execution of the command echo (line=4) finished successfully.
exit status=0 execution time=0.000s CPU time=0.000s
16:27:17 010496 KNAX6831-I The command definition matched the rule specified by the
environment settings parameter CHILDJOB_SHEBANG. script="child1.sh" shebang="/bin/sh"
>>>>>> [JOBLOG] /home/usr/child1.sh
16:27:17 010497 KNAX6571-I The child job CHILD1 started. parent job=ROOT parent job
ID=010496
16:27:17 010497 KNAX6572-I The child job CHILD1 will use the job environment file "/home/
usr/adsh_job.ase".
16:27:17 010497 KNAX7902-I The adshexec command will run in non-tty stdin mode.
16:27:17 010497 KNAX6112-I Execution of the command echo (line=4) finished successfully.
exit status=0 execution time=0.000s CPU time=0.000s
16:27:17 010497 KNAX6578-I The child job CHILD1 ended. exit status=0 execution time=0.001s
CPU time=0.000s
<<<<<< [JOBLOG] /home/usr/child1.sh

16:27:17 010496 KNAX6116-I Execution of the command child1.sh (line=5) finished
successfully. exit status=0 execution time=0.006s CPU time=0.000s
16:27:17 010496 KNAX6112-I Execution of the command echo (line=6) finished successfully.
exit status=0 execution time=0.000s CPU time=0.000s
16:27:17 010496 KNAX0092-I ROOT.S step started.
16:27:17 010496 KNAX6112-I Execution of the command echo (line=9) finished successfully.
exit status=0 execution time=0.000s CPU time=0.000s
16:27:17 010496 KNAX6831-I The command definition matched the rule specified by the
environment settings parameter CHILDJOB_SHEBANG. script="child2.sh" shebang="/bin/sh"
```

(Continued)

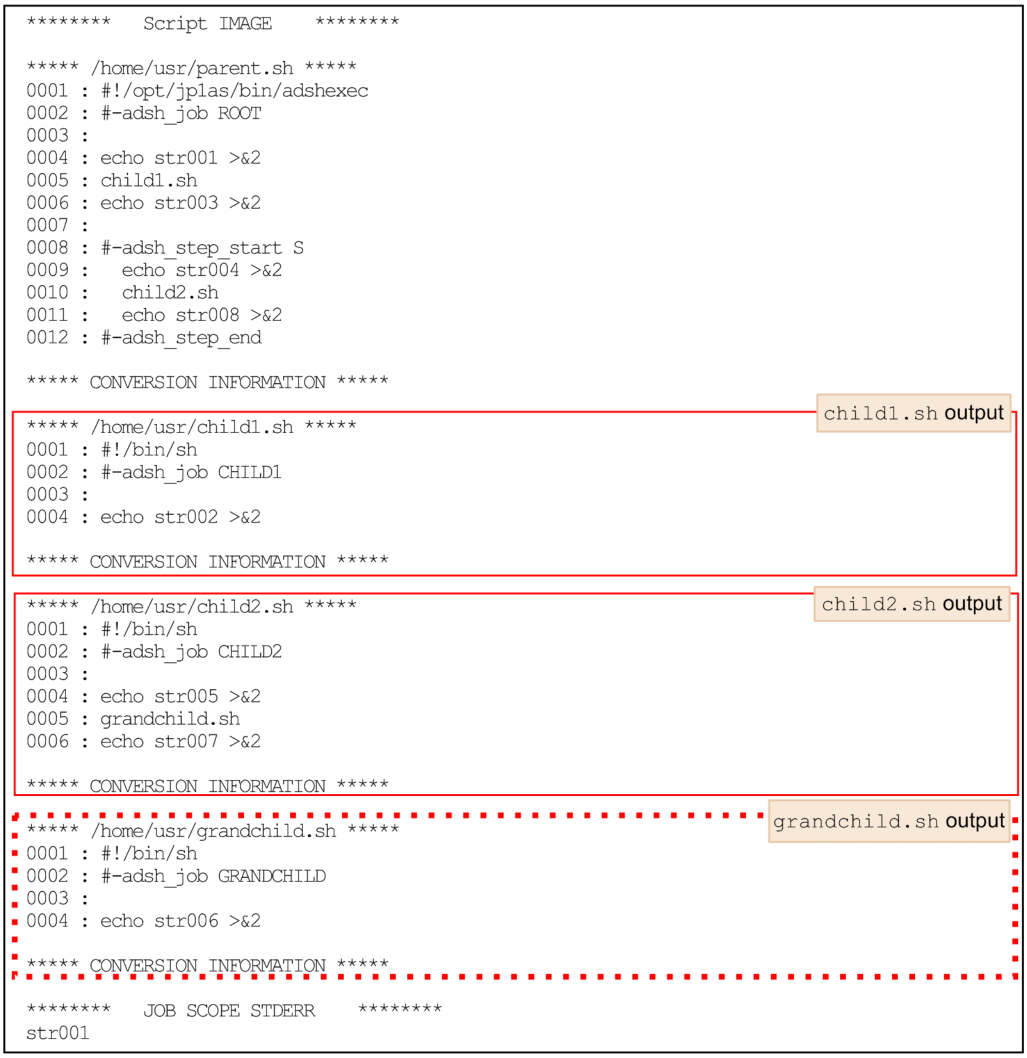
(Continued)



```
>>>>>> [JOBLOG] /home/usr/child2.sh
16:27:17 010498 KNAX6571-I The child job CHILD2 started. parent job=ROOT parent job
ID=010496
16:27:17 010498 KNAX6572-I The child job CHILD2 will use the job environment file "/home/
usr/adsh_job.ase".
16:27:17 010498 KNAX7902-I The adshexec command will run in non-tty stdin mode.
16:27:17 010498 KNAX6112-I Execution of the command echo (line=4) finished successfully.
exit status=0 execution time=0.000s CPU time=0.000s
16:27:17 010498 KNAX6831-I The command definition matched the rule specified by the
environment settings parameter CHILDJOB_SHEBANG. script="grandchild.sh" shebang="/bin/sh"
>>>>>> [JOBLOG] /home/usr/grandchild.sh
16:27:17 010499 KNAX6571-I The child job GRANDCHILD started. parent job=CHILD2 parent job
ID=010498
16:27:17 010499 KNAX6572-I The child job GRANDCHILD will use the job environment file "/
home/usr/adsh_job.ase".
16:27:17 010499 KNAX7902-I The adshexec command will run in non-tty stdin mode.
16:27:17 010499 KNAX6112-I Execution of the command echo (line=4) finished successfully.
exit status=0 execution time=0.000s CPU time=0.000s
16:27:17 010499 KNAX6578-I The child job GRANDCHILD ended. exit status=0 execution
time=0.001s CPU time=0.000s
<<<<<< [JOBLOG] /home/usr/grandchild.sh

16:27:17 010498 KNAX6116-I Execution of the command grandchild.sh (line=5) finished
successfully. exit status=0 execution time=0.007s CPU time=0.000s
16:27:17 010498 KNAX6112-I Execution of the command echo (line=6) finished successfully.
exit status=0 execution time=0.000s CPU time=0.000s
16:27:17 010498 KNAX6578-I The child job CHILD2 ended. exit status=0 execution time=0.010s
CPU time=0.010s
<<<<<< [JOBLOG] /home/usr/child2.sh

16:27:17 010496 KNAX6116-I Execution of the command child2.sh (line=10) finished
successfully. exit status=0 execution time=0.015s CPU time=0.010s
16:27:17 010496 KNAX6112-I Execution of the command echo (line=11) finished successfully.
exit status=0 execution time=0.000s CPU time=0.000s
16:27:17 010496 KNAX6597-I ROOT.S step succeeded. exit status=0 execution time=0.017s CPU
time=0.010s
16:27:17 010496 KNAX0098-I ROOT The job ended. exit status=0 execution time=0.027s CPU
time=0.020s
```

(Continued)

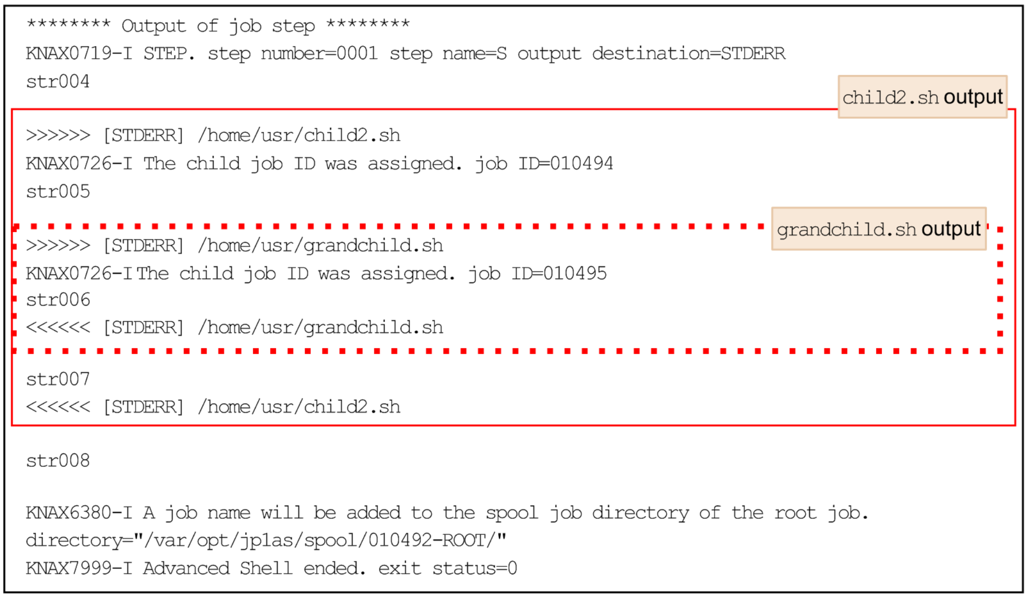
(Continued)



```
********   Script IMAGE    ********

***** /home/usr/parent.sh *****
0001 : #!/opt/jp1as/bin/adshexec
0002 : #-adsh_job ROOT
0003 :
0004 : echo str001 >&2
0005 : child1.sh
0006 : echo str003 >&2
0007 :
0008 : #-adsh_step_start S
0009 :   echo str004 >&2
0010 :   child2.sh
0011 :   echo str008 >&2
0012 : #-adsh_step_end

***** CONVERSION INFORMATION *****

***** /home/usr/child1.sh *****
0001 : #!/bin/sh
0002 : #-adsh_job CHILD1
0003 :
0004 : echo str002 >&2

***** CONVERSION INFORMATION *****

***** /home/usr/child2.sh *****
0001 : #!/bin/sh
0002 : #-adsh_job CHILD2
0003 :
0004 : echo str005 >&2
0005 : grandchild.sh
0006 : echo str007 >&2

***** CONVERSION INFORMATION *****

***** /home/usr/grandchild.sh *****
0001 : #!/bin/sh
0002 : #-adsh_job GRANDCHILD
0003 :
0004 : echo str006 >&2

***** CONVERSION INFORMATION *****

********   JOB SCOPE STDERR    ********
str001
```

(Continued)

(Continued)



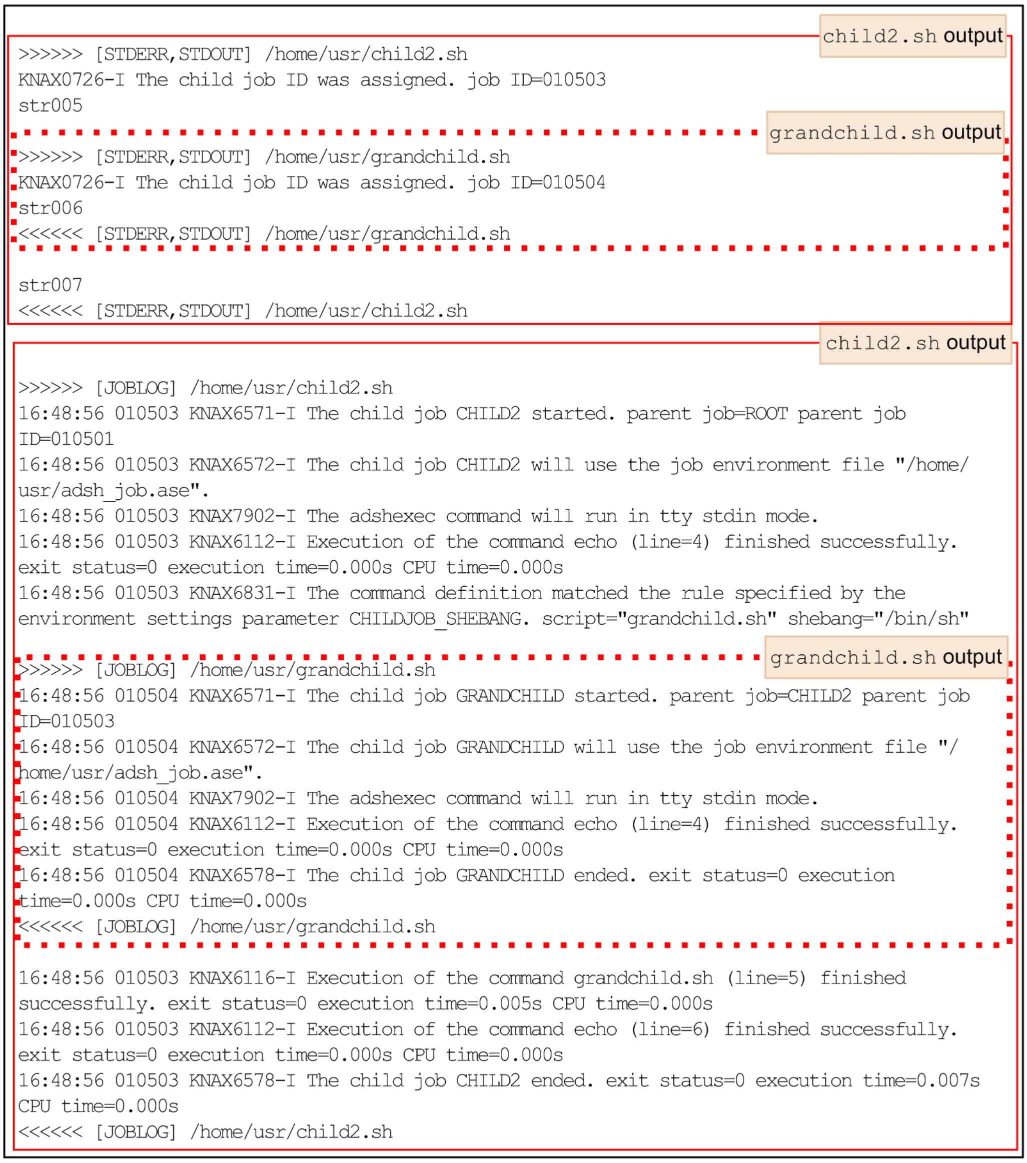
```
******** Output of job step ********
KNAX0719-I STEP. step number=0001 step name=S output destination=STDERR
str004

>>>>>> [STDERR] /home/usr/child2.sh
KNAX0726-I The child job ID was assigned. job ID=010494
str005

>>>>>> [STDERR] /home/usr/grandchild.sh
KNAX0726-I The child job ID was assigned. job ID=010495
str006
<<<<<< [STDERR] /home/usr/grandchild.sh

str007
<<<<<< [STDERR] /home/usr/child2.sh

str008

KNAX6380-I A job name will be added to the spool job directory of the root job.
directory="/var/opt/jp1as/spool/010492-ROOT/"
KNAX7999-I Advanced Shell ended. exit status=0
```

- Job execution log during debug execution

```
$ adshexec -d parent.sh
KNAX7901-I The adshexec command will wait for all asynchronous processes at the end of the
job.
KNAX0724-I The job ID was assigned. job ID=010500
(adshdb) run
KNAX7007-I Execution of the following script will now start: /home/usr/parent.sh

KNAX0724-I The job ID was assigned. job ID=010501
KNAX0091-I ROOT The job started.
KNAX7902-I The adshexec command will run in tty stdin mode.
str001
KNAX6112-I Execution of the command echo (line=4) finished successfully. exit status=0
execution time=0.000s CPU time=0.000s
KNAX6831-I The command definition matched the rule specified by the environment settings
parameter CHILDJOB_SHEBANG. script="child1.sh" shebang="/bin/sh"
                                                                          child1.sh output
>>>>>> [STDERR,STDOUT] /home/usr/child1.sh
KNAX0726-I The child job ID was assigned. job ID=010502
str002
<<<<<< [STDERR,STDOUT] /home/usr/child1.sh

                                                                          child1.sh output
>>>>>> [JOBLOG] /home/usr/child1.sh
16:48:56 010502 KNAX6571-I The child job CHILD1 started. parent job=ROOT parent job
ID=010501
16:48:56 010502 KNAX6572-I The child job CHILD1 will use the job environment file "/home/
usr/adsh_job.ase".
16:48:56 010502 KNAX7902-I The adshexec command will run in tty stdin mode.
16:48:56 010502 KNAX6112-I Execution of the command echo (line=4) finished successfully.
exit status=0 execution time=0.000s CPU time=0.000s
16:48:56 010502 KNAX6578-I The child job CHILD1 ended. exit status=0 execution time=0.001s
CPU time=0.000s
<<<<<< [JOBLOG] /home/usr/child1.sh

KNAX6116-I Execution of the command child1.sh (line=5) finished successfully. exit status=0
execution time=0.009s CPU time=0.000s
str003
KNAX6112-I Execution of the command echo (line=6) finished successfully. exit status=0
execution time=0.000s CPU time=0.000s
KNAX0092-I ROOT.S step started.
str004
KNAX6112-I Execution of the command echo (line=9) finished successfully. exit status=0
execution time=0.000s CPU time=0.000s
KNAX6831-I The command definition matched the rule specified by the environment settings
parameter CHILDJOB_SHEBANG. script="child2.sh" shebang="/bin/sh"
```

(Continued)

(Continued)



```
>>>>>> [STDERR,STDOUT] /home/usr/child2.sh
KNAX0726-I The child job ID was assigned. job ID=010503
str005
>>>>>> [STDERR,STDOUT] /home/usr/grandchild.sh
KNAX0726-I The child job ID was assigned. job ID=010504
str006
<<<<<< [STDERR,STDOUT] /home/usr/grandchild.sh

str007
<<<<<< [STDERR,STDOUT] /home/usr/child2.sh

>>>>>> [JOBLOG] /home/usr/child2.sh
16:48:56 010503 KNAX6571-I The child job CHILD2 started. parent job=ROOT parent job
ID=010501
16:48:56 010503 KNAX6572-I The child job CHILD2 will use the job environment file "/home/
usr/adsh_job.ase".
16:48:56 010503 KNAX7902-I The adshexec command will run in tty stdin mode.
16:48:56 010503 KNAX6112-I Execution of the command echo (line=4) finished successfully.
exit status=0 execution time=0.000s CPU time=0.000s
16:48:56 010503 KNAX6831-I The command definition matched the rule specified by the
environment settings parameter CHILDJOB_SHEBANG. script="grandchild.sh" shebang="/bin/sh"

>>>>>> [JOBLOG] /home/usr/grandchild.sh
16:48:56 010504 KNAX6571-I The child job GRANDCHILD started. parent job=CHILD2 parent job
ID=010503
16:48:56 010504 KNAX6572-I The child job GRANDCHILD will use the job environment file "/
home/usr/adsh_job.ase".
16:48:56 010504 KNAX7902-I The adshexec command will run in tty stdin mode.
16:48:56 010504 KNAX6112-I Execution of the command echo (line=4) finished successfully.
exit status=0 execution time=0.000s CPU time=0.000s
16:48:56 010504 KNAX6578-I The child job GRANDCHILD ended. exit status=0 execution
time=0.000s CPU time=0.000s
<<<<<< [JOBLOG] /home/usr/grandchild.sh

16:48:56 010503 KNAX6116-I Execution of the command grandchild.sh (line=5) finished
successfully. exit status=0 execution time=0.005s CPU time=0.000s
16:48:56 010503 KNAX6112-I Execution of the command echo (line=6) finished successfully.
exit status=0 execution time=0.000s CPU time=0.000s
16:48:56 010503 KNAX6578-I The child job CHILD2 ended. exit status=0 execution time=0.007s
CPU time=0.000s
<<<<<< [JOBLOG] /home/usr/child2.sh
```

(Continued)

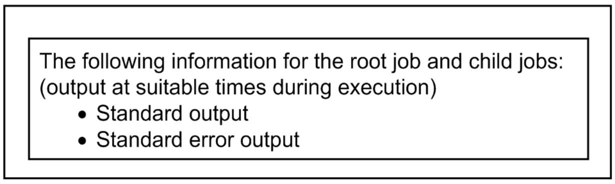
3.4.4 Examples of job execution log output (when the simple output mode or the minimum output mode is selected)

This subsection presents examples of the job execution log when the simple output mode or the minimum output mode is selected with the OUTPUT_MODE_ROOT or OUTPUT_MODE_CHILD environment setting parameter. Note that the error messages output in the simple output mode differ from those output in the minimum output mode.

(1) Configuration of job execution log

The following shows the configuration of the job execution logs.

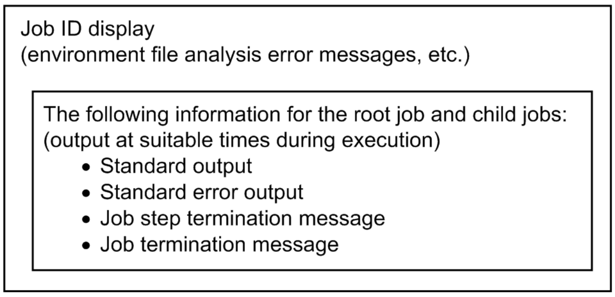
- Job execution log that can be referenced from other programs (such as JP1/AJS)



The following information for the root job and child jobs:
(output at suitable times during execution)

- Standard output
- Standard error output

- Job execution log during debug execution



Job ID display
(environment file analysis error messages, etc.)

The following information for the root job and child jobs:
(output at suitable times during execution)

- Standard output
- Standard error output
- Job step termination message
- Job termination message

(2) Examples of job execution log output

The following are examples of job execution log output:

- Job execution log that can be referenced from other programs (such as JP1/AJS)

■ Environment setting parameters

```
#-adsh_conf OUTPUT_MODE_ROOT SIMPLE
#-adsh_conf OUTPUT_MODE_CHILD SIMPLE
#-adsh_conf CHILDJOB_EXT ash
```

■ Job definition script: `logroot.ash`

```
#-adsh_job SampleJobRoot
#-adsh_file_temp WORK01
#-adsh_file_temp WORK02
./logsub.ash data tokyo   2>$WORK01
./logsub.ash data fukuoka 2>$WORK02
echo -E "***WORK01*****" >&2
cat $WORK01 >&2
echo -E "***WORK02*****" >&2
cat $WORK02 >&2
```

■ Job definition script: `logsub.ash`

```
#-adsh_job SampleSub
cat $1 | grep $2 >&2
```

■ Input data: `data`

```
aichi     nagoya     052
fukuoka   kurume     0942
fukushima iwaki      0246
tokyo     machida    042
tokyo     tachikawa  042
```

■ Execution example

```
[user001@HOST01 ~]$ /opt/jp1as/bin/adshexec logroot.ash
***WORK01*****
tokyo     machida    042
tokyo     tachikawa  042
***WORK02*****
fukuoka   kurume     0942
[user001@HOST01 ~]$
[user001@HOST01 ~]$
```

- Job execution log during debug execution

```
[user001@hosta user001]$ /opt/jp1as/bin/adshexec -d logroot.ash
KNAX7901-I The adshexec command will wait for all asynchronous processes at the end of the
job.
KNAX0724-I The job ID was assigned. job ID=000483
(adshdb) list
1: #-adsh_job SampleJobRoot
2: #-adsh_file_temp WORK01
3: #-adsh_file_temp WORK02
4: ./logsub.ash data tokyo 2>$WORK01
5: ./logsub.ash data fukuoka 2>$WORK02
6: echo -E "***WORK01*****" >&2
7: cat $WORK01 >&2
8: echo -E "***WORK02*****" >&2
9: cat $WORK02 >&2
(adshdb) b 4
KNAX7018-I Breakpoint "1": filename="logroot.ash" line=4
(adshdb) run
KNAX7007-I Execution of the following script will now start: /home/user001/logroot.ash

KNAX0724-I The job ID was assigned. job ID=000484

KNAX7018-I Breakpoint "1": filename="logroot.ash" line=4
KNAX7032-I The script "logroot.ash" stopped running.
4: ./logsub.ash data tokyo 2>$WORK01
Current: ./logsub.ash
(adshdb) info b
Num  Type        What
1    breakpoint  logroot.ash:4
(adshdb) s
KNAX7032-I The script "logroot.ash" stopped running.
5: ./logsub.ash data fukuoka 2>$WORK02
Current: ./logsub.ash
(adshdb) c
KNAX7034-I The script will continue.
***WORK01*****
tokyo     machida     042
tokyo     tachikawa   042
***WORK02*****
fukuoka   kurume      0942
KNAX0098-I SampleJobRoot The job ended. exit status=0 execution time=24.432s CPU time=0.020s

(adshdb) quit
[user001@hosta user001]$
```

3.4.5 Examples of job execution log output (when only the standard error output is output)

This subsection presents an example of the job execution log when `STDERR` (output only the standard error output to the job execution log) is selected with the `JOBEXECLOG_PRINT` environment setting parameter.

This example assumes that job definition scripts `sample.ash`, `samplesub1.ash`, and `samplesub2.ash` have been defined as shown in the following.

- Job definition script `sample.ash`

```
#-adsh_job SAMPLEJOB
echo JOB_STDERR_001 >&2
cd /home/user001/dir
cd xxx
cd /home/user001
#-adsh_step_start S1 -run always -onError cont
echo STEP_STDERR_001 >&2
cd /home/user001/dir
cd xxx
cd /home/user001
#-adsh_step_end
echo JOB_STDERR_002 >&2
#-adsh_step_start S2 -run always -onError cont
echo STEP_STDERR_002 >&2
./samplesub1.ash
#-adsh_step_end
cd /home/user001/dir
cd xxx
cd /home/user001
echo JOB_STDERR_003 >&2
./samplesub2.ash
```

- Job definition script `samplesub1.ash`

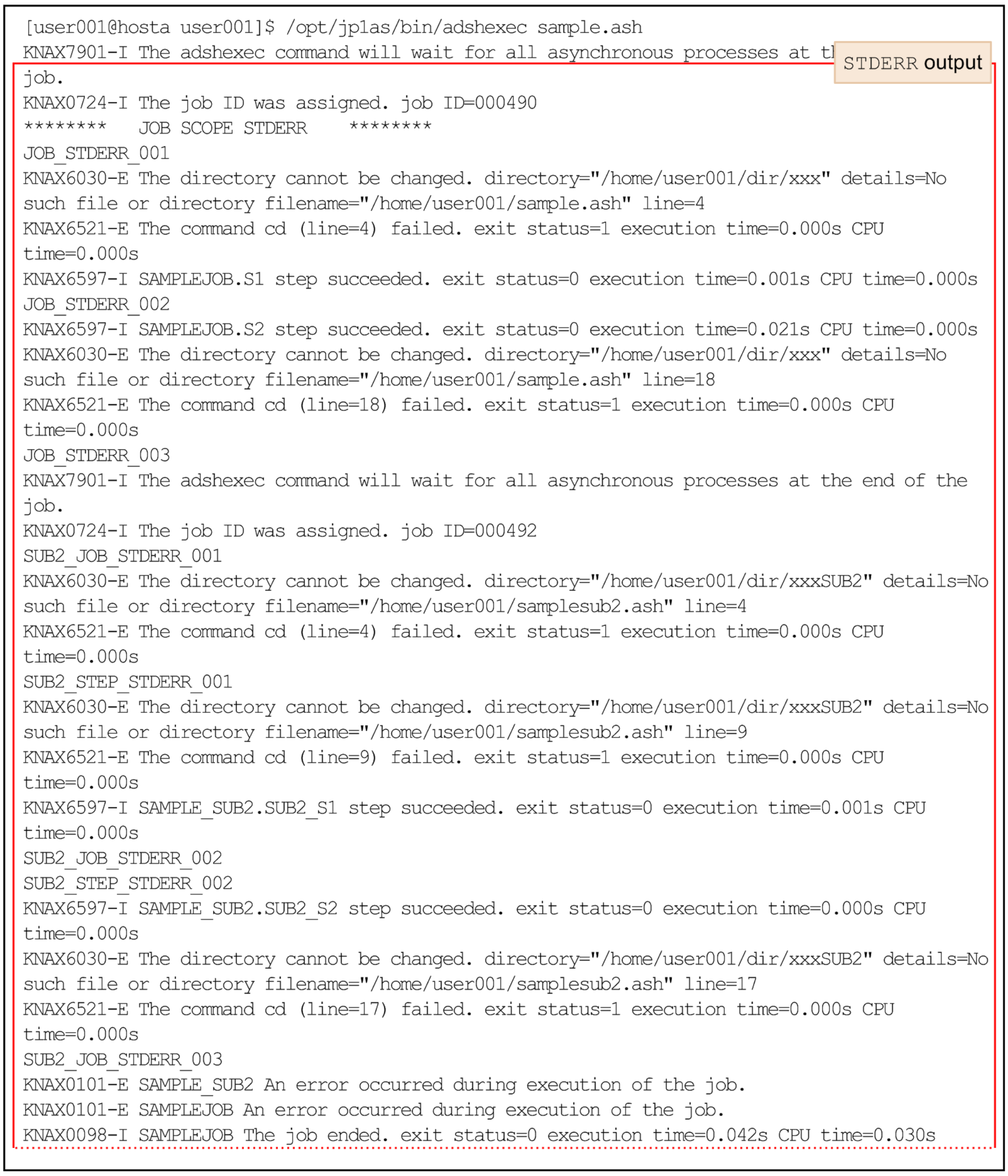
```
#-adsh_job SAMPLE_SUB1
echo SUB1_JOB_STDERR_001 >&2
cd /home/user001/dir
cd xxxSUB1
cd /home/user001
#-adsh_step_start SUB1_S1 -run always -onError cont
echo SUB1_STEP_STDERR_001 >&2
cd /home/user001/dir
cd xxxSUB1
cd /home/user001
#-adsh_step_end
echo SUB1_JOB_STDERR_002 >&2
#-adsh_step_start SUB1_S2 -run always -onError cont
echo SUB1_STEP_STDERR_002 >&2
#-adsh_step_end
cd /home/user001/dir
cd xxxSUB1
cd /home/user001
echo SUB1_JOB_STDERR_003 >&2
```

- Job definition script `samplesub2.ash`

```
#-adsh_job SAMPLE_SUB2
echo SUB2_JOB_STDERR_001 >&2
cd /home/user001/dir
cd xxxSUB2
cd /home/user001
#-adsh_step_start SUB2_S1 -run always -onError cont
echo SUB2_STEP_STDERR_001 >&2
cd /home/user001/dir
cd xxxSUB2
cd /home/user001
```

```
#-adsh_step_end
echo SUB2_JOB_STDERR_002 >&2
#-adsh_step_start SUB2_S2 -run always -onError cont
echo SUB2_STEP_STDERR_002 >&2
#-adsh_step_end
cd /home/user001/dir
cd xxxSUB2
cd /home/user001
echo SUB2_JOB_STDERR_003 >&2
```

The following shows an example of the job execution log output:



```
[user001@hosta user001]$ /opt/jp1as/bin/adshexec sample.ash
KNAX7901-I The adshexec command will wait for all asynchronous processes at th
job.
KNAX0724-I The job ID was assigned. job ID=000490
********   JOB SCOPE STDERR    ********
JOB_STDERR_001
KNAX6030-E The directory cannot be changed. directory="/home/user001/dir/xxx" details=No
such file or directory filename="/home/user001/sample.ash" line=4
KNAX6521-E The command cd (line=4) failed. exit status=1 execution time=0.000s CPU
time=0.000s
KNAX6597-I SAMPLEJOB.S1 step succeeded. exit status=0 execution time=0.001s CPU time=0.000s
JOB_STDERR_002
KNAX6597-I SAMPLEJOB.S2 step succeeded. exit status=0 execution time=0.021s CPU time=0.000s
KNAX6030-E The directory cannot be changed. directory="/home/user001/dir/xxx" details=No
such file or directory filename="/home/user001/sample.ash" line=18
KNAX6521-E The command cd (line=18) failed. exit status=1 execution time=0.000s CPU
time=0.000s
JOB_STDERR_003
KNAX7901-I The adshexec command will wait for all asynchronous processes at the end of the
job.
KNAX0724-I The job ID was assigned. job ID=000492
SUB2_JOB_STDERR_001
KNAX6030-E The directory cannot be changed. directory="/home/user001/dir/xxxSUB2" details=No
such file or directory filename="/home/user001/samplesub2.ash" line=4
KNAX6521-E The command cd (line=4) failed. exit status=1 execution time=0.000s CPU
time=0.000s
SUB2_STEP_STDERR_001
KNAX6030-E The directory cannot be changed. directory="/home/user001/dir/xxxSUB2" details=No
such file or directory filename="/home/user001/samplesub2.ash" line=9
KNAX6521-E The command cd (line=9) failed. exit status=1 execution time=0.000s CPU
time=0.000s
KNAX6597-I SAMPLE_SUB2.SUB2_S1 step succeeded. exit status=0 execution time=0.001s CPU
time=0.000s
SUB2_JOB_STDERR_002
SUB2_STEP_STDERR_002
KNAX6597-I SAMPLE_SUB2.SUB2_S2 step succeeded. exit status=0 execution time=0.000s CPU
time=0.000s
KNAX6030-E The directory cannot be changed. directory="/home/user001/dir/xxxSUB2" details=No
such file or directory filename="/home/user001/samplesub2.ash" line=17
KNAX6521-E The command cd (line=17) failed. exit status=1 execution time=0.000s CPU
time=0.000s
SUB2_JOB_STDERR_003
KNAX0101-E SAMPLE_SUB2 An error occurred during execution of the job.
KNAX0101-E SAMPLEJOB An error occurred during execution of the job.
KNAX0098-I SAMPLEJOB The job ended. exit status=0 execution time=0.042s CPU time=0.030s
```

(Continued)

(Continued)

STDERR output

```
******** JOBSTEP OUTPUT ********
KNAX0719-I STEP. step number=0001 step name=S1 output destination=STDERR
STEP_STDERR_001
KNAX6030-E The directory cannot be changed. directory="/home/user001/dir/xxx" details=No
such file or directory filename="/home/user001/sample.ash" line=9
KNAX6521-E The command cd (line=9) failed. exit status=1 execution time=0.000s CPU
time=0.000s

KNAX0719-I STEP. step number=0002 step name=S2 output destination=STDERR
STEP_STDERR_002
KNAX7901-I The adshexec command will wait for all asynchronous processes at the end of the
job.
KNAX0724-I The job ID was assigned. job ID=000491
SUB1_JOB_STDERR_001
KNAX6030-E The directory cannot be changed. directory="/home/user001/dir/xxxSUB1" details=No
such file or directory filename="/home/user001/samplesub1.ash" line=4
KNAX6521-E The command cd (line=4) failed. exit status=1 execution time=0.000s CPU
time=0.000s
SUB1_STEP_STDERR_001
KNAX6030-E The directory cannot be changed. directory="/home/user001/dir/xxxSUB1" details=No
such file or directory filename="/home/user001/samplesub1.ash" line=9
KNAX6521-E The command cd (line=9) failed. exit status=1 execution time=0.000s CPU
time=0.000s
KNAX6597-I SAMPLE_SUB1.SUB1_S1 step succeeded. exit status=0 execution time=0.001s CPU
time=0.000s
SUB1_JOB_STDERR_002
SUB1_STEP_STDERR_002
KNAX6597-I SAMPLE_SUB1.SUB1_S2 step succeeded. exit status=0 execution time=0.000s CPU
time=0.000s
KNAX6030-E The directory cannot be changed. directory="/home/user001/dir/xxxSUB1" details=No
such file or directory filename="/home/user001/samplesub1.ash" line=17
KNAX6521-E The command cd (line=17) failed. exit status=1 execution time=0.000s CPU
time=0.000s
SUB1_JOB_STDERR_003
KNAX0101-E SAMPLE_SUB1 An error occurred during execution of the job.

KNAX6380-I A job name will be added to the spool job directory of the root job. spool job
directory="/var/opt/jp1as/spool/000490-SAMPLEJOB/"
KNAX7999-I Advanced Shell ended. exit status=0
[user001@hosta user001]$
```

3.5 Outputting the executed commands and their arguments

When the `xtrace` shell option is enabled, the executed commands and their arguments are output to the standard error output as trace information.

There are three ways to enable the `xtrace` shell option:

- Executing the `set` command with the `-x` or `-o xtrace` option specified in job definition scripts
- Executing the `adshexec` command with the `-x` option specified during job execution
- Selecting **Enable xtrace** in JP1/Advanced Shell Editor's Runtime Environment Settings dialog box

The trace information is output in the following format:

- The value of the `PS4` shell variable is added at the beginning of the trace information.
- If the values of variables are referenced, the variable substitution results are output.
- If command arguments contain wildcard characters, the results of the wildcard replacement are output.

Example of trace information output

This example shows an executed job definition script and the trace information that is output.

Contents of the job definition script

```
set -o xtrace
typeset -i cnt=1
if [ $cnt -eq 1 ]
then
  echo "--- JOB START ---"
fi
date
```

Standard error output results

```
+ typeset -i cnt=1
+ [ 1 -eq 1 ]
+ echo --- JOB START ---
+ date
```

Notes about trace information

Even when the `xtrace` shell option is enabled, the following commands and their arguments are not output:

- Commands in the `[[ ]]` format that is the abbreviated form of the `test` command
- Extended script commands

Commands in the `(( ))` format that is the abbreviated form of the `let` command are replaced with the `let` command in the trace information. The following shows an example of a command in the `(( ))` format and the output information:

Contents of the job definition script

```
set -o xtrace
typeset -i a=0
(( a=(2+3)*9 ))
echo $a
```

Standard error output results

```
+ typeset -i a=0
+ let  a=(2+3)*9
+ echo 45
```

Trace information for functions themselves are output, but trace information for commands within the functions is not output. To output trace information for commands within a function, execute the `typeset` command to enable the function's trace mode. The following shows an example of the `typeset` command's specification and the output information:

Contents of the job definition script

```
set -o xtrace
fn1(){
  echo "call $1 $2"
  echo $LINENO
}
echo "in main"
fn1 "function" "1"
typeset -ft fn1
fn1 "function" "2"
```

Standard error output results

```
+ echo in main
+ fn1 function 1
+ typeset -ft fn1
+ fn1 function 2
+ echo call function 2
+ echo 4
```

When the `xtrace` shell option is enabled, trace information for the job definition script executed by the child job is output, but trace information for the commands within the child job is not output. To output trace information for a job definition script that is executed by a child job, also enable the `xtrace` shell option within the child job's job definition script.

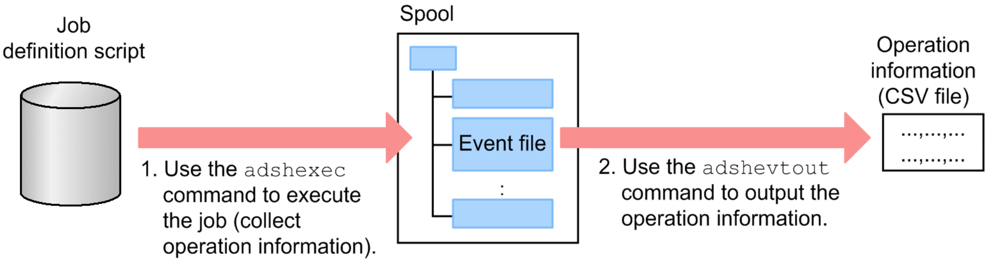
The input contents of here documents are not output as trace information even when the `xtrace` shell option is enabled.

3.6 Outputting job definition script operation information

Job definition script operation information includes the execution time of each command executed by the job, the CPU time, any output messages, and job step execution results. You can use this information to obtain the job execution status and investigate the causes of delays in job execution.

The following figure shows the general procedure for collecting and outputting job definition script operation information.

Figure 3–4: Collection and output of job definition script operation information


Job definition script
Spool
Event file
:
Operation information (CSV file)
...,...,...
...,...,...
1. Use the adshexec command to execute the job (collect operation information).
2. Use the adshevtout command to output the operation information.

1. When the user uses the adshexec command to execute a job, the adshexec command collects the job definition script operation information for the job and then outputs it to the event file on the spool.
2. The adshevtout command outputs the job definition script operation information that is contained in the event file to a CSV file.

   You can use a program such as a spreadsheet to analyze a CSV file that contains job definition script operation information.

3.6.1 Collecting job definition script operation information

The following table shows when job definition script operation information can be collected:

Environment	Execution status	Collection of job definition script operation information
Execution environment	Normal	Y
	Debugger mode	N
Development environment	(Not applicable)	N

Legend:

Y: Can be collected

N: Cannot be collected

In the normal status, when the adshexec command is used to execute a job, the adshexec command collects the job definition script operation information for the job and outputs it to the event file on the spool.

However, if the following specification is made in the environment file, the adshexec command does not output job definition script operation information to the event file:

```
#-adsh_conf EVENT_COLLECT NO
```

3.6.2 Outputting job definition script operation information

You use the adshevtout command to output job definition script operation information from the event file on the spool to a CSV file.

For details about how to specify the adshevtout command and an example output of job definition script operation information, see *adshevtout command (outputs job definition script operation information)* in *8. Commands Used During Operations*.

(1) Specifying a job whose job definition script operation information is to be output

You use the adshevtout command to specify a job whose job definition script operation information is to be output.

The following job information can be specified to select the job definition script operation information that is to be output:

- Range of job's execution start date and time
- JP1/AJS job name, job execution ID, and job number
- JP1/Advanced Shell job name, job ID, and path name of the job definition script file

If multiple conditions are specified, the information that satisfies all the specified conditions is output.

If no conditions are specified, the job definition script operation information for all jobs on the spool is output (except that job definition script operation information located in inaccessible event files will not be output).

(2) Controlling the job definition script operation information that is to be output

You can use the adshevtout command to control which information will be output, such as the following:

- Suppress output of header information
- Output only the header information (do not output the job definition script operation information itself)
- Output only the messages contained in the job definition script operation information
- Do not output information about environment variables in the job definition script operation information

(3) Spool to be referenced

The adshevtout command references the event files on the spool to output job definition script operation information.

The spool to be referenced by the adshevtout command is determined by the specified environment file in the same manner as for the adshexec command.

If a logical host is specified in the adshevtout command, the command uses the spool corresponding to the specified logical host in the same manner as for the adshexec command.

(4) Output destination of the job definition script operation information

The job definition script operation information is output to the adshevtout command's standard output (stdout).

If you use the redirect function, you can also output the operation information to a file.

3.6.3 Relationship between dates and times and time zones in the operation information

The adshevtout command interprets and outputs the date and the time in the formats *year−month−date* and *hour−minute−second* for the time zone in effect when the command is executed. Use the TZ environment variable to specify the time zone.

As shown in the three sets of examples in the following table, a given date and time (in the *year−month−date* and *hour−minute−second* formats) will be represented differently depending on the time zone:

Time zone	Date and time example 1	Date and time example 2	Date and time example 3
UTC-2	2012-06-10 08:00:00	2012-06-10 23:00:00	2012-06-11 15:00:00
UTC	2012-06-10 10:00:00	2012-06-11 01:00:00	2012-06-11 17:00:00
UTC+3	2012-06-10 13:00:00	2012-06-11 04:00:00	2012-06-11 20:00:00
UTC+9	2012-06-10 19:00:00	2012-06-11 10:00:00	2012-06-12 02:00:00

Remarks:

UTC+9 indicates the time zone that is nine hours ahead of coordinated universal time (UTC). The sign differs from when the value is set in the TZ environment variable.

3.6.4 Using multiple OR conditions for output of job definition script operation information

The adshevtout command outputs the job definition script operation information for the job that satisfies all the conditions specified in the arguments.

If you want to output the information that satisfies any one of multiple conditions, execute as many adshevtout commands as there are *OR* conditions, and then output the concatenated job definition script operation information to a single CSV file.

The following example outputs concatenated job definition script operation information to the outfile file.

```
adshevtout  -d                                   >  outfile
adshevtout  -t  option-specifying-condition-1       >> outfile
adshevtout  -t  option-specifying-condition-2       >> outfile
     :
adshevtout  -t  option-specifying-condition-n       >> outfile
```

- The first adshevtout command outputs only the header line (-d suppresses output of job definition script operation information).
- The second adshevtout command outputs the operation information for the job that satisfies *condition-1* without a header line (by specifying -t).
- The third adshevtout command outputs the operation information for the job that satisfies *condition-2* without a header line (by specifying -t).
- The adshevtout command *n* + 1 outputs the operation information for the job that satisfies *condition-n* without a header line (by specifying -t).

You output operation information that satisfies multiple *OR* conditions by executing the adshevtout command in this manner.

3.6.5 Outputting job definition script operation information from different spools

The job definition script operation information that can be output by executing the adshevtout command typically is for the job located on the spool that was specified in the environment file when the adshevtout command was executed. To output the job definition script operation information for a job located on another spool, you must use the corresponding environment file.

An example is shown below. In this example, a different spool root directory is specified in each environment file.

Example:

Environment file envfile1: Specifies spool root directory spooldir1.

Environment file envfile2: Specifies spool root directory spooldir2.

Environment file envfile3: Specifies spool root directory spooldir3.

To output the job definition script operation information for a job in each of these spool root directories, execute the adshevtout command with the correct environment file specified as follows:

```
export ADSH_ENV=envfile1
adshevtout

export ADSH_ENV=envfile2
adshevtout

export ADSH_ENV=envfile3
adshevtout
```

3.6.6 Format of operation information

The adshevtout command outputs operation information to a CSV file.

(1) Types of operation information

The operation information that is output by the adshevtout command consists mainly of the items listed below. Each item forms one record.

- When the adshexec command's execution began
- Environment variables
- Commands
- Messages
- Start of job step execution
- End of job step execution
- Skipped job step execution

- End of job execution

(2) Configuration of operation information

The operation information items are output in the following order for each job:

Order	Output item	Remarks
1	Header information	--
2	Start of the `adshexec` command's execution	--
3	Environment variables	Output if there are any environment variables
4	Commands Messages Start of job step execution End of job step execution Skipped job step execution End of job execution	Output according to the execution of the job definition script

Legend:

--: Not applicable

The order in which messages and the following records are output might differ:

- Commands
- Start of job step execution
- End of job step execution
- Skipped job step execution
- End of job execution

The order in which the following operation information items are output is not predefined:

- Output order among spool jobs
- Output order of the root and child jobs in a spool job

(3) Configuration of operation information records

The operation information records consist of multiple items.

Each item value is enclosed in double quotation marks (`"`). If an item value contains a double quotation mark, that double quotation mark is represented as two consecutive double quotation marks.

The items are delimited by the comma (`,`).

3.6.7 Operation information records in CSV format and output items

The following table shows the relationship between the operation information records in CSV format and the output items.

Table 3–2: Operation information records and output items

No.	Item name	Job start	Env var	Cmd	Extd Cmd	Msg	Job step start	Job step end	Job step skipd	Job end
1	EvtName	Y	Y	Y	Y	Y	Y	Y	Y	Y
2	RecTZ	Y	Y	Y	Y	Y	Y	Y	Y	Y
3	RecTime	Y	Y	Y	Y	Y	Y	Y	Y	Y
4	PhysicalHostName	Y	Y	Y	Y	Y	Y	Y	Y	Y
5	LogicalHostName	YN	YN	YN	YN	YN	YN	YN	YN	YN
6	OsName	Y	Y	Y	Y	Y	Y	Y	Y	Y
7	Jp1ajsService	YN	YN	YN	YN	YN	YN	YN	YN	YN
8	Jp1ajsRootJobnet	YN	YN	YN	YN	YN	YN	YN	YN	YN
9	Jp1ajsJobName	YN	YN	YN	YN	YN	YN	YN	YN	YN
10	Jp1ajsExecId	YN	YN	YN	YN	YN	YN	YN	YN	YN
11	Jp1ajsJobId	YN	YN	YN	YN	YN	YN	YN	YN	YN
12	Jp1asJobName	Y	Y	Y	Y	Y	Y	Y	Y	Y
13	Jp1asJobId	Y	Y	Y	Y	Y	Y	Y	Y	Y
14	Jp1asJobTime	Y	Y	Y	Y	Y	Y	Y	Y	Y
15	Jp1asJobPid	Y	Y	Y	Y	Y	Y	Y	Y	Y
16	Jp1asUid	Y	Y	Y	Y	Y	Y	Y	Y	Y
17	Jp1asGid	Y	Y	Y	Y	Y	Y	Y	Y	Y
18	Jp1asUserName	Y	Y	Y	Y	Y	Y	Y	Y	Y
19	Jp1asGroupName	Y	Y	Y	Y	Y	Y	Y	Y	Y
20	Jp1asScriptPath	Y	Y	Y	Y	Y	Y	Y	Y	Y
21	Jp1asEnvPath	Y	Y	Y	Y	Y	Y	Y	Y	Y
22	Jp1asSpoolPath	Y	Y	Y	Y	Y	Y	Y	Y	Y
23	Jp1asJobLang	Y	Y	Y	Y	Y	Y	Y	Y	Y
24	Jp1asJobEncode	Y	Y	Y	Y	Y	Y	Y	Y	Y
25	RecTZExec	Y	Y	Y	Y	Y	Y	Y	Y	Y
26	Jp1asJobParentJobName	YN	YN	YN	YN	YN	YN	YN	YN	YN
27	Jp1asJobParentJobId	YN	YN	YN	YN	YN	YN	YN	YN	YN
28	Jp1asJobParentJobTime	YN	YN	YN	YN	YN	YN	YN	YN	YN
29	Jp1asJobParentJobPid	YN	YN	YN	YN	YN	YN	YN	YN	YN
30	Jp1asJobRc	N	N	N	N	N	N	N	N	Y
31	Jp1asJobSig	N	N	N	N	N	N	N	N	Y
32	EnvVar	N	Y	N	N	N	N	N	N	N
33	Jp1asScriptLineNo	N	N	Y	Y	N	Y	Y	Y	N

No.	Item name	Job start	Env var	Cmd	Extd Cmd	Msg	Job step start	Job step end	Job step skipd	Job end
34	`Jp1asStepSeq`	N	N	Y	N	N	Y	Y	Y	N
35	`Jp1asStepName`	N	N	YN	N	N	YN	YN	YN	N
36	`Jp1asStepSkip`	N	N	N	N	N	N	N	Y	N
37	`Jp1asStepRc`	N	N	N	N	N	N	Y	N	Y
38	`Jp1asStepSig`	N	N	N	N	N	N	Y	N	Y
39	`Jp1asCmdExec`	N	N	Y	Y	N	N	N	N	N
40	`Jp1asCmdType`	N	N	Y	Y	N	N	N	N	N
41	`Jp1asCmdPath`	N	N	YN	Y	N	N	N	N	N
42	`Jp1asCmdArg`	N	N	YN	YN	N	N	N	N	N
43	`Jp1asLang`	N	N	Y	Y	N	N	N	N	N
44	`Jp1asCharEncode`	N	N	Y	Y	N	N	N	N	N
45	`Jp1asCmdStart`	N	N	Y	N	N	N	N	N	N
46	`Jp1asCmdEnd`	N	N	Y	N	N	N	N	N	N
47	`Jp1asCmdElaps`	N	N	Y	N	N	N	N	N	N
48	`Jp1asCmdRc`	N	N	Y	N	N	N	N	N	N
49	`Jp1asCmdSig`	N	N	Y	N	N	N	N	N	N
50	`Jp1asCmdPid`	N	N	YN	N	N	N	N	N	N
51	`Jp1asCmdCpuUser`	N	N	YN	N	N	N	N	N	N
52	`Jp1asCmdCpuSys`	N	N	YN	N	N	N	N	N	N
53	`Reserved1`	N	N	N	N	N	N	N	N	N
54	`Reserved2`	N	N	N	N	N	N	N	N	N
55	`Jp1asMsgId`	N	N	N	N	Y	N	N	N	N
56	`Jp1asMsgText`	N	N	N	N	Y	N	N	N	N
57	`Jp1asMsgLang`	N	N	N	N	Y	N	N	N	N
58	`Jp1asMsgEncode`	N	N	N	N	Y	N	N	N	N

Legend:

Job start: Start of `adshexec` command execution

Env var: Environment variables

Cmd: Commands

Extd cmd: Commands (extended script commands)

Msg: Messages

Job step start: Start of job step execution

Job step end: End of job step execution

Job step skipd: Skipped job step execution

Job end: End of job execution

Y: Information is output.

YN: Information is output, but if there is no information to be output, the null character string is set.

N: There is no information to be output (the null character string is set).

3.6.8 Output items for operation information in CSV format

The following table lists and describes the output items (columns) for operation information in CSV format.

Table 3–3: Output items (columns) for operation information

No.	Item name	Description
1	`EvtName`	Name indicating the type of operation information (record): • `adshStart`: Start of `adshexec` command execution • `envVar`: Environment variable • `command`: Command • `message`: Message • `stepStart`: Start of job step execution • `stepEnd`: End of job step execution • `stepSkip`: Skipped job step execution • `jobEnd`: End of job execution
2	`RecTZ`	Time zone in effect when the `adshevtout` command was executed.# This is the time zone that is used to represent the date and time information in the operation information.
3	`RecTime`	Date and time the operation information was recorded#
4	`PhysicalHostName`	Physical host name
5	`LogicalHostName`	Logical host name. If a logical host is not used, the null character string is set.
6	`OsName`	OS name. One of the following character strings: • `Windows` • `Linux` • `AIX` • `HP-UX` • `Solaris`
7	`Jp1ajsService`	JP1/AJS scheduler service name. This is the value set in the `AJS_AJSCONF` environment variable by JP1/AJS. This information is output when the job was started from JP1/AJS.
8	`Jp1ajsRootJobnet`	JP1/AJS's root jobnet name. This is the value set in the `AJSNETNAME` environment variable by JP1/AJS. This information is output if the job was started from JP1/AJS.
9	`Jp1ajsJobName`	JP1/AJS's job name. This is the value set in the `AJSJOBNAME` environment variable by JP1/AJS. This information is output when the job was started from JP1/AJS.
10	`Jp1ajsExecId`	JP1/AJS's job execution ID. This is the value set in the `AJSEXECID` environment variable by JP1/AJS. This information is output when the job was started from JP1/AJS.
11	`Jp1ajsJobId`	JP1/AJS's job number. This is the value set in the `JP1JobID` environment variable by JP1/AJS. This information is output when the job was started from JP1/AJS.
12	`Jp1asJobName`	JP1/Advanced Shell's job name
13	`Jp1asJobId`	JP1/Advanced Shell's job ID

No.	Item name	Description
14	`Jp1asJobTime`	`adshexec` command execution start date and time#
15	`Jp1asJobPid`	Process ID of the `adshexec` command
16	`Jp1asUid`	User ID of the `adshexec` command's process. In Windows, the null character string is set.
17	`Jp1asGid`	Group ID of the `adshexec` command's process. In Windows, the null character string is set.
18	`Jp1asUserName`	User name of the `adshexec` command's process.
19	`Jp1asGroupName`	Group name of the `adshexec` command's process. In Windows, the null character string is set.
20	`Jp1asScriptPath`	Path name of the job definition script#
21	`Jp1asEnvPath`	Path name of the environment file#
22	`Jp1asSpoolPath`	Path name of spool directory.# A spool job's directory name is *job-ID_job-name* once the job has terminated, but this item outputs only *job-ID*.
23	`Jp1asJobLang`	Value of the `LANG` environment variable when the `adshexec` command's execution started.
24	`Jp1asJobEncode`	Character encoding when the `adshexec` command's execution started.#
25	`RecTZExec`	Time zone in effect when the `adshexec` command was executing.# This time zone is not used to represent the date and time information in the operation information.
26	`Jp1asJobParentJobName`	Name of the parent job. For a root job, the null character string is set.
27	`Jp1asJobParentJobId`	Job ID of the parent job. For a root job, the null character string is set.
28	`Jp1asJobParentJobTime`	Parent job execution start date and time.# For a root job, the null character string is set.
29	`Jp1asJobParentJobPid`	Process ID of the parent job. For a root job, the null character string is placed.
30	`Jp1asJobRc`	`adshexec` command's return code
31	`Jp1asJobSig`	Signal number if the `adshexec` command terminated with a signal. If the command did not terminate with a signal, `0` is set. In Windows, the null character string is set.
32	`EnvVar`	Environment variables used when the `adshexec` command started. The format is *environment-variable-name*=*value*. For details, see *(1) Environment variables in the EnvVar item* below.
33	`Jp1asScriptLineNo`	Line number in the job definition script. For details, see *(2) Line number in the Jp1asScriptLineNo item* below.
34	`Jp1asStepSeq`	Job step number. If this is not a step, the null character string is set.
35	`Jp1asStepName`	Job step name. If the job step name is omitted or this is not a step, the null character string is set.

No.	Item name	Description
36	JP1asStepSkip	The preceding command or job step's status when the job step was skipped: - normal: Skipped because the status was normal. - abnormal: Skipped because an error resulting in termination occurred.
37	JP1asStepRc	Job step's return code.
38	JP1asStepSig	Signal number if the job step terminated with a signal. If the job step did not terminate with a signal, 0 is set. In Windows, the null character string is set.
39	JP1asCmdExec	Command execution mode: - *null-character-string*: Foreground execution. - back: Background execution. For details, see *(3) Command execution mode in the JP1asCmdExec item* below.
40	JP1asCmdType	Command type: - assign: Standard shell command that updates a shell variable. - embStd: Standard shell command or extended shell command. - embAdsh: Extended script command. - extCmd: External command.
41	JP1asCmdPath	Command name. For an external command, the command's path name.# For details, see *(4) Command name and command's path name in the JP1asCmdPath item* below.
42	JP1asCmdArg	Command arguments. The arguments are delimited by the space. These are not the arguments as specified on the command line or in the job definition script, rather they are the character strings obtained by evaluating (expanding) the arguments specified on the command line or in the job definition script and that are passed to the command.
43	JP1asLang	Value of the LANG environment variable during command execution
44	JP1asCharEncode	Character encoding during command execution#
45	JP1asCmdStart	Command execution start date and time#
46	JP1asCmdEnd	Command execution termination date and time#
47	JP1asCmdElaps	Duration of command execution. This is the difference between the command's execution start date and time and its termination date and time (microseconds).
48	JP1asCmdRc	Command's return code
49	JP1asCmdSig	Signal number if the command terminated with a signal. If the command did not terminate with a signal, 0 is set. In Windows, the null character string is set.
50	JP1asCmdPid	Process ID of an external command
51	JP1asCmdCpuUser	Command's user CPU time# (microseconds)
52	JP1asCmdCpuSys	Command's system CPU time# (microseconds)
53	Reserved1	Reserved item (null character string)
54	Reserved2	Reserved item (null character string)
55	JP1asMsgId	Message ID

No.	Item name	Description
56	`Jp1asMsgText`	Message text
57	`Jp1asMsgLang`	Value of the `LANG` environment variable when the message was output
58	`Jp1asMsgEncode`	Character encoding when the message was output#

#

The following table explains the output items.

Item	Output contents
End-of-line code	An end-of-line code contained in the value of an item is replaced with a space. Specifically, each of the character codes `LF(0x0A)` and `CR(0x0D)` is replaced with a space (`0x20`). For a new line in Windows, a pair of `CR(0x0D)` and `LF(0x0A)` is replaced with two bytes of spaces (`0x20`).
Date and time	The date and time are represented in the following format *YYYY*-*MM*-*DD* *hh*:*mm*:*ss*.*nnn* ● *YYYY*: Calendar year ● *MM*: Month ● *DD*: Date ● *hh*: Hour ● *mm*: Minute ● *ss*: Second ● *nnn*: Millisecond The date and time when the `adshevtout` command is executed are represented based on the time zone specified in the `TZ` environment variable. If acquisition of date and time information failed due to an error in the time acquisition function at the time the operation information was collected by the `adshexec` command, the date and time are represented as follows: `EEEE-EE-EE ee:ee:ee.eee`
Time zone	The difference from UTC is indicated in one of the formats shown below. The sign differs from the setting of the `TZ` environment variable. `+`*hh*`:`*mm*`:`*ss* `-`*hh*`:`*mm*`:`*ss* ● *hh*: Hour ● *mm*: Minute ● *ss*: Second If the time difference is 0, the value is `+00:00:00`. Example: The time zone that is nine hours ahead of coordinated universal time (UTC) is `+09:00:00`. If the time zone cannot be converted to the format shown above, the value in this item is `+EE:EE:EE`.
Character encoding	• In Windows This is always SJIS regardless of the actual character string encoding used. • In Linux, AIX, HP-UX, and Solaris In Japanese, the character encoding determined from the value of the `LANG` environment variable is set as follows: ● `SJIS`: Shift JIS code ● `EUC`: Extended Unix Code

Item	Output contents
Character encoding	● UTF8: UTF-8 In all other cases, the following value is set: ● OTHER(*LANG-environment-variable-value*)
Path name	Any number of consecutive directory separators in path names are output as is. Consecutive directory separators are not replaced with a single directory separator. For example, if the specified path name is "C:\\dir", "C:\\dir" is output.
CPU time	For external commands: CPU time of the process that executed the external command For other commands: The CPU time of the process of adshexec that executed the command. This is not the CPU time when each command was executed.

(1) Environment variables in the EnvVar item

The following environment variables are output

- The environment variables that were specified at the time the job started are output.
- If environment variables are specified in the environment file, the environment variables to which the settings in the environment file have been applied are output.
- The environment variables set by the adshexec command are output.
- The environment variables deleted by the adshexec command are not output.
- Value of the ADSH_JOB_NAME environment variable is the standard job name ADSH*xxxxxx* (*xxxxxx*: job ID). This is not the job name specified in #-adsh_job.
- The above also apply to child jobs.

(2) Line number in the Jp1asScriptLineNo item

The value of this item is the null character string in the following cases:

- A command whose action was specified in the trap command was executed.
- The #-adsh_script command was executed.
- A command specified by command substitution was executed.

If a function is executed, the line number assigned to the command that is defined in the body of the function is output.

(3) Command execution mode in the Jp1asCmdExec item

If cmd1 | cmd2 | cmd3 is executed, the item value is back because cmd1 and cmd2 are executed in separate processes.

(4) Command name and command's path name in the Jp1asCmdPath item

(a) If the command is executed in another process

If the command is executed in another process, the item value is Another process script as shown in the following example:

Example:

```
(echo1 a ; echo2 b)
(echo1 a) &
(echo1 a ; echo2 b) &
{ echo1 a ; echo2 b ; } &
{ echo1 a ; echo2 b ; } |&
```

In the following example, the value of the part `{ echo1 a ; echo2 b ; }` becomes `Another process script`.

```
{ echo1 a ; echo2 b ; } | echo3 c
```

(b) If the command is executed by using a pipe

If `cmd1 | cmd2 | cmd3` is executed and `cmd1`, `cmd2`, and `cmd3` are all external commands, the item values become as follows:

- `cmd1`: *command-name* (*file-name*)
- `cmd2`: *command-name* (*file-name*)
- `cmd3`: *path-name*

3.6.9 Job definition script operation information that is output

An example of job definition script operation information that is output is shown below. In this example, the header line is output on the first line.

```
"EvtName","RecTZ","RecTime","PhysicalHostName","LogicalHostName", ---
"adshStart","+09:00:00","2012-07-12 12:23:15.381","HOST01","", ---
"envVar","+09:00:00","2012-07-12 12:23:15.382","HOST01","", ---
    : (job definition script operation information that is output)
```

3.7 Using the user-reply functionality

The user-reply functionality notifies JP1/IM of batch job information by issuing JP1 events and enables replies to be sent. By using this functionality, you can notify the operator by issuing character strings even in an environment where a window used as the standard input and output is not available, such as when batch jobs are started from JP1/AJS.

The operator can monitor the JP1 events issued during command execution from a JP1/IM - View that is connected to JP1/IM - Manager. Because JP1/IM - View's integrated console can display multiple servers' JP1 events, the operator can monitor several servers from a single location.

3.7.1 Prerequisites

For details about the required programs, including JP1/IM, see *2.2.2 Programs required in each environment*. You must also perform the environment setup described in *2.8 Setting up the user-reply functionality*.

3.7.2 Execution method

You use the commands shown below to specify character strings that are to be issued as JP1 events by the user-reply functionality. You specify these commands in job definition scripts.

Command name	Usage	Section describing how to specify the command
adshecho	Issues an event notification message as a JP1 event.	*adshecho command (issues a specified event notification message as a JP1 event)* in *9.4 Extended shell commands*
adshread	Issues a reply-request message as a reply-waiting event.	*adshread command (issues a specified reply-request message as a reply-waiting event)* in *9.4 Extended shell commands*

An issued JP1 event is displayed in the JP1/IM - View window. The operator can enter a reply to a reply-request event from JP1/IM - View.

3.7.3 Relationship with JP1/IM - View

When the user-reply functionality is used, the event notification messages and reply-request messages specified in job definition scripts are output as JP1 events. You use JP1/IM - View to view event notification messages and reply-request messages output as JP1 events.

A reply-request message specified in the `adshread` command is treated as a reply-request event by JP1/IM - Manager. The reply-request events are accumulated and displayed in JP1/IM - View and the operator can enter replies to them from JP1/IM - View.

For details about how to enter replies, see the applicable JP1/IM manual. The following table lists and describes the statuses that are displayed in the Enter Replies window when replies are entered from JP1/IM - View.

Table 3–4: Statuses displayed in JP1/IM - View's Enter Replies window

Status	Meaning	Whether a reply can be entered from JP1/IM - View
READY TO RESPOND	A reply can be entered.	Y

Status	Meaning	Whether a reply can be entered from JP1/IM - View
NO LONGER MANAGED BY JP1/AS	A reply cannot be entered because the reply-request message is not managed by JP1/Advanced Shell.	N
RESPONDED SUCCESSFULLY	A reply entered to the reply-request message was successful.	N
ALREADY RESPONDED	A reply to the reply-request message has already been entered.	N
INTERNAL ERROR	A reply cannot be entered because an internal error has occurred in JP1/Advanced Shell.	N

Legend:

Y: A reply can be entered.

N: A reply cannot be entered.

3.7.4 How to specify the standard input and output as the input source and output destination of the user-reply functionality

Because the adshecho and adshread commands issue JP1 events, these commands terminate with an error if they are used in an environment that does not have JP1/Base or JP1/IM, such as when job definition scripts are debugged. To enable the user to debug job definition scripts in such cases, the user-reply functionality provides the following functions:

- Function to output character strings to the standard output, rather than to JP1/IM - View
- Function to enable replies to be entered from the standard input, rather than from JP1/IM - View

You use the USERREPLY_DEBUG_DESTINATION parameter to specify whether JP1 events are to be issued or the standard input and output are to be used when the adshecho and adshread commands are used. Specify the USERREPLY_DEBUG_DESTINATION parameter in the system environment file or the job environment file. You can also set the input source and output destination of character strings to the standard input and output by specifying the -d option in the adshecho and adshread commands.

The function for setting the input source and output destination for the user-reply functionality to the standard input and output is enabled only during debugging (performed by using adshexec -d in UNIX and in the development environment in Windows). When you use this function, the JP1/Base, JP1/Integrated Management - Manager, and JP1/Integrated Management - View programs that are related to the user-reply functionality are not needed. There is also no need to start the user-reply functionality's management daemon or service.

Note that the following parameters are ignored:

- HOSTNAME_JP1IM_MANAGER parameter
- USERREPLY_JP1EVENT_INTERVAL parameter
- USERREPLY_WAIT_MAXCOUNT parameter

3.7.5 How to handle adshecho and adshread commands that terminate with an error

If the `adshecho` or `adshread` command terminates with an error, its re-execution might result in successful command processing. To re-execute the `adshecho` or `adshread` command, create a job definition script for re-executing the command by referencing the following example:

```
#! /opt/jp1as/bin/adshexec

###
#Function executing the adshread command
#Argument: Reply-request message
#Return code: 0 (normal termination)
#        1 (terminated with a retryable error)
#        2 (terminated with a non-retryable error)
###

func_adshread()
{
    adshread ans "$1"

  case "$?" in
    # Normal termination
    0 )             return 0 ;;
    # When terminated with a retryable error
    3 | 4 | 6 | 8 ) return 1 ;;
    # When terminated with any other error
    * )             return 2 ;;
  esac
}

###
### Body of script
###

#
# Specify a process that is to be executed as a job
#

###
### Wait for the operator's reply before resuming processing
###

  while :
  do

    #Call the function that executes the adshread command
    func_adshread "Do you want to resume processing?(Y/N) [host name:
$HOSTNAME, script name: $0]"

    if [ $? = 0 ]; then       #The adshread command terminated normally
      break                   #Exit the loop
    elif [ $? = 1 ] ; then    #The adshread command terminated with a
retryable error
      continue                #Re-execute the adshread command
    else                      #The adshread command terminated with a non-
retryable error
```

```
      echo "The adshread command terminated with an error."
      exit 1                  #Terminate the script
    fi
  done

###
### Perform processing according to the reply received by adshread
###

  if [ "$ans" = "Y" ] ; then
    adshecho "Y was entered. The processing will be resumed."
  elif [ "$ans" = "N" ] ; then
    adshecho "N was entered. The processing will be terminated."
    exit 1                       #Terminate the script
  else
    adshecho "An invalid reply was entered. The processing will be
terminated."
    exit 1                       #Terminate the script
  fi
```

3.7.6 Notes

- You can enter replies only from a JP1/IM - View that is connected to the JP1/IM - Manager specified in the `HOSTNAME_JP1IM_MANAGER` parameter.
- Because there is a limit to the number of reply-waiting events that can be accumulated by JP1/IM - Manager, design your operations in such a manner that this limitation is observed.
- The character codes that can be entered from JP1/IM - View are those within the range of the ASCII character codes (excluding the control characters). If a character code outside this range is entered, an error message will be displayed. In such a case, re-enter a character code within the permitted range.
- When the `adshecho` and `adshread` commands are executed, JP1/Base closes a TCP/IP connection with the host specified in the `HOSTNAME_JP1IM_MANAGER` parameter and then establishes connection using a new port.

  The port that was being used becomes unavailable for a period equal to the OS's maximum segment lifetime (MSL) × 2 (seconds). If the value of MSL is large or there are only a few ports, a shortage of ports might occur.

  For this reason, set MSL, the number of JP1 events to be output during a period of MSL × 2 (seconds), and the number of ports to satisfy the following condition:

  $n \times \text{MSL} \times 2 \div 3 <$ *number-of-ports*

  n: Number of JP1 events that can be output by the user-reply functionality during a period of MSL × 2 (seconds)
- In UNIX, when the user-reply functionality's management daemon starts, a file with the following name is created in the spool directory to manage reply-request messages in the shared memory:
  - `.adsh_mqueue`
  - `.adsh_mqueue_`*logical-host-name* (applicable when the user-reply functionality's management daemon is started on a logical host)

  Do not delete this file while the user-reply functionality's management daemon is running. When the user-reply functionality's management daemon terminates, this file remains and can be reused the next time the user-reply functionality's management daemon starts.

  In HP-UX, in addition to the above file, a file with the following name is also created in the spool directory:
  - `.adsh_mqueueS`

    - `.adsh_mqueue_`*`logical-host-name`*`S` (applicable when the user-reply functionality's management daemon is started on a logical host)

  Do not delete this file while the user-reply functionality's management daemon is running. The file is deleted when the user-reply functionality's management daemon terminates.

- Before you shut down the OS, terminate the user-reply functionality's management daemon or service. If there are reply-request messages still waiting for replies when the user-reply functionality's management daemon or service stops, the daemon or service will cancel these reply-request messages and then stop. If OS shutdown processing is performed, the OS might be shut down before the reply-request messages are cancelled. If this happens, the accumulated reply-waiting events will remain in JP1/IM - View.

3.8 Deleting spool jobs

Spool jobs stored on a spool increase in size while they are stored in a spool directory. Therefore, make sure that you periodically delete old spool jobs to free up disk space.

- How to delete spool jobs

  To delete spool jobs, enter the `adshhk` command below. For details about how to specify the `adshhk` command, see *adshhk command (deletes spool jobs)*.

  ```
  adshhk target-list-file-name report-file-name log-file-name [number-of-days]
  ```

  Before you execute the `adshhk` command, specify in the file indicated as *target-list-file-name* the necessary information, including the name of the spool directory that contains the spool jobs to be deleted.

  The `adshhk` command's execution results are output to the file whose name is specified in *report-file-name*. The execution results are also output to trace logs.

  Error messages are output to the file whose name is specified in *log-file-name*.

  Spool jobs that have existed for more than the number of days specified in *number-of-days* are deleted. For example, if you specify 2, spool jobs that have existed for 2 or more days are deleted.

- Report file created by the `adshhk` command

  When the `adshhk` command has executed, the execution results are output to a report file. In the following example report file, the header information is output on the first line:

  ```
  "jobid","jobname","rc","start date","end
  date","act","info","spool","target days","execute date"
  "000056","JOB001","1","2011/06/13 09:03:31","2011/06/13
  09:03:31","delete","","C:\Documents and Settings\All Users\Documents
  \Hitachi\jp1as\jp1ase\spool","15","2011/06/30 18:19:58"
            :
  ```

  Legend:

  The first line of the execution results contains the headers listed below. The subsequent lines display the values corresponding to the header items.

Header	Meaning
`jobid`	Job ID
`jobname`	Job name
`rc`	Job's return code
`start date`	Job's execution start date and time (in the format *yyyy*/*mm*/*dd* *hh*:*mm*:*ss*). The spool allocated to the debugger itself when the job was started in the debugger mode is used to output the debugger's logs, not the job execution results. Therefore, the debugging start date and time are output.
`end date`	Job's termination date and time (in the format *yyyy*/*mm*/*dd* *hh*:*mm*:*ss*). The spool allocated to the debugger itself when the job was started in the debugger mode is used to output the debugger's logs, not the job execution results. Therefore, the job termination date and time are not output.
`act`	Applied action (`keep`: save, `delete`: delete, `error`: an error occurred during deletion processing)
`info`	Detailed error information
`spool`	Spool directory
`target days`	Target days

Header	Meaning
`execute date`	Command execution start date and time (in the format *yyyy*/*mm*/*dd* *hh*:*mm*:*ss*)

Important note

If the spool directory contains any user-specified file or directory that was not created by JP1/Advanced Shell, the `adshhk` command outputs a message such as `KNAX4419-E`, and then terminates.

3.9 Acquiring coverage information

If the user specifies during job execution that coverage information is to be acquired, JP1/Advanced Shell records in a coverage information file (`asc` file) such information as whether commands in the job definition script were executed.

You can use the coverage information manipulation commands to merge and display coverage information.

3.9.1 Overview of coverage information

Coverage information consists of indicators used to determine the coverage of tests of programs. The two types of coverage information indicators are the C0 information and the C1 information.

(1) C0 information and C1 information

The following explains the C0 and C1 information.

- C0 (statement coverage information): Command coverage

  This indicator determines the percentage of the commands in the tested job definition script that were executed. It is calculated as follows:

  C0 = *number of executed commands* ÷ *total number of commands* × 100 (%)

- C1 (branch coverage information): Branch coverage

  This indicator determines the percentage of the branches in the tested job definition script that were executed. It is calculated as follows:

  C1 = *number of executed branches* ÷ *total number of branches that could be executed* × 100 (%)

(2) Usage of coverage information

You can use the coverage information as reference data when you test your job definition scripts. You can also accumulate, display, and merge coverage information. For details about the coverage information that is acquired, see *A. Coverage Information That Is Acquired*.

The differences between Windows and UNIX in how coverage information is collected, displayed, and merged, are described in the following table.

Table 3–5: Differences between Windows and UNIX in how coverage information is collected, displayed, and merged

Environment	Collecting coverage information	Displaying coverage information	Merging coverage information
Windows development environment	Use JP1/Advanced Shell Editor's debugging function to collect coverage information. The collected coverage information is accumulated in a coverage information file. Coverage information is not collected unless **Accumulate coverage information** is selected during execution environment setup. The coverage auto-acquisition functionality is not available.	Either of the following methods can be used: • `adshcvshow` command • JP1/Advanced Shell Editor	The `adshcvmerg` command# is used to merge coverage information. An editor cannot be used to merge coverage information.
Windows execution environment	When a job definition script is run, use the `adshexec` command with the coverage accumulation option (`-t`) specified to collect coverage information. The collected	The `adshcvshow` command is used to	The `adshcvmerg` command# is used to

Environment	Collecting coverage information	Displaying coverage information	Merging coverage information
Windows execution environment	coverage information is accumulated in a coverage information file. The debugging function is not available.	display coverage information.	merge coverage information.
UNIX execution environment	Specify in the `adshexec` command one or both of the following options: • Coverage accumulation option (`-t`) to collect coverage information in a coverage information file. • Debugging option (`-d`) to collect coverage information in memory.	• If the coverage information was collected in a coverage information file, the `adshcvshow` command is used to display it. • If the coverage information was collected in memory, the `info coverage` command is used to display it.	The `adshcvmerg` command# is used to merge coverage information.

Note

The following notes apply to sharing coverage information between different platforms:

- Do not transfer coverage information between different platforms.

- A coverage information file created in one OS cannot be processed with the commands of a different OS.

#

You can use the `adshcvmerg` command to merge coverage information from two coverage information files at a time. To merge coverage information from three or more coverage information files, execute this command multiple times.

3.9.2 Managing coverage information

The coverage information files (`asc` files) are used to manage coverage information.

(1) File name and storage directory for coverage information files

(a) File name

Coverage information is collected for each job definition script and each user. Therefore, the default `asc` file name consists of a job definition script name and a user name.

In the execution environment, you can specify any `asc` file name by using a command option.

In the editor, the default `asc` file names are used. Non-default `asc` file names cannot be specified.

The following shows the default `asc` file name:

```
job-definition-script-name(without-the-extension)_user-name.asc
```

If the length of an `asc` file name exceeds the maximum length supported by the OS being used, collection of coverage information will fail. For this reason, you must pay attention to the number of characters in the file name of each job definition script that is to be executed.

(b) Storage directory

In the execution environment, the default asc files are created in the current directory during command execution.

In the development environment, if coverage information is accumulated from a Windows's editor, the asc files are created in the directory that contains the job definition script file.

(2) Updating asc files

The asc files are updated whenever coverage information is accumulated or merged.

An asc file cannot be shared among multiple users at the same time. If an attempt is made to use an asc file that is in use by another user, the command issues the KNAX6211-E message and results in an error.

(3) asc file output processing

If a coverage information write operation on an asc file fails for a reason such as insufficient disk capacity, that coverage information is lost.

To prevent loss of collected coverage information, if a specified asc file (including a file with the default asc file name when specification is omitted) already exists, JP1/Advanced Shell updates the asc file as described below. JP1/Advanced Shell does not directly update the specified asc file (including a file with the default asc file name when specification is omitted).

- Outputs new coverage information to a temporary asc file.
- Renames the existing asc file to be the backup asc file.
- Renames the temporary asc file to be the specified asc file (including a file with the default asc file name when specification is omitted).
- Deletes the backup asc file.

Therefore, if a process to output coverage information to an asc file terminates prematurely and the command is re-executed, JP1/Advanced Shell recovers the coverage information and then resumes command processing.

When command processing terminates successfully, there will be no temporary asc file or backup asc file. If a process to output coverage information to an asc file terminates prematurely, the temporary and backup asc files might remain. These files will be deleted if the command is re-executed or when the command is processed and terminates normally.

You can delete temporary asc files manually. Do not delete any backup asc file. If a backup asc file is deleted, the coverage information accumulated up to the point of a command's error termination will be lost.

Normally, when you use the coverage functionality, you need not know the above details.

You can determine whether a specified asc file (including a file with the default asc file name when specification is omitted) has been updated by new coverage information because the KNAX6242-I message will have been output during the previous command execution.

If the KNAX6242-I has been output, the contents of the specified asc file (including a file with the default asc file name when specification is omitted) have been updated by new coverage information.

(4) File names associated with asc files

When coverage information is collected in an asc file, a temporary file is created. This file is called a temporary coverage information file (temporary asc file). The file name, which always begins with a fixed character string, is as shown in the following:

Name of a temporary asc file

The file name always begins with a fixed character string as follows:

- For the adshexec command: adshexec_temp_*any-character-string*

Name of a backup asc file

When an asc file is renamed temporarily (backup asc file), the following file name is used:

- For the adshexec command: adshexec_backup_*any-character-string*

When you specify an asc file name (including the default asc file name) in an argument of a command, you must make sure that the specified file name does not exceed the maximum length (in bytes) permitted for a path name by the OS being used.

If processing is cancelled during command execution, temporary and backup asc files might remain. The remaining files are handled as follows:

- You can re-execute the command with the temporary asc file remaining in the system. Alternatively, you can manually delete the temporary asc file and then re-execute the command.
- If a backup asc file remains in the system, do not delete it. The adshexec command automatically restores the original asc file name from the remaining backup asc file name and then collects coverage information. Alternatively, you can manually restore the original asc file name and then re-execute the command.

Do not create a user file that has the same name as a temporary asc file or backup asc file. If a file with the same name as a temporary asc file for an asc file exists in the same directory, that user file will be deleted. If a file with the same name as a backup asc file for an asc file exists in the same directory, that user file might be treated as an asc file and deleted.

(5) Using temporary and backup asc files

In the execution environment, whether temporary and backup asc files are used depends on the specification of the -t and -d options in the adshexec command. The following table shows whether temporary and backup asc files are used in the execution environment.

Table 3–6: Whether temporary and backup asc files are used in the execution environment

adshexec command's option		Windows		UNIX	
-t	-d	Temporary asc file	Backup asc file	Temporary asc file	Backup asc file
Omitted	Omitted	N	N	N	N
Omitted	Specified	--	--	Y	N
Specified	Omitted	Y	Y	Y	Y
Specified	Specified	--	--	Y	Y

Legend:

Y: File is used.

N: File is not used.

--: The debugging option is not supported in Windows.

In the Windows development environment, whether temporary and backup `asc` files are used depends on the specification of **Coverage information** in the Runtime Environment Settings dialog box. The following table shows whether temporary and backup `asc` files are used in the development environment.

Table 3–7: Whether temporary and backup `asc` files are used in the development environment

Specification of Coverage information	Windows development environment (editor)	
	Temporary asc file	Backup asc file
Do not accumulate	N	N
Accumulate	Y	Y
Accumulate (and overwrite on update)	Y	Y

Legend:

Y: File is used.

N: File is not used.

(6) Commands and temporary asc files

Temporary `asc` files are used as work files during command execution. If a file with the same name as a temporary `asc` file exists before command execution, that file will be deleted.

(7) Processing of asc files during execution of the adshexec command

The following table explains how `asc` files are processed during execution of the `adshexec` command.

Table 3–8: Processing of asc files during execution of the adshexec command

Status when the command starts		Command processing		
job.asc	adshexec_backup_job.asc	job.asc	adshexec_backup_job.asc	Status of asc file and how it is processed
N	N	Created	None	There is neither a new nor an old `asc` file. An `asc` file is created.
N	Y	None	Renamed to `job.asc`	The old `asc` file `job.asc` has been renamed as backup file `adshexec_backup_job.asc`. The `asc` file `job.asc` is restored from backup `asc` file `adshexec_backup_job.asc`.
Y	N	Used	None	`job.asc` is either the old or the new `asc` file. If `job.asc` is the new `asc` file, the `KNAX6242-I` message was issued during the previous execution.
Y	Y	Used	Deleted	`job.asc` is the new `asc` file and `adshexec_backup_job.asc` is the old `asc` file. The old `asc` file `adshexec_backup_job.asc` is deleted. The new `asc` file `job.asc` is used.

Legend:

Y: The file exists.

N: The file does not exist.

`job.asc`: Name of `asc` file

`adshexec_backup_job.asc`: Name of backup `asc` file

3.9.3 Accumulating coverage information

(1) How to accumulate coverage information and the format of coverage information

When you execute a job definition script, you use an option of the execution command to specify that coverage information is to be accumulated. When this specification is made, coverage information will be accumulated in an `asc` file.

The option for accumulating coverage information is the `-t` option. You can also use the `-o` option to rename the `asc` file to a name of your choosing. The format of the execution command with the option for accumulating coverage information specified is shown below. If there is conflicting information in the job definition script that is to be executed, an error will result. However, if the `-f` option is specified, the coverage information will be overwritten without resulting in an error.

```
adshexec [other-options...] [-t [-f ][-o path-name-of-asc-file]]
path-name-of-job-definition-script-file [run-time-parameters]
```

In UNIX, if the `adshexec` command is executed with the `-d` option specified (and not the `-t` option), coverage information is collected only in memory. In this case, you can use the debugger's `info coverage` command to display the coverage information. If you exit the debugger by entering the debugger's `quit` command, the collected coverage information will be discarded and the memory released.

(2) Accumulation methods

The two coverage information accumulation methods are initial accumulation, which is the first accumulation, and continued accumulation, which is any subsequent accumulation. The accumulation method that is used (initial or continued) is determined by whether there is already an `asc` file.

If a change is made to a job definition script, the changed information will no longer have the same line numbers. Therefore, a backup of the job definition script is stored in the `asc` file. If there is a difference between the job definition script file and the backup `asc` file, the command terminates with an error without executing the job definition script.

In the initial accumulation, the command creates an `asc` file and writes the coverage information into it during execution. In a continued accumulation, the command reads the contents of the `asc` file and updates it by adding coverage information for the current execution.

(a) Examples of initial accumulation

The following are examples of initial accumulation.

Example 1:

This example collects coverage information when there is no coverage information file (`asc` file).

Example 2:

If all the following conditions are satisfied, the command performs initial accumulation:

- There is a coverage information file (`asc` file).

- The job definition script file differs from that used when coverage information was collected in the existing coverage information file.
- A different job definition script file is used and the option for initializing the coverage information file (`-f` option in the `adshexec` command) is specified.

(b) Examples of continued accumulation

If all the following conditions are all satisfied, the command performs a continued accumulation:

- A coverage information file (`asc` file) already exists.
- The job definition script file is the same as the one used when coverage information was collected in the existing coverage information file.

One of the conditions for performing continued accumulation is that the job definition script is the same as the previous one used to collect coverage information. The command determines that job definition scripts are identical in the following manner:

- A binary comparison that is performed on the job definition scripts shows that their sizes and contents are the same.

When these conditions are satisfied, the command treats the job definition scripts as being identical even if their file names and paths differ.

(3) File names of job definition scripts that are registered in coverage information files

In a continued accumulation, the file name of the job definition script that was used the first time coverage information was collected takes effect.

Example:

The following files are treated as being the same job definition scripts although their file names differ:

- `/dir1/file1`
- `/dir2/file2`

If a continued accumulation is performed by using an output `asc` file named `out.asc` and the above job definition scripts, the scrip file name in the `out.asc` will be as follows:

1. If `adshexec -t -o out.asc /dir1/file1` is executed, the script file name in `out.asc` is `/dir1/file1`.
2. If `adshexec -t -o out.asc /dir2/file2` is executed, the script file name in `out.asc` is `/dir2/file2`.
3. If `adshexec -t -o out.asc /dir1/file1` is re-executed, the script file name in `out.asc` is `/dir1/file1`.

(4) Extension of coverage information files

The default extension for coverage information files (`asc` files) is `.asc`. The extension for coverage information files does not have to be `.asc`. When coverage information is collected, the command treats a file with any extension that was specified as coverage information as an `asc` file.

(5) Size of a job definition script

The size of a job definition script file must not exceed 2 gigabytes.

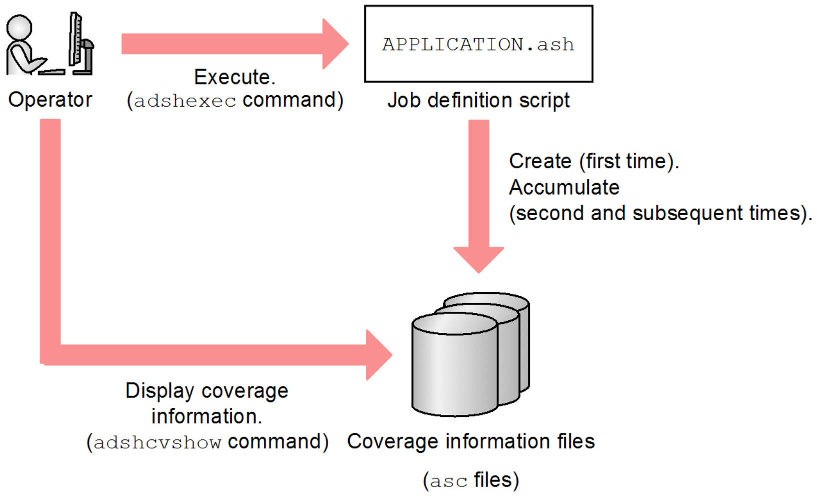
(6) Initializing accumulated coverage information

To initialize accumulated coverage information, delete the corresponding `asc` file with a command such as `rm`, and then collect coverage information using the initial accumulation method.

3.9.4 Displaying coverage information

The following figure shows the general procedure from executing a job definition script to displaying the coverage information.

Figure 3–5: General procedure for displaying coverage information



You can also use an editor to display coverage information in the development environment. For details about how to use an editor to display coverage information, see *4.4.7 Displaying coverage information*.

(1) How to display and command format

The `adshcvshow` coverage information display command is used to display coverage information. This command displays the contents of a specified `asc` file. If you wish to display only a desired range of a job definition script's coverage information, you can do so by specifying the range in the `-l` option.

If the `-s` option is specified, the command displays only the contents of a job definition script that has been backed up. You use the `-s` option to check the contents of a job definition script that has been backed up and to determine if there is any differential information between job definition scripts.

The following shows the format of the coverage information display command:

```
adshcvshow {[-l n1[-[n2]][,n3[-[n4]]]...]|-s} path-name-of-asc-file
```

To specify lines, use the comma (`,`) to separate individual line numbers and the hyphen (`-`) to specify a range of lines numbers. For example, to specify lines 1 through 10, line 15, and lines 21 through 30, specify the command as follows:

```
adshcvshow -l 1-10,15,21-30 path-name-of-asc-file
```

If no number follows a hyphen, the command assumes a range from the specified line number through the last line. For example, to specify lines 21 through the last line, specify the command as follows:

```
adshcvshow -l 21- path-name-of-asc-file
```

(2) Coverage information display format

The following table explains the coverage information display format.

Table 3–9: Explanation of the coverage information display format

Item	Description
Title line (`Advanced Shell Coverage Information`)	Main title line indicating that this is coverage information acquired by JP1/Advanced Shell.
Date and time (top right)	Displays the date and time the `adshcvshow` command was executed, in the format *yyyy*-*mm*-*dd hh*:*mm*:*ss*. If the month, day, hour, minute, or second value consists of one digit, a leading zero is added.
Header section (`Header Information`)	Section title line indicating display of header information.
Job definition script name (`Shellscript Name`)	Displays the absolute path name of the job definition script.
Version of `asc` file (`Asc version`)	Displays the `asc` file's version number.
Coverage information collection start time (`Coverage Start Time`)	Displays the time collection of coverage information started. The format is the same as for the date and time.
Coverage information collection end time (`Coverage End Time`)	Displays the time collection of coverage information ended. The format is the same as for the date and time.
Number of times coverage information was collected (`Test Count`)	Displays the number of times coverage information was collected. If the coverage information collection count exceeds 9,999, `9999+` is displayed. How the collection count is obtained depends on an option specified in the batch job execution command (`adshexec`). `adshexec` command with `-t` and `-d` specified • For coverage information in memory ● Initial value If there is an `asc` file, the coverage information collection count for the `asc` file is used. If there is no `asc` file, the collection count is 0. ● Updating The coverage information collection count is incremented by one each time the debugger's `run` command is executed. • For an `asc` file When the `adshexec` command terminates, the coverage information collection count is increased by the number of times the debugger's `run` command executed. `adshexec` command with `-t` only specified When the `adshexec` command terminates, the coverage information collection count is increased by one. `adshexec` command with `-d` only specified • Coverage information in memory ● Initial value

Item	Description
Number of times coverage information was collected (`Test Count`)	The initial value for the coverage information collection count is 0. ● Updating The coverage information collection count is incremented by one each time the debugger's `run` command is executed. • For an `asc` file `asc` files are not updated.
Main information section (`Main Information`)	Section title line indicating display of coverage information (C0 and C1 information).
Line number (`Line`)	The line numbers begin with `1`. A line number exceeding `9999` is displayed as `9999+`
Additional information (`Info`)	This is the header for C0 and C1 information. The coverage information is displayed in units of lines. If a command spans multiple lines, the C0 and C1 information is displayed on the lines containing the command name. If the numbers of C0 and C1 information items are both 32 or fewer, the coverage information can be recorded and this item is blank. The character strings that are displayed when the coverage information cannot be recorded are explained below: `overC0`: The number of C0 information items exceeds 32. `overC1`: The number of C1 information items exceeds 32. `over`: The numbers of C0 and C1 information items both exceed 32.
C0 information (`C0`)	Displays the C0 information: `@`: Command was executed. `-`: Command was not executed. If a line contains multiple commands, the C0 information for a maximum of the first four commands is displayed as four characters.
C1 information (`C1`)	Displays the C1 information: `@`: Execution path was executed. `-`: Execution path was not executed. If a line contains multiple execution paths, the C1 information for a maximum of the first four execution paths is displayed as four characters.
Job definition script (`<Shellscript Image>`)	Displays the contents of the job definition script in units of lines. If a range is specified, only the lines in the specified range are displayed.
Totals section (`Total Information`)	Section title line indicating display of totals of the C0 and C1 information. If a range is specified, the `Total Information` line and lines subsequent to it are not displayed. If the count exceeds 99,999,999, `99999999+` is displayed.
Totals subject to C0 and C1 targets (`Target Total`)	`<C0>` displays the total number of target commands, and `<C1>` displays the total number of execution paths.
`<C0>`	Includes the total number of commands subject to C0 in the job definition script. All target commands are counted even if there is a line that contains more than 32 commands subject to C0.
`<C1>`	Includes the total number of execution paths subject to C1 in the job definition script. All execution paths are counted even if there is a line that contains more than 32 execution paths subject to C1.
Totals subject to C0 and C1 that were executed (`Executed Total`)	`<C0>` displays the total number of commands executed, and `<C1>` displays the total number of execution paths executed.
`<C0>`	Execution is recorded as coverage information for a maximum of the first 32 commands subject to C0 in each line. Of these 32 commands, the commands that were executed are counted.

Item	Description
`<C1>`	Execution is recorded as coverage information for a maximum of the first 32 execution paths subject to C1 in each line. Of these 32 execution paths, the execution paths that were executed are counted.
Totals subject to C0 and C1 that were not executed (`Unexecuted Total`)	`<C0>` displays the total number commands that were not executed, and `<C1>` displays the total number of execution paths that were not executed.
`<C0>`	This is the total number subject to C0 (`Target Total`) - total number subject to C0 that were executed (`Executed Total`).
`<C1>`	This is the total number subject to C1 (`Target Total`) - total number subject to C1 that were executed (`Executed Total`).
Execution percentage rate (`Coverage Rate`)	Displays the execution percentages of C0 and C1 (%). The values are rounded off to the first decimal place.
`<C0>`	This is the total subject to C0 that were executed (`Executed Total`)/total subject to C0 (`Target Total`). If there was a line containing more than 32 commands subject to C0, this value would be less than 100%, even if all commands were executed.
`<C1>`	This is the total subject to C1 that were executed (`Executed Total`)/total subject to C1 (`Target Total`). If there was a line containing more than 32 execution paths subject to C1, this value would be less than 100%, even if all execution paths were executed.

The following subsections presents example coverage information displays. One is for when a maximum of one C0 and one C1 information item is displayed per line. The second display is for when a maximum of four information items are displayed per line.

(a) Example display of commands for which coverage information is displayed (maximum of one C0 and one C1 information item displayed per line)

In this example, a single `@` and a single `-` in `Main Information` indicate that C0 and C1 were acquired.

```
                    * Advanced Shell Coverage Information *

                                                            2013-12-06
12:22:50
****    Header Information
**************************************************
Shellscript Name     : /home/testuser/sample
Asc version          : 1.0
Coverage Start Time  : 2013-12-06 12:21:38
Coverage End Time    : 2013-12-06 12:21:39
Test Count           :    1

****    Main Information
**************************************************
Line   Info   C0   C1   <Shellscript Image>
    1
    2         @         echo 1
    3
    4         @    @    if true
    5                   then
    6         @           echo 2
    7              -    fi
    8
    9         @         echo 3
   10
   11         @    -    if false
   12                   then
   13         -           echo 4
   14              @    fi
   15
   16         @         echo 5
   17
   18         @    @    if true
   19                   then
   20         @           echo 6
   21              -    else
   22         -           echo 7
   23                   fi
   24
   25         @         echo 8
   26
   27         @    -    if false
   28                   then
   29         -           echo 9
   30              @    else
   31         @           echo 10
   32                   fi
   33
   34         @         echo 11
   35
   36

****    Total Information
**************************************************
                         <C0>      <C1>
  Target     Total         15         8
  Executed   Total         12         4
  Unexecuted Total          3         4
----------------------------------------------------------------------------
----
                         <C0>      <C1>
  Coverage   Rate        80.0 %    50.0 %
```

(b) Example display of commands for which coverage information is displayed (maximum of four C0 and C1 information items displayed per line)

In this example, lines 13 and 37 in `Main Information` indicate that multiple C0 and C1 information items were acquired.

- Line 13 displays the contents of lines 3 through 7.

  `@@@@` in the `C0` column indicates that the commands `echo 1`, `echo 2`, `echo 3`, and `echo 4` were executed in this order.

  `@@@@` in the `C0` column does not indicate whether command `echo 5` was executed.

- Line 37 displays the contents of lines 20 through 31.
  - Each character in `@@--` in the `C0` column corresponds to a command from the top.

    The first character `@` indicates that the command `true` was executed.

    The second character `@` indicates that the command `echo 1` was executed.

The third character - indicates that the command `true` that follows the first `elif` was not executed.

The fourth character - indicates that the command `echo 2` was not executed.

The characters `@@--` in the `C0` column show only whether the first four commands above were executed; whether the commands starting with `true` that follows the second `elif` were executed is not indicated.

- Each character in `@---` in the `C1` column corresponds to each command from the beginning.

  The first character `@` indicates that the execution path of the first `then` of `if` was executed.

  The second character - indicates that the execution path of `then` for the first `elif` was not executed.

  The third character - indicates that the execution path of `then` for the second `elif` was not executed.

  The fourth character - indicates that the execution path of `else` was not executed.

- Line 73 displays the contents of lines 43 through 67.

  The meaning of each character in `@@--` in the `C0` column and in `@---` in the `C1` column is the same as for line 37.

  `@---` in the `C1` column does not indicate whether the execution paths that follow the execution path of `then` for `if` (the fifth execution path from the top) were executed.

```
                    * Advanced Shell Coverage Information *

                                                            2013-12-06
12:24:27
****    Header Information
**************************************************
Shellscript Name     : /home/testuser/sample1.ash
Asc version          : 1.0
Coverage Start Time  : 2013-12-06 12:21:49
Coverage End Time    : 2013-12-06 12:21:50
Test Count           :    1

****    Main Information
**************************************************
Line   Info   C0   C1   <Shellscript Image>
    1
    2
    3         @         echo 1
    4         @         echo 2
    5         @         echo 3
    6         @         echo 4
    7         @         echo 5
    8
    9
   10
   11
   12
   13         @@@@      echo 1;echo 2;echo 3;echo 4;echo 5
   14
   15
   16
   17
   18
   19
   20         @    @    if true
   21                   then
   22         @           echo 1
   23         -    -    elif true
   24                   then
   25         -          echo 2
   26         -    -    elif true
   27                   then
   28         -          echo 3
   29              -    else
   30         -          echo 4
   31                   fi
   32
   33
   34
   35
   36
   37         @@-- @--- if true ;then echo 1 ;elif true ;then echo 2 ;elif
true ;then echo 3 ;else echo 4 ;fi
   38
   39
   40
   41
   42
   43         @    @    if true
   44                   then
   45         @          echo 1
   46         -    -    elif true
   47                   then
   48         -          echo 2
   49         -    -    elif true
   50                   then
   51         -          echo 3
   52              -    else
   53         -          echo 4
   54                   fi
   55
   56         @    @    if true
   57                   then
   58         @          echo 5
   59         -    -    elif true
   60                   then
   61         -          echo 6
   62         -    -    elif true
   63                   then
   64         -          echo 7
   65              -    else
   66         -          echo 8
   67                   fi
   68
   69
   70
   71
   72
   73         @@-- @--- if true ;then echo 1 ;elif true ;then echo 2 ;elif
true ;then echo 3 ;else echo 4 ;fi; if true ;then echo 5 ;elif true ;then
echo 6 ;elif true ;then echo 7 ;else echo 8 ;fi
   74

****    Total Information
**************************************************
                       <C0>      <C1>
  Target     Total       52        24
  Executed   Total         22         6
  Unexecuted Total         30        18
----------------------------------------------------------------------------
----
                         <C0>      <C1>
  Coverage   Rate        42.3 %    25.0 %
```

(3) How to display C0 and C1 information

The target subject to collection of coverage information varies depending on how script control statements are executed in a job definition script. When coverage information is displayed, an at mark (`@`) is displayed for a target that was executed, and a hyphen (`-`) is displayed for a target that was not executed.

(a) if statements

- When there is no `else`
  - If a path of `then` was executed, the following information is displayed:

```
C0   C1   Job definition script
     @    if          <--  C1 is acquired
@           true      <--  C0 is acquired
          then
@           cmd2       <--  C0 is acquired
@           cmd3       <--  C0 is acquired
     -    fi           <--  C1 is not acquired
```

  - If a path of `then` was not executed, the following information is displayed:

```
C0   C1  Job definition script
     -    if           <--  C1 is not acquired
@           false      <--  C0 is acquired
          then
-           cmd2       <--  C0 is not acquired
-           cmd3       <--  C0 is not acquired
     @    fi           <--  C1 is acquired
```

  - If a path of `then` and a path that is not `then` were both executed, the following information is displayed:

```
C0   C1  Job definition script
     @    if           <--  C1 is acquired
@           false      <-  C0 is acquired
          then
@           cmd2       <--  C0 is acquired
@           cmd3       <--  C0 is acquired
     @    fi           <--  C1 is acquired
```

- When there is `else`
  - If `then` was executed, the following information is displayed:

```
C0   C1  Job definition script
     @    if           <--  C1 is acquired
@           true       <--  C0 is acquired
          then
@           cmd2       <--  C0 is acquired
     -    else         <--  C1 is not acquired
-           cmd3       <--  C0 is not acquired
          fi           <--  None
```

- If `else` was executed, the following information is displayed:

```
C0   C1  Job definition script
     -    if           <--  C1 is not acquired
@           false      <--  C0 is acquired
          then
-           cmd2       <--  C0 is not acquired
     @    else         <--  C1 is acquired
@           cmd3       <--  C0 is acquired
          fi           <--  None
```

- If `then` and `else` were both executed, the following information is displayed:

```
C0   C1  Job definition script
     @    if           <--  C1 is acquired
@           false      <--  C0 is acquired
          then
@           cmd2       <--  C0 is acquired
     @    else         <--  C1 is acquired
@           cmd3       <--  C0 is acquired
          fi           <--  None
```

(b) for statements

- If a loop was executed, the following information is displayed:

```
C0   C1  Job definition script
     @    for          <--  C1 is acquired
          do
@           cmd1       <--  C0 is acquired
     -    done         <--  C1 is not acquired
```

- If a loop was not executed, the following information is displayed:

```
C0   C1  Job definition script
     -    for          <--  C1 is not acquired
          do
-           cmd1       <--  C0 is not acquired
     @    done         <--  C1 is acquired
```

- If execution involved executing a loop and then skipping a loop, the following information is displayed:

```
C0   C1  Job definition script
     @    for          <--  C1 is acquired
          do
@           cmd1       <--  C0 is acquired
     @    done         <--  C1 is acquired
```

(c) while and until statements

This subsection describes how `while` statements are displayed. `until` statements are displayed in the same manner.

- If a loop was executed, the following information is displayed:

```
C0   C1  Job definition script
     @    while        <--  C1 is acquired
          do
@           cmd1       <--  C0 is acquired
     -    done         <--  C1 is not acquired
```

- If a loop was not executed, the following information is displayed:

```
C0   C1  Job definition script
     -    while        <--  C1 is not acquired
          do
-           cmd1       <--  C0 is not acquired
     @    done         <--  C1 is acquired
```

- If execution involved executing a loop and then skipping a loop, the following information is displayed:

```
C0   C1  Job definition script
     @    while        <--  C1 is acquired
          do
@           cmd1       <--  C0 is acquired
     @    done         <--  C1 is acquired
```

(d) case statements

Whether an `*` pattern is used determines how the C1 information is displayed. An `*` pattern means that none of the patterns was a match in the `case` statement.

- If there is an `*` pattern

  The C1 information is displayed for `esac`.

- If there is no `*` pattern

  The C1 information is displayed for `esac`.

- Display method when there is an `*` pattern

  - If case 1 was executed, the following information is displayed:

```
C0   C1  Job definition script
          case $A in
     @      1)              <--  C1 is acquired
@              echo "abc"  <--  C0 is acquired
               ;;
     -      *)              <--  C1 is not acquired
-              echo "efg"  <--  C0 is not acquired
               ;;
          esac              <--  None
```

  - If an `*` pattern was executed, the following information is displayed:

```
C0   C1  Job definition script
          case $A in
     -      1)              <--  C1 is not acquired
               echo "abc"
               ;;
     @      *)              <--  C1 is acquired
@              echo "efg"  <--  C0 is acquired
```

```
               ;;
          esac             <--  None
```

- If case 1 and an `*` pattern were both executed, the following information is displayed:

```
C0   C1  Job definition script
          case $A in
     @      1)             <--  C1 is acquired
@              echo "abc"  <--  C0 is acquired
               ;;
     -      *)             <--  C1 is not acquired
-              echo "efg"  <--  C0 is not acquired
               ;;
          esac             <--  None
```

- Display method when there is no `*` pattern
    - If case 1 was executed, the following information is displayed:

```
C0   C1  Job definition script
          case $A in
     @      1)             <--  C1 is acquired
@              echo "abc"  <--  C0 is acquired
               ;;
     -      2)             <--  C1 is not acquired
-              echo "efg"  <--  C0 is not acquired
               ;;
     -    esac             <--  C1 is not acquired
```

- If case 2 was executed, the following information is displayed:

```
C0   C1  Job definition script
          case $A in
     -      1)             <--  C1 is not acquired
-              echo "abc"  <--  C0 is not acquired
               ;;
     @      2)             <--  C1 is acquired
@              echo "efg"  <--  C0 is acquired
               ;;
     -    esac             <--  C1 is not acquired
```

- If an `*` pattern was executed, the following information is displayed:

```
C0   C1  Job definition script
@         case $A in       <--  C0 is acquired
     -      1)             <--  C1 is not acquired
-              echo "abc"  <--  C0 is not acquired
               ;;
     -      2)             <--  C1 is not acquired
-              echo "efg"  <--  C0 is not acquired
               ;;
     @    esac             <--  C1 is acquired
```

(e) #-adsh_step_start command

Specification of the argument shown below in the `#-adsh_step_start` command sets whether a job step's execution is to be determined by the preceding job step and the status of the extended script command in the job definition script.

```
[ -run {normal | abnormal | always} ]
```

The following information is displayed in the C1 information to indicate whether a job step was executed:

- `--`: Execution did not reach the `#-adsh_step_start` command.
- `N-`: The preceding job step or job definition script is normal.
- `-A`: The preceding job step or job definition script is erroneous.
- `NA`: Cases `N-` and `-A` were both executed.

(f) #-adsh_step_error command

If an error occurs in a job step, the job definition script following the `#-adsh_step_error` command is executed. To indicate whether the error was handled, the following information is displayed in the C1 information.

- `--`: Execution did not reach the step containing the `#-adsh_step_error` command.
- `N-`: No error handling procedure executed because no error occurred in a job step.
- `-E`: An error handling procedure was executed because an error occurred in a job step.
- `NE`: Cases `N-` and `-E` were both executed.

(g) Functions

The following shows an example of function execution:

```
C0   C1   Job definition script
           funcAAA(){                 --> 1.
@              echo "start funcAAA"   --> 2.
     @         if true                --> 2.
               then                   --> 2.
@                echo true            --> 2.
     -         else                   --> 2.
-                echo false           --> 2.
               fi                     --> 2.
           }
             :
@          funcAAA                    --> 3.
```

1. When a function is executed, neither C0 nor C1 information is displayed at the location where the function is defined.
2. In the body of the function, C0 and C1 information is displayed for the commands and execution paths that executed
3. When a function has executed, the C0 information is displayed at the location where the function was called.

(h) (cmd1; cmd 2)

Commands enclosed in parentheses are executed as a separate process. In this case, coverage information is not collected either for the entire command group or for the individual commands in the command group.

(i) {cmd1; cmd2}

Commands enclosed in curly brackets are executed in the same process as the `adshexec` command. In this case, coverage information is collected for each command in the command group.

(j) cmd1 &

A separate process is generated, and the command is executed in the background in parallel with execution of the job definition script by the adshexec command. No coverage information is collected for job definition scripts that are executed in the background.

(k) trap actions

No coverage information is collected for trap actions.

- Example

```
trap "date; echo xxx" INT
```

(l) Command substitution

No coverage information is collected for a command or a script control statement that is executed by command substitution.

- Example

```
ls `which adshexec`
```

(m) Arguments of the time command

No coverage information is collected for a command that is executed as an argument of the time command.

- Example

```
time adshexec script1
```

(n) Arguments of the eval command

No coverage information is collected for a command that is executed as an argument of the eval command.

- Example

```
eval ls dir1
```

(o) Pipe function

No coverage information is collected for a command that is executed by using the pipe function.

- Example

```
ls | cat
```

(p) External scripts

Coverage information is not collected for called external scripts. Coverage information is collected for a process that calls an external script. An external script call is subject to collection of C0 information, but is not subject to collection of C1 information.

(4) Displaying coverage information collected in memory (UNIX only)

If you have used the `info coverage` command for debugging, you can display coverage information collected in memory.

The coverage information to be displayed depends on the accumulation type (initial or continued accumulation). For the initial accumulation, the coverage information up to the breakpoint is displayed. For a continued accumulation, the accumulated coverage information plus the coverage information up to the breakpoint is displayed. If accumulation is not specified, the coverage information up to the breakpoint is displayed in the same manner as for initial accumulation.

The types and format of the information that is displayed are the same as when the `adshcvshow` command is used to display coverage information.

(5) Case where the C1 execution percentage rate is not 100%

If the `#-adsh_step_start` command is used and no job step or command precedes the job step of `#-adsh_step_start`, the C1 execution percentage rate will never be 100%, even if all the execution paths are executed. `#-adsh_step_start` collects C1 information in the cases described below. However, if no job step or command precedes the job step of `#-adsh_step_start`, C1 information cannot be collected for case 2 below:

1. All the preceding job steps and commands terminated normally.
2. At least one of the preceding job steps or commands did not terminate normally.

In this case, you can enable the fault injection mode during debugging to simulate errors at the corresponding locations. This method enables you to collect CI information and improve the CI execution percentage rate to 100%. The following explains how to simulate errors.

- Debugging in GUI (Windows only)

  You can simulate errors by using JP1/Advanced Shell Editor's **Fault Injection Mode** menu. For details about the procedure, see *4.4.6(4) Simulating errors*.
- Debugging in CUI (started with the `-d` option of the `adshexec` command) (UNIX only)

  You can simulate errors by using the `joberrmode` command. For details about the `joberrmode` command, see *6.2.20 Enabling and disabling the fault injection mode (joberrmode command)*.

  You can use the `info status` command to check whether the fault injection mode is enabled. For details about the `info status` command, see *6.2.18 Displaying the status (info status command)*.

3.9.5 Merging coverage information

The purpose of merging coverage information is to combine the results obtained from testing by multiple users of the same job definition script. If different test cases were performed on a specific job definition script by different users, you can merge the separate pieces of coverage information into a single entity.

(1) How to merge

You use the `adshcvmerg` command to merge coverage information. This command merges two specified `asc` files. The following shows the command's format:

```
adshcvmerg -o output-file asc-file-1 asc-file-2
```

The command merges the information in *asc-file-1* and *asc-file-2* and outputs the results to the specified output file in `asc` file format.

(2) Types of information to be merged

The information to be merged includes the test counts and coverage information. For example, if the command `adshcvmerg -o out c1 c2` is entered to perform merge processing, the information is changed as follows:

- Full path name of job definition script: Full path name of `c1`
- Test count: Test count for `c1` + test count for `c2`
- Coverage information collection start date and time: The start date and time for `c1` or `c2`, whichever was earlier
- Coverage information collection end date and time: The end date and time for `c1` or `c2`, whichever was later

3.9.6 Coverage auto-acquisition functionality

The coverage auto-acquisition functionality enables you to collect coverage information without having to change parameters in the `adshexec` command.

If you use the environment setting parameters listed below to set the coverage information to be collected, there is no need to specify the `-t` option for collecting coverage information when you execute batch jobs with the `adshexec` command.

- `BATCH_CVR` parameter: Specifies that the coverage auto-acquisition functionality is to be used.
- `ASC_FILE` parameter: Defines the naming rules for accumulation files used by the coverage auto-acquisition functionality.

The following shows an example of specifying the `adshexec` command:

```
adshexec job-definition-script-name.ash
```

This command executes the specified job definition script without having to specify the options (`-t` and `-o`) for collecting coverage information.

When the coverage auto-acquisition functionality is used, the `-t` option cannot be specified in the `adshexec` command. If the `adshexec` command is executed with the `-t` option specified in such a case (such as `adshexec -t sample.ash)`, it will terminate with an error and set return code `1`.

You collect the coverage information by specifying `#-adsh_conf BATCH_CVR YES` in the environment file.

When the coverage function is enabled by the coverage auto-acquisition functionality, the `asc` files (coverage information files) are output to the current directory in which each command was executed.

If `#-adsh_conf ASC_FILE cvr/ver001-*` is specified in the environment file, the above command produces the same results as when `adshexec -t -o cvr/ver001-`*job-definition-script-name job-definition-script-name*`.ash` is executed.

If the current directory differs for each command, the `asc` files are created in various directories. By specifying `#-adsh_conf ASC_FILE`, you can designate a specific directory to which the `asc` files are to be output. You can also standardize the `asc` file names.

For details about the settings in the environment files, see *2.6.11 Enabling coverage information collection without having to specify the option during batch job execution*.

3.10 Forcibly terminating jobs

This section explains forced termination of jobs.

3.10.1 How to forcibly terminate jobs

(1) How to forcibly terminate a job

There are two ways to forcibly terminate a job:

- If the job was started from JP1/AJS, use JP1/AJS's forced termination procedure.
  To be able to forcibly terminate from JP1/AJS a job for a job icon that was executed in Windows or UNIX, you must have specified the `AJS_BJEX_STOP=TERM` environment variable beforehand. For details about jobs for job icons that are run in Windows or UNIX, see *2.7.2 Defining and executing a jobnet*.
- Send a termination request signal to the `adshexec` command's process. In Windows, you can use a command such as `taskkill` to terminate the `adshexec` process.

When a job is forcibly terminated, the job controller forcibly terminates its child or descendant process that are executing. For details, see *(2) Forcibly terminating child or descendant processes*.

After forcibly terminating the child or descendant process, the job controller performs postprocessing on the allocated files, and then terminates the job without executing any subsequent job steps or commands. The job controller does not execute a subsequent job step even if `abnormal` or `always` is specified in its `run` attribute. In UNIX, when a job is forcibly terminated, the `adshexec` command terminates with an error by signal. For details about the job processing in UNIX when `SIGTERM` is received, see *3.10.2 Processing when signals are received (UNIX only)*. For details about the job processing in Windows when jobs are forcibly terminated, see *3.10.3 Job processing during forced termination (Windows only)*.

Important note

In Windows, when the `adshexec` command is started, the `adshexecsub` command is also started, and when the `adshexec` command is forcibly terminated, the `adshexecsub` command is also terminated. Therefore, do not forcibly terminate the `adshexecsub` command. If an attempt is made to forcibly terminate the `adshexecsub` command, the following events might occur:

- A descendant process that is executing might not terminate.
- Temporary files might remain in the system.

If these events occur, use the `taskkill` command or the task manager to forcibly terminate the descendant process and delete the temporary files manually.

Important note

Because JP1/Advanced Shell in a Windows environment uses job objects to forcibly terminate descendant processes, note the following;

- A descendant process generated from JP1/Advanced Shell cannot be associated with a job object.

- If a process of JP1/Advanced Shell has already been associated with a job object, forced termination of the job will not terminate the process generated by the child process of JP1/Advanced Shell.

Important note

In Windows, if a job that executes an external command and generates a child process is terminated forcibly and more than 255 processes that are at its grandchild or lower levels exist concurrently, the `KNAX6381-E` message might be issued and renaming of the spool job directory might fail. Note the following three points about this:

- To reference a spool job directory that has failed, use the directory name displayed in the immediately following `KNAX6382-I` message that is issued.
- A spool job directory whose renaming has failed cannot be deleted by the `adshhk` command. If necessary, delete it manually.
- In the case of a job that has failed in renaming a spool job directory in the execution environment, job definition script operation information is not output by the `adshevtout` command.

(2) Forcibly terminating child or descendant processes

If a job is forcibly terminated, the job controller forcibly terminates its child or descendant processes, and then terminates the job.

(a) In UNIX

How child or descendant processes are forcibly terminated depends on the job input mode, as described in the following:

- Terminal input mode

  `SIGTERM` is sent only to the child processes of the `adshexec` command. `SIGTERM` is not sent to any of the descendant processes of the `adshexec` command, including grandchild processes. If you want to perform postprocessing on these processes, use one of the following methods to create and execute a job:

  - If the user creates external commands, design the external commands to perform postprocessing on the descendant processes, for example, by automatically sending `SIGTERM` also to the descendant processes after `SIGTERM` is received.
  - When a job is forcibly terminated in the terminal input mode, not only the `adshexec` command but all the descendant processes, including grandchild processes, are subject to operations such as **Ctrl** + **C** and **Ctrl** + \.

  If grandchild processes remain after forced termination, use the `ps` command to obtain the process IDs of the remaining processes, and then manually terminate them with the `kill` command.

- Non-terminal input mode

  `SIGTERM` is sent to the descendant processes of the `adshexec` command.

(b) In Windows

The `TerminateProcess` and `TerminateJobObject` functions are used to forcibly terminate the descendant processes of the `adshexec` command. The forced termination method is the same regardless of the job input mode.

(3) Notes about operations including Ctrl+C (UNIX only)

If a job is executed in the non-terminal input mode, operation such as **Ctrl** + **C** and **Ctrl** + \ might not be able to terminate simultaneously the root job, child jobs, and other external commands that were started.# If you wish to forcibly terminate these jobs and commands all at once, use the `kill` command to send a termination request signal such as `SIGTERM` to the root job immediately under the login shell.

#

If a job is executed in the non-terminal input mode, the `adshexec` command's process and its child processes belong to separate process groups. Therefore, if an operation such as **Ctrl** + **C** or **Ctrl** + \ is performed from the login shell while the job is executing, `SIGINT` or `SIGQUIT` is sent only to the process group currently running in the foreground.

The jobs and external commands running as descendant processes of the job that received the signal are forcibly terminated, but those jobs and external commands running as higher processes, including the parent process, are not forcibly terminated.

3.10.2 Processing when signals are received (UNIX only)

This subsection explains the processing that occurs when the job controller has received signals during normal execution and during debug execution.

(1) During normal execution

This subsection explains for the `SIGTERM` signal and for other signals the processing that occurs when the job controller has received signals during normal execution.

(a) SIGTERM

The processing that occurs when `SIGTERM` has been received depends on the specified `TRAP_ACTION_SIGTERM` environment setting parameter.

Table 3–10: When DISABLE is specified in the TRAP_ACTION_SIGTERM environment setting parameter#1

When an operation is not defined with the trap command	When an operation is defined with the trap command
• If the root job received `SIGTERM` First time: Outputs a message, performs postprocessing, and then terminates without performing any subsequent processing. If the root job was not started from JP1/AJS, the root job sends `SIGTERM` to itself and then terminates with the signal. Second time: Terminates immediately. • If a child job received `SIGTERM` The child job that received `SIGTERM` outputs a message, performs postprocessing, and then terminates itself without performing any subsequent processing. In this case, the child job sends `SIGTERM` to itself and then terminates with the signal.#2	Operation cannot be defined with the `trap` command.#3

#1

This includes when the `TRAP_ACTION_SIGTERM` environment setting parameter is not specified.

#2

For details about the behavior of child jobs when signals are received, see *3.2.3(3) Behavior of child jobs when signals are received*.

#3

When the action for SIGTERM is specified by the trap command, the job terminates with a trap command error.

Table 3–11: When TERM is specified in the TRAP_ACTION_SIGTERM environment setting parameter

When an operation is not defined with the trap command	When an operation is defined the trap command
• If the root job received SIGTERM First time: Outputs a message, performs postprocessing, and then terminates without performing any subsequent processing. If the root job was not started from JP1/AJS, the root job sends SIGTERM to itself and then terminates with the signal. Second time: Terminates immediately. • If a child job received SIGTERM The child job that received SIGTERM outputs a message, performs postprocessing, and then terminates itself without performing any subsequent processing. In this case, the child job sends SIGTERM to itself and then terminates with the signal. The parent job of the child job performs subsequent processing according to the results of the child process that terminated with termination code 128 + *signal number of SIGTERM*.#	• If the root job received SIGTERM Outputs a message and then executes the action defined for SIGTERM with the trap command. After executing the action, the root job terminates without performing any subsequent processing. If the root job was not started from JP1/AJS, the root job sends SIGTERM to itself and then terminates with the signal. • If a child job received SIGTERM Outputs a message and then executes the action defined for SIGTERM with the trap command. After executing the action, the child job terminates without performing any subsequent processing. The parent job of the child job performs subsequent processing according to the results of the child process that received SIGTERM.#

#

For details about the behavior of child jobs when signals are received, see *3.2.3(3) Behavior of child jobs when signals are received*.

Table 3–12: When CONT is specified in the TRAP_ACTION_SIGTERM environment setting parameter

Job start method	When an operation is not defined with the trap command	When an operation is defined with the trap command
Started from JP1/AJS (Started from a custom job or with TERM set in the AJS_BJEX_STOP environment variable)	The job definition script is not run and the job terminates with an error (error during environment file analysis).	
Started using a method that does not involve JP1/AJS (Started with a method other than the above)	• If the root job received SIGTERM First time: Outputs a message, performs postprocessing, and then terminates without performing any subsequent processing. The root job sends SIGTERM to itself and then terminates with the signal. Second time: Terminates immediately. • If a child job received SIGTERM The behavior of a child job that has received SIGTERM is the same as that of the root job. The parent job of the child job performs subsequent processing according to the results of the child process that terminated with termination code 128 + *signal number of SIGTERM*.#	• If the root job received SIGTERM Outputs a message and then executes the action defined for SIGTERM with the trap command. After executing the action, the root job performs any subsequent processing in the job definition script. • If a child job received SIGTERM The behavior of a child job that has received SIGTERM is the same as that of the root job. The parent job of the child job performs subsequent processing according to the results of the child process that received SIGTERM.#

#

For details about the behavior of child jobs when signals are received, see *3.2.3(3) Behavior of child jobs when signals are received*..

Table 3–13: When AUTO is specified in the TRAP_ACTION_SIGTERM environment setting parameter

Job start method	When an operation is not defined with the trap command	When an operation is defined with the trap command
Started from JP1/AJS (Started from a custom job or with TERM set in the AJS_BJEX_STOP environment variable)	Same processing as when TERM is specified	
Started using a method that does not involve JP1/AJS (Started with a method other than the above)	Same processing as when CONT is specified	

(b) Other than SIGTERM

Table 3–14: Processing when signals are received

Type of signal		When an operation is not defined with the trap command	When an operation is defined the trap command
Termination request signal	SIGHUP, SIGINT, SIGXCPU, SIGXFSZ, SIGQUIT, SIGUSR1, SIGUSR2, SIGPIPE, SIGALRM, SIGVTALRM, SIGPROF	• If the root job received the signal Performs postprocessing, such as termination of descendant processes and deletion of temporary files, and then terminates with an error by signal without executing any subsequent instruction. • If a child job received the signal The processing of the child job that received the signal is the same as when the signal was received by the root job. The parent job of the child job that received the signal performs subsequent processing according to the results of the terminated child job.[#1]	• If the root job received the signal The processing depends on the operation defined by the trap command. • If a child job received the signal The processing of the child job that received the signal is the same as when the signal was received by the root job. The parent job of the child job that received the signal performs subsequent processing according to the results of the child job.
	SIGMSG, SIGDANGER, SIGMIGRATE, SIGPRE, SIGVIRT, SIGALRM1, SIGRECONFIG, SIGCPUFAIL, SIGGRANT, SIGRETRACT, SIGSOUND	Same as above. (AIX only)	Same as above. (AIX only)
	SIGLOST	Same as above. (HP-UX and Solaris only)	Same as above. (HP-UX and Solaris only)
Error notification signal	SIGILL, SIGTRAP, SIGABRT, SIGFPE, SIGBUS, SIGSEGV, SIGSYS	• If the root job received the signal Terminates the program according to the default OS processing for the corresponding signal. • If a child job received the signal The processing of the child job that received the signal is the same as when the signal was received by the root job. The parent job of the child job that received the signal performs subsequent processing according to the results of the terminated child job.[#1]	• If the root job received the signal The processing depends on the operation defined by the trap command. • If a child job received the signal The processing of the child job that received the signal is the same as when the signal was received by the root job. The parent job of the child job that received the signal performs subsequent processing according to the results of the child job.
	SIGIOT, SIGEMT	Same as above. (AIX, HP-UX, and Solaris only)	Same as above. (AIX, HP-UX, and Solaris only)

Type of signal		When an operation is not defined with the trap command	When an operation is defined the trap command
Error notification signal	`SIGLOST`	Same as above. (AIX only)	Same as above. (AIX only)
Other		• If the root job received the signal Depends on the default OS processing for the corresponding signal. • If a child job received the signal The processing of the child job that received the signal is the same as when the signal was received by the root job. The parent job of the child job that received the signal performs subsequent processing according to the results of the child job.[#1]	• If the root job received the signal The processing depends on the operation defined by the `trap` command.[#2] • If a child job received the signal The processing of the child job that received the signal is the same as when the signal was received by the root job. The parent job of the child job that received the signal performs subsequent processing according to the results of the child job.

#1

- For details about the behavior of child jobs when signals are received, see *3.2.3(3) Behavior of child jobs when signals are received*.

#2

- For `SIGKILL` and `SIGSTOP`, the `trap` command cannot be used to define an operation.
- For `SIGWAITING`, the `trap` command cannot be used to define an operation (AIX only).

Important note

If you set - for the operation when you are using the `trap` command, the operation to be performed when signals are received is reset to the default.

With some signals, the operation during debug execution differs from that described in the tables. For details about the differences in signal processing depending on whether an operation is defined with the `trap` command, see *(2) During debug execution*.

(2) During debug execution

Table 3–15: Processing when signals are received during debug execution

Type of signal	When an operation is not defined with the trap command	When an operation is defined with the trap command
`SIGINT`	The debugger terminates execution of the job definition script and then waits for entry of a command.[#]	The debugger terminates execution of the job definition script and then waits for entry of a command.[#] The processing depends on the operation defined by the `trap` command.
`SIGCHLD`, `SIGTSTP`, `SIGTTOU`, `SIGURG`, `SIGWINCH`, `SIGIO`, `SIGPWR`	Performs the next processing.	The processing depends on the operation defined by the `trap` command.
`SIGSTKFLT` (Linux only)		
`SIGWAITING`, `SIGLWP`, `SIGFREEZE`, `SIGTHAW`, `SIGCANCEL`, `SIGXRES`, `SIGJVM1`, `SIGJVM2` (Solaris only)		

Type of signal	When an operation is not defined with the trap command	When an operation is defined with the trap command
Real-time signal (HP-UX, Linux, and Solaris only)	Performs the next processing.	The processing depends on the operation defined by the `trap` command.

#

For details about terminating a job definition script, see *6.2 CUI debugger (UNIX only)*.

3.10.3 Job processing during forced termination (Windows only)

The following table describes job processing in Windows during forced termination.

If you use the `trap` command to define processing for immediate termination of a process by using a function such as TerminateProcess, specify `TERM` in the `TRAP_ACTION_SIGTERM` parameter.

Table 3–16: Job processing during forced termination

Forced termination method		When an operation is not defined with the trap command	When an operation is defined with the trap command
Control signal	CTRL + C CTRL + BREAK `CTRL_CLOSE_EVENT`	The control signal is sent to all process groups that are running as the root job, child jobs, and commands. • Processing of the root job (`adshexec.exe`) that received the control signal Child process `adshexecsub.exe` performs postprocessing, and then terminates without executing subsequent scripts. `adshexec.exe` that received the control signal waits for termination of the child process, and then terminates itself. • Processing of the root job (`adshexecsub.exe`) that received the control signal and its child jobs `adshexecsub.exe` that received the control signal outputs the `KNAX7896-I` message, performs postprocessing, and then terminates without executing subsequent scripts.	Operation cannot be defined with the `trap` command.
	`CTRL_LOGOFF_EVENT`	Terminates immediately without performing postprocessing because OS logoff and shutdown processing take precedence.	Operation cannot be defined with the `trap` command.
	`CTRL_SHUTDOWN_EVENT`		

Forced termination method	When an operation is not defined with the trap command	When an operation is defined with the trap command
Immediate termination of process by a means such as `TerminateProcess`	- If the root job (`adshexec.exe`) is subject to immediate termination The target `adshexec.exe` is terminated immediately, but `adshexecsub.exe` of its child processes and the child jobs perform postprocessing, and then terminate without executing subsequent scripts. - If the root job (`adshexecsub.exe`) is subject to immediate termination The target `adshexecsub.exe` is terminated immediately without performing postprocessing (do not perform immediate termination on this process). - If a child job is subject to immediate termination The target child job is terminated immediately without performing postprocessing (do not perform immediate termination on this process). The parent job of the terminated child job performs postprocessing according to the processing that occurs when a child process is terminated with an error and return code `1`.	- If the root job (`adshexec.exe`) is subject to immediate termination The target `adshexec.exe` is terminated immediately, but `adshexecsub.exe` of its child processes and the child jobs perform the operation defined with the `trap` command, perform postprocessing, and then terminate. - If the root job (`adshexecsub.exe`) is subject to immediate termination The target `adshexecsub.exe` is terminated immediately without performing postprocessing (do not perform immediate termination on this process). - If a child job is subject to immediate termination The target child job is terminated immediately without performing postprocessing (do not perform immediate termination on this process). The parent job of the terminated child job performs postprocessing according to the processing that occurs when a child process is terminated with an error and return code `1`.

Note

When the `trap` command is used and - is set for the operation, the command resets the previously specified `action` setting for the specified method so that the method is not associated with any `action` setting.

4 Using JP1/Advanced Shell - Developer (Windows Only)

This chapter explains how to employ JP1/Advanced Shell - Developer so that you can use JP1/Advanced Shell Editor to develop job definition scripts in a Windows environment. The chapter also explains how to use the editor to debug job definition script files.

4.1 Starting and terminating JP1/Advanced Shell - Developer (Windows only)

You can create and debug job definition script files in JP1/Advanced Shell's development environment. This section explains how to start and terminate JP1/Advanced Shell's development environment.

4.1.1 Starting JP1/Advanced Shell - Developer

This subsection explains how to start JP1/Advanced Shell - Developer. You start the editor to create and edit job definition scripts files. There are two ways to start the editor.

(1) Starting from the Start menu

1. From the **Start** menu, select **All Programs**, and then **Advanced Shell - Developer**.

2. From the Advanced Shell - Developer group, select the **Editor** icon.

(2) Starting from the right-click menu

1. From Explorer, right-click the job definition script file.

2. Select **Edit**.

4.1.2 Terminating JP1/Advanced Shell - Developer

To terminate JP1/Advanced Shell - Developer, do one of the following:

- Select **File**, and then **Exit**.
- Click the **Exit** button on the toolbar.

The editor function terminates.

4.2 JP1/Advanced Shell Editor modes (Windows only)

The editor has two modes, the edit mode and the debug mode.

4.2.1 Edit mode

The edit mode is used to create and edit job definition script files. The editor is in this mode when it starts.

4.2.2 Debug mode

The debug mode is used to debug created job definition script files. In this mode, the editor's edit window is grayed-out and the job definition script cannot be edited. The debug mode supports two functions:

- Syntax checking

  Selecting the **Check Syntax** menu from the **Debug** menu or clicking the **Check Syntax** button on the toolbar starts syntax checking.

- Debug execution

  Making the following menu item selection or clicking the following button executes debugging:

  - Selecting the **Run to Breakpoint** item from the **Debug** menu or clicking the **Run to Breakpoint** button on the toolbar
  - Selecting the **Step In**, **Step Over**, or **Step Out** item from the **Debug** menu, or clicking the **Step In**, **Step Over**, or **Step Out** button on the toolbar

For details about syntax checking, see *4.4.4 Checking syntax*. For details about debugging, see *4.4.6 Debugging*.

4.3 JP1/Advanced Shell Editor operation (Windows only)

The editor is a program you use to create job definition scripts and to edit existing job definition scripts. This section explains the JP1/Advanced Shell Editor window that is displayed when the editor starts. The section also explains the editor's functions by menu item.

The following lists the operations available in the JP1/Advanced Shell Editor window (the applicable section or subsection number is enclosed in parentheses):

- Creating job definition scripts (*4.4.1*)
- Setting up an operating environment for the editor (*4.4.2*)
- Setting up an execution environment for job definition scripts (*4.4.3*)
- Checking syntax (*4.4.4*)
- Searching for and replacing character strings (*4.4.5*)
- Setting and releasing breakpoints during debugging (*4.4.6(1)*)
- Performing and canceling debugging (*4.4.6(2)*)
- Adding variables to the watch list (*4.4.6(3)*)
- Displaying coverage information (*4.4.7*)
- Editing existing job definition scripts (*4.5*)
- Saving job definition scripts (*4.6*)
- Printing the contents of job definition script files[#]
- Undoing the previous operation[#]
- Redoing the previous operation[#]
- Cutting a selected character string and saving it on the clipboard[#]
- Copying a selected character string onto the clipboard[#]
- Pasting the character string from the clipboard to a specified location[#]
- Selecting all character strings[#]
- Jumping to the execution-point line[#]
- Changing toolbar view/hide settings[#]
- Changing the status bar view/hide setting[#]
- Aligning toolbars[#]
- Changing the ruler view/hide setting[#]
- Changing the vertical scroll bar view/hide setting[#]
- Changing the horizontal scroll bar view/hide setting[#]
- Changing the line numbers view/hide setting[#]
- Displaying the beginning of the file[#]
- Displaying the end of the file[#]
- Changing the Watch List window view/hide setting[#]

- Changing the error window view/hide setting#
- Displaying Help#

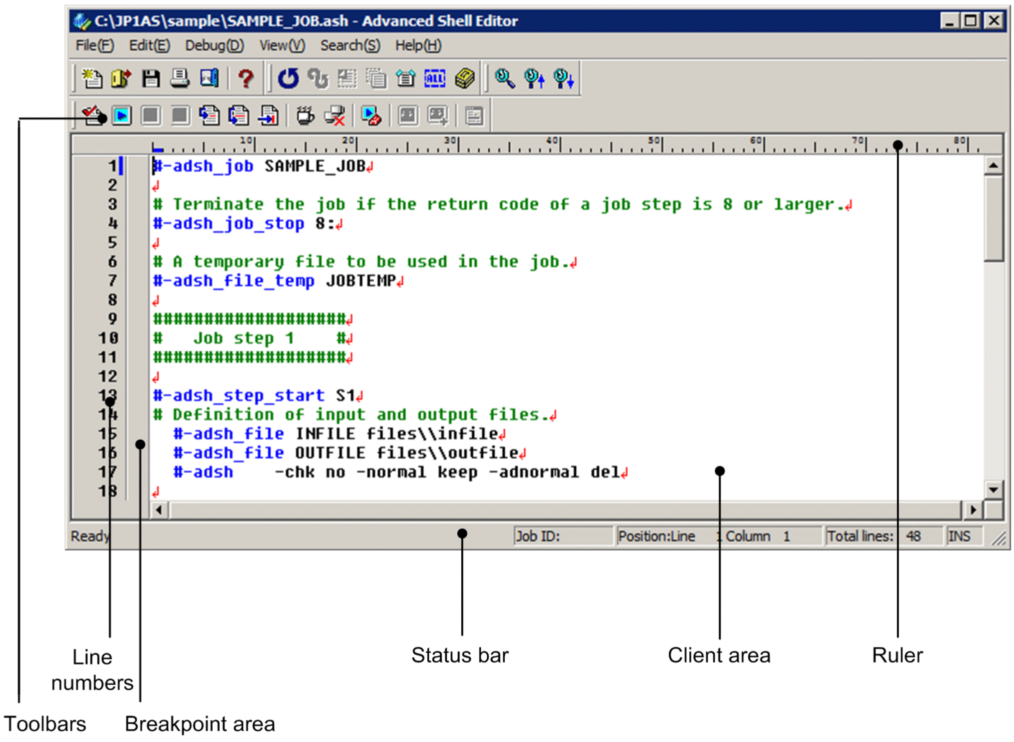
#

These operations are not explained in this manual (they are implemented by the standard Windows operating procedures or by selecting the corresponding items from the applicable menu lists).

4.3.1 JP1/Advanced Shell Editor window

The following figure shows the JP1/Advanced Shell Editor window and the names of the window's components.

Figure 4–1: JP1/Advanced Shell Editor window



```
#-adsh_job SAMPLE_JOB
 
# Terminate the job if the return code of a job step is 8 or larger.
#-adsh_job_stop 8:
 
# A temporary file to be used in the job.
#-adsh_file_temp JOBTEMP
 
####################
#   Job step 1    #
####################
 
#-adsh_step_start S1
# Definition of input and output files.
  #-adsh_file INFILE files\\infile
  #-adsh_file OUTFILE files\\outfile
  #-adsh    -chk no -normal keep -adnormal del
 
```

(1) Toolbars

The toolbars display buttons for the most frequently used of the commands that can be selected from the menu bar. You can execute a command by clicking its button on a toolbar. You can also use the **View** menu to hide the toolbars. Hovering the mouse cursor over a button displays a description of the button's function.

In the JP1/Advanced Shell Editor window, the toolbars that can be displayed are the Standard Toolbar, Edit Toolbar, Debug Toolbar, and Search Toolbar.

- Standard Toolbar

  The following table lists the buttons on the Standard Toolbar and describes their functions.

Button	Description
New button	Creates a job definition script file.
Open button	Enables an existing job definition script file to be opened.

Button	Description
Save button	Saves the job definition script file being edited.
Print button	Prints the job definition script file being edited.
Exit button	Terminates JP1/Advanced Shell Editor and enables you to select whether the file is to be saved.
Help button	Displays Help for JP1/Advanced Shell.

- Edit Toolbar
  The following table lists the buttons on the Edit Toolbar and describes their functions.

Button	Description
Undo button	Undoes the previous operation.
Redo button	Re-executes the previous operation.
Cut button	Cuts a selection and saves it on the clipboard.
Copy button	Copies a selection to the clipboard.
Paste button	Pastes the contents of the clipboard to the selected location.
Select All button	Selects the entire file.
Options button	Enables an operating environment for the editor to be set up.

- Debug Toolbar
  The following table lists the buttons on the Debug Toolbar and describes their functions.

Button	Description
Check Syntax button	Checks the syntax of the job definition script.
Run to Breakpoint button	Starts and restarts execution up to a breakpoint.
Stop Script button	Stops the job definition script. Completes the command whose execution is underway when the **Stop Script** button is clicked is completed and stops the job definition script before the next command executes.
Quit Debugging button	Completes the command whose execution is underway when the **Quit Debugging** button is clicked, stops execution before the next command is executed, then stops the job definition script and cancels debugging.
Step In button	Executes the next command or statement. If a function is called, this button also executes one line in the function and then stops execution.
Step Over button	Executes the next command or statement. If a function is called, this button does not stop after executing one line in the function, but stops only when a breakpoint is reached.
Step Out button	Executes through the end of a function call and stops on the line immediately following the function call or at a breakpoint.
Set/Remove Breakpoint button	Sets a breakpoint or releases a selected breakpoint.
Remove All Breakpoints button	Releases all breakpoints that have been set.
Runtime Environment Settings button	Sets up a script file execution environment.
View/Hide Watch List Window button	Changes the view/hide setting for the Watch List window.
Add Variable to Watch List button	Adds a specified variable to the Watch List window.
View Coverage Information button	Displays coverage information during debugging.

- Search Toolbar

The following table lists the buttons on the Search Toolbar and describes their functions.

Button	Description
Search button	Enables entry of a character string to be searched for and (optionally) a character string that is replace the search character string.
Find Previous button	Searches for the search character string in the up direction.
Find Next button	Searches for the search character string in the down direction.

(2) Ruler

This is a tick-marked bar that displays the horizontally-arranged columns.

(3) Line number area

This area displays the line numbers in a job definition script.

(4) Breakpoint area

This area displays the following symbols: a symbol at the line where a breakpoint is set (●), a symbol that indicates the line that is to execute next (➤), and a symbol that indicates the line where the debugger process ends (➤).

(5) Status bar

The status bar displays messages related to the current processing being executed by JP1/Advanced Shell Editor and messages related to the status after processing has terminated. The following table describes the status bar functions in the JP1/Advanced Shell Editor window.

Table 4–1: Functions of the status bar in the JP1/Advanced Shell Editor window

Status bar	Description
Job ID	Displays the job ID of the job that is being debugged.
Position	Displays the location of the cursor.
Total lines	Displays the total number of lines in the job definition script file being edited.
INS or OVR	Displays the overwrite mode that can be switched by toggling the **Insert** key. The two modes are the following (where the Insert mode is the default): • **OVR:Overwrite** mode • **INS:Insert** mode

(6) Client area

The client area displays the job definition script file you are working on.

4.3.2 Menus in the JP1/Advanced Shell Editor window

This subsection explains the menus displayed on the menu bar and the pop-up menus that are displayed in the JP1/Advanced Shell Editor window.

(1) Menus on the menu bars

This subsection explains the menus that are displayed in the editor window. The following table lists the menus in the JP1/Advanced Shell Editor window and describes the functions of the items you can select on these menus.

Table 4–2: Menus in the JP1/Advanced Shell Editor window and their functions

Menu		Description
File	**New**	Creates a job definition script file.
	Open	Enables an existing job definition script file to be opened.
	Save	Saves the job definition script file being edited.
	Save As	Saves the job definition script file being edited as a new job definition script file under a specified name.
	Print	Prints the job definition script file being edited.
	Exit	Exits the editor and enables you to select whether the file is to be saved.
	(*file-name*)	Opens the file whose name is displayed. The names of the most recent job definition script files that were saved are displayed (maximum of nine).
Edit	**Undo**	Undoes the previous operation.
	Redo	Re-executes the previous operation.
	Cut	Cuts a selection and saves it on the clipboard.
	Copy	Copies a selection to the clipboard.
	Paste	Pastes the contents of the clipboard to the selected location.
	Select All	Selects the entire file.
	Options	Enables an operating environment for the editor to be set up.
Debug	**Check Syntax**	Checks the syntax of the job definition script.
	Run to Breakpoint	Starts and restarts execution up to a breakpoint in the debug mode.
	Stop Script	Stops execution of the job definition script at the next line. Completes the command whose execution is underway when **Stop Script** is selected and stops the job definition script before the next command executes.
	Quit Debugging	Completes the command whose execution is underway when **Quit Debugging** is selected, stops execution before the next command executes, then stops the job definition script and cancels debugging.
	Step In	Executes the next command or statement in the debug mode. If a function is called, this menu item also executes one line in the function and then stops execution.
	Step Over	Executes the next command or statement in the debug mode. If a function is called, this menu item does not stop after executing one line in the function, but stops only when a breakpoint is reached.
	Step Out	Executes through the end of a function call and stops on the line immediately following the function call or at a breakpoint.
	Set or Remove Breakpoint	Sets a breakpoint or releases the selected breakpoint.
	Remove All Breakpoints	Releases all breakpoints that have been set.
	Runtime Environment Settings	Sets up a script file execution environment.

Menu		Description
Debug	**Add Variable to Watch List**	Adds a specified variable to the Watch List window.
	Fault Injection Mode	Enables or disables the fault injection mode while execution of the job definition script is stopped.
	Execute a trap action	Executes the `trap` command's action and continues processing up to a breakpoint.
	Jump to the currently executing line	Jumps to the line that is executing currently.
View	**Toolbar** > **Standard Toolbar**	Changes the view/hide setting for the standard toolbar.
	Toolbar > **Edit Toolbar**	Changes the view/hide setting for the edit toolbar.
	Toolbar > **Debug Toolbar**	Changes the view/hide setting for the debug toolbar.
	Toolbar > **Search Toolbar**	Changes the view/hide setting for the search toolbar.
	Status Bar	Changes the view/hide setting for the status bar.
	Align Toolbars	Aligns the toolbars.
	Ruler	Changes the view/hide setting for the ruler.
	Vertical Scrollbar	Changes the view/hide setting for the vertical scrollbar.
	Horizontal Scrollbar	Changes the view/hide setting for the horizontal scrollbar.
	Show Line Numbers	Changes the view/hide settings for the line number.
	Show First Line	Displays the first line of the job definition script file.
	Show Last Line	Displays the last line of the job definition script file.
	Show Watch List	Changes the view/hide setting for the Watch List window.
	Show Error List	Changes the view/hide setting for the Error List window.
	View Coverage Information	Displays coverage information during debugging.
Search	**Search for**	Enables entry of a character string to be searched for.
	Replace with	Enables entry of a character string that is to be searched for and a character string that is to replace the retrieved character string.
	Find Previous	Searches for the search character string in the up direction.
	Find Next	Searches for the search character string in the down direction.
Help	**Open Help**	Displays Help for JP1/Advanced Shell.
	About	Displays program information, version, and copyright information.

(2) Pop-up menus

Clicking the mouse's right button while the cursor is in the client area of the JP1/Advanced Shell Editor window displays a pop-up menu. The pop-up menu's contents depend on whether the mode is the edit mode or the debug mode.

- Pop-up menu in the edit mode

  The following table lists and describes the pop-up menu items that are displayed in the edit mode.

Pop-up menu item	Description
New	Creates a job definition script file.
Open	Enables an existing job definition script file to be opened.

Pop-up menu item	Description
Save	Saves the job definition script file being edited.
Undo	Undoes the previous operation.
Redo	Re-executes the previous operation.
Cut	Cuts a selection and saves it on the clipboard.
Copy	Copies a selection to the clipboard.
Paste	Pastes the contents of the clipboard to the selected location.
Select All	Selects the entire file.

- Pop-up menu in the debug mode
  The following table lists and describes the pop-up menu items that are displayed in the debug mode.

Pop-up menu item	Description
Copy	Copies a selection to the clipboard.
Set/Remove Breakpoint	Sets a breakpoint or releases the selected breakpoint.
Add Variable to Watch List	Adds a selected variable to the Watch List window.

4.3.3 Mouse and key operations in the JP1/Advanced Shell Editor window

This subsection explains the mouse and key operations in the JP1/Advanced Shell Editor window.

(1) Mouse operations

The following table describes the mouse operations in the client area of the JP1/Advanced Shell Editor window.

Table 4–3: Mouse operations in the JP1/Advanced Shell Editor window

Operation	Description
Click	Selects a target for an operation or releases an existing selection.
Double-click	Selects a character string.
Right-click	Displays a pop-up menu.

(2) Key operations

The following table describes the key operations while the cursor is positioned in the client area of the JP1/Advanced Shell Editor window and indicates the modes in which each operation is applicable.

Table 4–4: Key operations in the JP1/Advanced Shell Editor window

Operation	Edit mode	Debug mode	Description
Ctrl+A	Y	Y	Selects the entire file.
Ctrl+C	Y	Y	Copies a selection.
Ctrl+E	Y	N	Sets up a script file execution environment.
Ctrl+F	Y	P	Enables entry of a character string to be searched for.

Operation	Edit mode	Debug mode	Description
Ctrl+H	Y	N	Enables entry of a character string to be searched for and a character string that is to replace the retrieved character string.
Ctrl+N	Y	N	Creates a job definition script file.
Ctrl+O	Y	N	Enables an existing job definition script file to be opened.
Ctrl+P	Y	N	Prints the job definition script file being edited.
Ctrl+S	Y	N	Saves the job definition script file being edited.
Ctrl+V	Y	N	Pastes the contents of the clipboard to the selected location.
Ctrl+X	Y	N	Cuts a selection.
Ctrl+Z	Y	N	Undoes the previous operation.
Ctrl+Home	Y	Y	Displays the first line of the job definition script file.
Ctrl+End	Y	Y	Displays the last line of the job definition script file.
F1	Y	Y	Displays Help for JP1/Advanced Shell.
F3	Y	P	Searches for a character string in the down direction.
F5	Y	N	Starts and restarts execution up to a breakpoint.
F7	Y	N	Checks the syntax of the job definition script.
F9	Y	Y	Sets a breakpoint or releases the selected breakpoint.
F11	Y	Y	Executes the next command or statement. If a function is called, this key also executes one line in the function and then stops execution.
Alt+0	N	Y	Adds a specified variable to the Watch List window.
Alt+1	N	Y	Changes the view/hide setting for the Watch List window.
Alt+2	N	Y	Changes the view/hide setting for the Error List window.
Alt+F4	Y	Y	Exits JP1/Advanced Shell Editor and enables whether the file is to be saved to be specified.
Shift+F3	Y	P	Searches for a character string in the up direction.
Shift+F5	N	Y	Terminates the job definition script and cancels debugging.
Shift+F9	Y	Y	Releases all breakpoints that have been set.
Shift+F11	Y	Y	Executes through the end of a function call and stops on the line immediately following the function call or at a breakpoint.
Shift+Ctrl+F11	Y	Y	Executes the next command or statement. If a function is called, this key does not stop after executing one line in the function, but stops only when a breakpoint is reached.
Shift+Ctrl+Z	Y	N	Re-executes the previous operation.
Enter	Y	N	Creates a new line by copying the spaces and tabs from the beginning of the selected line. If an end-of-line code is entered after `{`, this key adds a tab to the next line and `}` on the following line.

Legend:

Y: Applicable

P: Partially applicable

N: Not applicable

4.4 Creating job definition scripts (Windows only)

This section explains how to create job definition scripts in JP1/Advanced Shell Editor.

4.4.1 Creating job definition scripts

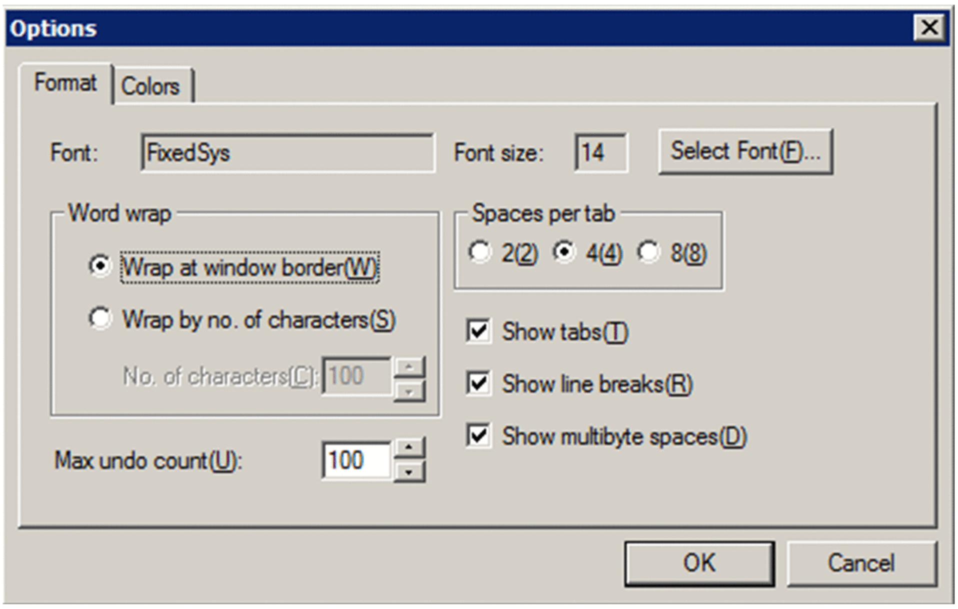
To create a job definition script:

1. From the **File** menu, select **New**.
   A new JP1/Advanced Shell Editor window is displayed.

4.4.2 Setting up an operating environment for the editor

To set up an operating environment for the editor:

1. From the **Edit** menu, select **Options**.
   The Options (Format) dialog box is displayed.



Options

Format | Colors

Font: FixedSys

Font size: 14

Select Font(F)...

Word wrap
- ● Wrap at window border(W)
- ○ Wrap by no. of characters(S)
- No. of characters(C): 100

Spaces per tab: ○ 2(2) ● 4(4) ○ 8(8)

☑ Show tabs(T)

☑ Show line breaks(R)

☑ Show multibyte spaces(D)

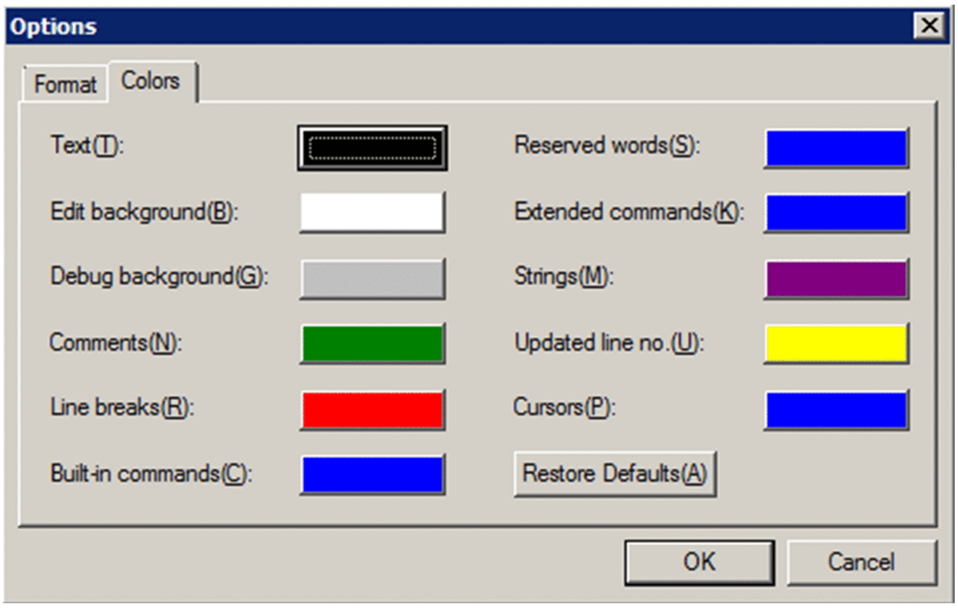
Max undo count(U): 100

OK | Cancel

   For details about how to specify settings in this dialog box, see *4.7.1 Options (Format) dialog box*.

2. Specify the format-related settings.

3. Click the **Colors** tab.
   The Options (Colors) dialog box is displayed.



4. Specify the display color information.
   To reset the display colors to the defaults, click the **Reset to Default** button.
   For details about how to specify settings in this dialog box, see *4.7.2 Options (Colors) dialog box*.

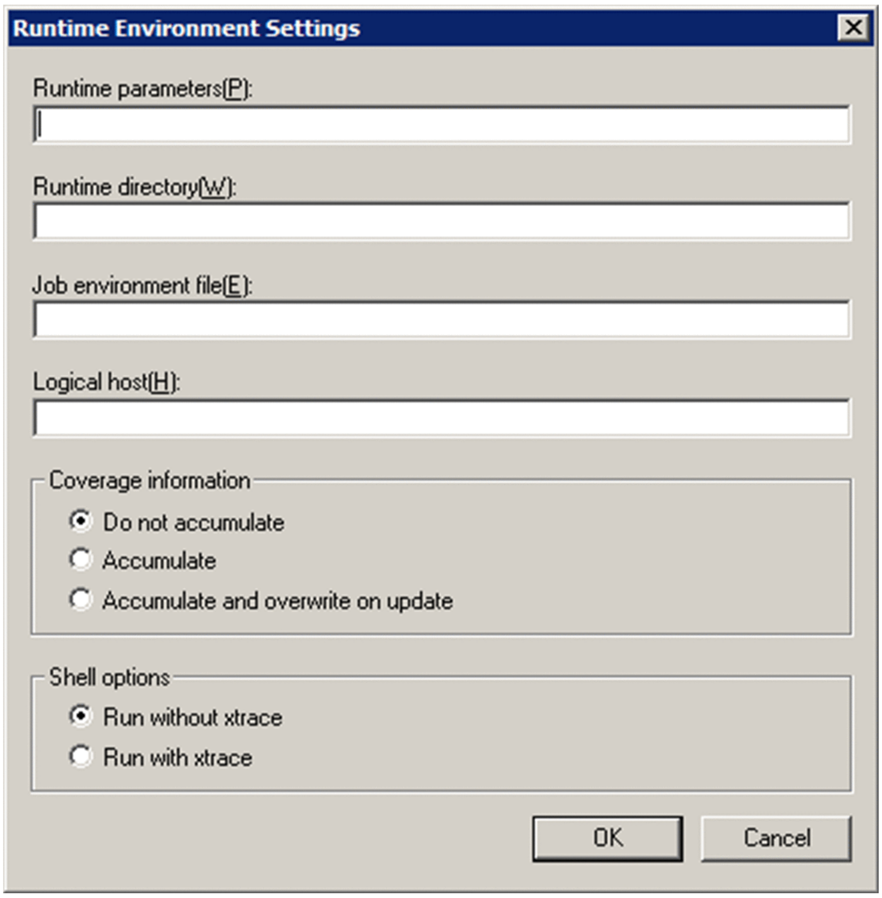
5. Click the **OK** button.
   An operating environment is now set up for the editor, and the dialog box closes.

4.4.3 Setting up an execution environment for job definition scripts

You can specify for each job definition script file run-time parameters, a run-time directory, a job environment file, and a logical host. The specified information is stored in the debugging information file.

To set up an execution environment for a job definition script:

1. From the **Debug** menu, select **Runtime Environment Settings**.
   The Runtime Environment Settings dialog box is displayed.



Runtime parameters(P):

Runtime directory(W):

Job environment file(E):

Logical host(H):

Coverage information
● Do not accumulate
○ Accumulate
○ Accumulate and overwrite on update

Shell options
● Run without xtrace
○ Run with xtrace

For details about how to specify settings in this dialog box, see *4.7.3 Runtime Environment Settings dialog box*.

2. Click the **OK** button.
   An execution environment is now set up, and the dialog box closes.
   If **Do not accumulate** is selected in **Coverage information**, coverage information will not be collected.

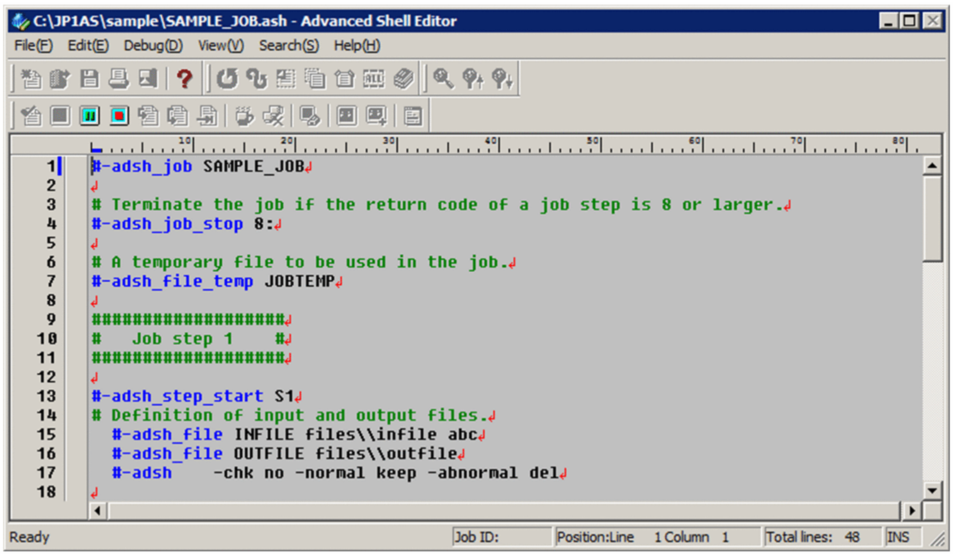
4.4.4 Checking syntax

You can check the syntax of a job definition script file. The editor checks for any syntax errors, but does not execute the script. Coverage information is not collected even if the option for accumulating coverage information is specified (this corresponds to the `-c` option in the `adshexec` command).

The console is not displayed. Errors are displayed in the Error List window.

To check syntax:

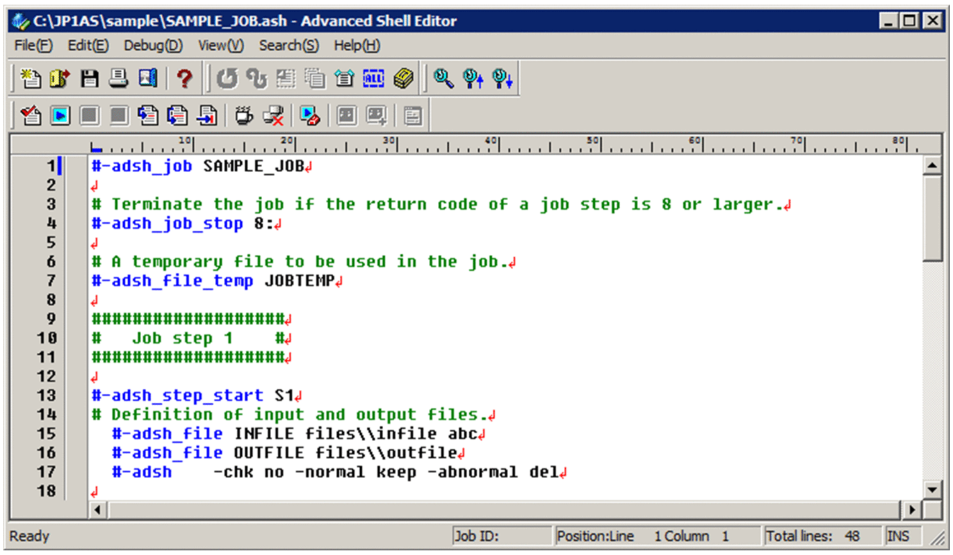
1. From the **Debug** menu, select **Check Syntax**.
   The editor is placed in the debug mode and starts syntax checking.
   The window will be grayed out while syntax checking is underway.
   - Display during syntax checking


C:\JP1AS\sample\SAMPLE_JOB.ash - Advanced Shell Editor
File(F) Edit(E) Debug(D) View(V) Search(S) Help(H)
Ready
Job ID:
Position:Line 1 Column 1
Total lines: 48
INS

```
 1  #-adsh_job SAMPLE_JOB
 2
 3  # Terminate the job if the return code of a job step is 8 or larger.
 4  #-adsh_job_stop 8:
 5
 6  # A temporary file to be used in the job.
 7  #-adsh_file_temp JOBTEMP
 8
 9  ###################
10  #   Job step 1    #
11  ###################
12
13  #-adsh_step_start S1
14  # Definition of input and output files.
15    #-adsh_file INFILE files\\infile abc
16    #-adsh_file OUTFILE files\\outfile
17    #-adsh     -chk no -normal keep -abnormal del
18
```

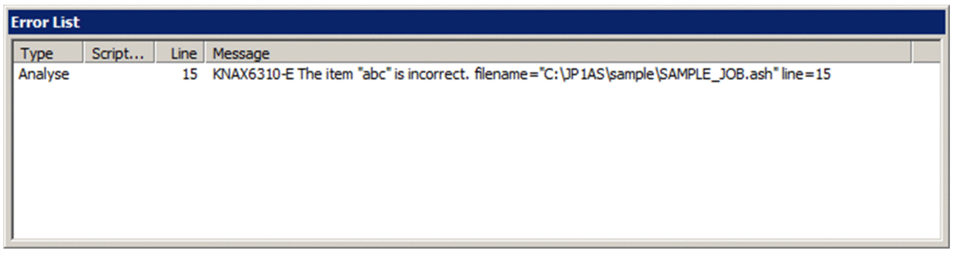
- Display when syntax checking has been completed

  If any syntax errors are detected, information about the errors is displayed in the Error List window.


C:\JP1AS\sample\SAMPLE_JOB.ash - Advanced Shell Editor
File(F) Edit(E) Debug(D) View(V) Search(S) Help(H)
Ready
Job ID:
Position:Line 1 Column 1
Total lines: 48
INS

```
 1  #-adsh_job SAMPLE_JOB
 2
 3  # Terminate the job if the return code of a job step is 8 or larger.
 4  #-adsh_job_stop 8:
 5
 6  # A temporary file to be used in the job.
 7  #-adsh_file_temp JOBTEMP
 8
 9  ###################
10  #   Job step 1    #
11  ###################
12
13  #-adsh_step_start S1
14  # Definition of input and output files.
15    #-adsh_file INFILE files\\infile abc
16    #-adsh_file OUTFILE files\\outfile
17    #-adsh     -chk no -normal keep -abnormal del
18
```

2. Check the information displayed in the Error List window.



Type	Script...	Line	Message
Analyse		15	KNAX6310-E The item "abc" is incorrect. filename="C:\JP1AS\sample\SAMPLE_JOB.ash" line=15

For details about the Error List window, see *4.7.4 Error List window*.

Notes:

- In the debug mode, the menus are grayed out and **Check Syntax** cannot be selected from the **Debug** menu.
- If you attempt to perform a syntax check on a job definition script file that does not yet have a name, the Save As dialog box for specifying a name for the file and saving it will be displayed. A syntax check cannot be performed for such a file until it is saved with a new job definition script file name (`.ash`).

- If the contents of the job definition script file have changed, a message asking whether the file is to be updated is displayed. To update the file, save it, and then perform the syntax check.

4.4.5 Searching for and replacing character strings

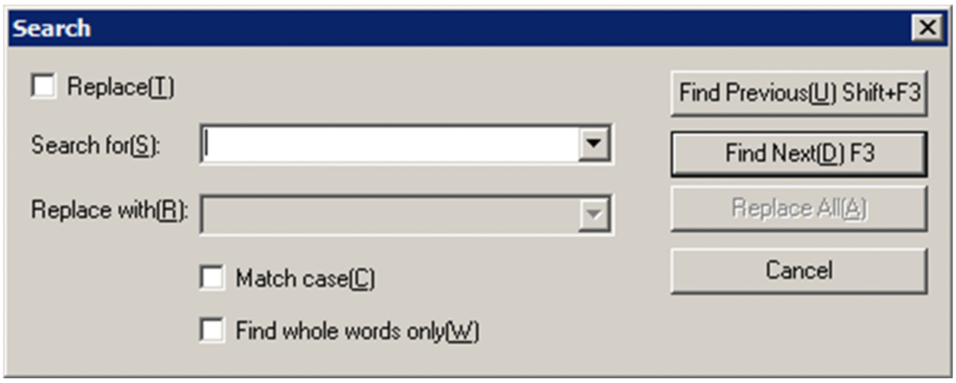
This subsection explains how to search for and replace character strings in job definition script files.

(1) Searching for character strings

To search a job definition script file for a character string:

1. From the **Search** menu, select Search.

   The Search dialog box is displayed.

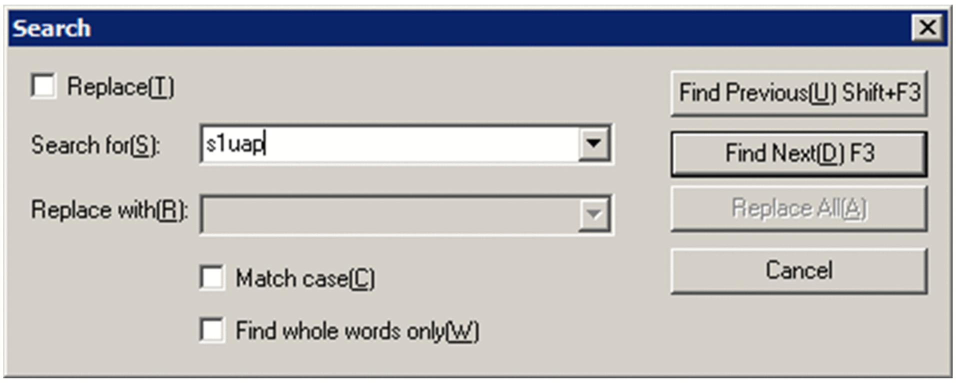
   

   For details about how to specify settings in this dialog box, see *4.7.5 Search dialog box*.

2. Make sure that the **Replace** check box is not selected.

   If the **Replace** check box is selected, clear it.

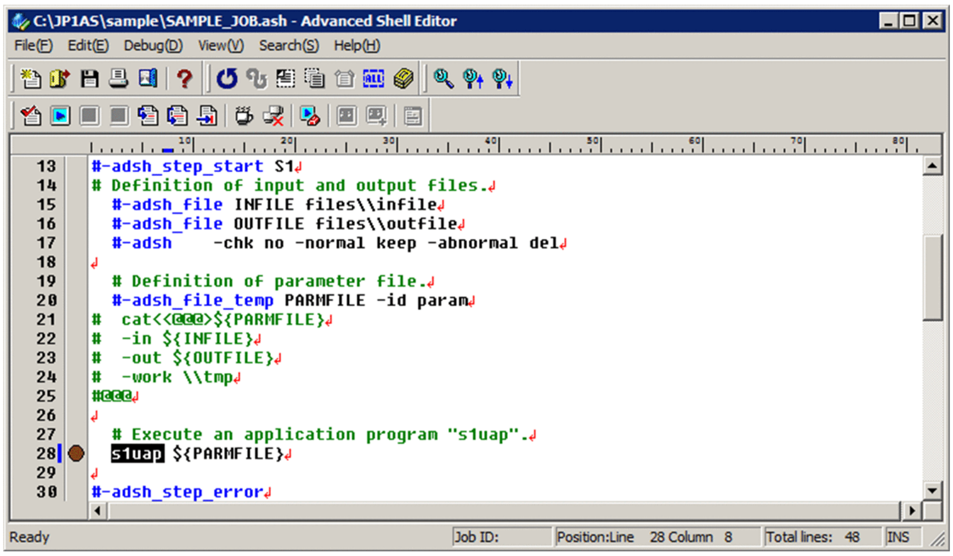
3. In **Search for**, enter the character string to be searched for. If necessary, select the **Match case** and **Find whole words only** check boxes.

   Search for(S): s1uap

4. Click the **Find Previous** or **Find Next** button.

   The specified character string is searched for. If there is no matching character string, the editor sounds a beep tone.


C:\JP1AS\sample\SAMPLE_JOB.ash - Advanced Shell Editor
File(F) Edit(E) Debug(D) View(V) Search(S) Help(H)
Ready
Job ID:
Total lines: 48
INS

```
#-adsh_step_start S1
# Definition of input and output files.
  #-adsh_file INFILE files\\infile
  #-adsh_file OUTFILE files\\outfile
  #-adsh    -chk no -normal keep -abnormal del

  # Definition of parameter file.
  #-adsh_file_temp PARMFILE -id param
#  cat<<@@@>${PARMFILE}
#  -in ${INFILE}
#  -out ${OUTFILE}
#  -work \\tmp
#@@@

  # Execute an application program "s1uap".
  s1uap ${PARMFILE}

#-adsh_step_error
```

Position:Line 28 Column 8

5. To end the search, click the **Cancel** button.

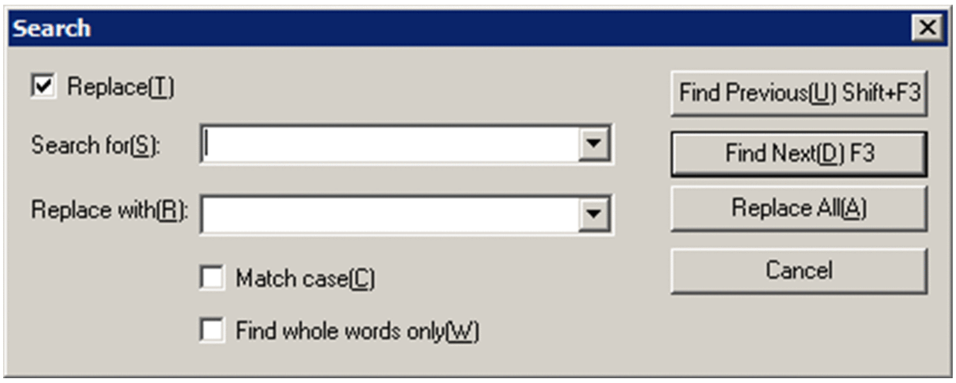
   The Search dialog box closes.

(2) Replacing character strings

To replace a character string in a job definition script file:

1. From the **Search** menu, select **Replace**.

   The Search dialog box is displayed.



☑ Replace(T)

Search for(S):

Replace with(R):

☐ Match case(C)

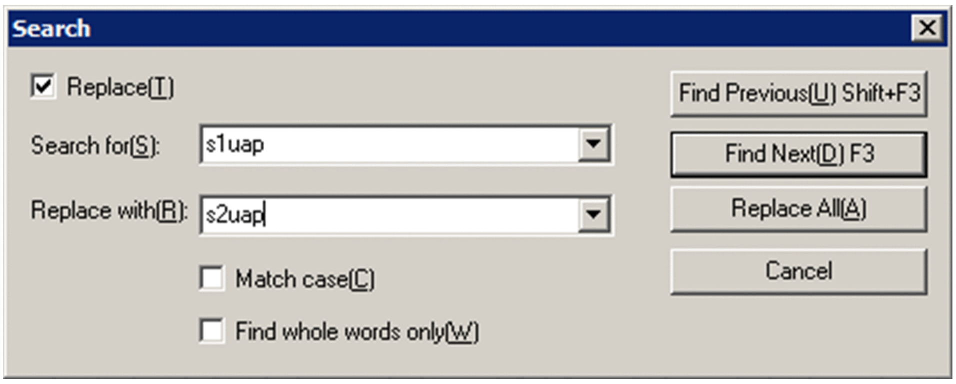
☐ Find whole words only(W)

   For details about how to specify settings in this dialog box, see *4.7.5 Search dialog box*.

2. Make sure that the **Replace** check box is selected.

   If the **Replace** check box is not selected, select it.

3. In **Search for**, enter the character string to be searched for and replaced. If necessary, select the **Match case** and **Find whole words only** check boxes.



☑ Replace(T)

Search for(S): s1uap

Replace with(R): s2uap

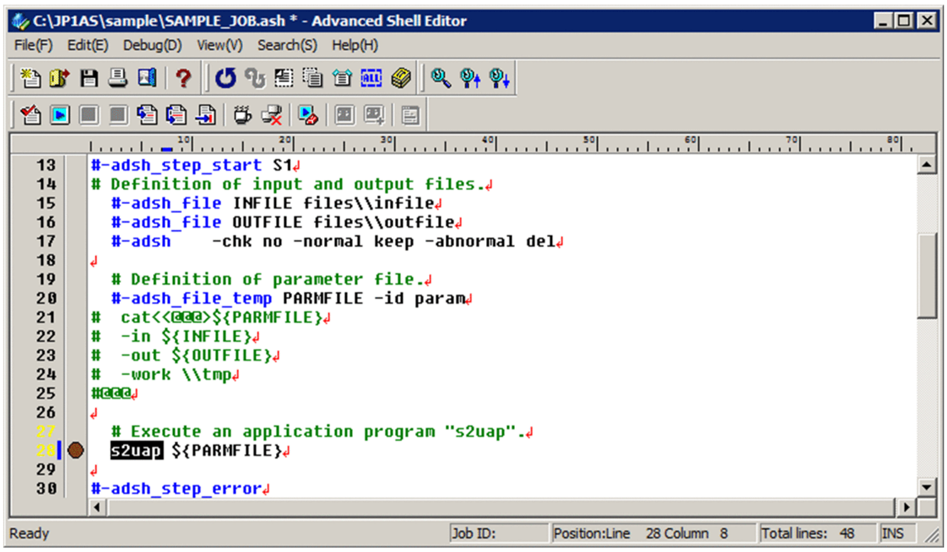
☐ Match case(C)

☐ Find whole words only(W)

4. In **Replace with**, enter the character string that is to replace the specified **Search for** character string.

5. Click the **Find Previous** or **Find Next** button.

   The editor starts searching for the specified **Search for** character string and replacing it with the specified **Replace with** character string. If there is no character string to be replaced, the editor sounds a beep tone.


C:\JP1AS\sample\SAMPLE_JOB.ash * - Advanced Shell Editor
File(F) Edit(E) Debug(D) View(V) Search(S) Help(H)

```
#-adsh_step_start S1
# Definition of input and output files.
  #-adsh_file INFILE files\\infile
  #-adsh_file OUTFILE files\\outfile
  #-adsh    -chk no -normal keep -abnormal del

  # Definition of parameter file.
  #-adsh_file_temp PARMFILE -id param
#  cat<<@@@>${PARMFILE}
#  -in ${INFILE}
#  -out ${OUTFILE}
#  -work \\tmp
#@@@

  # Execute an application program "s2uap".
  s2uap ${PARMFILE}

#-adsh_step_error
```

Ready | Job ID: | Position:Line 28 Column 8 | Total lines: 48 | INS

6. To end the replace operation, click the **Cancel** button.

   The Search dialog box closes.

4.4.6 Debugging

Checking the behavior of a job definition script file is called debugging.

This is equivalent to the `adshexec` command with the `-d` option specified. The console is displayed while debugging is being performed. Error messages are displayed in the Error List window.

There are two ways to perform debugging:

Method	Operation	Overview
Execution	From the **Debug** menu, select **Run to Breakpoint**.	Starts and restarts execution up to a breakpoint.
Step-by-step execution	From the **Debug** menu, select **Step In**.	Executes one line of the job definition script, and then stops execution. If a function is

Method	Operation	Overview
Step-by-step execution	From the **Debug** menu, select **Step In**.	called, the editor also executes one line in the function and stops execution.
	From the **Debug** menu, select **Step Over**.	Executes one line of the job definition script and then stops execution. If a function is called, the editor does not stop after executing a single line in the function, but stops only when a breakpoint is reached.
	From the **Debug** menu, select **Step Out**.	Stops on the line immediately following a function call or at a breakpoint.

Notes:

- If you attempt to debug a job definition script file that does not yet have a name, the Save As dialog box for specifying a name for the file and saving it will be displayed. A file cannot be debugged until it is saved with a new job definition script file name (`.ash`).
- If the contents of the job definition script file have changed, a message asking whether the file is to be updated is displayed. If you update the file, you can then debug it.
- If the editor is forcibly terminated during debugging (because, for example, **End now** is selected in the Exit the Program dialog box), the debugger's `adshesub.exe` process might keep running and the console might remain displayed. If this happens, terminate the `adshesub.exe` process with the `taskkill` command or from the task manager.

(1) Setting and releasing breakpoints during debugging

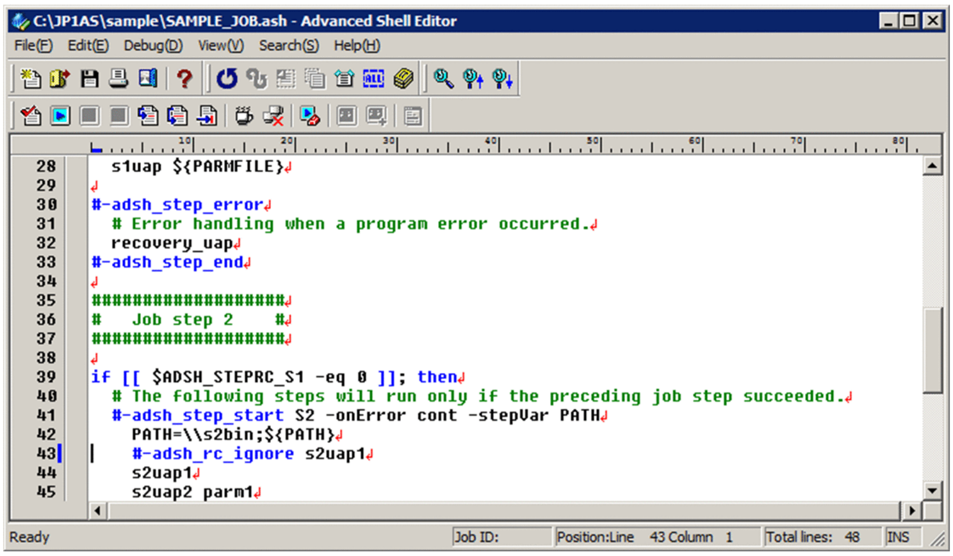
You set breakpoints at locations where you want execution to stop temporarily during debugging. You can also release breakpoints that have been set.

Because JP1/Advanced Shell Editor sets a breakpoint at the line where the cursor is located, breakpoints cannot be set in external scripts. Even if breakpoints have already been set in an external script, execution will not stop at such breakpoints. You can set a maximum of 999 breakpoints.

(a) Setting breakpoints

To set a breakpoint:

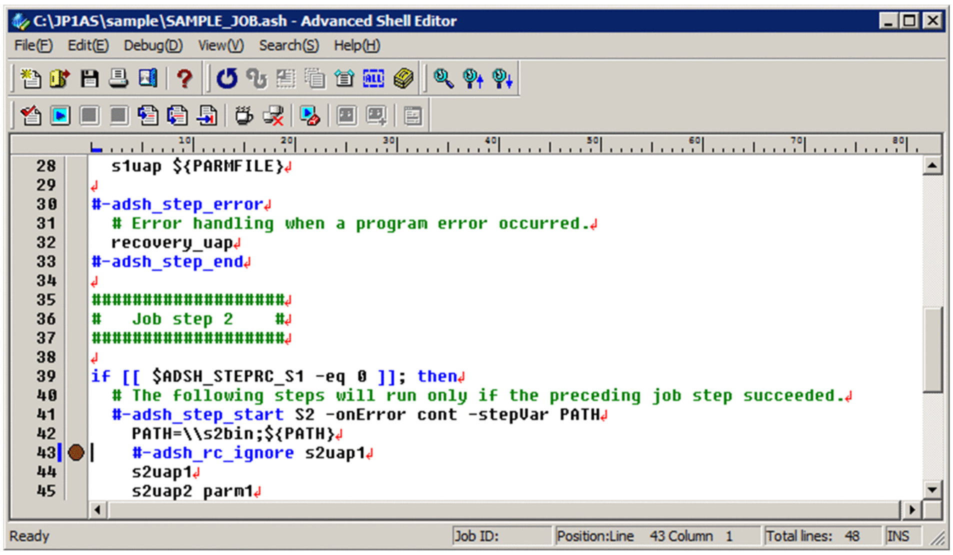
1. Move the cursor to the line where you want to set a breakpoint.


C:\JP1AS\sample\SAMPLE_JOB.ash - Advanced Shell Editor
File(F) Edit(E) Debug(D) View(V) Search(S) Help(H)
Ready
Job ID:
Position:Line 43 Column 1
Total lines: 48
INS

```
  s1uap ${PARMFILE}

#-adsh_step_error
  # Error handling when a program error occurred.
  recovery_uap
#-adsh_step_end

####################
#   Job step 2    #
####################

if [[ $ADSH_STEPRC_S1 -eq 0 ]]; then
  # The following steps will run only if the preceding job step succeeded.
  #-adsh_step_start S2 -onError cont -stepVar PATH
    PATH=\\s2bin;${PATH}
    #-adsh_rc_ignore s2uap1
    s2uap1
    s2uap2 parm1
```

2. From the **Debug** menu, select **Set Breakpoint**.

   A breakpoint is set at the line where the cursor is located. The ● symbol is displayed on the left end of the line to indicate that a breakpoint has been set. The job definition script executes up to, but not including, the line where the breakpoint is set and then stops.

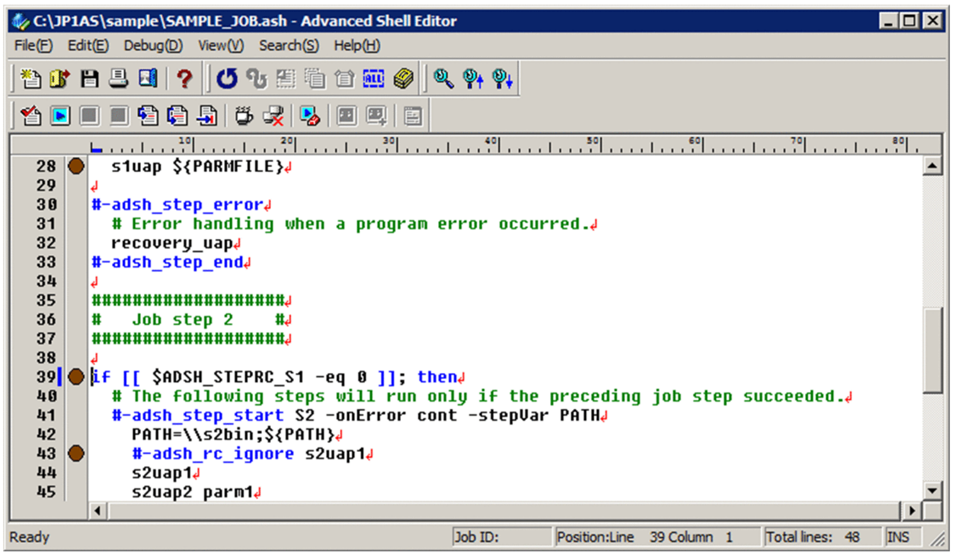

C:\JP1AS\sample\SAMPLE_JOB.ash - Advanced Shell Editor
File(F) Edit(E) Debug(D) View(V) Search(S) Help(H)
Ready
Job ID:
Position:Line 43 Column 1
Total lines: 48
INS

```
  s1uap ${PARMFILE}

#-adsh_step_error
  # Error handling when a program error occurred.
  recovery_uap
#-adsh_step_end

####################
#   Job step 2    #
####################

if [[ $ADSH_STEPRC_S1 -eq 0 ]]; then
  # The following steps will run only if the preceding job step succeeded.
  #-adsh_step_start S2 -onError cont -stepVar PATH
    PATH=\\s2bin;${PATH}
    #-adsh_rc_ignore s2uap1
    s2uap1
    s2uap2 parm1
```

(b) Removing a breakpoint

To remove a breakpoint:

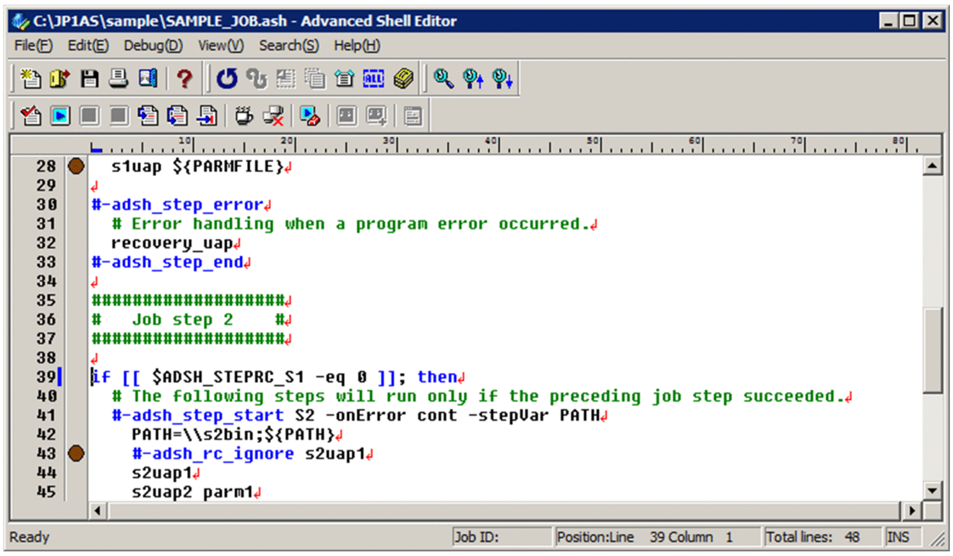
1. Move the cursor to the line where a breakpoint is to be removed.


C:\JP1AS\sample\SAMPLE_JOB.ash - Advanced Shell Editor
File(F) Edit(E) Debug(D) View(V) Search(S) Help(H)
Ready
Job ID:
Position:Line 39 Column 1
Total lines: 48
INS

```
  s1uap ${PARMFILE}

#-adsh_step_error
  # Error handling when a program error occurred.
  recovery_uap
#-adsh_step_end

####################
#   Job step 2    #
####################

if [[ $ADSH_STEPRC_S1 -eq 0 ]]; then
  # The following steps will run only if the preceding job step succeeded.
  #-adsh_step_start S2 -onError cont -stepVar PATH
    PATH=\\s2bin;${PATH}
    #-adsh_rc_ignore s2uap1
    s2uap1
    s2uap2 parm1
```

2. From the **Debug** menu, select **Remove Breakpoint**.

   The breakpoint is removed from the line where the cursor is located.

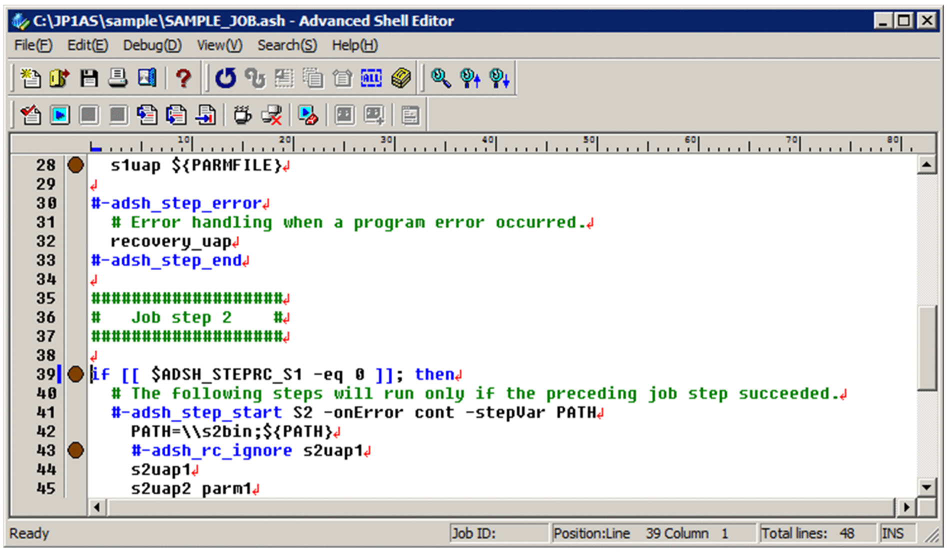

C:\JP1AS\sample\SAMPLE_JOB.ash - Advanced Shell Editor
File(F) Edit(E) Debug(D) View(V) Search(S) Help(H)
Ready
Job ID:
Position:Line 39 Column 1
Total lines: 48
INS

```
  s1uap ${PARMFILE}

#-adsh_step_error
  # Error handling when a program error occurred.
  recovery_uap
#-adsh_step_end

####################
#   Job step 2    #
####################

if [[ $ADSH_STEPRC_S1 -eq 0 ]]; then
  # The following steps will run only if the preceding job step succeeded.
  #-adsh_step_start S2 -onError cont -stepVar PATH
    PATH=\\s2bin;${PATH}
    #-adsh_rc_ignore s2uap1
    s2uap1
    s2uap2 parm1
```

(c) Removing all breakpoints

To remove all breakpoints:

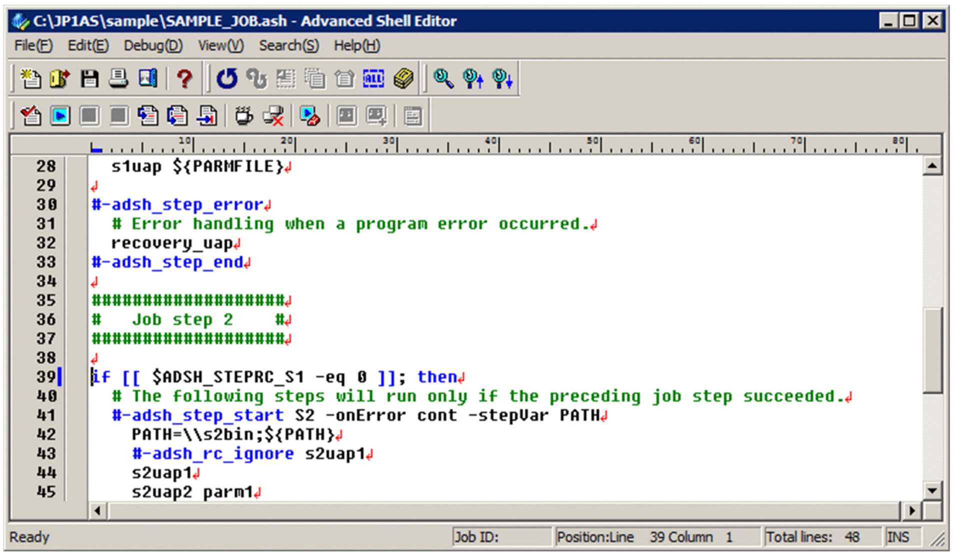
1. Display a job definition script in which breakpoints are set.



```
  s1uap ${PARMFILE}

#-adsh_step_error
  # Error handling when a program error occurred.
  recovery_uap
#-adsh_step_end

###################
#   Job step 2    #
###################

if [[ $ADSH_STEPRC_S1 -eq 0 ]]; then
  # The following steps will run only if the preceding job step succeeded.
  #-adsh_step_start S2 -onError cont -stepVar PATH
    PATH=\\s2bin;${PATH}
    #-adsh_rc_ignore s2uap1
    s2uap1
    s2uap2 parm1
```

Ready | Job ID: | Position:Line 39 Column 1 | Total lines: 48 | INS

2. From the **Debug** menu, select **Remove All Breakpoints**.
   All breakpoints are removed from the displayed job definition script file.



```
  s1uap ${PARMFILE}

#-adsh_step_error
  # Error handling when a program error occurred.
  recovery_uap
#-adsh_step_end

###################
#   Job step 2    #
###################

if [[ $ADSH_STEPRC_S1 -eq 0 ]]; then
  # The following steps will run only if the preceding job step succeeded.
  #-adsh_step_start S2 -onError cont -stepVar PATH
    PATH=\\s2bin;${PATH}
    #-adsh_rc_ignore s2uap1
    s2uap1
    s2uap2 parm1
```

Ready | Job ID: | Position:Line 39 Column 1 | Total lines: 48 | INS

Notes:

- You can set a breakpoint at any line while you are in the edit mode. In the debug mode, a breakpoint can be set only on a line for a command or a statement that is to be executed.
- If a breakpoint is set on a non-execution line, the editor searches downwards in the script file for an appropriate location for a breakpoint and sets a breakpoint on that line when the debug mode begins.
- You can set a maximum of 999 breakpoints.

(2) Performing and canceling debugging

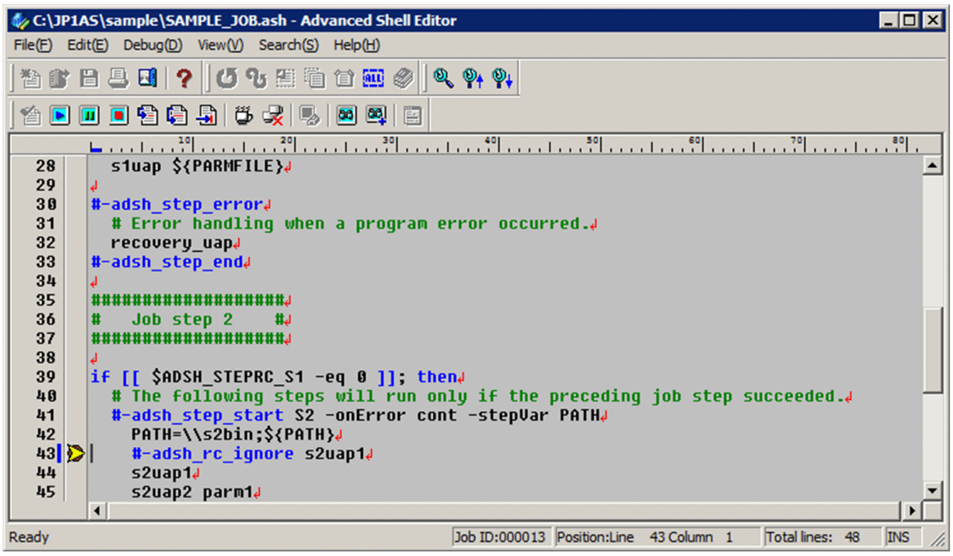
(a) Debugging up to a breakpoint

To debug up to a breakpoint:

1. From the **Debug** menu, select **Run to Breakpoint**, or on the toolbar, click the **Run to Breakpoint** button.

JP1/Advanced Shell Editor is placed in the debug mode and begins debugging. A symbol (▶) indicating the next location to be executed is displayed to the left of the corresponding line. Comment lines and lines containing only spaces are ignored.

Execution stops temporarily when the script has executed through the line preceding a line on which a breakpoint (●) is set.

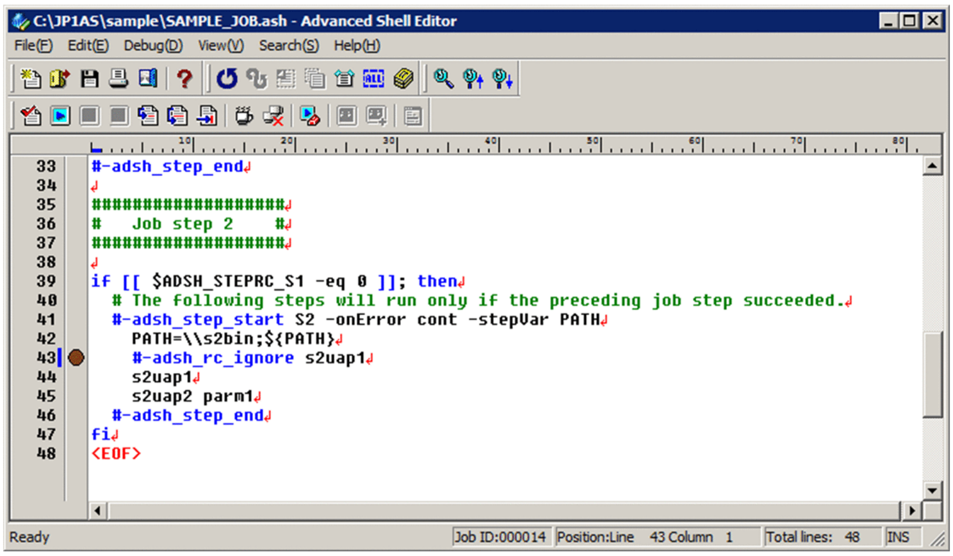

C:\JP1AS\sample\SAMPLE_JOB.ash - Advanced Shell Editor
File(F) Edit(E) Debug(D) View(V) Search(S) Help(H)
Ready
Job ID:000013
Position:Line 43 Column 1
Total lines: 48
INS

```
  s1uap ${PARMFILE}

#-adsh_step_error
  # Error handling when a program error occurred.
  recovery_uap
#-adsh_step_end

####################
#   Job step 2    #
####################

if [[ $ADSH_STEPRC_S1 -eq 0 ]]; then
  # The following steps will run only if the preceding job step succeeded.
  #-adsh_step_start S2 -onError cont -stepVar PATH
    PATH=\\s2bin;${PATH}
    #-adsh_rc_ignore s2uap1
    s2uap1
    s2uap2 parm1
```

For details about how to set breakpoints, see *(1) Setting and releasing breakpoints during debugging*.

2. To cancel debugging, from the **Debug** menu, select **Quit Debugging**, or on the toolbar, click the **Quit Debugging** button.

   The editor stops debugging when this selection is made, displays a message, performs postprocessing, and then terminates debugging. The editor terminates debugging without stopping even if the end of the process has not been reached, and returns to the edit mode.


C:\JP1AS\sample\SAMPLE_JOB.ash - Advanced Shell Editor
File(F) Edit(E) Debug(D) View(V) Search(S) Help(H)
Ready
Job ID:000014
Position:Line 43 Column 1
Total lines: 48
INS

```
#-adsh_step_end

####################
#   Job step 2    #
####################

if [[ $ADSH_STEPRC_S1 -eq 0 ]]; then
  # The following steps will run only if the preceding job step succeeded.
  #-adsh_step_start S2 -onError cont -stepVar PATH
    PATH=\\s2bin;${PATH}
    #-adsh_rc_ignore s2uap1
    s2uap1
    s2uap2 parm1
  #-adsh_step_end
fi
<EOF>
```

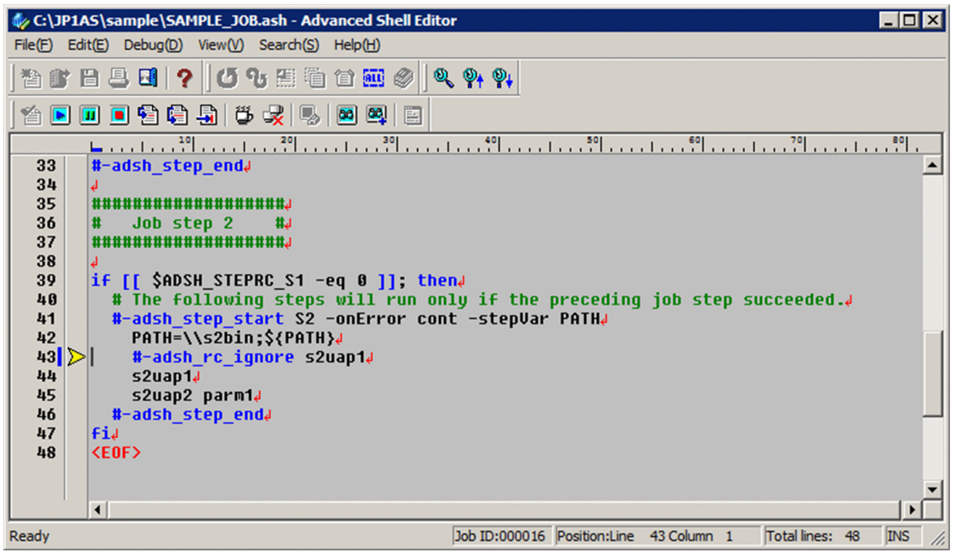
(b) Executing one line at a time (performing step-by-step execution in functions)

To execute one line at time and perform step-by-step execution in functions:

1. From the **Debug** menu, select **Step In**, or on the toolbar, click the **Step In** button.

The editor is placed in the debug mode and begins debugging. A symbol (▷) indicating the next location to be executed is displayed to the left of the corresponding line. Comment lines and lines containing only spaces are ignored. The editor also executes only one line at a time inside functions. When the job definition script has been executed through the last line, a symbol (▶) indicating the end of the debugger process is displayed.

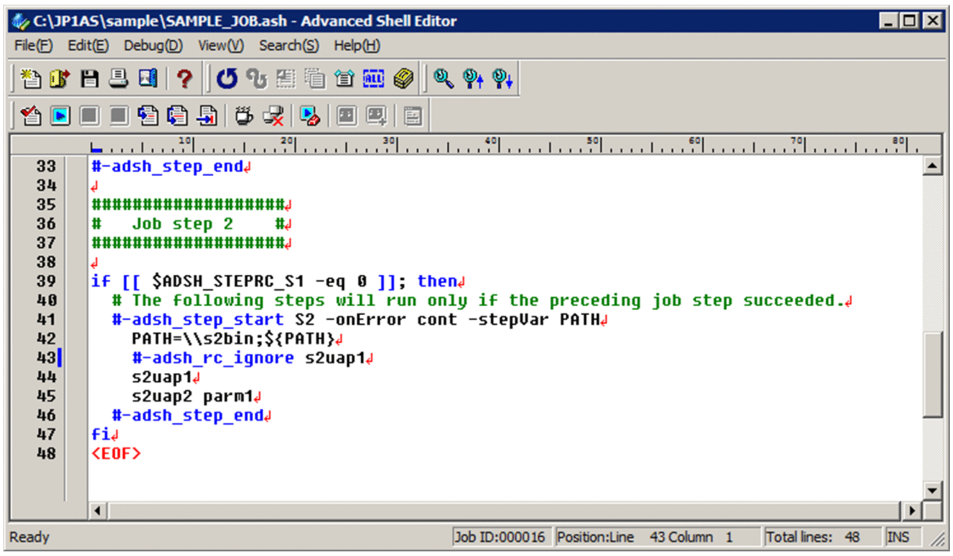
Unlike CUI, when an external script is executed, execution does not stop within the external script. Execution stops when it reaches the next command in the job definition script being displayed by the editor.


C:\JP1AS\sample\SAMPLE_JOB.ash - Advanced Shell Editor
File(F) Edit(E) Debug(D) View(V) Search(S) Help(H)
Ready
Job ID:000016
Position:Line 43 Column 1
Total lines: 48
INS

```
#-adsh_step_end

####################
#   Job step 2     #
####################

if [[ $ADSH_STEPRC_S1 -eq 0 ]]; then
  # The following steps will run only if the preceding job step succeeded.
  #-adsh_step_start S2 -onError cont -stepVar PATH
    PATH=\\s2bin;${PATH}
    #-adsh_rc_ignore s2uap1
    s2uap1
    s2uap2 parm1
  #-adsh_step_end
fi
<EOF>
```

2. To cancel debugging, from the **Debug** menu, select **Quit Debugging**, or on the toolbar, click the **Quit Debugging** button.

   The editor stops debugging when this selection is made, displays a message, performs postprocessing, and then terminates debugging. The editor terminates debugging without stopping even if the end of the process has not been reached, and returns to the edit mode.


C:\JP1AS\sample\SAMPLE_JOB.ash - Advanced Shell Editor
File(F) Edit(E) Debug(D) View(V) Search(S) Help(H)
Ready
Job ID:000016
Position:Line 43 Column 1
Total lines: 48
INS

```
#-adsh_step_end

####################
#   Job step 2     #
####################

if [[ $ADSH_STEPRC_S1 -eq 0 ]]; then
  # The following steps will run only if the preceding job step succeeded.
  #-adsh_step_start S2 -onError cont -stepVar PATH
    PATH=\\s2bin;${PATH}
    #-adsh_rc_ignore s2uap1
    s2uap1
    s2uap2 parm1
  #-adsh_step_end
fi
<EOF>
```

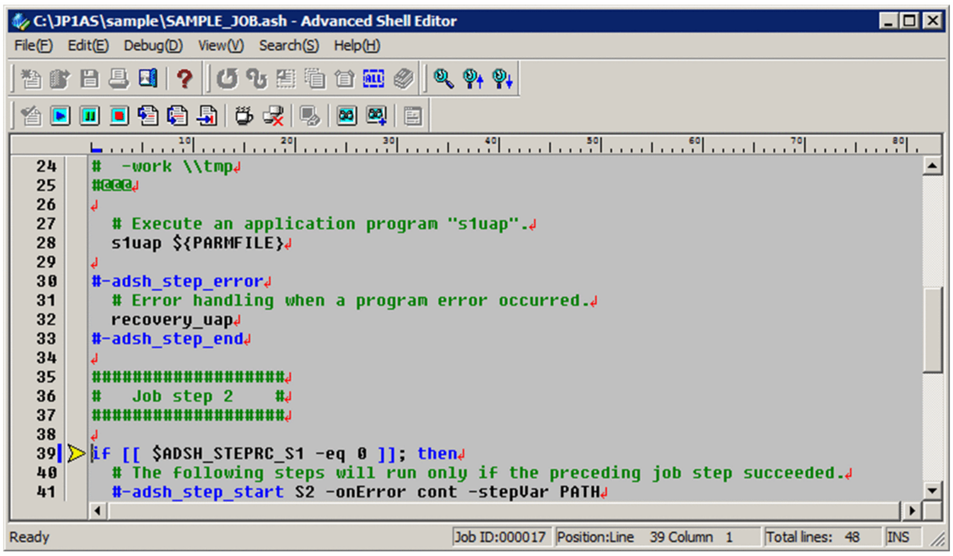
(c) Executing one line at a time (not performing step-by-step execution in functions)

To execute one line at a time and not perform step-by-step execution in functions:

1. From the **Debug** menu, select **Step Over**, or on the toolbar, click the **Step Over** button.

The editor is placed in the debug mode and begins debugging. A symbol (▷) indicating the next location to be executed is displayed to the left of the corresponding line. Comment lines and lines containing only spaces are ignored. When the job definition script has been executed through the last line, a symbol (▶) indicating the end of the debugger process is displayed.

Unlike CUI, when an external script is executed, execution does not stop within the external script. Execution stops when it reaches the next command in the job definition script being displayed by the editor.

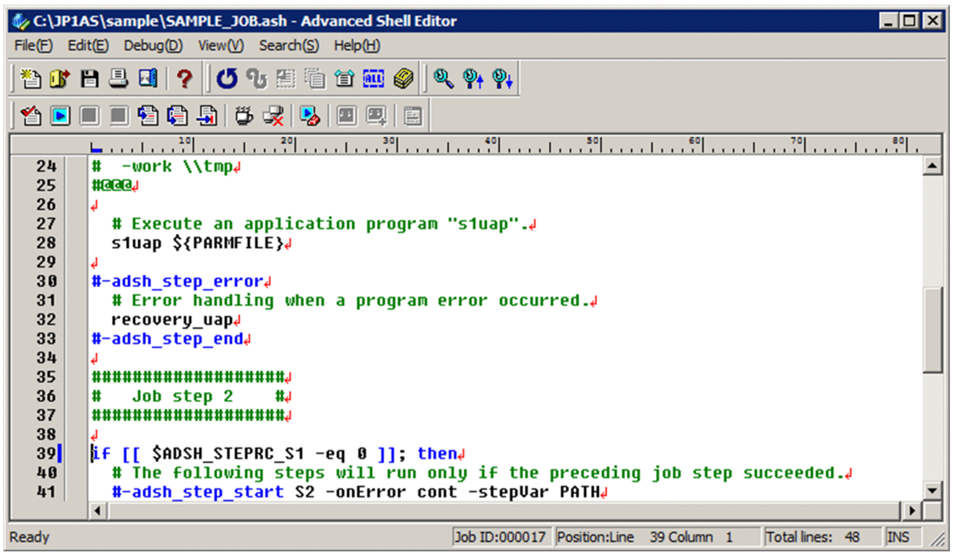

C:\JP1AS\sample\SAMPLE_JOB.ash - Advanced Shell Editor
File(F) Edit(E) Debug(D) View(V) Search(S) Help(H)

```
#  -work \\tmp
#@@@

  # Execute an application program "s1uap".
  s1uap ${PARMFILE}

#-adsh_step_error
  # Error handling when a program error occurred.
  recovery_uap
#-adsh_step_end

###################
#   Job step 2    #
###################

if [[ $ADSH_STEPRC_S1 -eq 0 ]]; then
  # The following steps will run only if the preceding job step succeeded.
  #-adsh_step_start S2 -onError cont -stepVar PATH
```

Ready | Job ID:000017 | Position:Line 39 Column 1 | Total lines: 48 | INS

2. To cancel debugging, from the **Debug** menu, select **Quit Debugging**, or on the toolbar, click the **Quit Debugging** button.

   The editor stops debugging when this selection is made, displays a message, performs postprocessing, and then terminates debugging. The editor terminates debugging without stopping even if the end of the process has not been reached, and returns to the edit mode.


C:\JP1AS\sample\SAMPLE_JOB.ash - Advanced Shell Editor
File(F) Edit(E) Debug(D) View(V) Search(S) Help(H)

```
#  -work \\tmp
#@@@

  # Execute an application program "s1uap".
  s1uap ${PARMFILE}

#-adsh_step_error
  # Error handling when a program error occurred.
  recovery_uap
#-adsh_step_end

###################
#   Job step 2    #
###################

if [[ $ADSH_STEPRC_S1 -eq 0 ]]; then
  # The following steps will run only if the preceding job step succeeded.
  #-adsh_step_start S2 -onError cont -stepVar PATH
```

Ready | Job ID:000017 | Position:Line 39 Column 1 | Total lines: 48 | INS

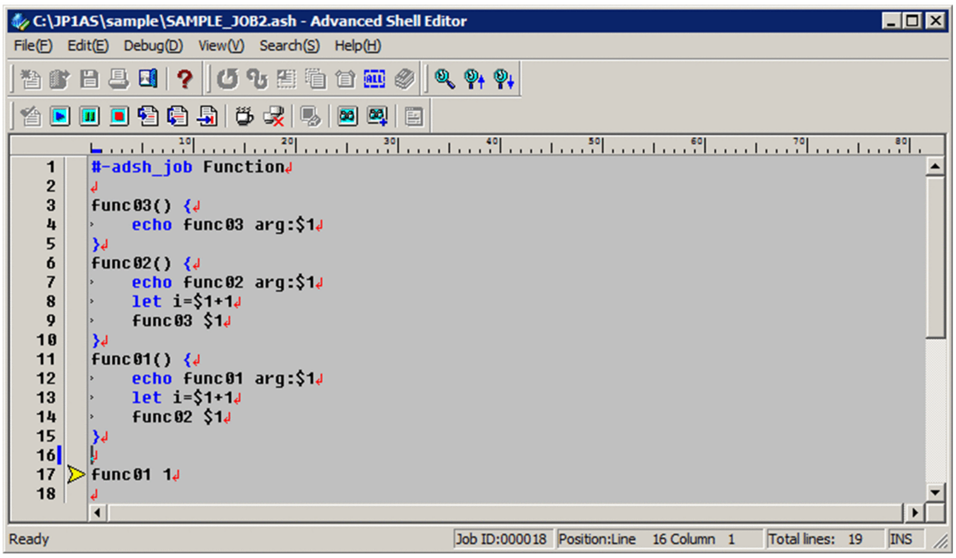
(d) Executing through the end of a function

To execute through the end of a function:

1. From the **Debug** menu, select **Step Out**, or on the toolbar, click the **Step Out** button.

The editor is placed in the debug mode and begins debugging. A symbol (▷) indicating the next location to be executed is displayed to the left of the corresponding line. Comment lines and lines containing only spaces are ignored. When the job definition script has been executed through the last line, a symbol (▶) indicating the end of the debugger process is displayed.

Unlike CUI, execution continues through the end of the job definition script even when control is not inside a function. However, if there is a breakpoint before control is returned from the function, execution stops.



```
#-adsh_job Function

func03() {
    echo func03 arg:$1
}
func02() {
    echo func02 arg:$1
    let i=$1+1
    func03 $1
}
func01() {
    echo func01 arg:$1
    let i=$1+1
    func02 $1
}

func01 1

```

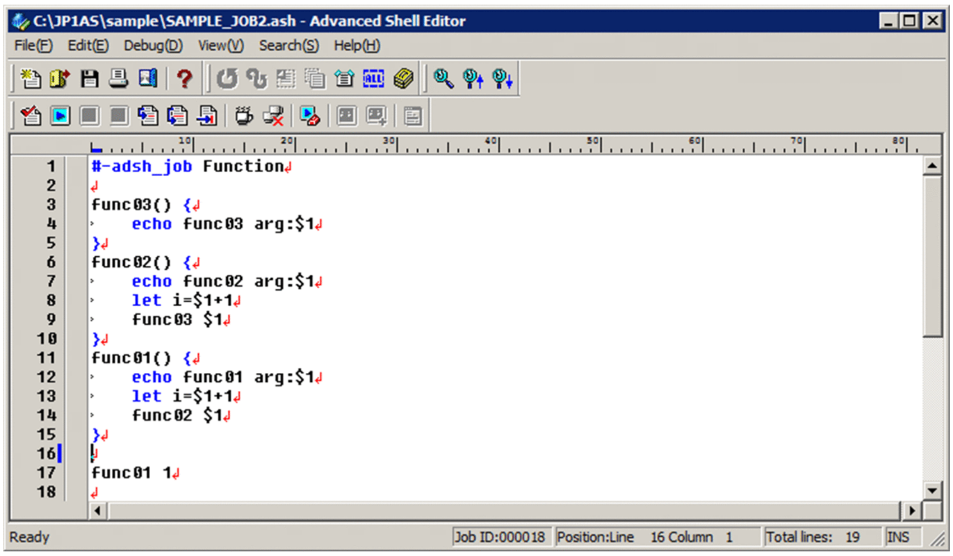
Ready | Job ID:000018 | Position:Line 16 Column 1 | Total lines: 19 | INS

2. To stop executing the job definition script, from the **Debug** menu, select **Stop Script**, or on the toolbar, click the **Stop Script** button.

   The command whose execution is underway when the **Stop Script** button is clicked is completed, and the job definition script stops before the next command is executed.

3. To cancel debugging, from the **Debug** menu, select **Quit Debugging**, or on the toolbar, click the **Quit Debugging** button.

   The editor stops debugging when this selection is made, displays a message, performs postprocessing, and then terminates debugging. The editor terminates debugging without stopping even if the end of the process has not been reached, and returns to the edit mode.



```
#-adsh_job Function

func03() {
    echo func03 arg:$1
}
func02() {
    echo func02 arg:$1
    let i=$1+1
    func03 $1
}
func01() {
    echo func01 arg:$1
    let i=$1+1
    func02 $1
}

func01 1

```

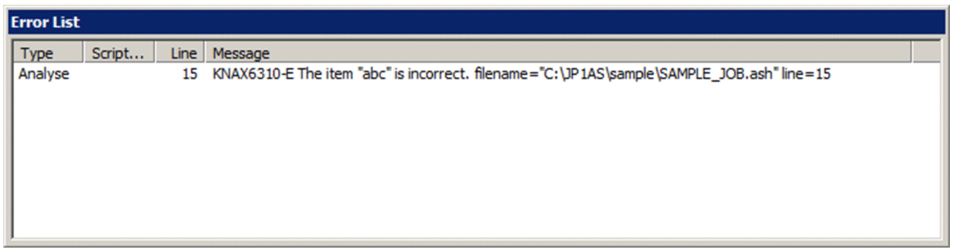
Ready | Job ID:000018 | Position:Line 16 Column 1 | Total lines: 19 | INS

(e) Error List window, Watch List window, and console during debugging

- Error List window

  Analysis and execution errors detected during debugging are displayed in the Error List window. The values of variables added to the Watch List window are updated and displayed when execution stops.

  For details about the Error List window, see *4.7.4 Error List window*.

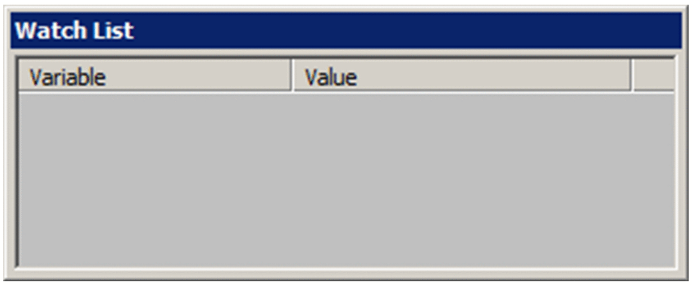


Type	Script...	Line	Message
Analyse		15	KNAX6310-E The item "abc" is incorrect. filename="C:\JP1AS\sample\SAMPLE_JOB.ash" line=15

- Watch List window

  The values of the variables displayed in the Watch List window are updated and displayed when execution of the job definition script stops. To display the Watch List window, while the job definition script is not running, select **Show Watch List** from the **View** menu or click the **Show Watch List** button on the toolbar. You can select this menu item only while the job definition script is not running. Double-clicking a variable name in the Watch List window displays the Edit Value dialog box, where you can update the variable's value.

  For details about the Watch List window, see *4.7.6 Watch List window*. For details about how to specify settings in the Edit Value dialog box, see *4.7.8 Edit Value dialog box*.



- Console

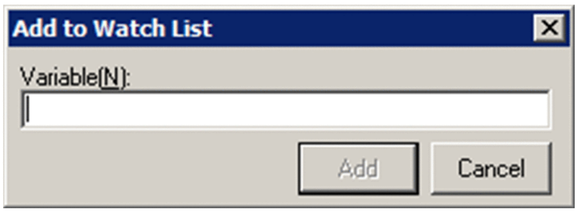
  During debugging, information equivalent to the standard output, standard error output, and job execution logs is output to the console. The console is displayed while a process executing a job definition script is running and closes when the process terminates. While a job definition script is executing, the console is active; while a job definition script is stopped, the editor is active.

(3) Adding variables to the watch list

To add a variable to the Watch List window:

1. In the debug mode, from the **Debug** menu, select **Add Variable to Watch List**.

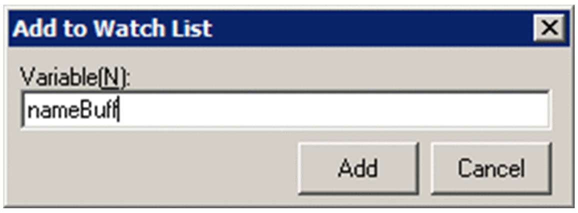
   The Add to Watch List dialog box is displayed.



   For details about how to specify a setting in this dialog box, see *4.7.7 Add to Watch List dialog box*.

2. Enter a name for the variable that you want to add.

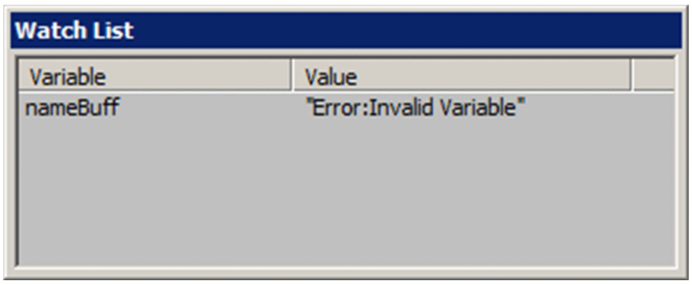
   You can enter a maximum of 99 bytes. A variable name longer than 99 bytes cannot be entered.



Variable(N): nameBuff

3. Click the **Add** button.

   The entered variable name is added to the Watch List window.



Variable	Value
nameBuff	"Error:Invalid Variable"

4. Click the **Cancel** button.

   The Add to Watch List dialog closes.

- In the debug mode, selecting a character string in the client area of the JP1/Advanced Shell Editor window, and then selecting **Debug** and then **Add Variable to Watch List**, sets the selected character string in **Variable name**.
- To delete an added variable from the Watch List window, right-click the variable name and select the **Delete from List** menu item.

(4) Simulating errors

The C1 execution percentage rate might not be 100%, even when all execution paths have executed. This situation occurs when no job step or command that results in an error precedes the job step of the `#-adsh_step_start` command.

In such a case, if you simulate an error at a location preceding the `#-adsh_step_start` command, the C1 information indicating errors in the preceding job steps or job definition scripts can be acquired, thus achieving a 100% C1 execution percentage rate. For details about the behavior of a job definition script when the fault injection mode is enabled, see *6.2.20 Enabling and disabling the fault injection mode (joberrmode command)*.

To simulate an error:

1. Move the cursor to a row in which you want to simulate an error and set a breakpoint.

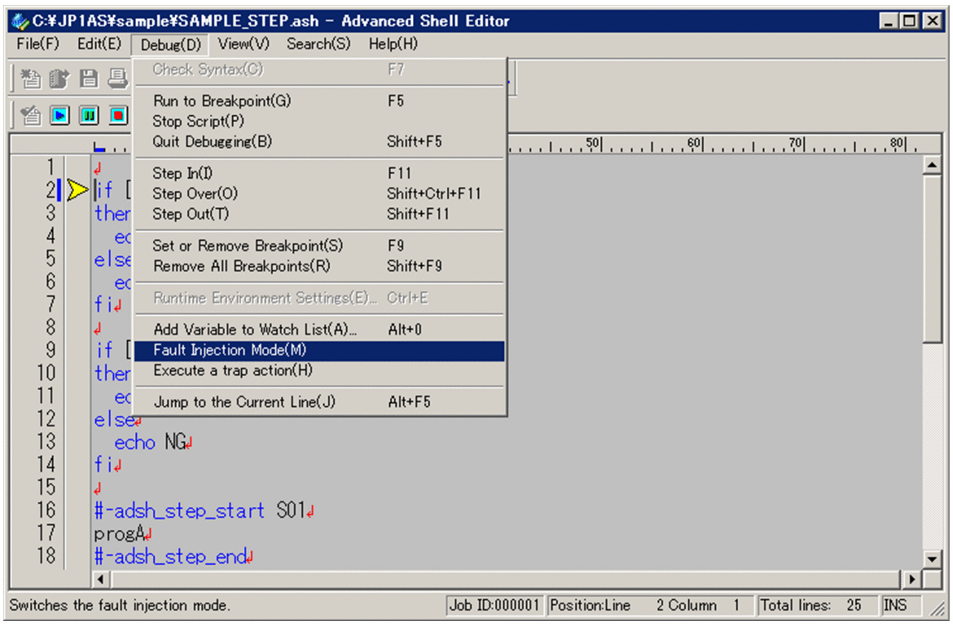
   For details about how to set breakpoints, see *(1) Setting and releasing breakpoints during debugging*.

2. Perform debugging up to the line in which you intend to simulate an error.

   Perform debugging up to the breakpoint set in step 1. Debugging stops temporarily at the breakpoint. For details about how to perform debugging up to a breakpoint, see *(2) Performing and canceling debugging*.

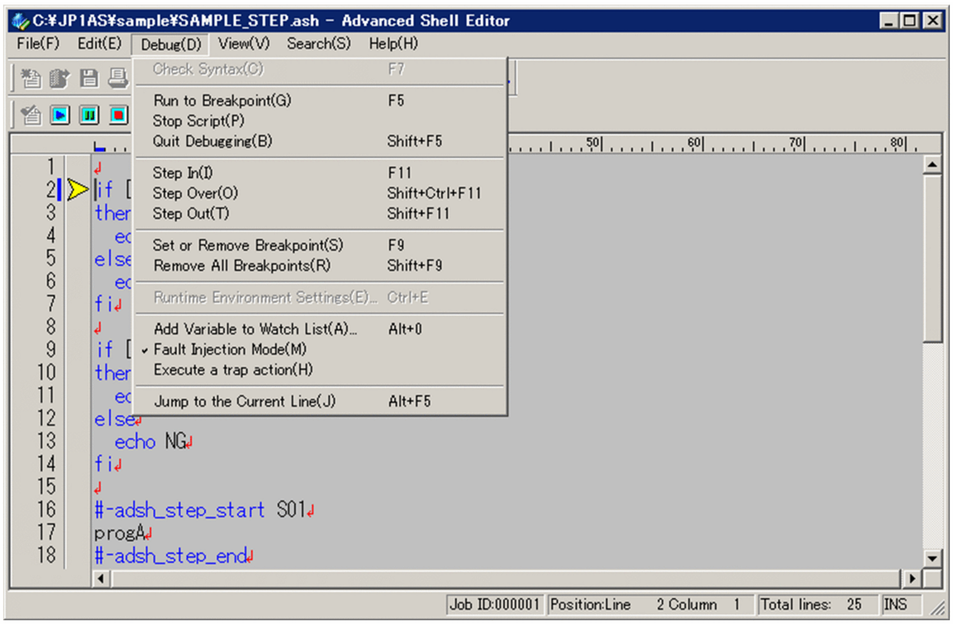
3. From the **Debug** menu, select **Fault Injection Mode**.

   The **Fault Injection Mode** menu item can be selected only when the job definition script is stopped for a reason such as a breakpoint.


C:¥JP1AS¥sample¥SAMPLE_STEP.ash - Advanced Shell Editor
File(F) Edit(E) Debug(D) View(V) Search(S) Help(H)
Check Syntax(C) F7
Run to Breakpoint(G) F5
Stop Script(P)
Quit Debugging(B) Shift+F5
Step In(I) F11
Step Over(O) Shift+Ctrl+F11
Step Out(T) Shift+F11
Set or Remove Breakpoint(S) F9
Remove All Breakpoints(R) Shift+F9
Runtime Environment Settings(E)... Ctrl+E
Add Variable to Watch List(A)... Alt+0
Fault Injection Mode(M)
Execute a trap action(H)
Jump to the Current Line(J) Alt+F5
echo NG
#-adsh_step_start S01
progA
#-adsh_step_end
Switches the fault injection mode.
Job ID:000001
Position:Line 2 Column 1
Total lines: 25
INS

The fault injection mode is enabled and a check mark is displayed to the left of the **Fault Injection Mode** menu item, as shown below.

If you restart debugging in this status, an error is simulated and C1 information is acquired.


C:¥JP1AS¥sample¥SAMPLE_STEP.ash - Advanced Shell Editor
File(F) Edit(E) Debug(D) View(V) Search(S) Help(H)
Check Syntax(C) F7
Run to Breakpoint(G) F5
Stop Script(P)
Quit Debugging(B) Shift+F5
Step In(I) F11
Step Over(O) Shift+Ctrl+F11
Step Out(T) Shift+F11
Set or Remove Breakpoint(S) F9
Remove All Breakpoints(R) Shift+F9
Runtime Environment Settings(E)... Ctrl+E
Add Variable to Watch List(A)... Alt+0
✓ Fault Injection Mode(M)
Execute a trap action(H)
Jump to the Current Line(J) Alt+F5
echo NG
#-adsh_step_start S01
progA
#-adsh_step_end
Job ID:000001
Position:Line 2 Column 1
Total lines: 25
INS

To release the fault injection mode, select the **Fault Injection Mode** item again before you restart debugging.

4. Restart debugging.

   By stepping in, stepping over, stepping out, or executing up to the breakpoint, the editor is placed in the debug mode and debugging resumes.

   When debugging is completed through the last line of the job definition script, the fault injection mode is released.

(5) Executing the trap command's action

You use the `trap` command's action to define an operation that is to be performed when the job controller has received a forced termination request.

To execute the `trap` command's action during debug execution:

1. Start debug execution, and then stop the job definition script at any line after the trap action is defined.

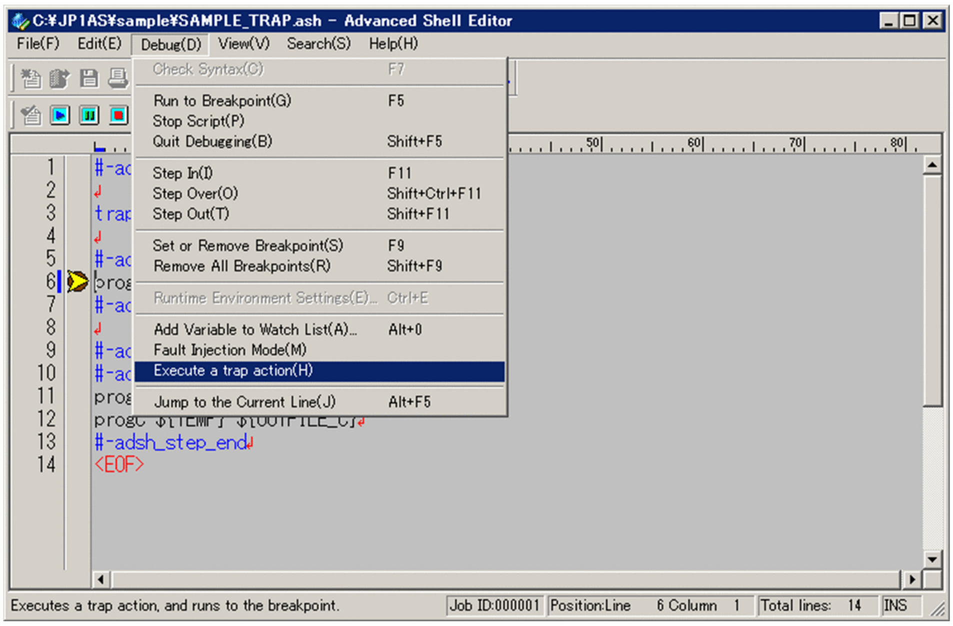
   For details about how to perform debug execution, see *(2) Performing and canceling debugging*.

2. Stop the job definition script at a desired line by using a method such as breakpoints.

   For details about how to set breakpoints, see *(1) Setting and releasing breakpoints during debugging*.

3. From the **Debug** menu, select **Execute a trap action**.

   The **Execute a trap action** menu item can be selected only when the job definition script is stopped by a means such as breakpoints.


C:¥JP1AS¥sample¥SAMPLE_TRAP.ash - Advanced Shell Editor
File(F)
Edit(E)
Debug(D)
View(V)
Search(S)
Help(H)
Check Syntax(C) F7
Run to Breakpoint(G) F5
Stop Script(P)
Quit Debugging(B) Shift+F5
Step In(I) F11
Step Over(O) Shift+Ctrl+F11
Step Out(T) Shift+F11
Set or Remove Breakpoint(S) F9
Remove All Breakpoints(R) Shift+F9
Runtime Environment Settings(E)... Ctrl+E
Add Variable to Watch List(A)... Alt+0
Fault Injection Mode(M)
Execute a trap action(H)
Jump to the Current Line(J) Alt+F5
#-adsh_step_end
<EOF>

Executes a trap action, and runs to the breakpoint. | Job ID:000001 | Position:Line 6 Column 1 | Total lines: 14 | INS

   Once the menu item is selected, the job definition script is run up to the breakpoint. In this case, the commands are executed in the following order:

   1) Current command located on the line where execution is stopped

   2) Commands in the action part

   3) Commands following the command in step 1

   If `DISABLE` is specified in the `TRAP_ACTION_SIGTERM` environment setting parameter or no action by the `trap` command has been defined and this menu is selected, JP1/Advanced Shell runs the job definition script up to the next breakpoint without executing the `trap` command's action.

Notes

- The job definition script cannot be stopped while the `trap` command's action is executing.
- If the job is terminated while the action by this function is underway, the termination code of the last command executed is applied as the job's termination code, unlike when the job is terminated forcibly. For example, if action `exit 2` is executed by using this function, the job is terminated with termination code `2`. On the other hand, if the job is terminated forcibly and then action `exit 2` is executed, the job is terminated with error and termination code `1`.
- After this menu is selected, if execution of the command that immediately follows is skipped, the action will not be executed.[#1]

#1

For example, execution of a command is skipped in the following cases:

- The fault injection mode is enabled.
- A command inside a job step for which `stop` was specified for the `onError` attribute terminates with an error.

4.4.7 Displaying coverage information

If coverage information has been collected, it can be displayed for the job definition script that is currently open in the editor or that is being debugged. Coverage information being collected is displayed during debugging. After debugging is completed, the last coverage information collected is displayed. The coverage information is output to a temporary file and can be displayed using Notepad (`notepad.exe`).

The **View Coverage Information** menu item used to display coverage information is enabled if coverage information is set to **Accumulate** in **Runtime Environment Settings**. If **Accumulate** is not selected, the **View Coverage Information** menu item is grayed out and cannot be selected.

Notepad remains displayed when the editor and debugger have terminated. Coverage information that is being displayed is not refreshed even when the coverage information is changed during debugging.

To display the coverage information and then cancel display of coverage information:

1. From the **View** menu, select **View Coverage Information**, or on the toolbar, click the **View Coverage Information** button.
   Notepad opens and the coverage information is displayed. You can save the displayed coverage information under a desired file name.

2. To close the display of coverage information, exit Notepad.
   The display of coverage information is closed.

4.5 Editing existing job definition scripts (Windows only)

This section explains how to edit an existing job definition script in JP1/Advanced Shell Editor. There are three ways to start editing a job definition script file, as described in the following.

Starting from the right-click menu

1. From Explorer, find and right-click the desired job definition script file.

2. Select **Edit**.

Starting by using a drag-and-drop operation

1. From Explorer, drag the desired job definition script file.

2. Drop the dragged job definition script file onto the **Editor** icon or in an active editor window.

For details about the editor window, see *4.3 JP1/Advanced Shell Editor operation (Windows only)*.

Starting from the editor's menu

1. From the **File** menu, select **Open**, or from the **File** menu, select an edit history to start editing.

2. Select the existing job definition script file that is to be edited.

4.6 Saving job definition scripts (Windows only)

To save the job definition script in JP1/Advanced Shell Editor:

1. To save (overwrite) the job definition script file, from the **File** menu, select **Save**.
   The job definition script file is saved (overwritten).

2. To save the job definition script file under a new name, from the **File** menu, select **Save As**.
   The job definition script file is saved under the new name that you specify.

4.7 Details of the JP1/Advanced Shell Editor window (Windows only)

While you use the JP1/Advanced Shell Editor window, the following dialog boxes and windows are displayed:

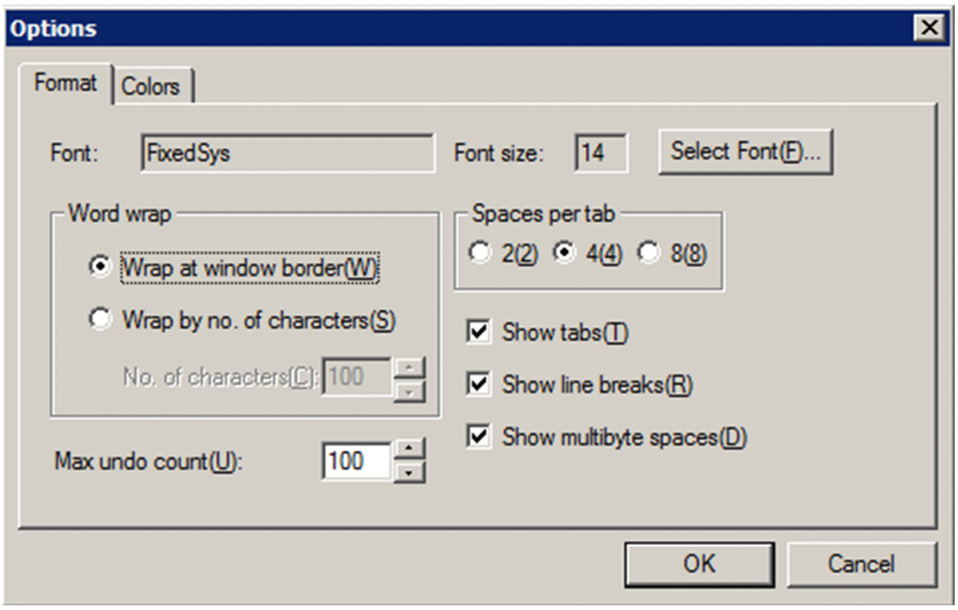
- Options (Format) dialog box
- Options (Colors) dialog box
- Runtime Environment Settings dialog box
- Error List window
- Search dialog box
- Watch List window
- Add to Watch List dialog
- Edit Value dialog box
- Console

4.7.1 Options (Format) dialog box

In the JP1/Advanced Shell Editor window, from the **Edit** menu, selecting **Options** displays the Options dialog box.

The Options dialog box contains the **Format** tab and the **Colors** tab.

Selecting the **Format** tab displays the Options (Format) dialog box.



Font: FixedSys

Font size: 14

Word wrap

● Wrap at window border(W)

○ Wrap by no. of characters(S)

No. of characters(C): 100

Max undo count(U): 100

Spaces per tab

○ 2(2) ● 4(4) ○ 8(8)

☑ Show tabs(T)

☑ Show line breaks(R)

☑ Show multibyte spaces(D)

(1) Items in the dialog box

Font

Displays the name of the font currently being used. To change the font, click the **Select Font** button.
By default, **FixedSys** is selected.

Font size

Specifies the current font size for characters. To change the font size, click the **Select Font** button.

By default, **14** is selected.

Word wrap

Selects the word wrap method.

By default, **Wrap at window border** is selected.

Wrap at window border

Select this item to wrap text at the window border.

Wrap by No. of characters

Select this item to wrap text at a specified number of characters.

No. of characters

Specifies the number of characters for wrapping text.

This item is enabled only when **Wrap by No. of characters** is selected.

Specify a value in the range from 20 through 512 (bytes).

By default, this is set to **100**.

Spaces per tab

Selects the number of spaces (bytes) for setting tabs.

By default, **4** is selected.

Show tabs

Specifies the view/hide setting for tabs.

By default, this item is selected.

Show line breaks

Specifies the view/hide setting for line breaks.

By default, this item is selected.

Show multibyte spaces

Specifies the view/hide setting for the symbol that indicates double-byte spaces.

By default, this item is selected.

Max undo count

Specifies the maximum permitted number of undo operations (for when **Undo** is selected from the **Edit** menu).

Specify a value in the range from 10 through 999. By default, **100** is set.

(2) Operations in the dialog box

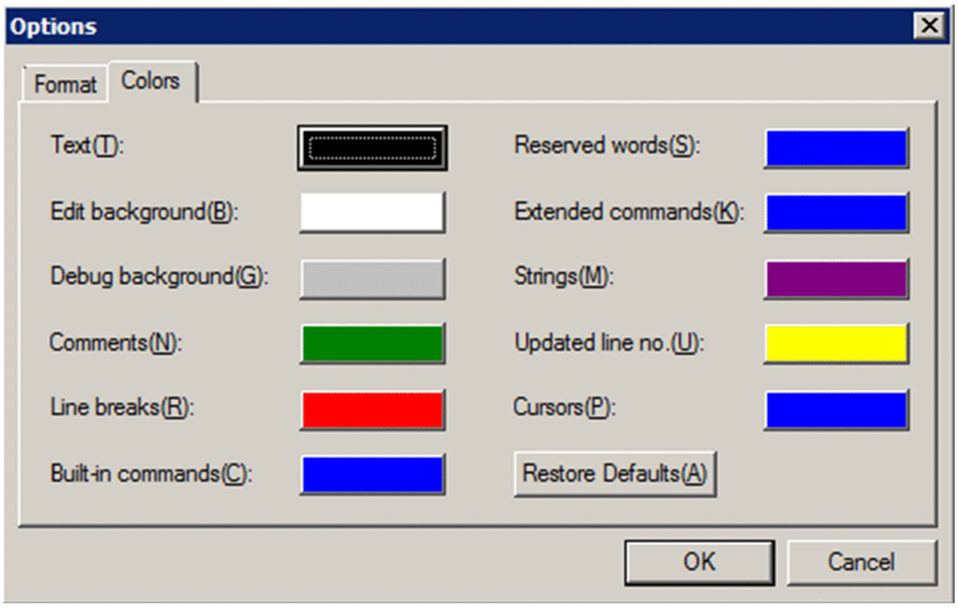
- Clicking the **OK** button applies the specified format settings and then closes the dialog box.
- Clicking the **Cancel** button closes the dialog box without changing any format settings.

4.7.2 Options (Colors) dialog box

In the JP1/Advanced Shell Editor window, from the **Edit** menu, selecting **Options** displays the Options dialog box.

The Options dialog box contains the **Format** tab and the **Colors** tab.

Selecting the **Colors** tab displays the Options (Colors) dialog box.



(1) Items in the dialog box

Text

Specifies a color for text.

The default is the system color.

Edit background

Specifies the background color in the edit mode.

The default is the system color.

Debug background

Specifies the background color in the debug mode.

The default is gray.

Comments

Specifies a color for comment lines.

The default is green.

Line breaks

Specifies a color for line breaks.

The default is red

Built-in commands

Specifies a color for built-in commands.

The default is blue.

Reserved words

Specifies a color for reserved words and `]]`.

The default is blue.

Extended commands

Specifies a color for extended commands.

The default is blue.

Strings

Specifies a color for character strings.

The default is dark purple.

Updated line No.

Specifies a color for updated line numbers.

The default is yellow.

Cursors

Specifies a color for the cursor that indicates the current location.

The default is blue.

Reset to Default

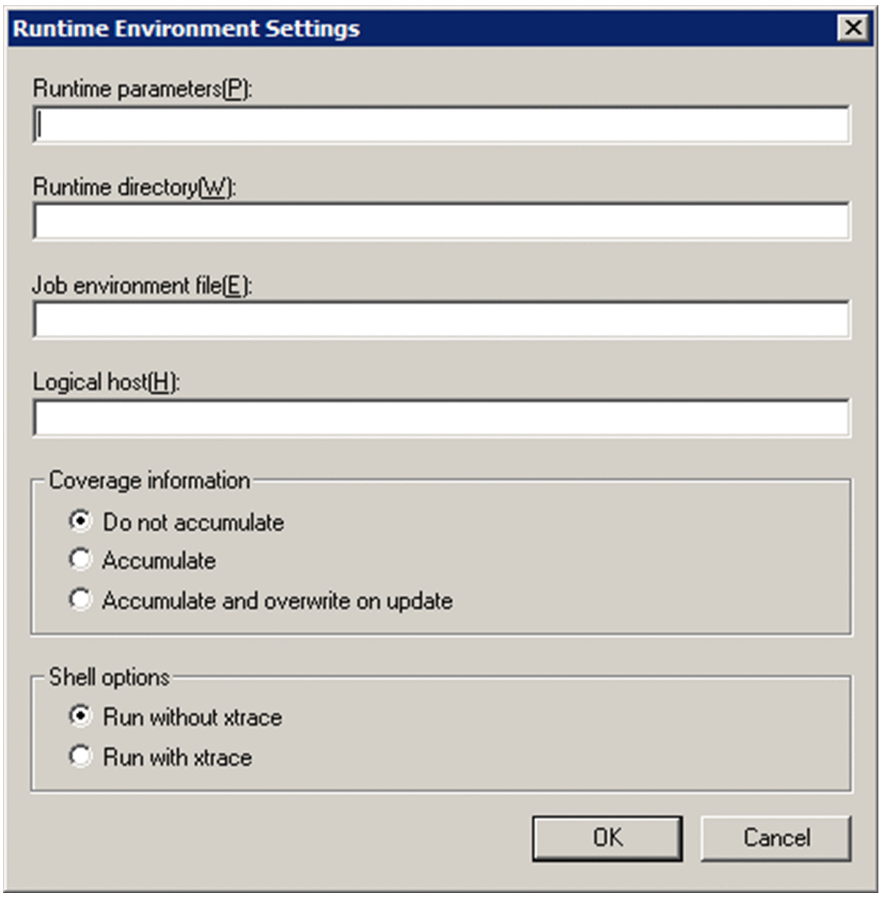
Resets the color settings for all items to the defaults.

(2) Operations in the dialog box

- Selecting any button other than **Reset to Default** opens the Set Colors dialog box where you can select a desired color. In the Set Colors dialog box, clicking the **Create Colors** button displays the Create Colors dialog box in which you can create a desired color.
- Clicking the **OK** button applies the specified color settings and then closes the dialog box.
- Clicking the **Cancel** button closes the dialog box without changing any color settings.

4.7.3 Runtime Environment Settings dialog box

In the JP1/Advanced Shell Editor window, from the **Debug** menu, selecting **Runtime Environment Settings** displays the Runtime Environment Settings dialog box.



Runtime Environment Settings

Runtime parameters(P):

Runtime directory(W):

Job environment file(E):

Logical host(H):

Coverage information

● Do not accumulate

○ Accumulate

○ Accumulate and overwrite on update

Shell options

● Run without xtrace

○ Run with xtrace

(1) Items in the dialog box

Runtime parameters

Specifies run-time parameter to be passed to the job definition script.

Example:

If the name of the job definition script to be used is `a.ash` and `ABC` is specified for a run-time parameter, `ABC` is used as the first argument for `a.ash`.

Runtime directory

Specifies the current drive or folder in which the job definition script is to be executed.

If this information is omitted, the editor's current folder is assumed.

The editor's current folder is one of the following:

- Folder containing the job definition script file
- Program folder
- Program's current folder
- Shortcut work folder

Example:

In this example, job definition script `pwd` is specified in the job definition script named `a.ash`. Specifying `C:\` for the run-time directory produces the same results as when the following specification is made in a Windows directory:

```
C:\> a.ash
C:\
C:\>
```

Job environment file

Specifies the job environment file to be used during debugging. The specified file is used by the job to be debugged.

If this information is omitted, the file specified in the `ADSH_ENV` environment variable is used. If no file is specified in the `ADSH_ENV` environment variable, the default value is assumed.

Logical host

Specifies the logical host to be used by the user-reply functionality. The user-reply functionality used by the job subject to debugging is run on the specified logical host.

If this information is omitted, the user-reply functionality is run on the physical host.

Coverage information

Specifies whether coverage information is to be accumulated:

- **Do not accumulate**

  Does not accumulate coverage information. This is equivalent to omitting the `-t` option in the `adshexec` command. The function for displaying coverage information during debugging is disabled.

- **Accumulate**

  Accumulates coverage information. If the job definition script file has been modified, the job is terminated without executing the job definition script. This is equivalent to specifying the `-t` option in the `adshexec` command. The coverage information can be displayed from the editor.

- **Accumulate (and overwrite on update)**

  Accumulates coverage information. If the job definition script file has been modified, the editor discards the accumulated coverage information and then starts accumulating the new coverage information. This is equivalent to specifying both the `-t` and `-f` options in the `adshexec` command. The coverage information can be displayed from the editor.

By default, **Do not accumulate** is selected.

Coverage information is saved in a coverage information file (`asc` file). The `asc` file is created in the directory where the job definition script file is located. If there is already an `asc` file in the directory where the job definition script file is located, the editor uses that `asc` file.

To display the coverage information accumulated in the coverage information file, select **View Coverage Information** from the editor's **View** menu or use the `adshcvshow` command. To merge coverage information contained in two separate coverage information files, use the `adshcvmerg` command. For details about coverage information, see *3.9 Acquiring coverage information*.

Shell option

Specifies whether the `xtrace` shell option is to be set.

- **Disable xtrace**

  Does not set the `xtrace` shell option when debugging starts. This is the same as the `adshexec` command with the `-x` option omitted.

- **Enable xtrace**

  Sets the `xtrace` shell option when debugging starts. This is the same as the `adshexec` command with the `-x` option specified.

  When this item is selected, executed commands and their arguments are output to the standard error output as trace information. For details, see *3.5 Outputting the executed commands and their arguments*.

The default is that **Disable xtrace** is selected.

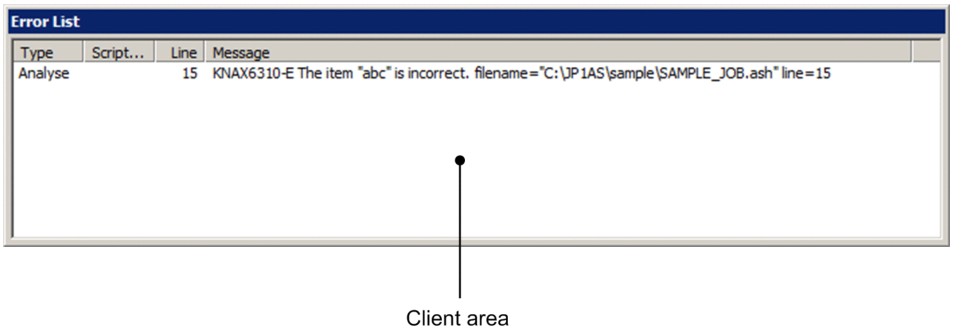
The shell options also include the option for limiting available functions and the option for switching the execution mode. For details about the shell options, see *5.6 Shell options*.

(2) Operations in the dialog box

- Clicking the **OK** button creates an execution environment file containing the information specified in the dialog box.
- Clicking the **Cancel** button closes the dialog box without creating an execution environment file.

4.7.4 Error List window

The Error List window displays the errors that have occurred during debugging. This window is displayed while the job definition script is not running.



Error List

Type	Script...	Line	Message
Analyse		15	KNAX6310-E The item "abc" is incorrect. filename="C:\JP1AS\sample\SAMPLE_JOB.ash" line=15

(1) Client area

Type

Displays the type of error. One of the following error types is displayed:

- **Analysis**: A parsing error occurred.
- **Runtime**: A run-time error occurred.

Script File Name

Displays the name of the job definition script file in which the error occurred. For a file being edited by the editor, this field is blank.

Line

Displays the line number where the error occurred.

Message

Displays the nature of the error.

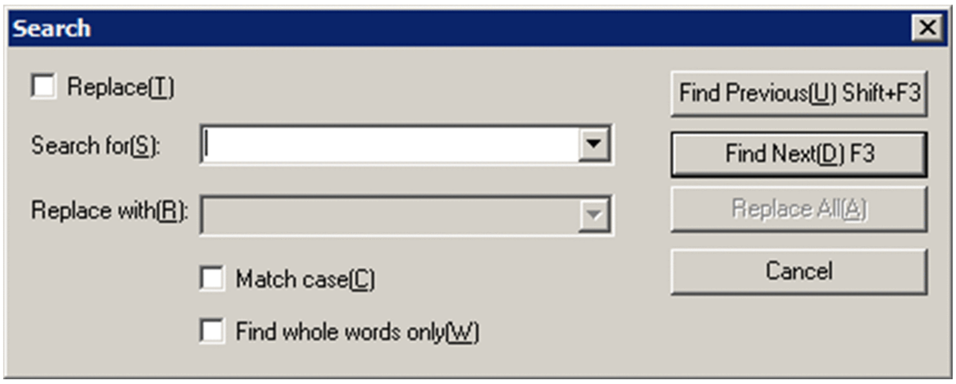
(2) Operations in the Error List window

- If the name of the job definition script file is blank, double-clicking the contents of the **Type** column moves the cursor to the beginning of the corresponding line where the error occurred.
- If the name of the job definition script file is blank, right-clicking the contents of the **Type** column displays the **Jump to Error** pop-up menu to move the cursor to the location of the corresponding error.

4.7.5 Search dialog box

In the JP1/Advanced Shell Editor window, from the **Search** menu, selecting **Search** displays the Search dialog box.

This Search dialog box is also displayed when **Find Previous** or **Find Next** is selected from the **Search** menu.



(1) Items in the dialog box

Replace

Select this check box to replace the character string that is to be searched for.

By default, this check box is cleared.

Search for

Specifies the character string to be searched for.

Replace with

Specifies the character string that is to replace the character string specified in **Search for**. This field is enabled only when the **Replace** check box is selected.

Match case

Select this check box to conduct a case-sensitive search (when a character string to be retrieved must exactly match the character string specified in **Search for** in terms of case).

By default, this check box is cleared.

Find whole words only

Select this check box if only whole words are to be retrieved.

By default, this check box is cleared.

(2) Operations in the dialog box

- Clicking the **Find Previous** button searches for the specified character string upwards.
- Clicking the **Find Next** button searches for the specified character string downwards.
- Clicking the **Replace All** button replaces all occurrences of the character string in the job definition script file with the character string specified in **Replace with**.
- Clicking the **Cancel** button closes the dialog box.

(3) Notes

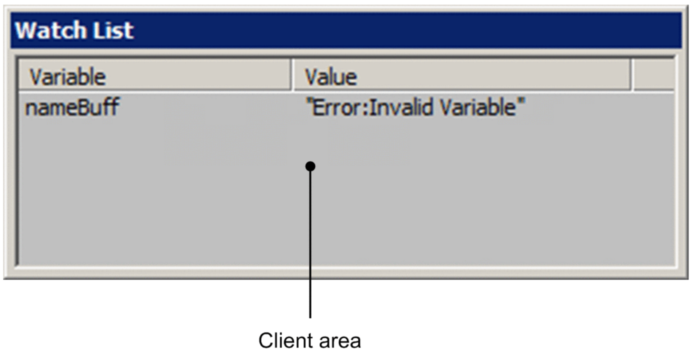
- The character strings specified most recently in **Replace with** and in **Search for** are stored in drop-down lists (up to 10 such character strings in each drop-down list).

- If the specified character string is not found, the editor sounds a beep tone.

4.7.6 Watch List window

The Watch List window displays the names and values of variables during debugging.

For details about how to add variables to the Watch List window, see *4.4.6(3) Adding variables to the watch list*.



(1) Client area

Variable Name

Displays the names of the variables.

Value

Displays the values of the variables. If a specified variable is invalid, one of the following character strings is displayed:

- `Error:Non-numeric Value`

  A non-numeric value was specified for a numeric-type variable.
- `Error:Internal Error`

  An internal error occurred.
- `Error:Invalid Variable`

  The variable name is not valid. This text is also displayed if an index number with only an array variable name specified falls beyond the valid range, or a dimension of a specified array variable is invalid because values cannot be referenced.
- `Error:Insufficient Memory`

  A memory shortage occurred.
- `Error:Read-Only Variable`

  A value was specified for a read-only variable.
- `No Value`

  The specified variable has no value.

This error is displayed when an attempt is made to change a read-only variable or to specify a character string for a numeric-type variable. If debugging is restarted, the previous value before the change is set.

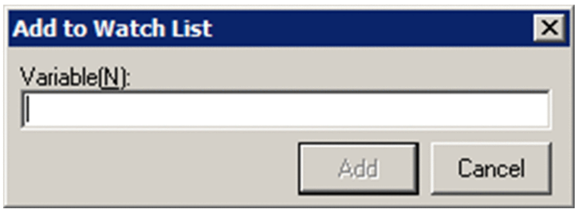
(2) Operations in the Watch List window

- Clicking a variable name displays a pop-up menu for selecting one of the following two items:

    - **Edit Value**
      Enables the specified variable's value to be updated.
    - **Delete from List**
      Deletes the specified variable.
- Double-clicking a variable name displays the Edit Value dialog box.

4.7.7 Add to Watch List dialog box

In the debug mode, from the **Debug** menu, selecting **Add Variable to Watch List** displays the Add to Watch List dialog box.



(1) Items in the dialog box

Variable name
Specifies a name for a variable that is to be added.

(2) Operations in the dialog box

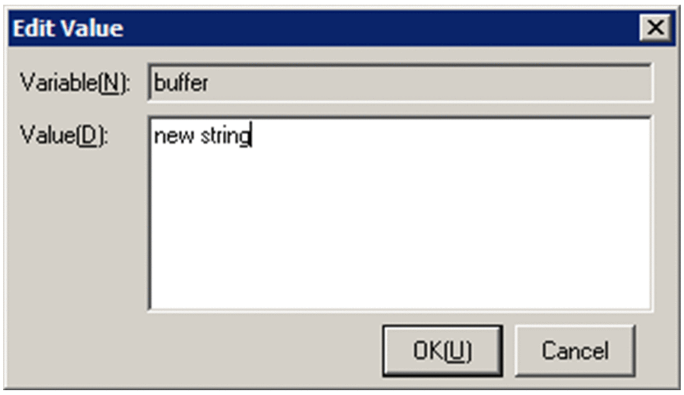
- Clicking the **Add** button adds the specified variable name to the Watch List window and then closes the Add to Watch List dialog box.
- Clicking the **Cancel** button closes the dialog box without adding a variable.

(3) Notes

- **Add Variable to Watch List** can be selected from the **Debug** menu only in the debug mode.
- No error results if the same variable is specified more than once or nonexistent variables are specified.
- A maximum of 99 bytes can be specified for a variable name. If a specified variable name exceeds 99 bytes, the excess part of the name is discarded.
- You can specify a maximum of 99 variable names.

4.7.8 Edit Value dialog box

In the Watch List window, double-clicking a variable name displays the Edit Value dialog box.



Variable(N): buffer

Value(D): new string

(1) Items in the dialog box

Variable name

Displays the name of the variable that is to be updated.

Value

Enter a desired value. The maximum length for a value is 1,024 bytes. If no name has been defined for the variable that is to be updated, the value will not be changed.

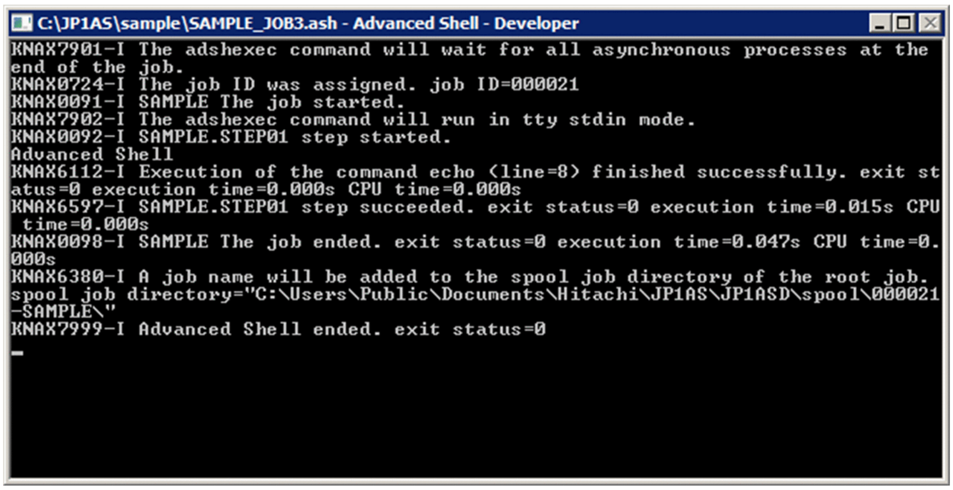
By default, the current value is set.

(2) Operations in the dialog box

- Clicking the **OK** button updates the specified variable's value and then closes the Edit Value dialog box.
- Clicking the **Cancel** button closes the dialog box without updating the variable's value.

4.7.9 Console

During debugging, the console displays information equivalent to the standard output, standard error output, and job execution logs.


C:\JP1AS\sample\SAMPLE_JOB3.ash - Advanced Shell - Developer

```
KNAX7901-I The adshexec command will wait for all asynchronous processes at the
end of the job.
KNAX0724-I The job ID was assigned. job ID=000021
KNAX0091-I SAMPLE The job started.
KNAX7902-I The adshexec command will run in tty stdin mode.
KNAX0092-I SAMPLE.STEP01 step started.
Advanced Shell
KNAX6112-I Execution of the command echo (line=8) finished successfully. exit st
atus=0 execution time=0.000s CPU time=0.000s
KNAX6597-I SAMPLE.STEP01 step succeeded. exit status=0 execution time=0.015s CPU
 time=0.000s
KNAX0098-I SAMPLE The job ended. exit status=0 execution time=0.047s CPU time=0.
000s
KNAX6380-I A job name will be added to the spool job directory of the root job.
spool job directory="C:\Users\Public\Documents\Hitachi\JP1AS\JP1ASD\spool\000021
-SAMPLE\"
KNAX7999-I Advanced Shell ended. exit status=0
_
```

(1) Console operation

- The console is closed when the job being debugged terminates. If an environment file or the job definition script contains any errors, including syntax errors, check the execution results output to the spool on the basis of the job ID and the information displayed in the Error List window, and then take appropriate action.
- The method for specifying properties is the same as when the command prompt is used.

5 Creating Job Definition Scripts

This chapter explains the syntax for job definition scripts.

5.1 Basic elements of job definition scripts

This section explains the basic elements of job definition scripts.

> **Important note**
>
> Note the following about coding job definition scripts:
>
> - A line of a job definition script cannot exceed 8,191 bytes. If a line's length exceeds 8,191 bytes, the job definition script terminates with an error. A continuation line is treated as a separate line for purposes of line length. For details about continuation lines, see *5.1.6(3) Line continuation*.
> - For a line using an extended script command, the maximum length including the continuation lines is 8,191 bytes according to the limitations for extended script commands. For details about the limitations of extended script commands, see *9.1.3(1) Limitations*.
> - If a file path is converted during a file input or output operation, the number of bytes changes depending on the timing of the file input or output operation. In such a case, even if the maximum length of a line is exceeded due to conversion, processing continues and no error occurs.

5.1.1 Reserved words

In JP1/Advanced Shell, special words and phrases that are used in control statements in job definition scripts are registered as reserved words. A reserved word has a special meaning when it is used as the first word of a command and is always recognized as a reserved word (unless it is not enclosed in quotation marks). A reserved word used as the second or any subsequent word is treated as a normal variable. Therefore, special attention is important when you use the reserved words.

To use reserved words, use the standard shell commands `command -V` and `whence -v`. For details about the `command` and `whence` commands, see *command command (executes a command)* and *whence command (displays how character strings would be interpreted if used as commands)* in *9.3 Standard shell commands*.

The following are the reserved words:

```
! [[ { } case do done elif else esac fi for function if in select then time
until while
```

5.1.2 Variables

Variables are placeholders that are replaced by values in job definition scripts. You can create variables and reference their values. Variables can be inherited as environment variables to child processes by exporting them. The variables are also referred to as shell variables.

(1) Naming conventions for variables

You can assign to a created variable any name that observes the naming conventions. Alphabetic characters are always case sensitive. Therefore, two variables with the same name but with differences in case will be treated as different variables.

In a Windows environment, if you attempt to export a variable that name includes lowercase letters and use it as an environment variable, an error might result depending on the specification of the VAR_ENV_NAME_LOWERCASE parameter.

The naming conventions for variables are described below. For details about the naming conventions for environment variables, see *(a) Naming conventions for environment variables (Windows only)*.

- The only permitted characters are alphanumeric characters and the underscore (_).
- A variable must being with a non-numeric character.
- There is no limit to the length of a variable name. However, there is a limit to the length of an input line, and the CUI debugger imposes a limit to the length of a command (number of characters). We recommend that you observe these limitations for variables that are used in job definition scripts.

  For details about the maximum length of an input line, see *5.1 Basic elements of job definition scripts*. For details about the maximum length of a command entry in the CUI debugger, see *6.1.4 List of debugger commands (UNIX only)*.

(a) Naming conventions for environment variables (Windows only)

The supported environment variable names depend on the specification of the VAR_ENV_NAME_LOWERCASE parameter, as described in the following:

- When DISABLE is specified in the VAR_ENV_NAME_LOWERCASE parameter

  Environment variable names that include lowercase letters are not supported.

  An attempt to export a variable name that includes lowercase letters results in an error.
- When ENABLE is specified in the VAR_ENV_NAME_LOWERCASE parameter

  Environment variable names that include lowercase letters are supported, except that you must not use lowercase letters in an environment variable name that begins with ADSH.

  Following are notes about specification:

  - If a shell variable name consisting of lowercase letters is exported by a job definition script, the environment variable name that includes lowercase letters is passed as is to an external command that is called subsequently. Because environment variable names are not case-sensitive in Windows, environment variable names with the same spelling are treated as the same name and the last value exported is passed as the environment variable. However, shell variables are case-sensitive and they represent different values.

    For example, in the following specification, lowercase is set first as the environment variable value and uppercase is set last as the environment variable value because environment variable names ABC and abc are not distinguished:

    ```
    export abc=lowercase
    export ABC=uppercase
    ```
  - If a shell variable is exported, the export attribute of shell variables with the same spelling within the scope of that shell variable becomes invalid. For arrays also, the export attribute of all elements becomes invalid.

    For a local shell variable, the export attribute of shell variables with the same spelling outside the scope of that shell variable is valid, but outside the function processing, the environment variable value is recovered.

The setting of the VAR_ENV_NAME_LOWERCASE parameter determines whether lowercase letters are supported by commands and parameters, as described in the following:

Command or parameter	VAR_ENV_NAME_LOWERCASE parameter setting	
	DISABLE	ENABLE
export parameter (environment setting parameter)	Uppercase and lowercase letters are permitted, but they are not distinguished.	Same as at the left
export command, typeset -x command	Shell variable names in lowercase letters are not supported.	Shell variable names in lowercase letters are supported, but environment variable names are not case sensitive. Shell variables are case sensitive and treated as different shell variables.
set command (-a option)	All shell variables following the set command are exported, but an attempt to set values in shell variables consisting of lowercase letters results in an error.	All shell variables following the set command are exported. If values are set in shell variables consisting of lowercase letters, those shell variables are also exported. However, environment variable names are not case sensitive.
unset command, readonly command, read command	Lowercase letters are supported because they specify shell variable names.	Same as at the left
#-adsh_file command, #-adsh_file_temp command, #-adsh_spoolfile command	Lowercase letters are not supported in file environment variable definition names.	Lowercase letters are supported in file environment variable definition names. However, environment variable names are not case sensitive.
#-adsh_step_start command	Job step names are used as shell variable names for storing step return values, but lowercase letters are supported. Lowercase letters are not supported in shell variable names that are specified in -stepVar.	Same as at the left
#-adsh_path_var command	Lowercase letters are supported in variable names.	Same as at the left
Substitution by {*environment-variable-name*} of extended script commands	Lowercase letters are supported in environment variable names.	Same as at the left
Shell variables watched by the GUI debugger	Lowercase letters are supported in shell variable names.	Same as at the left
adshread command	Lowercase letters are supported in shell variable names.	Same as at the left
for *shell-variable*	Lowercase letters are supported in shell variable names.	Same as at the left
ENVIRON built-in variable in the awk command	Environment variable names can be specified for subscripts, but lowercase letters are also supported.	Same as at the left
Arguments passed to the adshjava command's batch applications	Environment variable names can be specified in system properties, but the adshjava command does not check character types.	Same as at the left

Command or parameter	VAR_ENV_NAME_LOWERCASE parameter setting	
	DISABLE	ENABLE
Environment variables collected by the operation information acquisition functionality	Environment variable names and values are collected, but the environment variable names are not checked for character types.	Same as at the left

(2) Creating variables and assigning values to them

The following shows the format used to create a variable and to assign a value to it:

```
variable-name=value
```

You create a variable by specifying a variable name followed by an equal sign (=). You can use your created variables to write and to read values. To assign a value to the variable, specify the value to the right of the equal sign (=). Note the following about creating variables:

- If a variable has the read-only attribute, an attempt to assign a value to the variable will result in an error and the job will terminate. You use the `readonly` standard shell command to change a variable's attribute to read-only (for details about the `readonly` command, see *readonly command (sets the read-only attribute for variables or displays all read-only variables)* in *9.3 Standard shell commands*).
- If you specify the name of a variable that has not been created, the variable will be created and the specified value will be assigned to it. When the value to be assigned is a character string, its length can consist of any number of characters.

  When numeric values are assigned to variables defined by the `typeset` command as the integer type, or numeric values are assigned to variables used in arithmetic operations, the variable values and the results of the arithmetic operations must be within the range of `-2147483648` to `2147483647`. If a value outside this range is specified, correct results cannot be obtained.
- Do not enter any spaces before or after the equal sign. If there is such a space, the variable will not be created.
- To assign to a variable a character string that contains spaces or metacharacters, you must enclose the character string in quotation marks (`'` or `"`), which will disable the metacharacters or cause space characters to be recognized correctly as spaces. For details about metacharacters and disabling metacharacters, see *5.1.6 Metacharacters*.

(3) Referencing the values of variables

(a) Referencing method

The following shows the formats used to reference the value of a variable:

```
$variable-name
or
${variable-name}
```

You reference the value assigned to a variable by specifying a dollar sign (`$`) before the variable name. The variable with the matching variable name is referenced. If the variable name you specify contains an invalid character, only the characters up to that invalid character will be recognized as a variable name.

Example in which an invalid character is specified in the name of a variable to be referenced

```
abc=xxx
echo $abc@zzz
-->xxx@zzz is output to the standard output.
```

To explicitly specify a variable to be referenced, enclose the variable name in curly brackets (`{}`). When a variable name is enclosed in curly brackets and characters that are not permitted for a variable name are used, those characters are processed as part of the variable name.

Example for explicitly specifying variable abc and referencing its value

```
abc=xxx
abcdef=yyy
echo ${abc}def
-->xxxdef is output to the standard output.
```

(b) Referencing method using offset (start point for referencing) and length (length to be referenced)

The following shows the formats used to reference a variable value by specifying *offset* (start point for referencing) and *length* (length to be referenced):

```
${variable-name:offset}
```

or

```
${variable-name:offset:length}
```

or

```
${variable-name::length}
```

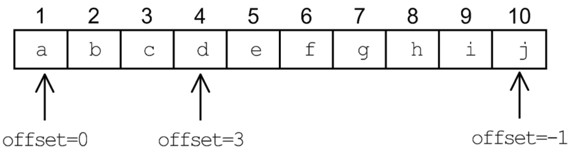
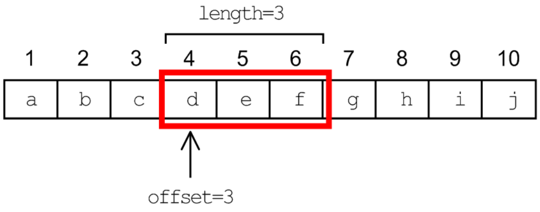
To reference a specific part of the value assigned to the variable to be referenced, specify `:`*offset* or `:`*offset*`:`*length* following *variable-name*.

Only those characters and numeric values permitted by the variable naming rules can be specified for *offset* and *length*. You must observe the following rules in specifying variable names and numeric values for *offset* and *length*:

Specification value	Specification rules
Numeric value (unsigned)	Space and tab cannot be specified before or after the value.
Numeric value (signed)	There must be a space immediately before a sign. Space and tab cannot be specified before or after any other value.
Variable	Space and tab cannot be specified before or after the value.
Default for *offset*	Spaces are permitted, but tabs are not permitted.

If the specification spans multiple lines, enter a slash (\) at the end of each line that is to be continued.

The following table describes the value ranges permitted for *offset* and *length*:

Type	Value range (in characters)#	Specification example
offset	`-65535` to `65535`	If `0` is specified, the character string is output from the beginning. If a positive value is specified, the start point at which the character string is output is determined by counting the specified number of characters from the beginning of the character string + 1. If a negative value is specified, the start point is determined by counting the specified number of characters from the end of the character string. Example: The following shows the reference target when `0`, `3`, or `-1` is specified for *offset* for data consisting of 10 characters:  1 2 3 4 5 6 7 8 9 10 a b c d e f g h i j offset=0 offset=3 offset=-1
length	`0` to `65536`	Example: The portion of the character string enclosed in the highlighted box indicates the range of characters to be output when *offset*=`3` and *length*=`3` are specified for data consisting of 10 characters:  length=3 1 2 3 4 5 6 7 8 9 10 a b c d e f g h i j offset=3

#

You can use either of the following methods to specify a numeric value:

- As an octal, decimal, or hexadecimal value

  The specified numeric value is identified automatically as follows:

  - Any character string beginning with `0x` is assumed to be a hexadecimal number (for example, `0xa`).
  - Any character string beginning with `0` is assumed to be an octal number (for example, `012`.
  - Any numeric value that is neither octal nor hexadecimal is assumed to be a decimal number.
  - `-0` is treated as being the same as `0`.
  - If a numeric value has a sign (- or +), you must specify at least one space between the colon (`:`) and the sign.

- As a variable names to which a numeric value is assigned

  If no variable with the specified variable name exists or no value has been assigned to the specified variable, `0` is assumed as the variable value.

  An error results if the specified variable references multiple variables recursively and the recursion count for *offset* exceeds 1,024 or the recursion count for *length* exceeds 1,025.

The notes below apply to specifying *offset* and *length*.

- The following specifications result in a syntax error:
  - The character string specified for *offset* or *length* is not a numeric value or a variable name.

    ```
    echo ${ABC:123D}
    ```

    ➜ An error results because `123D` is not a numeric value or a variable name.

    - A character string that is not a numeric value or a variable name is specified as a value to set to an *offset* or *length* variable name.

      ```
      CNT=123D
      echo ${ABC:CNT}
      ```

      ➜ An error results because the specified variable value `123D` is not a numeric value or a variable name.

    - *offset* is omitted illegally.

      ```
      echo ${ABC:}
      ```

      ➜ An error results because *offset* is not specified.

    - An arithmetic expression is specified for *offset* or *length*.

      ```
      echo ${ABC:10-2}
      ```

      ➜ An error results because an arithmetic expression is specified.

    - The variable specified for *offset* or *length* contains `$` or `${}`.

      ```
      ABC=abcdefghijklmn
      AA=1
      echo ${ABC:$AA}
      ```

      ➜ An error results because the variable specified for *offset* or *length* contains `$` or `${}`.

    - A tab is specified immediately before *offset* or *length*.

      (The following example specifies a tab immediately before *offset*, where ➜ indicates a tab:)

      ```
      ABC=abcdefghijklmn
      AA=1
      BB=1
      echo ${ABC:➜AA:BB}
      ```

      ➜ An error results because a tab is specified immediately before *offset*.

- If the value specified for *offset* or for *variable* exceeds the length of the character string set in *variable*, the character string set in *variable* is not retrieved.

  ```
  ABC=abcdefghijklmn
  echo ${ABC:20}
  ```

  ➜ The character string set in variable `ABC` cannot be retrieved because the length of that character string is only 14 characters.

- If the variable name specified for *offset* or *length* is undefined or the variable value is null, `0` is assumed as the value of *offset* or *length*.

- If the attribute of the variable value that is specified for *offset* or *length* is changed by executing the `typeset` command, the number base might change depending on the method used (if zeros padding is specified with the `-Z` option, the value is treated as being in octal), resulting in a change to the reference range. Furthermore, the specified value might become invalid due to the change to the number base.

  The example below uses the `typeset` command to change the attribute and specify zeros padding. As a result, the value `12` specified for variable `L1` is interpreted as being an octal value.

  ```
  ABC=abcdefghijklmnopqrstuvwxyz
  typeset -Z3 D1=4
  typeset -Z3 L1=12
  echo ${ABC:D1:L1}
  ```

  ➜ `012` (octal) is interpreted as `10` (decimal) and `efghijklmn` (10 characters) is output to `STDOUT`.

The example below uses the `typeset` command to change the attribute and specify zeros padding. As a result, `8` specified for variable `D1` becomes `008` and is interpreted as an octal value. However, an error results because octal numbers must be in the range from 0 to 7.

```
ABC=abcdefghijklmnopqrstuvwxyz
typeset -Z3 D1=8
typeset -Z3 L1=12
echo ${ABC:D1:L1}
```

➜ An error results because `8` is specified as an octal number.

If the variable specified for *offset* or *length* is a recursive reference specification or circular reference specification, an error results. Examples are shown in the following:

- Example of recursive reference specification

  `ABC=abcdefghijklmnopqrstuvwxyz`

  `D1=D1`

  `echo ${ABC:D1}`

- Example of circular reference specification

  `ABC=abcdefghijklmnopqrstuvwxyz`

  `D1=D2`

  `D2=D1`

  `echo ${ABC:D1}`

The following are examples of specifying *offset* and *length*.

- Specifying `offset(5)`

```
ABC=abcdefghijklmn
echo ${ABC:5}
```

  `-->fghijklmn` is output to the standard output.

- Specifying `offset(5)` and `length(4)`

```
ABC=abcdefghijklmn
echo ${ABC:5:4}
```

  `-->fghi` is output to the standard output.

- Specifying `offset(-1)`

```
ABC=abcdefghijklmn
echo ${ABC: -1}
```

  `-->n` is output to the standard output.

- Specifying the name of a variable that defines *offset*

```
DEF=abcdefghijklmn
CNT=5
echo ${DEF:CNT}
```

  `-->fghijklmn` is output to the standard output.

- Specifying the names of variables that define *offset* and *length*

```
DEF=abcdefghijklmn
CNT=5
```

```
LEN=4
echo ${DEF:CNT:LEN}
```

--> `fghi` is output to the standard output.

- Specifying the name of a variable that defines *offset*

```
DEF=abcdefghijklmn
CNT=-1
echo ${DEF:CNT}
```

-->`n` is output to the standard output.

- Specifying variable `xyz` explicitly and referencing three characters of the value specified for the variable beginning with character 5, where variable `xyz` contains multibyte characters and the values of *offset* and *length* are assigned to variables `CNT` and `LEN`.

```
xyz=あいうえおかきくけこさしすせabcdefghそたち
CNT=4
LEN=3
echo ${xyz: CNT: LEN}
```

--> `おかき` is output to the standard output.

- Specifying variable `xyz` explicitly and referencing 14 characters of the value specified for the variable beginning with character -17, where variable `xyz` contains multibyte characters and the values of *offset* and *length* are assigned to variables `CNT` and `LEN`.

```
xyz=あいうえおかきくけこさしすせabcdefghおかき
CNT=-17
LEN=14
echo ${xyz: CNT: LEN}
```

-->`fけこさしすせabcdefgh` is output to the standard output.

(4) Formats and attributes permitted for variables

In JP1/Advanced Shell, you can specify formats and attributes for variables. The following tables describe the formats and attributes that can be specified for variables.

Table 5–1: Formats that can be specified for variables

Format	Description
Left-justified	Left-justifies the value assigned to the variable.
Right-justified	Right-justifies the value assigned to the variable.
Zero-padded	Right-justifies the value assigned to the variable; if the value is a numeric value, pads any leading spaces with zeros.
Lowercase conversion	Converts all uppercase letters in the value assigned to the variable to lowercase letters.
Uppercase conversion	Converts all lowercase letters in the value assigned to the variable to uppercase letters.

Table 5–2: Attributes that can be specified for variables

Attribute	Description
Integer-type attribute	Treats the value assigned to the variable as an integer. This attribute enables a base number used during output to be defined.

Attribute	Description
Read-only attribute	Sets the variable as being read-only.
Export attribute	Exports the variable.

You use the `typeset` command to specify formats and attributes. For details about the `typeset` command, see *typeset command (declares explicitly the attributes and values of variables and functions)*.

Examples of specifying formats for variables are shown below. These examples assume that each of the variables shown has been defined (Δ indicates a space). Each line of code is followed by an explanation of the coding.

```
    STRn="ΔAbCdeFgHiJkΔ"
    NUMn="12345"
```

- Contents of job definition script

```
typeset -L STR1    # Change STR1 to left-justified format
echo $STR1         # Outputs STR1
typeset -L5 STR2   # Change STR2 to left-justified format
                     # with an area length of 5 bytes
echo $STR2         # Outputs STR2
typeset -R STR3    # Change STR3 to right-justified format
echo $STR3         # Outputs STR3
typeset -R4 NUM1   # Change NUM1 to right-justified format
                     # with an area length of 4 bytes
echo $NUM1         # Outputs NUM1
typeset -Z9 STR4   # Change STR4 to zero-padded format with
                     # an area length of 9 bytes
echo $STR4         # Outputs STR4
typeset -Z9 NUM2   # Change NUM2 to zero-padded format with
                     # an area length of 9 bytes
echo $NUM2         # Outputs NUM2
typeset -l STR5    # Change STR5 to lowercase conversion
                     # format
echo $STR5         # Outputs STR5
typeset -u STR6    # Change STR6 to uppercase conversion
                     # format
echo $STR6         # Outputs STR6
typeset -i16 NUM3  # Change NUM3 to integer-type attribute
                     # with hexadecimal representation
echo $NUM3         # Outputs NUM3
```

- `STDOUT` file contents of the executed job

```
********   JOB SCOPE STDOUT    ********
AbCdeFgHiJk             <-- Output result of STR1
AbCde                   <-- Output result of STR2
AbCdeFgHiJk             <-- Output result of STR3
2345                    <-- Output result of NUM1
CdeFgHiJk               <-- Output result of STR4
000012345               <-- Output result of NUM2
abcdefghijk             <-- Output result of STR5
ABCDEFGHIJK             <-- Output result of STR6
16#3039                 <-- Output result of NUM3
```

5.1.3 Arrays

In JP1/Advanced Shell, you can create and reference an array as a type of variable.

You can create a one-dimensional array that can hold a maximum of 65,536 elements with element numbers from 0 to 65,535. If no element is specified, no array is set.

(1) Creating arrays

The following explains how to create an array.

- Creating multiple elements at one time (using the `set` command)

  ```
  set -A array-name value value ...
  set +A array-name value value ...
  ```

  Example:

  ```
  set -A abc 1 2 3
  echo ${abc[1]}
  -->2 is output to the standard output.
  ```

  This method can create multiple elements at the same time. For details about the `set -A` command, see *set command (sets shell options, creates an array, or displays variable values)* in *9.3 Standard shell commands*.

- Creating a single element

  ```
  array-name[element-number]=value
  ```

  Example:

  ```
  abc[0]=1
  abc[1]=2
  abc[2]=3
  echo ${abc[1]}
  ```

  `-->2` is output to the standard output.

  This method creates one element at a time. To create multiple elements, perform the step as many times as there are elements to be created. An array with element number `0` is treated in the same manner as variables.

- Creating multiple elements at the same time (without using the `set` command)

  ```
  array-name=(value value ...)
  ```

  Example:

  ```
  abc=(1 2 3)
  echo ${abc[1]}
  ```

  `-->2` is output to the standard output.

  This method can create multiple elements at the same time. For details about the creation method, see *(2) Creating arrays by using array-name=(value value ...)*.

(2) Creating arrays by using array-name=(value value ...)

An array defined in the format *array-name*=`(`*value value* `...)` is registered in the format `set -A` *array-name value value* `....` In `JOBLOG`, array creation is output as if the `set` command had been executed, not in the format *array-name*=`(`*value value* `...)`.

Even when arrays are created in the format *array-name*=`(`*value value* `...)`, array elements are managed in the same manner as for other arrays. For example, an array created with the following definition is the same as the array created by `set -A ARRAY x1 x2 x3 x4 x5`:

Array elements for an array defined as ARRAY=(x1 x2 x3 x4 x5)

```
ARRAY[0]=x1
ARRAY[1]=x2
ARRAY[2]=x3
ARRAY[3]=x4
ARRAY[4]=x5
```

Therefore, output to `JOBLOG`, coverage collection, and output of the `xtrace` shell option all have the same result as when the `set` command is used to define arrays.

If *array-name*=`()` is defined, a shell variable whose name is *array-name* and value is the null string is created. This is the same as when *array-name*= is defined.

(a) Examples of array creation

The table below shows examples of creating array elements that contain shell variables by using the following variables:

```
A=a
B=b
C=c
MA=' a b c' #
MB=d
```

#

The single quotation mark (`'`) is used to indicate a space. It is not part of the actual variable value.

Table 5–3: Example of array element creation

Array definition	Array elements that are created	Number of array elements created
`(a b c)`	`[0]=a [1]=b [2]=c`	3
`($A $B $C)`	`[0]=a [1]=b [2]=c`	3
`(${A}${B}${C})`	`[0]=a [1]=b [2]=c`	3
``($A $B `echo 1`)``	`[0]=a [1]=b [2]=1`	3
`($A$B $C)`	`[0]=ab [1]=c`	2
`(${A}xyz ${B}stu)`	`[0]=axyx [1]=bstu`	2
`($MA $MB)`	`[0]=a [1]=b [2]=c [3]=d`	4
`($MA$MB)`	`[0]=a [1]=b [1]=cd`	3

(b) Example of JOBLOG output when arrays are used

Examples of array definition and the resulting `JOBLOG` output are shown in the following:

- Specifying 3 as the number of arrays `SEQ1` and `(x1 x2 x3)` as the number of elements

```
SEQ1=(x1 x2 x3)
echo ${SEQ1[@]}
```

-->`x1 x2 x3` is output to the standard output.

The following shows an example of the `JOBLOG` output when array `SEQ1` is used:

```
KNAX7901-I The job controller will wait for all asynchronous processes at
the end of the job.
KNAX0724-I The job ID was assigned. job ID=000053
---------------------------------------------------------------
 Advanced Shell 10-50

 [Information]
   Job ID          : 000053
   Spool directory : /var/opt/jp1as/spool/000053/
   Date            : 2014/02/05
   EnvFile(system) :
   EnvFile(job)    : /opt/jp1as/conf/adsh.conf
   Host name       : vm002149
 [Environment variable from Automatic Job Management System]
---------------------------------------------------------------
********  JOB CONTROLLER MESSAGE  ********
22:17:21 000053 KNAX0091-I ADSH000053 The job started.
22:17:21 000053 KNAX7901-I The job controller will wait for all
asynchronous processes at the end of the job.
22:17:21 000053 KNAX7902-I The job controller will run in tty stdin mode.
22:17:21 000053 KNAX6110-I Execution of the command SEQ1[0]=x1 (line=1)
finished successfully. exit status=0 execution time=0.000s CPU time=0.000s
22:17:21 000053 KNAX6110-I Execution of the command SEQ1[1]=x2 (line=1)
finished successfully. exit status=0 execution time=0.000s CPU time=0.000s
22:17:21 000053 KNAX6110-I Execution of the command SEQ1[2]=x3 (line=1)
finished successfully. exit status=0 execution time=0.000s CPU time=0.000s
22:17:21 000053 KNAX6112-I Execution of the command echo (line=2)
finished successfully. exit status=0 execution time=0.000s CPU time=0.000s
22:17:21 000053 KNAX0098-I ADSH000053 The job ended. exit status=0
execution time=0.001s CPU time=0.000s

********   Script IMAGE    ********

***** /home/jp1asuser1/shell/A.sh *****
0001 : SEQ1=(x1 x2 x3)
0002 : echo ${SEQ1[@]}

***** CONVERSION INFORMATION *****

********   JOB SCOPE STDERR    ********
KNAX0098-I ADSH000053 The job ended. exit status=0 execution time=0.001s
CPU time=0.000s

******** JOBSTEP OUTPUT ********
KNAX6380-I A job name will be added to the spool job directory of the
root job. spool job directory="/var/opt/jp1as/spool/000053-ADSH000053/"
```

```
KNAX7999-I Advanced Shell ended. exit status=0

********   JOB SCOPE STDOUT    ********
x1 x2 x3
```

- Defining variable names to which array elements are assigned as shown below, and specifying 3 as the number of arrays SEQ1

```
ARR1=x1
ARR2=x2
ARR3=x3
SEQ1=($ARR1 $ARR2 $ARR3)
echo ${SEQ1[@]}
```

-->x1 x2 x3 is output to the standard output.

The following shows an example of the JOBLOG output when array SEQ1 is used:

```
KNAX7901-I The job controller will wait for all asynchronous processes at
the end of the job.
KNAX0724-I The job ID was assigned. job ID=000051
---------------------------------------------------------------
 Advanced Shell 10-50

 [Information]
   Job ID           : 000051
   Spool directory : /var/opt/jp1as/spool/000051/
   Date             : 2014/02/05
   EnvFile(system) :
   EnvFile(job)     : /opt/jp1as/conf/adsh.conf
   Host name        : vm002149
 [Environment variable from Automatic Job Management System]
---------------------------------------------------------------
********  JOB CONTROLLER MESSAGE  ********
22:10:35 000051 KNAX0091-I ADSH000051 The job started.
22:10:35 000051 KNAX7901-I The job controller will wait for all
asynchronous processes at the end of the job.
22:10:35 000051 KNAX7902-I The job controller will run in tty stdin mode.
22:10:35 000051 KNAX6110-I Execution of the command ARR1=x1 (line=1)
finished successfully. exit status=0 execution time=0.000s CPU time=0.000s
22:10:35 000051 KNAX6110-I Execution of the command ARR2=x2 (line=2)
finished successfully. exit status=0 execution time=0.000s CPU time=0.000s
22:10:35 000051 KNAX6110-I Execution of the command ARR3=x3 (line=3)
finished successfully. exit status=0 execution time=0.000s CPU time=0.000s
22:10:35 000051 KNAX6110-I Execution of the command SEQ1[0]=x1 (line=4)
finished successfully. exit status=0 execution time=0.000s CPU time=0.000s
22:10:35 000051 KNAX6110-I Execution of the command SEQ1[1]=x2 (line=4)
finished successfully. exit status=0 execution time=0.000s CPU time=0.000s
22:10:35 000051 KNAX6110-I Execution of the command SEQ1[2]=x3 (line=4)
finished successfully. exit status=0 execution time=0.000s CPU time=0.000s
22:10:35 000051 KNAX6112-I Execution of the command echo (line=5)
finished successfully. exit status=0 execution time=0.000s CPU time=0.000s
22:10:35 000051 KNAX0098-I ADSH000051 The job ended. exit status=0
execution time=0.001s CPU time=0.000s

********   Script IMAGE    ********

***** /home/jp1asuser1/shell/B.sh *****
0001 : ARR1=x1
```

```
0002 : ARR2=x2
0003 : ARR3=x3
0004 : SEQ1=(${ARR1} ${ARR2} ${ARR3})
0005 : echo ${SEQ1[@]}

***** CONVERSION INFORMATION *****

********   JOB SCOPE STDERR    ********
KNAX0098-I ADSH000051 The job ended. exit status=0 execution time=0.001s
CPU time=0.000s

******** JOBSTEP OUTPUT ********
KNAX6380-I A job name will be added to the spool job directory of the
root job. spool job directory="/var/opt/jp1as/spool/000051/000051-
ADSH000051/"
KNAX7999-I Advanced Shell ended. exit status=0

********   JOB SCOPE STDOUT    ********
x1 x2 x3
```

(c) Notes

A maximum of 8,192 bytes can be specified per line. If an array whose array element numbers have been extended is defined and the maximum number of arrays are specified all on one command line, an error will result. If the specification exceeds 8,192 bytes, use the continuation line specification (\) to continue specification onto the next line so that no line exceeds 8,192 bytes.

Definition examples that use the continuation line specification (\) are shown in the following.

Example of definition using the set command

```
set -A ARRAY x0\
 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 ... x1000 \
x1001 x1002 x1003 x1004 x1005 x1006 x1007 x1008 x1009 x1010 x1011 ...
x2000 \
    :
x65001 x65002 x65003 x65004 x65005 x65006 x65007 x65008 x65009 ... x65535
```

Example of definition using an assignment expression

```
ARRAY=( x0\
 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 ... x1000 \
x1001 x1002 x1003 x1004 x1005 x1006 x1007 x1008 x1009 x1010 x1011 ...
x2000 \
    :
x65001 x65002 x65003 x65004 x65005 x65006 x65007 x65008 x65009 ... x65535)
```

(3) Referencing the values of arrays

The following explains how to reference the values in an array.

- Referencing the value of element 1 of an array

  ```
  ${array-name[element-number]}
  ```

  Example:

  ```
  set -A abc 1 2 3
  ```

`echo ${abc[1]}`

`-->` `2` is output to the standard output.

- Referencing the values of all elements of an array

  There are four ways to reference the values of all elements of an array:

  Referencing method 1:

  `${`*array-name*`[*]}`

  Example:

  ```
  set -A abc 1 2 3
  echo ${abc[*]}
  ```

  `-->` `1 2 3` is output to the standard output.

  Referencing method 2:

  `${`*array-name*`[@]}`

  Example:

  ```
  set -A abc 1 2 3
  echo ${abc[@]}
  ```

  `-->` `1 2 3` is output to the standard output.

  Referencing method 3:

  `"${`*array-name*`[*]}"`

  Note: When this method is used, each value is separated by the value of the `IFS` shell variable.

  Example:

  ```
  set -A abc 1 2 3
  IFS=:
  echo "${abc[*]}"
  ```

  `-->` `1:2:3` is output to the standard output.

  Referencing method 4:

  `"${`*array-name*`[@]}"`

  Example:

  ```
  set -A abc 1 2 3
  echo "${abc[@]}"
  ```

  `-->` `1 2 3` is output to the standard output.

The following shows an example of referencing the values of an array:

Contents of job definition script

```
set -A myArray a01 a02 a03     # Define myArray as an array
for myElement in ${myArray[*]} # Expand all elements of myArray to
wordlists in the for statement
do
  echo $myElement
done
```

Results output to the standard output

```
a01
a02
a03
```

5.1.4 Functions

You can use functions by defining them in the same job definition script files and external files. The following shows the formats used to define functions.

Format 1

```
function-name() {
       command
       :
       (omitted)
}
```

Format 2

```
function function-name {
       command
         :
       (omitted)
}
```

The naming conventions for functions are the same as for variables. A function cannot have the same name as any standard shell command or extended shell command. For details about the naming conventions for variables, see *5.1.2(1) Naming conventions for variables*.

If you define a function in a job definition script that is not the job definition script in which the function is to be executed, you must use the `.` (dot) command or the `#-adsh_script` command to call the job definition script in which the function is defined before the function executes.

The formats are shown below. For details about the `.` (dot) command, see *. command (executes a shell script)* in *9.3 Standard shell commands*. For details about the `#-adsh_script` command, see *#-adsh_script command (calls an external job definition script file from the job definition script that is running)* in *9.5 Extended script commands*.

```
. name-of-file-defining-the-function
```

or

```
#-adsh_script name-of-file-defining-the-function
```

If a function defined in a job definition script has the same name as another function in the same job definition script, in a job definition script called by the `.` (dot) command, or in a job definition script called by the `#-adsh_script` command, the function defined immediately before the execution location of the identically named function is executed.

The format used to execute a function is shown below. You can specify arguments in a function. The specified arguments are stored after the positional parameter `$1` within the function.

```
function-name [args]
```

In the positional parameter `$0` in the function, the following information is stored according to the format used:

- For a function defined in format 1, its shell script name is stored.
- For a function defined in format 2, its function name is specified.

The following table explains the relationship between positional parameters and whether function arguments are specified.

Table 5–4: Relationship between positional parameters and whether function arguments are specified

Positional parameter in the program calling the function	Function argument	Positional parameter when the function begins	Positional parameter when control returns from the function
Specified	Specified	Value specified in the argument	Value of the positional parameter in the program calling the function
Specified	Omitted	No value	Value of the positional parameter in the program calling the function
Omitted	Specified	Value specified in the argument	No value (value of the positional parameter in the program calling the function)
Omitted	Omitted	No value	No value (value of the positional parameter in the program calling the function)

(1) Local variables in functions

In JP1/Advanced Shell, you can use the `typeset` command in a function to define local variables that are valid within the function. When the function is completed, the defined variables are restored to their status before the function was executed. The following shows an execution example of local variables in a function.

Contents of job definition script:

```
myfunc(){             # Define the myfunc function
  typeset -r var=abc  # Define the var local variable with the read-only attribute in the function
  echo $var           # Output the value of the var variable
  readonly -p         # Output the variable with the read-only attribute
  return 0
}
typeset -i var=123    # Define the var variable with the integer-type attribute
typeset | grep var    # Output the attribute of the var variable
myfunc                # Execute the myfunc function
echo $var             # Output the value of the var variable after function execution
readonly -p           # Output the variable with the read-only attribute
typeset | grep var    # Output the attribute of the var variable
exit 0
```

Contents of the `STDOUT` file of execution job:

```
********   JOB SCOPE STDOUT    ********
typeset -i var             <-- Result of line 8. The var variable has been
defined as having the integer type.
abc                         <-- Result of line 3. The var variable has
been updated to abc.
readonly var=abc            <-- Result of line 4. The var variable has the
read-only attribute.
readonly ADSH_DIR_BIN=/opt/jp1as/bin/
readonly ADSH_DIR_CMD=/opt/jp1as/cmd/
123                         <-- Result of line 10. The value of the var
variable has been restored after function execution.
readonly ADSH_DIR_BIN=/opt/jp1as/bin/
readonly ADSH_DIR_CMD=/opt/jp1as/cmd/
typeset -i var              <-- Result of line 12. Same output results as
in line 8.
```

(2) Trace mode

By enabling the trace mode for a function, you can execute the commands specified in the function at the same time that the contents of the commands are output to the standard error output.

You use the `typeset` standard shell command to place a function in the trace mode. For details about the `typeset` command, see *typeset command (declares explicitly the attributes and values of variables and functions)* in *9.3 Standard shell commands*.

The following shows an example of a `typeset` command specification and its standard error output results:

Contents of job definition script

```
fn(){
  echo "--- `date`"
}
typeset -ft fn
fn
```

Standard error output results of the function in the trace mode

```
+ date
+ echo --- Thu Mar 28 16:00:00 JST 2013
```

(3) Function preload functionality

The function preload functionality enables you to define only the code functions used in the shell script to be defined during execution. When the function preload functionality is used, batch job execution performance improves compared with when this functionality is not used because you can avoid executing common code functions regardless of the execution.

The following explains how to use the function preload functionality.

1. Create a file containing function definitions (function definition file) and save it using the function name as the file name.

   If you define multiple functions in the function definition file, all the specified functions are defined. Note that if a function name defined in the function definition file already exists, that function definition is overwritten.

If information other than function definitions, such as commands, is specified in the function definition file, that information functions in the same manner as external scripts.

2. In the job definition script, specify the following command to enable the function preload functionality:

```
typeset -fu function-name [function-name-2...]
```

For *function-name*, specify the name of the function with the same name as the function definition file name.

JP1/Advanced Shell provides `autoload` as an alias of the `typeset -fu` command. The format of `autoload` is as follows:

```
autoload function-name [function-name-2...]
```

For details about the `typeset` command, see *typeset command (declares explicitly the attributes and values of variables and functions)* in *9.3 Standard shell commands*.

For details about `autoload`, see *5.1.5 Command alias definitions*.

3. In the `FPATH` shell variable, specify the directory that contains the function definition file.

If a function with the preload functionality enabled is not defined in the specified function definition file, the contents of the function definition file with the specified function name are loaded from the directory specified in the `FPATH` shell variable and then all the functions specified in the file are defined. You can specify the `FPATH` shell variable in job definition scripts and environment files.

(a) Example of using the preload functionality (using the typeset -fu and autoload commands)

This example uses the preload functionality with the `typeset -fu` and `autoload` commands. This example applies the function preload functionality to the functions `auto1`, `auto2`, and `auto3`.

Contents of the function definition file (`/home/jp1as/autoload/auto1`):

```
function auto1 {      # Defines the auto1 function
 echo "start auto1 in FPATH file"
 return 11
}
```

Contents of the function definition file (`/home/jp1as/autoload/auto2`):

```
autoxx() {           # Defines the autoxx function (auto2 is not defined)
  echo "start autoxx in FPATH file"
  return 255
}
```

Function definition files in the `/home/jp1as/autoload` directory:

`auto1`

`auto2`

(Function definition file `auto3` does not exist)

Contents of the job definition script (`/home/jp1as/test.ash`):

```
export FPATH="/home/jp1as/autoload"  # Specify FPATH

typeset -fu auto1 auto2  # Apply the preload functionality to the auto1 and auto2 functions
autoload auto3           # Apply the preload functionality to the auto3 function
```

```
0005 :
0006 : auto1                      # Execute the auto1 function
0007 : auto2                      # Execute the auto2 function
0008 : auto3                      # Execute the auto3 function
```

Execution results:

```
********  JOB CONTROLLER MESSAGE  ********
14:40:37 152263 KNAX0091-I ADSH152263 The job started.
14:40:37 152263 KNAX7901-I The adshexec command will wait for all
asynchronous processes at the end of the job.
14:40:37 152263 KNAX7902-I The adshexec command will run in tty stdin
mode.
14:40:37 152263 KNAX6112-I Execution of the command export (line=1)
finished successfully. exit status=0 execution time=0.000s CPU time=0.000s
14:40:37 152263 KNAX6112-I Execution of the command typeset (line=3)
finished successfully. exit status=0 execution time=0.000s CPU time=0.000s
14:40:37 152263 KNAX6112-I Execution of the command typeset (line=4)
finished successfully. exit status=0 execution time=0.000s CPU time=0.000s
14:40:37 152263 KNAX6112-I Execution of the command echo (line=2)
finished successfully. exit status=0 execution time=0.000s CPU time=0.000s
14:40:37 152263 KNAX6112-I Execution of the command return (line=3)
finished successfully. exit status=11 execution time=0.000s CPU
time=0.000s#1
14:40:37 152263 KNAX6046-E The function "auto2" is not defined in the
function definition file "/home/jp1as/autoload/auto2". filename="/home/
jp1as/test.ash" line=7#2
14:40:37 152263 KNAX6044-E The function definition file "auto3" was not
found in the FPATH directory. filename="/home/jp1as/test.ash" line=8#3
14:40:37 152263 KNAX0101-E ADSH152263 An error occurred during execution
of the job.
14:40:37 152263 KNAX0098-I ADSH152263 The job ended. exit status=127
execution time=0.004s CPU time=0.000s
```

#1

Indicates that the execution result of the `auto1` function is normal termination.

#2

The `auto2` function already exists in the function definition file with the same name in the directory specified in `FPATH`. Because that function definition file does not contain a definition of the `auto2` function, the `KNAX6046-E` message is issued and the function terminates with an error.

#3

For the `auto3` function, the directory specified in `FPATH` does not contain a function definition file with the function name. Therefore, the `KNAX6044-E` message is issued and the function terminates with an error.

(b) Notes

- Once a function is defined by loading its function definition file by using the preload functionality, no function definition file with the same file name will be loaded again. However, if the function is invalidated by the `unset` command and then is re-executed, the function definition file with the same name will be loaded again.
- If the function preload functionality is enabled for a function but the function has already been defined in the same shell script, in a shell script called by the `.` (dot) command, or in a shell script called by the `#-adsh_script` extended script command, the function definition file is not loaded.

- If the same function is defined in multiple function definition files, the last function defined takes effect. If you define multiple functions in a single function definition file, make sure that there is no duplication of function names.

  The following shows an example definition of the `fn1` function in multiple function definition files and the output results.

  Contents of the function definition file `/home/jp1as/autoload/fn1`:

  ```
  fn1(){ # fn1 function definition 1
  echo "start fn1"
  return 1
  }
  ```

  Contents of the function definition file `/home/jp1as/autoload/fn2`:

  ```
  fn2(){ # fn2 function definition
  echo "start fn2"
  return 2
  }
  fn1(){ # fn1 function definition 2
  echo "start fn1 in fn2"
  return 21
  }
  ```

  Contents of the job definition script `/home/jp1as/test.ash`:

  ```
  export FPATH="/home/jp1as/autoload" # Specify FPATH
  typeset -fu fn1 fn2 # Enable the preload functionality for the fn1 and fn2 functions
  fn1 # Execute the fn1 function
  fn2 # Execute the fn2 function
  fn1 # Re-execute the fn1 function
  ```

  Results output to the standard output

  ```
  start fn1  ← fn1 of definition 1 is executed.
  start fn2
  start fn1 in fn2  ← fn1 of definition 2 is executed.
  ```

- If the function preload functionality is enabled for an undefined function, the job definition script is placed in undefined status until a function definition file is loaded. Therefore, nothing is displayed by using the CUI debugger's `info functions` command.

5.1.5 Command alias definitions

In JP1/Advanced Shell, you can define an alias for a command. You use the `alias` command for the alias definition. The following shows the format:

Format:

```
alias alias-name=value
```

If you specify for *value* a character string that contains a space, you must enclose the entire value in quotation marks.

You can specify for *alias-name* a built-in command to redefine the command, but no reserved word can be redefined as an alias. The following shows an example.

Redefining a built-in command:

```
alias read="read STR"  # Define "read STR" for the alias name read
uname | read           # read is executed as "read STR"
echo $STR              # The result of uname set in the STR variable is
output
```

Specifying a reserved word as an alias:

```
alias while="echo JP1/AS"  # Define an alias for the reserved word while
while                      # while is interpreted as a reserved word and
                           # the while statement is terminated with a
format error
```

You use the `alias` command to define aliases or to output a list of aliases. To invalidate a defined alias definition, you use the `unalias` command. For details about these commands, see *alias command (defines aliases)* and *unalias command (removes alias definitions)* in *9. Job Definition Script Commands and Control Statements*.

In JP1/Advanced Shell, the following aliases are defined:

Alias	Definition
autoload	typeset -fu
functions	typeset -f
integer	typeset -i
local	typeset
type	whence -v

5.1.6 Metacharacters

Metacharacters are characters that have special meaning in job definition scripts. The following are the JP1/Advanced Shell metacharacters:

|, &, ;, <, >, (,), $, `, ', \, ", ~, #, *, [,], ?

If you want to use a metacharacter as a normal character, you must invalidate its metacharacter usage. The following table explains how to invalidate metacharacters.

Table 5–5: How to invalidate metacharacters

Invalidation method	Description
'*str*'	The character string *str* processes all characters other than the single quotation mark (') as normal characters.
"*str*"	The character string *str* processes all characters other than the dollar sign ($), backslash (\), and grave accent mark (`) as normal characters. However, if \ is immediately followed by $, \, `, or ", then \ is treated as a metacharacter. To include \ as a normal character in the character string *str*, specify \\.
\char	Invalidates (escapes) the special meaning of the characters *char*.

When invalidated metacharacters are output, the output processing depends on the specifications of the built-in command used, such as the `echo` and `print` commands.

Contents of job definition script:

```
echo 'JP1/AS\n'         # 1.
echo "JP1/AS\n"         # 2.
echo JP1/AS\\n          # 3.
echo 'JP1/AS\\n'        # 4.
```

`STDOUT` file contents of the executed job:

```
********   JOB SCOPE STDOUT    ********
JP1/AS          ← As a result of 1, the echo command outputs \n as a line 
break.

JP1/AS          ← Result of 2

JP1/AS          ← Result of 3

JP1/AS\n        ← As a result of 4, \ is treated as an escape character 
and \n is output as distinct characters.
```

The following subsections explain functions using metacharacters.

(1) Positional parameters

When you execute a job definition script, you can pass parameters to the job definition script as arguments by specifying run-time parameters following the job definition script file name. JP1/Advanced Shell assigns these arguments to special variables called *positional parameters*. There are 10 positional parameters, `$0` through `$9`. The file name of the executed job definition script is assigned to `$0`, and arguments are assigned to `$1` through `$9` in the order specified. The following table lists and describes the positional parameters of JP1/Advanced Shell and the related special characters.

Table 5–6: Positional parameters of JP1/Advanced Shell and the related special characters

Positional parameter or special character		Description
Positional parameters	`$0`	File name of the job definition script.
	`$`*n*	Value of argument *n* specified in the job definition script (*n*: 1 through 9)
Related special characters	`$#`	Number of arguments specified in the job definition script
	`$*`	All arguments specified in the job definition script
	`$@`	All arguments specified in the job definition script
	`"$*"`	Handles all arguments specified in the job definition script as a group. **Example:** `"$1 $2 $3 ... "` If the value of the `IFS` shell variable has been changed, the values are delimited by the new value of the `IFS` shell variable.
	`"$@"`	Handles the arguments specified in the job definition script individually. **Example:** `"$1" "$2" ... "` If the `IFS` shell variable has been changed, the values are delimited by the space.

You use the set standard shell command to change the positional parameters in a job definition script. For details about the set command, see *set command (sets shell options, creates an array, or displays variable values)* in *9.3 Standard shell commands*.

(2) String separators

In JP1/Advanced Shell, the characters specified in the IFS shell variable are treated as string separators. The space and tab characters are treated as string separators because the initial values of the IFS shell variable are the space, tab, and end-of-line characters. Any number of consecutive spaces and tab characters are treated as a single separator. For details about the IFS shell variable, see *5.5 Shell variables*.

To use space or tab characters in character strings, you must enclose the character string containing the space or tab character in quotation marks (' or ").

(3) Line continuation

The following explains how to specify a command over multiple lines:

1. Specify a backslash (\) at the end of a line that is to continue onto the next line.
2. Enclose the entire multi-line specification in a single set of quotation marks (but do not specify a quotation mark at the end of each line that is continued onto the next line).

JP1/Advanced Shell treats such a set of lines as continued lines and processes them as a single line.

(4) Comments

You can specify comments in job definition scripts. When a hash mark (#) is specified in a line, the rest of the line from the hash mark through the end of the line is treated as a comment. However, if a hash mark is followed by -adsh, the line is processed as follows:

- Line that begins with #-adsh

  This line is treated as an extended script command.
- Specification of #-adsh continues onto the next line

  The continuation line is treated as a comment line. The following shows an example:

  ```
  echo ABC \
  #-adsh
  ```

  However, anytime #-adsh is not continued from the previous line or is not at the beginning of a line, an error results.
- Coding other than #-adsh is specified

  The line is treated as a comment line.

Note that a comment cannot be specified on a line that contains an extended script command.

(5) Wildcards

You can use wildcard characters to obtain file names and directory names that satisfy specified conditions and to compare such names with desired character strings.

The following table describes the wildcard characters supported in JP1/Advanced Shell.

Table 5–7: Wildcard characters supported in JP1/Advanced Shell

Wildcard character	Description
?	Matches any single character, except the dot (.) in dot files.
*	Matches a character string consisting of any number of characters, except the dot (.) in dot files.
[...]	Match is based on the character string enclosed in the square brackets ([]). When ! is specified at the beginning of the character string enclosed in the square brackets, the wildcard matches any character string other than the character string that is enclosed in the square brackets. To specify] as a character, specify it at the beginning of the character string. If two characters are separated by a hyphen (-), the wildcard matches any character that falls between those two characters, inclusive. To specify the hyphen (-) as a character, specify it at the beginning or end of the character string. For example specifications, see *Table 5-8 Specification examples using the square brackets wildcard*.
{str,...}	Expands the *str* strings delimited by the comma by using brace expansion. This expansion is not performed in the following cases: • The braceexpand shell option is invalid. • The noglob shell option is valid.

The following table shows example specifications of the square brackets wildcard ([]).

Table 5–8: Specification examples using the square brackets wildcard

Specification example	Description
[]a]	Matches the character string]a.
[!abc]	Matches any character string other than abc.
[0-9]	Matches one of the numbers from 0 through 9.
[a-z]	Matches any lowercase letter.
[A-Z]	Matches any uppercase letter.
[0-9a-zA-Z]	Matches any alphanumeric character.
[-abc]	Matches the character string -abc.
[!-abc]	Matches any character string other than -abc.

(6) Substitution

The following three substitution functions are available:

- Variable substitution

  Replaces a variable's status, the length (in bytes) of a character string constituting the value of a variable, the number of elements in an array variable, or the value of a variable expanded by matching the variable value and a specified pattern.

- Command substitution

  Treats a command's standard output as a variable value.

- File name substitution

  Expands a file name that matches a specified condition by using wildcard characters, such as * and ?. Note that if the noglob shell option is enabled, file names are not replaced. For details about the noglob shell option, see *5.6.1 Shell options that can be specified with the set command*.

(a) Variable substitution

Variable substitution includes substitution of a variable based on the status of the variable, substitution of a variable for the length of a character string constituting the value of a variable or for the number of elements in an array, substitution of a variable based on the result of pattern matching, and substring expansion.

- Variable substitution depending on the status of the variable

  The table below lists and describes the formats used to perform variable substitution depending on the status of the variable. In these formats, *variable* represents a variable name and *word* represents the variable that is expanded according to the status of *variable*. In the examples and results, `a` indicates an undefined variable, `b=NULL`, and `c=1`.

Table 5–9: Formats used to perform variable substitution depending on the status of the variable

Format	Description	Examples	Results
${*variable*:-*word*}	If *variable* is defined as a variable and a value is assigned to it, the value of *variable* is returned. If *variable* is defined but its value is `NULL` or undefined, the expansion result of *word* is returned. The value of *variable* remains unchanged.	`cnt=${a:-7}`	`7` is assigned to `cnt`.
		`cnt=${b:-8}`	`8` is assigned to `cnt`.
		`cnt=${c:-9}`	Value of `c` is assigned to `cnt`.
${*variable*-*word*}	If *variable* is defined as a variable and a value is assigned to it, the value of *variable* is returned. If *variable* is defined and its value is `NULL`, the value of *variable* (`NULL`) is returned. If *variable* is undefined, the expansion result of *word* is returned. The value of *variable* remains unchanged.	`cnt=${a-7}`	`7` is assigned to `cnt`.
		`cnt=${b-8}`	`NULL` is assigned to `cnt`.
		`cnt=${c-9}`	Value of `c` is assigned to `cnt`.
${*variable*:=*word*}	If *variable* is defined as a variable and a value is assigned to it, the value of *variable* is returned. If *variable* is defined and its value is `NULL` or undefined, the expansion result of *word* is assigned to *variable*, and then the value of *variable* is returned. Note that this format is applicable only to variables, not to positional parameters.	`cnt=${a:=7}`	`7` is assigned to `a` and the value of `a` is assigned to `cnt`.
		`cnt=${b:=8}`	`8` is assigned to `b` and the value of `b` is assigned to `cnt`.
		`cnt=${c:=9}`	Value of `c` is assigned to `cnt`.
${*variable*=*word*}	If *variable* is defined as a variable and a value is assigned to it, the value of *variable* is returned. If *variable* is defined and its value is `NULL`, the value of *variable* (`NULL`) is returned. If *variable* is undefined, the expansion result of *word* is assigned to *variable*, and then the value of *variable* is returned. Note that this format is applicable only to variables, not to positional parameters.	`cnt=${a=7}`	`7` is assigned to `cnt`.
		`cnt=${b=8}`	`NULL` is assigned to `cnt`.
		`cnt=${c=9}`	Value of `c` is assigned to `cnt`.
${*variable*:?[*word*]}	If *variable* is defined as a variable and a value is assigned to it, the value of *variable* is returned. If *word* is specified and *variable* is defined and its value is `NULL` or undefined, the expansion result of *word* is output to the standard error output, and then the job definition script is terminated. If *word* is omitted and *variable* is defined and its value is `NULL` or undefined, the `KNAX6050-E` message indicating that *variable* is undefined is issued, and then the job definition script is terminated. The value of *variable* remains unchanged.	`cnt=${a:?7}`	The expansion result is output to the standard error output, and then the shell is terminated.
		`cnt=${a:?}`	A message is output, and then the shell is terminated.
		`cnt=${b:?8}`	The expansion result is output to the standard error output, and then the shell is terminated.

Format	Description	Examples	Results
${*variable*:?[*word*]}	If *variable* is defined as a variable and a value is assigned to it, the value of *variable* is returned. If *word* is specified and *variable* is defined and its value is NULL or undefined, the expansion result of *word* is output to the standard error output, and then the job definition script is terminated. If *word* is omitted and *variable* is defined and its value is NULL or undefined, the KNAX6050-E message indicating that *variable* is undefined is issued, and then the job definition script is terminated. The value of *variable* remains unchanged.	cnt=${c:?9}	Value of c is assigned to cnt.
${*variable*?[*word*]}	If *variable* is defined as a variable and a value is assigned to it, the value of *variable* is returned. If *word* is specified and *variable* is undefined, the expansion result of *word* is output to the standard error output, and then the job definition script is terminated. If *word* is omitted and *variable* is undefined, the KNAX6050-E message indicating that *variable* is undefined is issued, and then the job definition script is terminated. The value of *variable* remains unchanged.	cnt=${a?7}	The expansion result is output to the standard error output, and then the shell is terminated.
		cnt=${a?}	A message is output, and then the shell is terminated.
		cnt=${b?8}	NULL is assigned to cnt
		cnt=${c?9}	Value of c is assigned to cnt. Value of c is assigned to cnt.
${*variable*:+*word*}	If *variable* is defined as a variable and a value is assigned to it, the expansion result of *word* is returned. Otherwise, NULL is returned. The value of *variable* remains unchanged.	cnt=${a:+7}	NULL is assigned to cnt.
		cnt=${b:+8}	NULL is assigned to cnt.
		cnt=${c:+9}	9 is assigned to cnt.
${*variable*+*word*}	If *variable* is defined as a variable and a value is assigned to it or the value is NULL, the expansion result of *word* is returned. Otherwise, NULL is returned. The value of *variable* remains unchanged.	cnt=${a+7}	NULL is assigned to cnt.
		cnt=${b+8}	8 is assigned to cnt.
		cnt=${c+9}	9 is assigned to cnt.

- Variable substitution to the length of the character string constituting the value of the variable or to the number of array elements

  The table below lists and describes the formats used to perform variable substitution to the length of the character string constituting the value of the variable or to the number of array elements. In these formats, *variable* represents a variable name and *array* represents an array name.

Table 5–10: Formats used to perform variable substitution to the length of the character string constituting the value of the variable or to the number of array elements

Format	Description
${#*variable*}	If *variable* is * or @, the variable is replaced with the number of positional parameters. Otherwise, the variable is replaced according to the setting specified for the VAR_SHELL_GETLENGTH environment setting parameter: • BYTE Replaces the length of the value stored in *variable* with the number of bytes. • CHARACTER Replaces the length of the value stored in *variable* with the number of characters. If the VAR_SHELL_GETLENGTH environment setting parameter is not specified, the operation is the same as when BYTE is specified in the VAR_SHELL_GETLENGTH environment setting parameter.
${#*array*[*]}	The value is replaced with the number of elements of the array specified by *array*.

Format	Description
${#*array*[@]}	The value is replaced with the number of elements of the array specified by *array*.

- Variable substitution based on the result of pattern matching

  The table below lists and describes the formats used to perform variable substitution based on the result of pattern matching. In these formats, *variable* represents a variable name and *pattern* represents a character string used to perform pattern matching with *variable*. Wildcard characters can be used in *pattern*.

  Table 5–11: Formats used to perform variable substitution based on the result of pattern matching

Classification	Format	Description
Leading match	${*variable*#*pattern* }	If *pattern* matches the leading part of the variable value, the value in variable is replaced with its value less the shortest matching part (the shortest-matching part is deleted). Otherwise, the variable is replaced with the value of *variable*.
	${*variable*##*pattern* }	If *pattern* matches the leading part of the variable value, the variable is replaced with its value less the longest matching part (the longest-matching part is deleted). Otherwise, the variable is replaced with the value of *variable*.
Trailing match	${*variable*%*pattern* }	If *pattern* matches the trailing part of the variable value, the variable is replaced with its value less the shortest matching part (the shortest-matching part is deleted). Otherwise, the variable is replaced with the value of *variable*.
	${*variable*%%*pattern* }	If *pattern* matches the trailing part of the variable value, the variable is replaced with its value less the longest matching part (the longest-matching part is deleted). Otherwise, the variable is replaced with the value of *variable*.

  The following shows an example of outputting a variable value with deletion of a specified character string (leading match).

  Contents of the job definition script:

```
abc=abcd1234xyz987abcd1234efg
echo ${abc#abcd}          # 1.
echo ${abc#a*2}           # 2.
echo ${abc##a*2}          # 3.
echo ${abc#*1234}         # 4.
echo ${abc##*1234}        # 5.
echo ${abc#1234}          # 6.
```

  `STDOUT` file contents of the executed job:

```
********   JOB SCOPE STDOUT    ********
1234xyz987abcd1234efg      ← Result of 1: The leading character string
abcd is deleted.
34xyz987abcd1234efg        ← Result of 2: The string of characters from
a through 2 is deleted (shortest match).
34efg                      ← Result of 3: The string of characters from
a through 2 is deleted (longest match).
xyz987abcd1234efg          ← Result of 4: The string of characters from
the beginning through 1234 is deleted (shortest match).
efg                        ← Result of 5: The string of characters from
the beginning through 1234 is deleted (longest match).
abcd1234xyz987abcd1234efg  ← Result of 6:The value of abc is output
because there is no leading match.
```

The following shows an example of outputting a variable value with deletion of a specified character string (trailing match).

Contents of the job definition script:

```
abc=abcd1234xyz987abcd1234
echo ${abc%1234}         # 1.
echo ${abc%d*4}          # 2.
echo ${abc%%d*4}         # 3.
echo ${abc%34*}          # 4.
echo ${abc%%34*}         # 5.
echo ${abc%abcd}         # 6.
```

STDOUT file contents of the executed job:

```
********   JOB SCOPE STDOUT    ********
abcd1234xyz987abcd      ← Result of 1: The trailing character string 1234
is deleted.
abcd1234xyz987abc       ← Result of 2: The trailing character string from
d through 4 is deleted (shortest match).
abc                     ← Result of 3: The trailing character string from
d through 4 is deleted (longest match).
abcd1234xyz987abcd12    ← Result of 4: The character string beginning
with 34 is deleted (shortest match).
abcd12                  ← Result of 5: The character string beginning
with 34 is deleted (longest match).
abcd1234xyz987abcd1234 ← Result of 6: The value of abc is output because
there is no trailing match.
```

- Substring expansion

  The following table lists and describes the formats for variable substitution that is performed based on the result of substring expansion.

Table 5–12: Formats for substring expansion

Format	Description
${*variable*:*offset*}	Retrieves characters from the expansion result of *variable*. *offset* specifies the start position of the characters to be retrieved.
${*variable*:*offset*:*length*}	Retrieves as many characters as there are in maximum *length* from the expansion result of *variable*. *offset* specifies the start position of the characters to be retrieved.
${*array*[*]:*offset*}	Retrieves elements that begin with ${*array*[*offset*]} of the array.
${*array*[*]:*offset*:*length*}	Retrieves as many elements as there are in *length* that begin with ${*array*[*offset*]} of the array.
${*array*[@]:*offset*}	Retrieves elements that begin with ${*array*[*offset*]} of the array.
${*array*[@]:*offset*:*length*}	Retrieves as many elements as there are in *length* that begin with ${*array*[*offset*]} of the array.

Legend:

variable: Specifies a variable name.

array: Specifies an array name.

offset: Specifies the start position in the character string or array element subject to partial expansion.

length: Specifies the number of characters or array elements to be expanded.

(b) Command substitution

The table below lists and describes the formats used to perform command substitution. In these formats, `command` represents the name of the command and the arguments to be executed.

In Windows, command substitution is performed in the current process except for external commands.

Table 5–13: Formats used to perform command substitution

Format name	Format	Description
`$()` format	$(*command*)	If the character string of *command* contains a backslash (\\), \\ has no special meaning.
Grave character format	`*command*`	If the character string of *command* contains \\, \\ has a special meaning. To specify one command substitution within another, specify \\ immediately before the inner grave accent mark character string such as `*command* \\`*command*\\``.

In the grave character format, a backslash (\) in the character string of `command` is treated as a metacharacter. Therefore, if the character string of `command` contains \, the execution results differ between the `$()` and the grave character formats. We recommend that you use the `$()` format.

The following shows example specifications and execution results.

- `$()` format

  Specification example:

  ```
  echo $(echo '\$x')
  ```

  Execution result:

  ```
  \$x
  ```

- Grave character format

  Specification example:

  ```
  echo `echo '\$x'`
  ```

  Execution result:

  ```
  $x
  ```

(c) File name substitution

The table below lists and describes the formats used to perform file name substitution. In these formats, *pattern* represents a character string used for pattern matching. Wildcard characters can be used in *pattern*.

Table 5–14: Formats used to perform file name substitution with multiple patterns

Format	Description	Example	Matching character string
?(*pattern* | *pattern* ...)	Matches one of the character strings specified as *pattern*.	`?(h)`	Null character string, `h`
*(*pattern* | *pattern* ...)	Matches none or any number of the character strings specified as *pattern*.	`*(h)`	Null character string, `h`, `h`, `hh`, `hhh`, ...

Format	Description	Example	Matching character string
+(*pattern*|*pattern* ...)	Matches at least one of the character strings specified as *pattern*.	+(h)	h, hh, hhh, ...
@(*pattern*|*pattern* ..)	Matches only one of the character strings specified as *pattern*.	@(h)	h
!(*pattern*|*pattern* ...)	Matches all but one of the character strings specified as *pattern*.	!(h)	Any character string without h

You can replace file names by specifying multiple patterns delimited by the vertical bar (|). Do not specify any spaces before or after the vertical bar delimiter. If there is such a space, it will be regarded as part of the pattern.

The following shows examples.

Example of file name substitution

Contents of job definition script:

```
ls -C                        # 1. Display the file in the current directory
ls -C ?(*.sh|*.exe|*.dot)  # 2. Display the file names whose extension is
sh,exec, or dot
ls -C *(*.sh|*.exe)        # 3. Display the file names whose extension is
either sh or exe
ls -C +(*.jhs|*h)          # 4. Display the file names whose extension is
jhs or that end with h
ls -C @(*.c|*.jhs)         # 5. Display the file names whose extension is c
or jhs
ls -C !(*.c|*.jhs)         # 6. Display the file names whose extension is
neither c nor jhs
```

STDOUT file contents of the executed job:

```
********   JOB SCOPE STDOUT    ********
a.jhs  a.sh   a.txt  func.c  ← Execution result of 1
a.sh                         ← Execution result of 2
a.sh                         ← Execution result of 3
a.jhs a.sh                   ← Execution result of 4
a.jhs  func.c                ← Execution result of 5
a.sh  a.txt                  ← Execution result of 6
```

File names beginning with a dot (.) are excluded as targets of pattern matching. To include file names beginning with a dot as targets of pattern matching, you must specify the dot explicitly.

(7) Arithmetic expansion

Arithmetic expansion involves performing an arithmetic operation and then assigning the result to a variable. The following shows the format and an example of arithmetic expansion.

```
$((arithmetic-expression))
```

Example of arithmetic expansion:

Contents of job definition script:

```
x=100          # 1. Value 100 is assigned to variable x
x=$((x-1))     # 2. Perform the arithmetic operation and then assign
the result to x
echo $x        # 3. Output the value of x to the standard output
```

`STDOUT` file contents of the executed job:

```
********   JOB SCOPE STDOUT    ********
99             ← Value 99 is assigned to x.
```

(8) Input and output redirection

In job definition scripts, you can change the output destination of command execution results and the input source of information needed for command execution before commands are executed. These capabilities are called input and output redirection. This subsection explains input and output redirection as supported by JP1/Advanced Shell.

(a) Redirection

The table below describes the redirection methods supported by JP1/Advanced Shell. A redirection is interpreted from left to right.

Table 5–15: Redirection available in JP1/Advanced Shell

Redirection	Description
> *file*	Uses *file* as the standard output. If *file* does not exist, the file is created. If *file* already exists, the existing file is overwritten.
< *file*	Uses *file* as the standard input.
command_1 | *command_2*	This is a pipe. It uses the standard output of *command_1* as the standard input for *command_2*.
>>*file*	Uses *file* as the standard output. If *file* does not exist, the file is created. If *file* already exists, new data is added to the existing file.
>|*file*	Uses *file* as the standard output. If *file* does not exist, the file is created. If *file* already exists, the existing file is overwritten.
<>*file*	Opens *file* as the standard input for read and write operations.
<<*label*	This is a here document.
n>*file*	Redirects the output destination of file descriptor *n* to *file*.
n<*file*	Inputs file descriptor *n* from *file*.
>&*n*	Copies the standard output to file descriptor *n*.
<&*n*	Copies the standard input from file descriptor *n*.
>&-	Closes the standard output.
<&-	Closes the standard input.
|&[#1]	In UNIX, this direction starts a background process that involves input and output from the parent process.
>&p[#2]	Redirects the output destination of the background process to the standard output.

Redirection	Description
`<&p`[#2]	Redirects the input of the background process to the standard input.

#1

If `|&` is used to start multiple background processes that involve input and output operations from parent processes, the `KNAX6029-E` message might be issued and the job definition script might terminate with an error. Therefore, when you use `|&`, use a command such as `wait` to prevent multiple background processes from being started concurrently.

This is not supported in the Windows edition.

#2

This redirection method is not available in a Windows execution environment because background processes cannot be generated.

(b) File descriptors

The following table lists and describes the file descriptors in JP1/Advanced Shell by type of input and output.

Table 5–16: File descriptors in JP1/Advanced Shell by type of input and output

Input or output type	File descriptor	Remarks
Standard input	`0`	--
Standard output	`1`	If `SPOOL` is specified in the `-s` option of the `adshexec` command and the `OUTPUT_STDOUT` parameter in the environment file, the terminal is not opened by this file descriptor because JP1/Advanced Shell uses the standard output for output from the shell to the spool's `STDOUT` (`stdout`).
Standard error	`2`	The terminal is not opened by this file descriptor because JP1/Advanced Shell uses the standard error output for output from the shell to the spool's `STDERR` (`stderr`).
Other	`3` through `9`	The user can use these file descriptors in job definition scripts.

Legend:

--: None

(c) Here document

A *here document* refers to creation of the standard input within the job definition script. The following table lists and describes the formats related to here documents.

Table 5–17: Formats related to here documents

Format	Description
`<<` *label* *document* *label*	The *document* part enclosed by *label* and *label* is passed to the standard input. A variable specified for *document* is replaced and spaces and tab characters are passed as is to the standard input.
`<<-` *label* *document* *label*	The *document* part enclosed by *label* and *label* is passed to the standard input. A variable specified for *document* is replaced and spaces are passed as is to the standard input. The tab character at the beginning of each line is deleted.
`<< \`*label* *document* *label*	The *document* part enclosed by *label* and *label* is passed to the standard input. A variable specified for *document* is not replaced.
`<< '`*label*`'` *document* *label*	

(d) Pipes

When you want to connect multiple commands and use the standard output from one command as the standard input to another, you can use pipes to connect the commands. A set of commands connected by a pipe is called a pipeline.

- In Windows

  A pipeline is processed from left to right, and the commands in a pipeline are executed sequentially. Among built-in commands, functions, and external commands, only the external commands are executed as separate processes. Temporary files are used to transfer data between commands.

- In UNIX

  A pipeline is processed from left to right, and the commands in a pipeline are executed as separate processes.

  You can use the `PIPE_CMD_LAST` parameter to define whether the last command is to be executed in the current process or in a separate process. The default is the current process.#

  For details about the `PIPE_CMD_LAST` parameter, see *PIPE_CMD_LAST parameter (defines execution processing for the last command in a pipe) (UNIX only)* in *7. Parameters Specified in the Environment Files*.

  #

  > This is not applicable to commands executed in separate processes (such as external commands, child jobs, UNIX-compatible commands, shell operation commands, subshells, commands with a background specified, and commands with a background process specified).

  Also in UNIX, if multiple commands are connected by a pipe and then the output command outputs data to the pipe after the input command has terminated and while there is no process, the output command might receive `SIGPIPE`.

  In JP1/Advanced Shell, the command issues the `KNAX6522-E` message and terminates with an error unless the command handles `SIGPIPE`.

(9) Command separators

Command separators enable you to specify multiple commands on a single line of a job definition script. The following table lists and describes the formats for command separators.

Table 5–18: Formats for command separators

Format	Description
command`;`	The specification up to the semicolon (`;`) is interpreted as a command and its arguments.
command_1 Δ0`&&` Δ0*command_2*	This is the *AND* control operator. If the left-hand command terminates with return code `0`, the right-hand command is executed.
command_1 Δ0`||` Δ0*command_2*	This is the *OR* control operator. If the left-hand command terminates with any non-zero return code, the right-hand command is executed.

The command separators can be used to handle and execute a group of commands as a single command group.

(10) Grouping commands

You can execute multiple commands in a batch by grouping them. You can also group multiple command groups. The following table lists and describes the formats for grouping commands.

Table 5–19: Formats for grouping commands

Format	Description
(*command _1*; Δ0*command _2*; ...)	Executes the grouped commands in a subshell (child process).[#] If the environment is changed during command execution, the change is not inherited by the current shell. Delimit the commands in the group with the semicolon or end-of-line character.
(*command_1* (end-of-line) *command_2* (end-of-line) ...)	
{ Δ1*command_1*; Δ0*command _2*; ...;}	Executes the grouped commands in the current shell. If the environment is changed during command execution, the change is inherited after the commands have terminated. Delimit the commands in the group with the semicolon or end-of-line character. In this format, you must insert at least one space after the {, and you must insert a semicolon (;) or end-of-line character after the last command.

{ Δ1*command_1* (end-of-line)
command_2 (end-of-line)
...(end-of-line)
}	

#

Execution of grouped commands in a subshell (child process) is not supported in a Windows execution environment.

(11) Other metacharacters

The following table lists and describes the other metacharacters that are supported.

Table 5–20: Supported metacharacters

Metacharacter	Description
~[#1]	Replaced with the HOME shell variable.[#2]
~*any-character-string*	(UNIX only) Checks whether the user name matching the character string up to / or an argument delimiter is registered in the /etc/passwd file. If the user name is registered, this metacharacter is replaced with the corresponding user's login directory. If the user name is not registered, the specified character string is interpreted as a character string as is.
~+[#1]	Replaced with the PWD shell variable.
~-[#1]	Replaced with the OLDPWD shell variable.
&[#3]	Executes the job definition script or function in the background.

#1

The characters ~, ~+, and ~- are not replaced with the corresponding shell variables if they are enclosed in quotation marks or specified immediately before an escape character (\) or a character string enclosed in quotation marks. If such a case, you must use the shell variables themselves.

#2

The HOME shell variable is not specified automatically. You must define it as an environment variable.

#3

& is not supported in a Windows execution environment.

5.1.7 Execution in a separate process (UNIX only)

If the following formats appear in a job definition script, execution takes place in a different process from the current process. A change made by another process is not inherited to the current process. An example of execution in another process is shown in the following.

Executing a separate process by using a pipe (|):

The handling of execution in a separate process using a pipe (|) depends on the PIPE_CMD_LAST parameter specification.

Example:

```
hostname | read STR
```

- When CURRENT is specified in the PIPE_CMD_LAST parameter

  The hostname command is executed in a separate process, but the read command is executed in the current process.
- When OTHER is specified in the PIPE_CMD_LAST parameter

  The hostname and read commands are both executed in a separate process. Therefore, the result of the hostname command is not assigned to the STR variable.

Executing a separate process by using command substitution ($(), ``):

Example:

```
$(date '+%Y%m%d')  # Execute a command whose name is the output result
of the date command
`date '+%Y%m%d'`   # Execute a command whose name is the output result
of the date command
```

Executing a background process by using |&:

Example:

```
echo abc |&       # Output the character string abc by a background
process
sleep 1
read -p STR       # Read the data that was output by the background
process
echo $STR
```

Executing grouped commands in a subshell:

Example:

```
(TZ=GMT; export TZ; date) # Convert the TZ environment variable
temporarily
                          # to GMT and then output the time
```

Background execution by using &:

Example:

```
sleep 10 &
```

Replacement of character strings is performed in a separate process. Therefore, if the character string assigned to a variable by using one of the above formats is executed as a command, the variable name before substitution is output to the command execution results in the job execution log file. However, for aliases, the command name obtained after the alias was resolved is output because the alias is resolved in the current process before it is executed.

Contents of the job definition script:

```
ls="ls -lt"           # Assign "ls -lt" to variable ls
alias gt="grep test"  # Define "grep test" for alias gt
$ls | gt              # Execute ls -lt | grep test
```

Contents of the job execution log file for the executed job

```
********  JOB CONTROLLER MESSAGE  ********
16:01:11 152286 KNAX0091-I ADSH152286 The job started.
16:01:11 152286 KNAX7901-I The adshexec command will wait for all
asynchronous processes at the end of the job.
16:01:11 152286 KNAX7902-I The adshexec command will run in tty stdin
mode.
16:01:11 152286 KNAX6110-I Execution of the command ls=ls -lt (line=1)
finished successfully. exit status=0 execution time=0.000s CPU time=0.000s
16:01:11 152286 KNAX6112-I Execution of the command alias (line=2)
finished successfully. exit status=0 execution time=0.000s CPU time=0.000s
16:01:11 152286 KNAX6116-I Execution of the command $ls (line=3) finished
successfully. exit status=0 execution time=0.002s CPU time=0.000s
16:01:11 152286 KNAX6116-I Execution of the command /opt/jp1as/cmd/grep
(line=3) finished successfully. exit status=0 execution time=0.001s CPU
time=0.000s
16:01:11 152286 KNAX0098-I ADSH152286 The job ended. exit status=0
execution time=0.007s CPU time=0.000s
```

To enable specification of the `#-adsh_rc_ignore` command for a command specified in the above format, you must specify the base name of the character string before replacement as the command name in the argument of the `#-adsh_rc_ignore` command.

Important note

When the following standard shell commands are executed in a separate process, whether execution of the command results in normal termination or error termination differs from when these commands are executed in the current process, as described in the following:

- `let` command

  If this command is executed in a separate process without an arithmetic expression specified, it terminates normally with return code `1`.

- `exit` and `return` commands

  If either of these commands is executed in a separate process with a non-numeric value specified in its argument, it terminates normally with return code `1`.

- `getopts` command

  If this command is executed in a separate process and the end of options is detected, the command terminates with an error with return code `1`. Use the `successRC` attribute of the `#-adsh_step_start` command or the `#-adsh_rc_ignore` command to make sure that the `getopts` command will not terminated with an error.

- `read` command

  If this command is executed in a separate process and detects an end-of-file (EOF), it terminates with an error with return code `1`. You can prevent such a termination with an error by using the `successRC` attribute of the `#-adsh_step_start` or `#-adsh_rc_ignore` command.

If this command is executed in a separate process and that process receives a termination request signal, the command's execution might continue in the separate process depending on the type of signal. To forcibly terminate the command's processing in a separate process, send SIGTERM to that process.

5.1.8 Pattern matching

In JP1/Advanced Shell, you can compare patterns in some of the standard commands and script control statements. For example, if the format ${*variable*#*pattern*} is used, the shortest part of the *variable* value that matches the pattern *pattern* is deleted and then the remainder of the value is set in *variable*. The following shows an example.

Example:

```
$ var=abcd
$ echo ${var#a[b]}
cd
```

Whether a variable value can be compared with a specified pattern depends on the function. If the format of the pattern is invalid, the pattern is treated as a normal character string during comparison. For example, [a*(b] treats all [, *(, and], which normally have a special meaning as a pattern, as normal characters. The following shows an example.

Example:

```
$ var='[a*(b])cd'
$ echo ${var#[a*(b])}
cd
```

5.1.9 Escape characters

(1) List of escape characters

The echo and print commands interpret the characters listed in the following table as escape characters:

Escape character	Meaning	echo command	print command
\a	Alert character (bell)	Y	Y
\b	Backspace character	Y	Y
\c	Suppresses the linefeed character at the end of a line (characters following \c are not output)	Y	Y
\f	Formfeed character (page break)	Y	Y
\n	Linefeed character	Y	Y
\r	Carriage return character	Y	Y
\t	Tab character	Y	Y
\v	Vertical tab character	Y	Y
\0*nnn*#1	ASCII character represented by one, two, or three octal digits (0 to 7)	Y	Y

Escape character	Meaning	echo command	print command
`\x`*nn*[#2]	ASCII character represented by one or two hexadecimal digits (`0` to `9`, `a` to `f`, `A` to `F`)	Y	N
`\\`	A single backslash character	Y	Y

Legend:

Y: Can be specified.

N: Cannot be specified.

#1

If a specified ASCII character consists of one or two digits and two or one leading zeros, respectively, are added to make it three digits, the ASCII character will still be treated as consisting of only one or two digits. For example, the following three specifications are all interpreted as being the same, in which case the alert character (bell) is output three times:

```
echo -e "\07"
echo -e "\007"
echo -e "\0007"
```

#2

Enabled only when `YES` is specified in the `ESCAPE_SEQ_ECHO_HEX` environment setting parameter. For details about the `ESCAPE_SEQ_ECHO_HEX` parameter, see *ESCAPE_SEQ_ECHO_HEX parameter (specifies whether ASCII code characters in hexadecimal notation are to be interpreted as escape characters)* in *7. Parameters Specified in the Environment Files*.

If a specified ASCII character consists of one digit and a leading zero is added to make it two digits, the ASCII character will still be treated as consisting of only one digit. For example, the following two specifications are interpreted as being the same, in which case the linefeed character is output twice:

```
echo -e "\xA"
echo -e "\x0A"
```

(2) Handling of the echo command with neither -e nor -E option specified

If neither the `-e` option (interprets escape characters) nor the `-E` option (does not interpret escape characters) is specified in the `echo` command, the handling of escape characters depends on the setting in the `ESCAPE_SEQ_ECHO_DEFAULT` environment parameter. For details about the `ESCAPE_SEQ_ECHO_DEFAULT` environment parameter, see *ESCAPE_SEQ_ECHO_DEFAULT parameter (defines the action of the echo command when the escape-character option is omitted)* in *7. Parameters Specified in the Environment Files*.

5.1.10 Specifying extended script commands

In job definition scripts, a line beginning with `#-adsh` is treated as a request to an extended script command and the corresponding extended script command is executed.

For details about extended script commands, see *9.5 Extended script commands*.

5.1.11 Specifying external commands

Programs that are not built-in commands in job definition scripts are referred to collectively as external commands. External commands include UNIX-compatible commands, OS-provided commands, and user-created programs. You can execute external commands by specifying their command names in the job definition scripts.

The following show how to specify an external command:

```
$ path-of-external-command
```

(1) Executing external commands in Windows

(a) Defining the extensions of external commands

In Windows, you can execute executable files with the extensions .com, .exe, .cmd, and .bat from job definition scripts. If you register these extensions in the PATHEXT shell variable, you can execute such executable files from job definition scripts without having to specify their extensions.

External commands are executed from a job definition script in the order that the extensions are registered in the PATHEXT environment variable. For example, if the value of the PATHEXT environment variable is .COM;.EXE;, and ls.com and ls.exe are stored at the locations indicated by the PATH shell variable, ls.com is executed first.

An example search for an external command is shown below. In this example, the value of the PATHEXT environment variable is .COM;.EXE;.

```
ls      <-- Because no extension is specified, the extensions .com and .exe
are added in this order to locate and execute the corresponding external
command.
ls.exe  <-- ls.exe is executed because the extension is specified.
```

(b) Skipping processes before executing external commands

Before JP1/Advanced Shell executes an external command, it performs the processing explained below on the path and arguments of the external command. If this processing is not needed, execute the external command with the -w option specified in the command command.

- Converts a backslash (\) preceding a double quotation mark (") to \\.
- Adds a backslash \ in front of a double quotation mark (").
- Encloses in double quotation marks (").

For details about the -w option of the command command, see *command command (executes a command)* in *9.3 Standard shell commands*.

Example:

```
command -w ."\\prog.exe" "ABC"
```

(c) Limitations on batch file execution

The following limitations apply to batch file execution:

- When there is no argument
  If a batch file path contains any of the special characters &, (,), [,], {, }, ^, =, ;,!, ', +, ,, `, and ~, specify it in the format <*cmd.exe-path*> /c <*batch-file-path*> and then escape the special character in the batch file path by using "'\.
- When there are arguments
  All the following limitations apply;
  - Specify a batch file in the format <*cmd.exe-path*> /c <*batch-file-path*> <*argument-1*> <*argument-2*> ... <*argument-n*>.

- If a batch file path contains any of the special characters &, (,), [,], {, }, ^, =, ;, !, ', +, ,, `, and ~, use "'\ to escape the special character in the batch file path.
- If an argument contains a space or any of the special characters &, (,), [,], {, }, ^, =, ;, !, ', +, ,, `, and ~, use "'\ to escape the space or special character.
- Arguments enclosed in double quotation marks (") are passed to the batch file. The last argument has a double quotation mark (") only at the beginning. To reference arguments from a batch file, use the formats %~1, %~2, ... %~*n* to automatically remove the double quotation mark (").

(2) Executing external commands in UNIX

You can execute an executable binary file with execution permissions granted from a job definition script.

You can also execute a text file with execution permissions granted from a job definition script by specifying #! *executable-program-path* at the beginning of the file. In this case, the executable program is executed according to the specification of #!.

(3) Priority of command execution methods

When the job controller executes a file specified in a job definition script, it checks the conditions listed below in the order shown here.

Windows only

1. If the file satisfies the CHILDJOB_PGM parameter condition, the job controller executes the file as a child job.
2. If the file satisfies the CHILDJOB_SHEBANG parameter, the job controller executes the file as a child job.
3. If the file satisfies the default definition for the CHILDJOB_SHEBANG parameter, the job controller executes the file as a child job.
4. If the file satisfies the CHILDJOB_EXT parameter, the job controller executes the file as a child job.
5. If the file has the extension exe, bat, cmd, or com, the job controller executes the file as a command.
6. If none of the above conditions is satisfied, startup of the command fails. However, job execution continues.

UNIX only

1. If the file satisfies the CHILDJOB_PGM parameter condition, the job controller executes the file as a child job.
2. If file execution permissions have been granted, the job controller performs the checks beginning with 3. If execution permissions have not been granted, startup of the command fails and the command terminates with an error. However, job execution continues.
3. If the file satisfies the CHILDJOB_SHEBANG parameter condition, the job controller executes the file as a child job.
4. If the file satisfies the default definition for the CHILDJOB_SHEBANG parameter, the job controller executes the file as a child job.
5. If the file satisfies the CHILDJOB_EXT parameter condition, the job controller executes the file as a child job.
6. If the file is a binary file, the job controller executes the file as a command.
7. If the file is a text file, the job controller executes the file as a script. If #! is specified, the job controller executes the executable program specified in #!. If #! is not specified, the job controller executes the file with /bin/sh.

The following examples use parameters related to child jobs (test.sh exists and execution permissions have been granted).

Example 1:

- Contents of the environment file

```
#-adsh_conf CHILDJOB_PGM /bin/sh
#-adsh_conf CHILDJOB_SHEBANG /bin/ksh
#-adsh_conf CHILDJOB_EXT sh
```

- Contents of `test.sh`

```
#!/bin/ksh
echo JP1AS
```

- Contents of the job definition script

```
/bin/sh ./test.sh
```

→ The `CHILDJOB_PGM` parameter is applied and `test.sh` is executed as a child job.

Example 2:

- Contents of the environment file

```
#-adsh_conf CHILDJOB_PGM /bin/sh
#-adsh_conf CHILDJOB_SHEBANG /bin/ksh
#-adsh_conf CHILDJOB_EXT sh
```

- Contents of `test.sh`

```
#!/bin/ksh
echo JP1AS
```

Contents of the job definition script

```
./test.sh
```

→ The `CHILDJOB_SHEBANG` parameter is applied and `test.sh` is executed as a child job.

Example 3:

- Contents of the environment file

```
#-adsh_conf CHILDJOB_PGM /bin/sh
#-adsh_conf CHILDJOB_SHEBANG /bin/ksh
#-adsh_conf CHILDJOB_EXT sh
```

- Contents of `test.sh`

```
#!/bin/sh
echo JP1AS
```

- Contents of the job definition script

```
./test.sh
```

→ The `CHILDJOB_EXT` parameter is applied and `test.sh` is executed as a child job.

Example 4:

- Contents of the environment variable

```
#-adsh_conf CHILDJOB_PGM /bin/sh
#-adsh_conf CHILDJOB_SHEBANG /bin/ksh
#-adsh_conf CHILDJOB_EXT sh
```

- Contents of `test.sh`

```
#! /opt/jp1as/bin/adshexec
echo JP1AS
```

- Contents of the job definition script

```
./test.sh
```

➜ `test.sh` is executed as a child job because it satisfies the default definition for the `CHILDJOB_SHEBANG` parameter.

Example 5:

- Contents of the environment file

```
#-adsh_conf CHILDJOB_PGM /bin/sh
#-adsh_conf CHILDJOB_SHEBANG /bin/ksh
#-adsh_conf CHILDJOB_EXT sh
```

- Contents of `test.sh`

```
#!/bin/ksh
echo JP1AS
```

- Contents of the job definition script

```
/bin/ksh ./test.sh
```

➜ Because none of the `CHILDJOB_EXT`, `CHILDJOB_PGM`, `CHILDJOB_SHEBANG`, and `CHILDJOB_SHEBANG` parameters' default definitions apply, `test.sh` is executed with `/bin/ksh`, not as a child job.

Example 6:

- Contents of the environment variable

```
#-adsh_conf CHILDJOB_PGM /opt/jp1as/bin/adshexec
#-adsh_conf CHILDJOB_SHEBANG /bin/ksh
#-adsh_conf CHILDJOB_EXT sh
```

- Contents of `test.sh`

```
#!/bin/ksh
echo JP1AS
```

- Contents of the job definition script

```
/opt/jp1as/bin/adshexec ./test.sh
```

➜ The `CHILDJOB_PGM` parameter is applied and `test.sh` is executed as a child job.

Example 7:

- ■ Contents of the environment file

```
#-adsh_conf CHILDJOB_PGM ./test.sh
#-adsh_conf CHILDJOB_SHEBANG /bin/ksh
#-adsh_conf CHILDJOB_EXT sh
```

- ■ Contents of `test.sh`

```
#!/bin/ksh
echo JP1AS
```

- ■ Contents of the job definition script

```
./test.sh
```

➜ According to the order or condition check, the `CHILDJOB_PGM` parameter is checked. Because the `CHILDJOB_PGM` parameter definition is satisfied, the corresponding replacement is performed. However, because the file to be executed as a child job is not specified, an execution error results.

(4) Priority of child jobs or external commands that have the same name as the function

The following is the priority of child jobs or external commands that have the same name as the function:

- If the function has already been defined in the main script or the function definition file has been loaded by `#-adsh_script` or a `.` (dot) command, the function always takes effect.
- If the function has the automatic loading attribute and the directory specified in the `FPATH` shell variable contains a function definition file with the same name as the function, the function takes effect.
- If the function does not have the automatic loading attribute but the directory specified in the `FPATH` shell variable contains a function definition file with the same name as the function, the following conditions are checked in this order:
  1. If the directory specified in the `FPATH` shell variable contains a job definition script with the same name as the function or an external command, the child job or the external command is executed.
  2. If the directory specified in the `FPATH` shell variable does not contain a corresponding job definition script or external command, the function is executed.
- If neither the directory specified in the `PATH` environment variable nor the directory specified in the `FPATH` shell variable contains a corresponding file, the processing terminates with an error because there is no command to be executed.

5.1.12 Specifying UNIX-compatible commands

You can enter UNIX-compatible commands from the Window's command prompt or a UNIX shell. You can also use UNIX-compatible commands in a job definition script and then execute the script in Windows or UNIX. The UNIX-compatible commands enable you to use functions such as displaying, creating, and searching files and directories.

You can share in Windows and in UNIX job definition scripts that use executable UNIX-compatible commands if you use the `ADSH_DIR_CMD` shell variable to define the UNIX-compatible commands. For details about the `ADSH_DIR_CMD` shell variable, see *5.5.1 Shell variables set by JP1/Advanced*.

The following is an example of the `ADSH_DIR_CMD` shell variable that defines a UNIX-compatible command (`date` command) in a job definition script:

```
"${ADSH_DIR_CMD}date"
```

5.1.13 Specifying a shell for running job definition scripts and checking formats

(1) Specifying a shell

In UNIX, you can specify a shell to be used to execute a job definition script on the first line of the job definition script file. The following shows how to specify a shell:

```
#!executable-file-path
```

The command shown below executes a job definition script by using the shell with the executable file path specified on the first line of the job definition script (if the executable file path is omitted, `/bin/sh` is used):

```
$ job-definition-script-file-path
```

The command shown below that specifies the path of the `adshexec` command at the beginning executes a job definition script on the path of the `adshexec` command, regardless of the information specified on the first line of the job definition script:

```
$ path-of-adshexec-command job-definition-script-file-path
```

(2) Checking the lexical format

When the `adshexec` command is used to execute a job definition script file, the command checks the lexical format used in the job definition script file and then executes the file. Note that the command does not check the format of an external file that is read by a **.** (dot) command within the job definition script file.

If you want to only check the lexical format without executing the job definition script, execute the `adshexec` command with the `-c` option specified.

5.2 Conditionals

In job definition scripts, the processing to be executed is controlled based on the results of conditional expressions specified in control statements. This section explains control statements and conditional expressions as conditionals.

5.2.1 Control statements

JP1/Advanced Shell supports the control statements listed below. For details, see *9.6 Script control statements*.

- `case`
  Executes one of the processes depending on the contents of character strings.
- `for`
  Executes the same processing repeatedly by changing the value sequentially.
- `if`
  Executes the processing that satisfies each condition by branching out the processing depending on conditions.
- `until`
  Executes the same processing repeatedly until the condition is satisfied.
- `while`
  Executes the same processing repeatedly while the condition is satisfied.

You can specify command substitution and any commands other than extended script commands at the following locations in the control statements.

Locations where command substitution can be specified

- Expressions and patterns in `case` statements
- `wordlist` in `for` statements

Locations where command substitution and any commands other than extended script commands can be specified:

- Condition 1 (condition that follows `if`) and condition 2 (condition that follows `elif`) in `if` statements
- Conditions in `until` statements
- Conditions in `while` statements

The return code of a specified command is not applied as the return code of the job and job step. When execution of control statements is completed, the return code of the last command executed within the block of control statements becomes the return code of the job and job step.

However, if any of the conditions listed below is satisfied, the return code of a specified command is applied as the return code of the job and job step. When a job step is completed, the return code of the specified command becomes the return code for the job and job step.

1. The specified command is the `exit` or `return` command, not a command substitution.
2. The specified command is a special built-in command, not a command substitution, and it resulted in an error.
3. The specified command, which is neither 1 nor 2 above, resulted in an error and the control statement was in a job step for which `stop` was specified for the `onError` attribute.

If you want to apply the return code of a command resulting in an error as the job's return code, such as when a job is to result in an error in JP1/AJS, specify the control statement in a job step for which `stop` is specified for the `onError` attribute.

The following shows examples of the `if` statement:

Example 1:

```
if `cmdA true`   <-- cmdA is a nonexistent command name.
then
  echo "true"
else
  echo "false"  <-- The else clause is executed.
fi
# At this point, the return code of the job is 0.
```

In example 1, if `cmdA` for command substitution specified in condition 1 of the `if` statement is a nonexistent command name, the condition is determined to be false, in which case `echo "false"` in the `else` clause is executed. At the point where the `if` statement is completed, the return code of the job is `0`.

Example 2:

```
#-adsh_step_start S1 -onError stop
if `cmdA true`  <-- cmdA is a nonexistent command name.
then
  echo "true"
else
  echo "false"
fi
#-adsh_step_end <-- The job step is cancelled without executing the then
or else clauses.
# The return code of job step S1 and the job at this point is 127.
```

In example 2, if `cmdA` for command substitution specified in condition 1 of the `if` statement is a nonexistent command name, the job step is cancelled without executing either the `then` or the `else` clause of the `if` statement. When the job step is completed, the return code of the job is `127`, which indicates that the specified command does not exist. For details about the definitions of job steps and the `onError` attribute, see *#-adsh_step_start command, #-adsh_step_error command, #-adsh_step_end command (defines a job step)* in *9.5 Extended script commands*.

5.2.2 Conditional expressions

Numeric value comparisons, character string comparisons, file attributes, logical operators, and ternary operators are used in conditional expressions. The following explains the specifications common to all conditional expressions.

- The `test` or `let` command is used to evaluate conditions.

  The `test` command includes the `[[]]` substitution format. The `let` command includes the `(())` substitution format.

- When you use the `test` command to evaluate conditions, insert a space between a variable and an operator. When you use `[[]]` instead of the `test` command, insert a space immediately after `[[` and immediately before `]]`.

  The following shows an example of a conditional using `[[]]`.

```
if [[ $arg1 -eq $args ]]; then
    echo TRUE
fi
```

- If an argument of the `test` command (such as `-eq`) is specified as an argument of the `let` command, the `let` command interprets that argument of the `test` command as a variable.

(1) Numeric value comparison

The following table lists and describes the operators used for comparing numeric values.

Table 5–21: Operators used for comparing numeric values

Conditional expression using an operator	Evaluation	Usage in test command or [[]]	Usage in let command or (())
numeric-value-1 `-eq` *numeric-value-2*	True if *numeric-value-1* is equal to *numeric-value-2*	Y	N
numeric-value-1 `-ne` *numeric-value-2*	True if *numeric-value-1* is not equal to *numeric-value-2*	Y	N
numeric-value-1 `-ge` *numeric-value-2*	True if *numeric-value-1* is equal to or greater than *numeric-value-2*	Y	N
numeric-value-1 `-gt` *numeric-value-2*	True if *numeric-value-1* is greater than *numeric-value-2*	Y	N
numeric-value-1 `-le` *numeric-value-2*	True if *numeric-value-1* is equal to or less than *numeric-value-2*	Y	N
numeric-value-1 `-lt` *numeric-value-2*	True if *numeric-value-1* is less than *numeric-value-2*	Y	N
numeric-value-1 == *numeric-value-2*	True if *numeric-value-1* is equal to *numeric-value-2*	Y#1	Y
numeric-value-1 `!=` *numeric-value-2*	True if *numeric-value-1* is not equal to *numeric-value-2*	Y#1	Y
numeric-value-1 >= *numeric-value-2*	True if *numeric-value-1* is equal to or greater than *numeric-value-2*	N	Y
numeric-value-1 > *numeric-value-2*	True if *numeric-value-1* is greater than *numeric-value-2*	Y#1, #2	Y
numeric-value-1 < *numeric-value-2*	True if *numeric-value-1* is less than *numeric-value-2*	Y#1, #2	Y
numeric-value-1 <= *numeric-value-2*	True if *numeric-value-1* is equal to or less than *numeric-value-2*	N	Y

Legend:

Y: Permitted

N: Not permitted

#1

Compared as character strings, not numeric values. For details about character string comparison, see *5.2.2(2) Character string comparison*.

#2

Can be used only in `[[]]`; cannot be used in any other format.

The following shows an example of numeric value comparison.

```
a=1
b=2
if [[ $a -lt $b ]]; then
  echo TRUE
else
  echo FALSE
fi

while (( $a != $b )); do
  echo LOOP
  ((a+=1))
done
```

(2) Character string comparison

The following table lists and describes the operators used for comparing character strings.

Table 5–22: Operators used for comparing character strings

Conditional expression using an operator	Evaluation	Usage in test command or [[]]	Usage in let command or (())
character-string	True if the length of *character-string* is one or more characters. This operator cannot be used in `[[ ]]` commands.	Y	N
`-n` *character-string*	True if the length of *character-string* is one or more characters. If the length of *character-string* is `0`, the command outputs the `KNAX6041-E` message and terminates with an error with return code `2`. If no *character-string* is specified in the `[[ ]]` command, a format error results, in which case the command outputs the `KNAX6041-E` message and terminates with an error with return code `1`.	Y	N
`-z` *character-string*	True if the length of *character-string* is zero	Y	N
`-o` *character-string*	True if *character-string* matches the character string of the shell option that is currently valid. For details about the character string of the shell option, see *Name* in *Table 5-34 Shell options that can be specified with the set command* in *5.6.1 Shell options that can be specified with the set command*.	Y	N
character-string `= pattern`	True if *character-string* matches `pattern`.	Y	N
character-string `== pattern`	True if *character-string* matches `pattern`.	Y	N
character-string `!= pattern`	True if *character-string* does not match `pattern`.	Y	N
character-string-1 `<` *character-string-2*	*character-string-1* and *character-string-2* are compared in the order of ASCII codes. If *character-string-1* is less than *character-string-2*, the result is true.	Y#	N
character-string-1 `>` *character-string-2*	*character-string-1* and *character-string-2* are compared in the order of ASCII codes. If *character-string-1* is greater than *character-string-2*, the result is true.	Y#	N

Legend:

Y: Permitted.

N: Not permitted.

#

Can be used only in `[[]]`; cannot be used in any other format.

Because any character string to be compared might contain one or more spaces, we recommend that you always enclose the entire character string in double quotation marks ("). The following shows examples.

```
str1="aaa"
str2="bbb"
test "$str1" == "$str2"
[[ "$str1" == "$str2" ]]
```

You can specify the `*`, `?`, and `[...]` wildcard characters in character strings to be compared. Note that wildcards can be specified only in `[[ ]]`, not in any other format. Note that when a character string containing a wildcard character is enclosed in double quotation marks ("), the use of the wildcard character as a wildcard character becomes invalid. The following shows an example.

```
str1="adsh"
str2="ads?"
str3="ad*"
[[ "$str1" == "$str2" ]]    ← The wildcard character is invalid. The
character string "ads?" is compared.
[[ $str1 != $str3 ]]        ← The wildcard character is valid.
```

The following shows an example of character string comparison using `[[]]`. The `*`, `?`, and `[...]` wildcard characters can be used.

```
if [[ abc == ab* ]]; then
    echo TRUE
fi
```

For details about wildcard characters, see *5.1.6(5) Wildcards*.

(3) File attributes

The following table lists and describes the operators used for evaluating file attributes, such as file formats and permissions.

Table 5–23: Operators used for evaluating file attributes, such as file formats and permissions

Conditional expression using an operator	Evaluation	Usage in test command or [[]]	Usage in let command or (())
`-a` *file*	True if *file* exists.	Y	N
`-b` *file*	True if *file* exists and it is a block type device.#1	Y	N
`-c` *file*	True if *file* exists and it is a character type device.#1	Y	N
`-d` *file*	True if *file* exists and it is a directory.	Y	N
`-e` *file*	True if *file* exists.	Y	N
`-f` *file*	True if *file* exists and it is a regular file.	Y	N
`-g` *file*	True if *file* exists and the `setgid` bit is set.#1	Y	N

Conditional expression using an operator	Evaluation	Usage in test command or [[]]	Usage in let command or (())
`-h` *file*	True if *file* exists and it is a symbolic link.#2	Y	N
`-k` *file*	True if *file* exists and a sticky bit is set.#1	Y	N
`-p` *file*	True if *file* exists and it is a pipe file.#1	Y	N
`-r` *file*	In Windows, the result is true if *file* exists; in UNIX, the result is true if *file* exists and it can be read from the current process.	Y	N
`-s` *file*	True if the following conditions are all satisfied: In Windows: • *file* exists. • *file* is not a folder. • The file size is at least 1 byte. In UNIX: • *file* exists. • The file size is at least 1 byte or *file* is a directory. With `-s`, because the conditions for `true` differ between UNIX and Windows, use `-d` to check if *file* is a directory or folder.	Y	N
`-t fd`	True if this is `fd` whose terminal is open.#3	Y	N
`-u` *file*	True if *file* exists and the `setuid` bit is set.#1	Y	N
`-w` *file*	In Windows, the result is true if the read-only attribute is not set or this is a directory; in UNIX, the result is true if *file* exists and it can be written from the current process.	Y	N
`-x` *file*	In Windows, the result is true if one of the following is true: • The extension is `.com`, `.exe`, `.cmd`, or `.bat`. • This is a directory. • The file satisfies the condition specified in the `CHILDJOB_EXT` or `CHILDJOB_SHEBANG` parameter (including a default definition) in the environment file.#4 In UNIX, the result is true if *file* exists and it can be executed from the current process.	Y	N
`-G` *file*	True if *file* exists and the group to which *file* belongs matches the ID of the group executing the calling process.#2	Y	N
`-L` *file*	True if *file* exists and it is a symbolic link.#2	Y	N
`-O` *file*	True if *file* exists and its owner has a valid user ID for the process.#2	Y	N
`-S` *file*	True if *file* exists and it is a socket.#1	Y	N
file`1 -ef` *file2*	True if *file1* and *file2* exist and the entities of *file1* and *file2* are the same (if either their symbolic link or hard link targets are the same).#2	Y	N
file1 `-nt` *file2*	True if *file1* and *file2* exist and the most recent modification date and time of *file1* is more recent than the most recent modification date and time of *file2*; also true if *file1* exists and *file2* does not exist.	Y	N
file1 `-ot` *file2*	True if *file1* and *file2* exist and the most recent modification date and time of *file1* is earlier than the most recent modification date and time of *file2*; also true if *file2* exists and *file1* does not exist.	Y	N
`-H` *file*	Always false	Y	N

Legend:

Y: Permitted

N: Not permitted

#1

In a Windows environment, the result is always false because this check occurs on nonexistent file types and flags.

#2

If the `test` command is executed in a Windows environment, it always results in an error because the processing is not supported during evaluation. However, you can set it to issue a message and result in an error or to treat it as normal by specifying the `UNSUPPORT_TEST` parameter. For details about the parameter, see *UNSUPPORT_TEST parameter (defines the handling of an unsupported conditional expression) (Windows only)* in *7.3 Environment setting parameters*.

#3

Do not specify a value greater than `9` in `fd`. If you do, the value cannot be guaranteed.

#4

For details about the `CHILDJOB_SHEBANG` parameter, see *CHILDJOB_SHEBANG parameter (defines an executable program path for job definition script files that are to be executed as child jobs)* in *7.3 Environment setting parameters*.

For details about the `CHILDJOB_EXT` parameter, see *CHILDJOB_EXT parameter (defines an extension for job definition script files that are to be executed as child jobs)* in *7.3 Environment setting parameters*.

The following shows an example of evaluating a file attribute.

```
FILE="$HOME/script/test.ash"
if [[ -a $FILE ]];
then
  echo "$FILE exists."
else
  echo "$FILE does not exist."
fi
```

(4) Logical operations

The following table lists and describes the operators used for evaluation in logical operations.

Table 5–24: Operators used for evaluation in logical operations

Conditional expression using an operator	Evaluation	Usage in test command or [[]]	Usage in let command or (())
expr1 `-a` *expr2*	True if the results of *expr1* and *expr2* are both true	Y#	N
expr1 `-o` *expr2*	True if the result of either *expr1* or *expr2* is true	Y#	N
expr1 `&&` *expr2*	True if the results of *expr1* and *expr2* are both true	Y	Y
expr1 `||` *expr2*	True if the result of either *expr1* or *expr2* is true	Y	Y
`!` *expr*	True if the result of *expr* is false	Y	Y

Legend:

Y: Permitted

N: Not permitted

#

Cannot be used in `[[]]`.

The following shows an example of a logical operation.

```
DIR="/tmp"
FILE="/tmp/test.ash"
```

```
a=2
b=4
if test -d $DIR -a -a $FILE
then
  echo "$DIR is directory and $FILE exists."
else
  echo "$DIR is not directory or $FILE does not exist."
fi

while ((a*0 || b-3)); do
  echo LOOP
  let b-=1
done
```

If you use `&&` and `||` in the `test` command, specify them as follows:

```
a=1
b=2
c=3
if test "$a" == 1 && test "$b" == 2; then
    echo "True"
else
    echo "False"
fi
if test "$a" != "$b" || test "$a" != "$c"; then
    echo "True"
else
    echo "False"
fi
```

(5) Ternary operator

You can use the ternary operator, which is an abbreviated notation of `if-else`. The following table explains the ternary operator supported by JP1/Advanced Shell.

Table 5–25: Ternary operator supported by JP1/Advanced Shell

Conditional expression using an operator	Evaluation	Usage in test command or [[]]	Usage in let command or (())
expr1?*expr2*: *expr3*	If the result of *expr1* is true, the result of *expr2* is returned. If the result of *expr1* is false, the result of *expr3* is returned.	N	Y

Legend:

Y: Permitted

N: Not permitted

The following shows an example of the ternary operator.

```
VAR1=3
VAR2=2
ANSWER=0

((ANSWER=VAR1>VAR2?8+VAR1:8*VAR2))
echo $ANSWER
```

5.3 Arithmetic operations

In job definition scripts, a variable value is treated as characters unless it is declared explicitly in the `-i` option of the `typeset` command as the integer type. However, if you specify an operator used for an arithmetic operation in the `let` command or in `(())`, the values assigned to variables are treated as numeric values during the arithmetic operation.

JP1/Advanced Shell supports arithmetic operators, increment and decrement operators, bitwise logical operators, and assignment operators. The following describes the specifications common to all operators:

- When you use the `let` command to perform arithmetic operations, do not place any spaces between a variable and an operator. If there is a space between a variable and an operator, the operation terminates with an error because of invalid format. If you need to place a space between a variable and an operator, either use `(())` (abbreviation of the `let` command) or enclose the entire arithmetic expression in quotation marks.

  Examples

  ```
  let   NUM = 100 - 99      # Terminates with an error because the
  arithmetic expression is not enclosed in quotation marks
  let " NUM = 100 - 99 "    # Arithmetic expression is executed because it
  is enclosed in quotations marks
  (( NUM    = 100 - 99 ))   # Arithmetic expression is executed because the
  abbreviation (( )) is used
  ```

- In arithmetic expressions, you can specify variables to which numeric values or numbers are assigned. A base number notation (*base-number#value*) can be used to specify numeric values.

  If the base number notation is omitted, the value is interpreted as a decimal number during operation.

- If a non-numeric character is assigned to a specified variable, the operation terminates with an error.

5.3.1 Arithmetic operators

Arithmetic operators are used in job definition scripts to perform arithmetic operations on the values of variables. The following table lists and describes the arithmetic operators supported by JP1/Advanced Shell.

Table 5–26: Arithmetic operators supported by JP1/Advanced Shell

Arithmetic operator	Description
-*num*	This is a unary minus operator. It changes *num* to a negative value.
*num1***num2*	Returns the results obtained by multiplying *num1* by *num2*.
num1/*num2*	Returns the results obtained by dividing *num1* by *num2*.
num1%*num2*	Returns the remainder obtained when *num1* is divided by *num2*.
num1+*num2*	Returns the results obtained by adding *num1* and *num2*.
num1-*num2*	Returns the results obtained by subtracting *num2* from *num1*.
*num1****num2*	Returns the results obtained by exponentiating the value *num1* to the power *num2*. If a value smaller than 0 is specified for *num2*, the command outputs the `KNAX6068-E` message and terminates with an error with return code `2`.

5.3.2 Increment and decrement operators

The increment and decrement operators are used to represent succinctly incremental and decremental processing on the same variable. The following table lists and describes the increment and decrement operators supported by JP1/Advanced Shell.

Table 5–27: Increment and decrement operators supported by JP1/Advanced Shell

Increment or decrement operator	Description
num++	References *num*, and then adds 1 to *num*.
num--	References *num*, and then subtracts 1 from *num*.
++*num*	Adds 1 to *num*, and then references the value of *num*.
--*num*	Subtracts 1 from *num*, and then references the value of *num*.

5.3.3 Bitwise logical operators

The bitwise logical operators are used to perform logical operations on variable values in bits. The following table lists and describes the bitwise logical operators supported by JP1/Advanced Shell.

Table 5–28: Bitwise logical operators supported by JP1/Advanced Shell.

Bitwise logical operator	Description
num1&*num2*	Returns the result of bitwise *AND* operation on *num1* and *num2*.
num1 | *num2*	Returns the result of bitwise *OR* operation on *num1* and *num2*.
num1^*num2*	Returns the result of bitwise *EXCLUSIVE-OR* operation on *num1* and *num2*.
num1<<*num2*	Returns the result obtained by shifting *num1* by *num2* bits to the left.
num1>>*num2*	Returns the result obtained by shifting *num1* by *num2* bits to the right.
~*num*	This is the result of bitwise negation of *num*. It returns a complement of 1.

5.3.4 Assignment operators

The assignment operators are used to assign values to variables. They can assign the results of arithmetic operations and bitwise logical operations on variables. The following table lists and describes the assignment operators supported by JP1/Advanced Shell.

Table 5–29: Assignment operators supported by JP1/Advanced Shell

Assignment operator	Description
num1=*num2*	Assigns *num2* to *num1*.
*num1**=*num2*	Assigns to *num1* the result obtained by multiplying *num1* by *num2*.
num1/=*num2*	Assigns to *num1* the result obtained by dividing *num1* by *num2*.
num1%=*num2*	Assigns to *num1* the remainder obtained by dividing *num1* by *num2*.

Assignment operator	Description
num1+=*num2*	Assigns to *num1* the result obtained by adding *num1* and *num2*.
num1-=*num2*	Assigns to *num1* the result obtained by subtracting *num2* from *num1*.
num1<<=*num2*	Assigns to *num1* the result obtained by shifting *num1* to the left by *num2* bits.
num1>>=*num2*	Assigns to *num1* the result obtained by shifting *num1* to the right by *num2* bits.
num1&=*num2*	Assigns to *num1* the result obtained by performing bitwise *AND* operation on *num1* and *num2*.
num1|=*num2*	Assigns to *num1* the result obtained by performing bitwise *OR* operation on *num1* and *num2*.
num1^=*num2*	Assigns to *num1* the result obtained by performing bitwise *EXCLUSIVE-OR* operation on *num1* and *num2*.

5.4 Priority of conditional and arithmetic operations

Priority applies to the following operators that can be used in the `let` command:

- Numeric value comparisons
- Logical operators
- Ternary operator
- Arithmetic operations

The table below shows the priority of conditional expressions and arithmetic operations in descending order of priority level, where 1 is the highest priority. Operations are performed in descending order of the priority, starting from the highest.

Table 5–30: Priority of operators

Priority	Operator
1	- (unary minus operator), !, ++, --, ~
2	**
3	*, /, %
4	+, -
5	<<, >>
6	<, <=, >, >=
7	==, !=
8	&
9	^
10	|
11	&&
12	||
13	?: (ternary operator)
14	=, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=

In the example shown below, `3**3` is calculated first because `**` has a higher priority than `*`. As a result, the value 54 is assigned to `a` and `54` is output to the standard output.

```
let a=2*3**3   <-- 3 to the power of 3 is multiplied by 2
echo $a        <-- 54 is output as the value of a to the standard output
```

5.5 Shell variables

Shell variables are variables whose values are inheritable once they are defined until a job definition script terminates. JP1/Advanced Shell enables you to specify and use shell variables. You can customize an environment for executing job definition scripts by referencing and changing the values of shell variables.

In Windows, the names of environment variables set when the job controller starts, such as environment variables specified as system properties, are converted to uppercase, and then the environment variables are imported as shell variables if the VAR_ENV_NAME_LOWERCASE parameter is specified as described in the following:

VAR_ENV_NAME_LOWERCASE parameter value	Conversion of environment variable names set when the job controller starts
ENABLE	Only environment variable names HOME, PATH, and SHELL are converted to uppercase; all other environment variable names are imported as is as shell variables.
DISABLE	All environment variable names are converted to uppercase and then imported as shell variables.

If the name of an environment variable set when the job controller starts is not consistent with the naming conventions for variables supported by JP1/Advanced Shell, that environment variable is imported but cannot be used. For details about the naming conventions for variables, see *5.1.2(1) Naming conventions for variables*.

In JP1/Advanced Shell, some shell variables cannot be used as variable names. If these shell variables are used, an error message is displayed. The following are the shell variables that are not supported:

COLUMNS, EDITOR, EXECSHELL, FCEDIT, HISTFILE, HISTSIZE, KSH_VERSION, MAIL, MAILCHECK, MAILPATH, POSIXLY_CORRECT, SH_VERSION, TMOUT, VISUAL

You must also not use the shell variables shown below, because they are used internally by JP1/Advanced Shell (however, no error message will be displayed if these shell variables are used):

PS1, PS2, PS3

5.5.1 Shell variables set by JP1/Advanced Shell

The table below lists and describes the shell variables set by JP1/Advanced Shell. Do not set values for, change attributes of, or release settings of these shell variables.

Table 5–31: Shell variables that are set by JP1/Advanced Shell

Shell variable name	Value that is set
#	Number of arguments passed to the current job definition script or function.
-	An abbreviated character string of a shell option set in the shell. A shell option that does not have an abbreviation is not set in this variable.
?	Return code of the immediately preceding command that was executed.
$	The process ID of the following program is set as the shell's process ID: (Windows only) Process ID of adshexecsub.exe or adshesub.exe

Shell variable name	Value that is set
$	(UNIX only) Process ID of adshexec
_	Value at the time the adshexec command starts. If no value exists, the contents of argv[0] at the time the adshexec command starts are set. When an external command or child job is started as a child process, the contents of argv[0] are set.
!	Process ID of the last command that executed in the background.
ADSH_DIR_BIN[#1]	Path name of the JP1/Advanced Shell program folder (bin).[#2]
ADSH_DIR_CMD[#1]	Path name of the folder for JP1/Advanced Shell's UNIX-compatible commands (cmd).[#3]
ADSH_RC_STEPLAST[#1]	Return code of the most recent job step to have executed. If no job steps have executed, the shell variable is undefined.
ADSH_RC_STEPMAX[#1]	Maximum value among the return codes of all job steps that have executed in the past. If no job steps have executed, the shell variable is undefined.
ADSH_RC_STEPMIN[#1]	Minimum value among the return codes of all job steps that have executed in the past. If no job steps have executed, the shell variable is undefined.
ADSH_STEPRC_*job-step-name*[#1]	Return code of the job step whose name is indicated. If the job step with the indicated job step name has not executed, the shell variable is undefined. If there are duplicate job step names, the return code of the last job step executed is stored.
LINENO	Line number in the job definition script of the line that is executing currently.
OLDPWD	Immediately preceding work directory that was set by the cd command.
OPTARG	Value of the last option argument that was processed by the getopts command.
OPTIND	Index of the last option argument that was processed by the getopts command.
PPID	In Windows, the value is always 0. In UNIX, the number of the shell's parent process is set.
PWD	Current work directory.
RANDOM	A random integer in the range from 0 through 32767 (= 0x7FFF).
REPLY	The data read by the read command with no arguments specified.
SECONDS	Number of seconds that have elapsed since the shell started.
Function information arrays[#1]	One-dimensional arrays of information about the function being executed by the adshexec command. The following arrays are supported: • Called function name array • Function call line number array • Function definition script file name array For details about the function information arrays, see *5.5.3 Arrays of function information*.

#1

These shells that have a special meaning are referred to collectively as the extended shell variables.

#2

The following is a definition example that uses the adshfile command in job definition scripts:

```
"${ADSH_DIR_BIN}adshfile" -s job -n keep -a del ${VAL01}
```

#3

The following is a definition example that uses the `expr` command in job definition scripts:

```
num=`"${ADSH_DIR_CMD}expr" $NUM - 1`
```

The following shows usage examples of these shell variables.

This example checks conditions by using the `if` script control statement and controls the execution of a job step based on the execution results of the preceding job step:

```
#!/opt/jp1as/bin/adshexec
#-adsh_job JOB001
#-adsh_step_start STEP01
    uap01
#-adsh_step_end
if [[ $ADSH_STEPRC_STEP01 -eq 0 ]]; then   <-- Execute STEP02 only if the
return code of STEP01 is 0.
    #-adsh_step_start STEP02
        uap02
    #-adsh_step_end
fi
```

This example terminates the job definition script with the maximum job step return code value when the `exit` command is used to terminate the job definition script:

```
#!/opt/jp1as/bin/adshexec
#-adsh_job JOB001

#-adsh_step_start STEP01
  uap01
#-adsh_step_end

#-adsh_step_start STEP02 -run always
  uap02
#-adsh_step_end

#-adsh_step_start STEP03 -run always
  exit $ADSH_RC_STEPMAX        <-- Use the maximum job step return code
value as the job's return code.
#-adsh_step_end
```

5.5.2 Shell variables whose values are set by the user

The following table lists and describes the shell variables whose values can be set by the user in JP1/Advanced Shell.

Table 5–32: Shell variables that can be used in JP1/Advanced Shell

Shell variable name	Value set by the user
`CDPATH`	Specifies a candidate path for a search when the target directory specified in the `cd` command does not exist under the work directory.
`ENV`	• Windows and Linux only If `YES` is specified in the `KSH_ENV_READ` parameter or the parameter is omitted, this variable specifies the name of the `.env` file to be loaded when the shell starts.

Shell variable name	Value set by the user
ENV	• AIX, HP-UX, and Solaris only If YES is specified in the KSH_ENV_READ parameter, this variable specifies the name of the .env file to be loaded when the shell starts.
FPATH	Specifies the directory that stores the function definition file. The specified directory is searched when the preload functionality is enabled for a referenced function or the function to be executed is undefined. The variable reads the contents of the file with the same name as the function name, defines the function in the current environment, and then executes it.
HOME	Specifies the home directory.
IFS	Abbreviation of Internal Field Separator. The specified characters are used as string separators. The first characters in IFS, $*, are used to separate arguments for substitution. The initial values are the space, tab, and end-of-line characters.
PATH	Specifies a command search path.
PS4	Specifies the prompt character string placed at the beginning of each line when the xtrace shell option is enabled. The initial value is +.
SHELL	Specifies the path name of a shell that is retained during shell execution.
TMPDIR	Changing this shell variable has no effect on temporary files because all temporary files are created in the directory specified in the TEMP_FILE_DIR environment setting parameter.

You can use the #-adsh_path_var command to define and use shell variables for converting directory paths between Windows and UNIX. For details about defining shell variables, see *5.8.5 Defining shell variables that handle path names*.

5.5.3 Function information arrays

Information about the functions that are executed by the adshexec command is stored in single-dimensional function information arrays.

Whether function information arrays are used is defined with the VAR_SHELL_FUNCINFO environment setting parameter. The array names are also determined by the specification of the VAR_SHELL_FUNCINFO environment setting parameter. For details about the VAR_SHELL_FUNCINFO environment setting parameter, see *VAR_SHELL_FUNCINFO parameter (selects whether function information arrays are used)*.

Following are the characteristics of function information arrays:

- Arrays of function information exist from the time a job definition script is executed to the time the job definition script is terminated. However, if functions are executed within a .ENV file, there are no function information arrays.
- The range of the number of elements is from 0 to 65,535.
- Attributes can be changed to the character string format attribute. Attributes cannot be changed to local variables in a function.
- Because function information arrays have the read-only attribute, the values in arrays can be referenced only; values cannot be set nor can the arrays be disabled.

(1) Types of function information arrays

The following table lists and describes the types of function information arrays.

Table 5–33: Types and names of function information arrays

Type of array	Description	Array name	
		When TYPE_A is specified in VAR_SHELL_FUNCINFO	When TYPE_B is specified in VAR_SHELL_FUNCINFO
Called function name array	This array stores the names of all functions that are in the call stack. Element number 0 stores the name of the currently executing function. The lowest element stores main. If an external script was called with the . (dot) command (standard shell command) or the #-adsh_script extended script command, the array stores source.	ADSH_FUNCNAME	FUNCNAME
Function call line number array#1	This array stores the line numbers of the script files for which all functions in the call stack were called. Element number 0 stores the line number that called the currently executing function. The lowest element stores 0. For an external script, the array stores the line number that executed the . (dot) command (standard shell command) and the #-adsh_script extended script command. The attribute can be changed to the integer type.	ADSH_LINENO	BASH_LINENO
Function definition script file name array#2	This array stores the names of the script files in which the functions in the call stack have been defined. Element number 0 stores the name of the script file that defines the currently executing function. The lowest element stores the absolute path of the job definition script name. For an external script, this array stores the absolute path of the external script file.	ADSH_SOURCE	BASH_SOURCE

#1

If a function is called within a trap action after a signal or a forced termination request has been received, the function call line number array stores the line number of the processing that called the trap action, not the processing within the trap action.

For example, in the definition below, function fn1 is called on line number 4. Because line number 4 is within the trap action, line number 6 is stored in the array.

```
1  fn1(){
2    echo ${ADSH_LINENO[*]}
3  }
4  trap fn1 INT
5
6  kill -INT $$
7  pwd
```

#2

If the adshexec command is executed with the -r option specified, -r CMDLINE is stored as the script file name in the function definition script file name array. The following shows an example:

```
C:\tmp>adshexec -m SIMPLE -r "echo ${ADSH_SOURCE[*]}"
-r CMDLINE
```

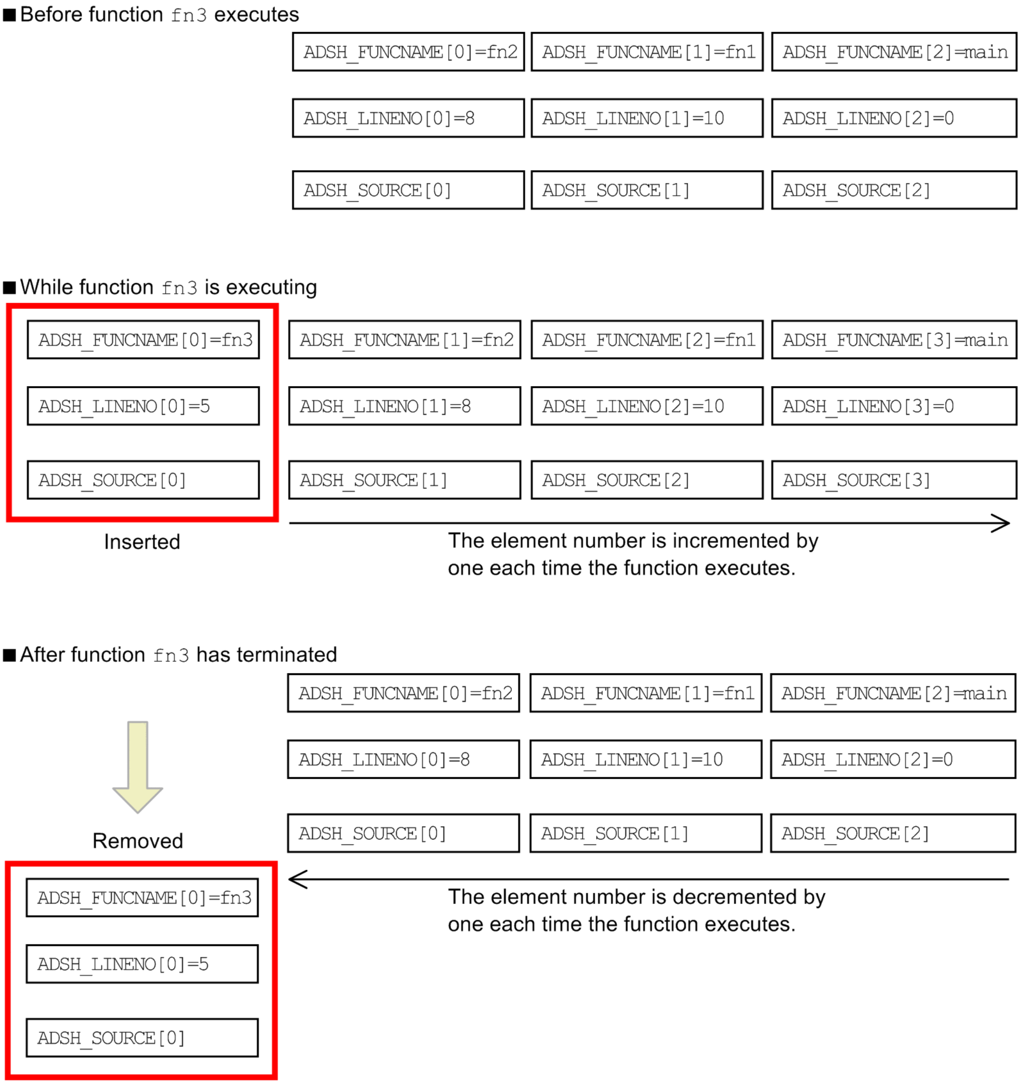
```
C:\tmp>
```

(2) Structures of function information arrays

This subsection explains the transitions among the arrays based on the example of the following job definition script (file name: `func.ash`):

```
fn3(){
  echo "JP1/AS"
}
fn2(){
  fn3
}
fn1(){
  fn2
}
fn1
```

When this job definition script is run, the array status changes as shown in the following by the execution of function `fn3`:



Note: The absolute path of func.ash is stored in ADSH_SOURCE[0] through ADSH_SOURCE[3].

(3) Notes about function information arrays

- Because function information arrays cannot be changed within job definition scripts, during debugging using CUI or GUI, the user can reference values (by using the watch, print, and info variables commands), but cannot update the values (with the set command).
- Arrays of function information are exported as follows:
  - The contents of element 0 are inherited as variables, not as an array. However, for child jobs, function information arrays are overwritten because they are reset when a job starts.
  - For the operation information for a child job, the exported values before being overwritten are output.
- The name of afunction information array cannot be specified for the stepVar attribute in the #-adsh_step_start command. If the name of a function information array is specified, the command issues the KNAX6312-E message and terminates with an error.
- (Windows only) The absolute path of the script file (including \ characters) is stored in the function definition script file name array. The echo and print commands handle this character (\) whenever it is encountered after shell variable expansion as an escape character. For this reason, if you output values from a function definition script file name array by using the echo and print commands, you must execute the commands as follows:

- If you use the echo command to output values, either specify the -E option or specify NO in the ESCAPE_SEQ_ECHO_DEFAULT environment setting parameter.
- If you use the print command to output values, specify the -r option.

5.6 Shell options

Shell options can be used to place limitations on the available functions and to switch execution modes. There are two ways to specify these shell options:

- Execute the `set` command in a job definition script.
- Specify the shell options as options of the `adshexec` command.

5.6.1 Shell options that can be specified with the set command

The following table lists and describes the shell options that can be specified with the `set` command. For details about the `set` command, see *set command (sets shell options, creates an array, or displays variable values)* in *9.3 Standard shell commands*.

Table 5–34: Shell options that can be specified with the set command

Name	Specification method	Meaning when the shell option is specified	Default value

`allexport`[#1]	`-a`
	`-o allexport`
		Automatically exports all variables.	Disabled

`braceexpand`	`-o braceexpand`	Enables brace expansion. Brace expansion means expanding to multiple words the portion enclosed in curly brackets (`{ }`). Each of the comma-separated words enclosed in curly brackets is expanded as a single variable by adding to it the characters that precede and follow the curly brackets. For example, `a{1,2,3}` is expanded to `a1`, `a2`, and `a3`.	Enabled
`bgnice`[#2]	`-o bgnice`	Lowers the priority for background jobs.	Disabled

`noglob`	`-f`
	`-o noglob`
		Prohibits file name substitution. For details about file name substitution, see *5.1.6(6) Substitution*.
		This shell option also prohibits brace expansion. To enable brace expansion, disable the `noglob` shell option. For details about how to disable the shell options, see *set command (sets shell options, creates an array, or displays variable values)* in *9.3 Standard shell commands*.
			Disabled

`nounset`	`-u`
	`-o nounset`
		If no value is set in a variable subject to substitution, the job terminates with an error and the shell terminates.	Disabled

`verbose`[#3]	`-v`
	`-o verbose`
		Reads the shell input lines and outputs them to the standard error output. The input lines are output before they are analyzed or executed.	Disabled

`xtrace`	`-x`
	`-o xtrace`
		Sets the value of shell variable `PS4` at the beginning of the line and then outputs the executed command and its arguments to the standard error output. Note that this shell option does not output the `[[]]` command, extended script commands, or their arguments. An arithmetic operation using `(())` is replaced with the `let` command and then is output.	Disabled

#1

If the `VAR_ENV_NAME_LOWERCASE DISABLE` environment setting parameter is specified in the Windows edition, any variable whose name contains lowercase letters cannot be exported. If shell variables containing lowercase letters are defined or their values are updated after the `allexport` shell option has been enabled, an error message is output and the batch job terminates.

#2

Not supported in a Windows execution environment.

#3

When you specify the `verbose` option in the `set` command, the output destination for the input lines of a job step definition command is changed.

- `#-adsh_step_start` command

  When the `#-adsh_step_start` command is executed, the input lines for `#-adsh_step_start` itself are output to `STDERR` for the job because `STDERR` is switched from job to job step.

- `#-adsh_step_end` command

  When the `#-adsh_step_end` command is executed, the input lines for `#-adsh_step_end` itself are output to `STDERR` for the job step because `STDERR` is switched from job step to job.

The following shows an example.

Job definition script

```
set -o verbose       <-- Or set -v.

#-adsh_step_start S1
  cmdA
#-adsh_step_error
  cmdB
#-adsh_step_end
  cmdC
```

`STDERR` for job

```
#-adsh_step_start S1
  cmdC
  :
```

`STDERR` for job step `S1`

```
  cmdA
#-adsh_step_error
  cmdB
#-adsh_step_end
```

Notes:

When run-time parameters and JP1/Advanced Shell Editor are used to debug job definition scripts, the command execution results are output to the standard error output. If the `set` command with the `verbose` option specified is used, a message containing the command execution results is output after the contents of the next line have been output. The following shows an example.

Job definition script

```
001:  set -o verbose
002:  echo "Line 002"
003:  echo "Line 003"
```

The following shows an output example during debugging:

```
KNAX7018-I Breakpoint "1": filename="test.ash" line=1
KNAX7032-I The script "test.ash" stopped running.
1: set -o verbose
Current: set
```

```
(adshdb) step <-- Execute the set command on line 001 of the job
definition script.
echo "Line 002" <-- Output the contents of line 002 of the job
definition script.
KNAX6112-I Execution of the command set (line=1) finished
successfully. exit status=0 execution time=0.000s CPU time=0.000s <--
Output the results of the set command on line 001 of the job
definition script.
KNAX7032-I The script "test.ash" stopped running.
2: echo "Line 002"
Current: echo
(adshdb) step
Line 002
        :
```

5.6.2 Shell options that can be specified with the adshexec command

The following table shows the shell option that is specified with the `adshexec` command. For details about the `adshexec` command, see *adshexec command (executes a batch job)* in *8.3 Shell operation commands*.

Table 5–35: Shell option that is specified with the adshexec command

Name	Specification method	Meaning when the shell option is specified
`noexec`	`-c`	Reads commands and checks for syntax errors, but does not execute the commands.
`xtrace`	`-x`	Same processing as when the `xtrace` shell option is enabled. For details, see *3.5 Outputting the executed commands and their arguments*.

5.7 Environment variables for job information

When you start a job or job step, you can specify the job name, job ID, and job step name in environment variables so that this information can be referenced by the job definition script files and user programs.

- `ADSH_JOB_NAME` (job name is specified when the job starts)
- `ADSH_JOBID` (job ID is specified when the job starts)
- `ADSH_STEP_NAME` (job step name is specified when the job step starts)

For details about these environment variables, see *2.5 Specifying environment variables*.

5.8 Defining jobs, job steps, and commands

You can use extended script commands to declare job names and define jobs, job steps, and commands. For details about the extended script commands, see *9.5 Extended script commands*.

5.8.1 Declaring job names

You use the `#-adsh_job` command to declare the job name of a job definition script.

The specification methods are shown below. Use either method 1 or method 2.

Specification method 1:

```
Line 1: #!any-character-string
Line 2: ∆0#-adsh_job job-name
```

Specification method 2:

```
Line 1: ∆0#-adsh_job job-name
```

If the `#-adsh_job` command is omitted, the default attribute value shown in the following table is used.

Attribute	Default value when omitted or undeclared	Example
Job name	`ADSH`*job-ID*	If the job ID is `000010`: `ADSH000010`

5.8.2 Defining the job end condition

You use the `#-adsh_job_stop` command to define the condition to be used to determine whether the job is to be cancelled when a job step terminates.

(1) Timing of evaluation

Each time a job step terminates, JP1/Advanced Shell checks if the return code for this attribute is defined. If such a return code is defined, JP1/Advanced Shell terminates the job without executing the subsequent job definition scripts.

(2) Scope

The end condition applies to execution of job definition scripts starting at the location immediately following where it is specified. If this command is specified in the preceding job definition script, that specification is reset and only the new condition specified takes effect.

The following shows an example.

```
01: #!/opt/jp1as/bin/adshexec
02: #-adsh_job JOB0001
03:
04: #-adsh_step_start STEP1
05: #-adsh_step_end
06:
07: #-adsh_job_stop 4:       ← The scope of this definition is from line 09
```

```
through 13.
08:
09: #-adsh_step_start STEP2
10: #-adsh_step_end
11:
12: #-adsh_step_start STEP3
13: #-adsh_step_end
14:
15: #-adsh_job_stop 8:16,24:32 ← The scope of this definition is from line
17 through 18.
16:
17: #-adsh_step_start STEP4
18: #-adsh_step_end
```

(3) Example of job end condition definition

If the `#-adsh_job_stop` command is used to define a job end condition, the following occurs:

- Even if a job step terminates with the return code specified in the `#-adsh_job_stop` command, commands outside the job step do not cancel the job.
- If a job step terminates with the return code specified in the `#-adsh_job_stop` command, the job step cancels the job.
- If a job is cancelled by executing the `#-adsh_job_stop` command, any subsequent commands outside the job step are not executed. The subsequent job steps are not executed either, regardless of the specification of the `run` attribute.

The following shows an execution example.

```
#-adsh_job JOB_STOP
#-adsh_rc_ignore CBLRTN
#-adsh_job_stop 4     ← Specifies that the job is to be cancelled at rc=4.

echo "Job start."
CBLRTN  004  # Command that succeeds at rc=4 ← The job is not canceled.
                         although a command outside the job step results in
rc=4.

#-adsh_step_start STEP01
echo "Step start."
CBLRTN  004  # Command that succeeds at rc=4
#-adsh_step_end       ← The job step terminates with rc=4 and the job is
cancelled.

#-adsh_step_start STEP03 -run always ← Does not execute any subsequent job
steps regardless of the run attribute.
  echo "command in step"
#-adsh_step_end

echo "Job end."       ← Does not execute any subsequent commands.
```

5.8.3 Defining job steps

The job step definition commands whose names begin with #-adsh_step are used to group a portion of a job definition script as a job step. A job step consists of a group of commands.

(1) How to group commands

A group of commands normally executed as a job step is specified in a block from the #-adsh_step_start command to the #-adsh_step_error or #-adsh_step_end command. This block is called a job step normal block.

A group of commands that is executed only if the last command in the job step normal block terminates with an error is specified in a block from the #-adsh_step_error command to the #-adsh_step_end command. This block is called a job step error block.

(2) Flow of job step execution

The following describes the flow of job step execution.

1. Whether the job step is to be skipped is determined based on the preceding job step having terminated with an error, based on whether any commands terminated with an error, or based on the run attribute's specification. For details about the run attribute, see *#-adsh_step_start command, #-adsh_step_error command, #-adsh_step_end command (defines a job step)* in *9.5 Extended script commands*.
2. The commands in the job step normal block are executed sequentially. If any of the commands terminates with an error and the onError attribute is stop, JP1/Advanced Shell exits the job step normal block without executing the subsequent commands. If the onError attribute is cont, JP1/Advanced Shell executes the subsequent commands and then exits the job step normal block.
3. If #-adsh_step_error is defined and the last command in the job step normal block terminates with an error, the commands in the job step error block are executed sequentially by JP1/Advanced Shell.

(3) Declaring shell variables that are valid only within a job step

You can declare shell variables that are to be valid only within the current job step by specifying the stepVar attribute. When the declared shell variables are exported, they are placed in exported status only within the job step.

When the job step begins, JP1/Advanced Shell automatically places the shell variables in undefined status. However, if the PATH shell variable is defined to be valid only within the job step, the values before the job step started are inherited.

When the job step terminates, JP1/Advanced Shell automatically resets the shell variables to their status when the job step started.

You can declare shell variables with the same names as shell variables outside the job step. The following notes apply when you declare such shell variables:

- A declared shell variable is treated as different from other shell variable with the same name outside the job step.
- When the job step starts, a declared shell variable is placed in undefined status. The value of a shell variable with the same name outside the job step is not inherited because it is treated as a separate shell variable.
- A shell variable with the same name outside the job step cannot be referenced or updated within the current job step.
- After the job step has terminated, a shell variable with the same name outside the job step can be referenced and updated again.

The following shows a usage example.

Usage example of a job definition script file:

```
VAL1=AAA
echo "Before starting the step (outside the step)"
echo "beforeStepVar1="$VAL1
echo "beforeStepVar2="$VAL2

#-adsh_step_start S1 -stepVar VAL1,VAL2
  echo "The step has started"
  echo "startStepVar1="$VAL1
  echo "startStepVar2="$VAL2
  VAL1=XXX
  VAL2=YYY
  echo "endStepVar1="$VAL1
  echo "endStepVar2="$VAL2
#-adsh_step_end

echo "The step was terminated"
echo "afterStepVar1="$VAL1
echo "afterStepVar2="$VAL2
```

This example declares `VAL1` that has a shell variable with the same name outside the job step and `VAL2` that does not have a shell variable with the same name outside the job step.

The following shows the execution results:

```
Before starting the step (outside the step)
beforeStepVar1=AAA <-- References VAL1 outside the job step. This is a
different variable from VAL1 inside the job step.
beforeStepVar2= <-- References VAL2 outside the job step, but a nonexistent
step was started.
startStepVar1= <-- References VAL1 inside the job step. This is a different
variable from VAL1 outside the job step.
startStepVar2=
endStepVar1=XXX
endStepVar2=YYY
The step was terminated
afterStepVar1=AAA <-- References VAL1 outside the job step. This is a
different variable from VAL1 inside the job step.
afterStepVar2= <-- References VAL2 outside the job step, but it does not
exist.
```

If you specify `PATH` in a shell variable that is valid only within the job step, you can add a path to the `PATH` shell variable that is valid only in the current job step. For the initial value of `PATH`, the value in effect before the job step starts is inherited.

The example shown below adds paths to the `PATH` shell variable that is valid only in the job step. This example assumes that the value of the `PATH` shell variable is `a:b` when execution of the job definition script begins.

```
#-adsh_job J1                          --> 1.
cmdA
PATH=x:$PATH                           --> 2.
cmdB
#-adsh_step_start S1 -stepVar PATH     --> 3.
```

```
  cmdC
  PATH=y:$PATH                            --> 4.
  cmdD
#-adsh_step_end                           --> 5.
```

The numbers in the example of adding paths to a PATH shell variable that is valid only in the current job step correspond to the numbers in the following explanation:

1. The initial value of PATH is a:b.
2. Valid inside the job step. The value of PATH becomes x:a:b.
3. Specifies PATH for stepVar. The shell variable remains undeleted and the value remains as x:a:b.
4. Valid inside the job step. The value of PATH becomes y:x:a:b.
5. Resets PATH to the value in effect when the job step started. The value of PATH becomes x:a:b.

When you use the #-adsh_path_var command, you can define and use the shell variables for converting directory paths between Windows and UNIX. For details about this function, see *5.8.5 Defining shell variables that handle path names*.

(4) Specifying the job step return code in the event of a job step error

You can set a desired return code for a job step in the event of a job step error. To do this, execute the exit command with the desired return code specified in its argument inside the job step error block. In this case, the value specified in the argument of the exit command is also set as the job's return code because the job is terminated by the exit command.

The value specified in the argument of the exit command also becomes the return code of job step when a . (dot) command or #-adsh_script command is used within the job step error block to call an external script and then the exit command with the argument specified is executed inside the called external script.

The following example executes the exit command with the argument specified inside the job step error block:

```
#-adsh_step_start STEP1
  cmdA
  cmdB    <-- Terminated in an error with return code 1.
  cmdC
#-adsh_step_error
  exit 4   <-- The job step terminates with an error and the exit command's
argument 4 becomes the return code of the job step.
#-adsh_step_end
```

If the exit command with no argument specified is executed inside the job step error block, the return code of the last command executed within the job step normal block becomes the job step's return code.

The following example executes the exit command with no argument specified inside the job step error block:

```
#-adsh_step_start STEP1
  cmdA
  cmdB               <-- Error termination with return code 1.
  cmdC
#-adsh_step_error
  exit               <-- The job step terminates with an error. Because the
exit command has no arguments, the return code of cmdB becomes the return
```

```
code of the job step.
#-adsh_step_end
```

(5) Job step execution examples

The following shows example job definition script file executions in which all commands terminate normally and in which an intermediate command terminates with an error.

- Example execution in which all commands terminate normally

```
#!/opt/jp1as/bin/adshexec
#-adsh_job JOB001
#-adsh_step_start STEP001
  command1                   <-- Executed.
  command2                   <-- Executed.
  command3                   <-- Executed.
#-adsh_step_error
  command4                   <--  Not executed.
  command5                   <-- Not executed.
#-adsh_step_end
#-adsh_step_start STEP002
  command6                   <-- Executed.
#-adsh_step_end
#-adsh_step_start STEP003 -run abnormal
  command7                   <-- Not executed.
#-adsh_step_end
#-adsh_step_start STEP004 -run always
  command8                   <-- Executed.
#-adsh_step_end
```

- Example execution in which `command2` terminates with an error (`onError` attribute is `stop`)

```
#!/opt/jp1as/bin/adshexec
#-adsh_job JOB001
#-adsh_step_start STEP001 -onError stop
  command1                   <-- Executed.
  command2                   <-- Executed (error termination).
  command3                   <-- Not executed.
#-adsh_step_error
  command4                   <-- Executed.
  command5                   <-- Executed.
#-adsh_step_end
#-adsh_step_start STEP002
  command6                   <-- Not executed.
#-adsh_step_end
#-adsh_step_start STEP003 -run abnormal
  command7                   <-- Executed.
#-adsh_step_end
#-adsh_step_start STEP004 -run always
  command8                   <-- Executed.
#-adsh_step_end
```

- Example execution in which `command2` terminates with an error (`onError` attribute is `cont`)

```
#!/opt/jp1as/bin/adshexec
#-adsh_job JOB001
#-adsh_step_start STEP001 -onError cont
  command1                   <-- Executed.
  command2                  <-- Executed (error termination).
  command3                  <-- Executed.
#-adsh_step_error
  command4                  <--  Not executed.
  command5                  <--  Not executed.
#-adsh_step_end
#-adsh_step_start STEP002
  command6                  <-- Executed.
#-adsh_step_end
#-adsh_step_start STEP003 -run abnormal
  command7                  <-- Not executed.
#-adsh_step_end
#-adsh_step_start STEP004 -run always
  command8                  <-- Executed.
#-adsh_step_end
```

If a command outside the job step results in an error, the subsequent job definition script is handled as follows:

- The subsequent commands outside the job step are executed.
- The subsequent commands whose `run` attribute is `normal` are not executed.
- The subsequent job steps whose `run` attribute is `abnormal` or `always` are executed.

The following shows an execution example:

```
#-adsh_job CMD_ERROR

echo "Job start."
cd -x  #Command that results in an error  <--This command results in an
error.
echo "Job end."                       <--Commands outside the job step
execute.

#-adsh_step_start STEP01 -run normal  <--Does not execute a job step for
which run normal is specified.
  echo "command in step"
#-adsh_step_end

#-adsh_step_start STEP02 -run abnormal  <--Executes a job step for which
run abnormal is specified.
  echo "command in step"
#-adsh_step_end

#-adsh_step_start STEP03 -run always  <--Executes a job step for which run
always is specified.
  echo "command in step"
#-adsh_step_end
```

If a job step results in an error, whether the subsequent job step is to be executed is determined by the `run` attribute of the subsequent job step. Commands outside the subsequent job step will not be executed. The following shows an execution example:

```
#-adsh_job STEP_ERROR

#-adsh_step_start STEP01
  echo "Step start."
  cd -x  #Command that results in an error
  echo "Step end."
#-adsh_step_end      <--The job step terminates with an error.

echo "Job end."      <--If the preceding job step resulted in an error,
commands outside the subsequent job step do not execute.
```

The return code of a command executed inside the job step error block has no effect on the return code of the job step. The return code of the last command executed inside the job step normal block becomes the return code of the job step. The following shows an execution example:

```
#-adsh_job STEP_ERRBLK_RC

#-adsh_step_start STEP01
  echo "Step start."
  cd -x  #Command that results in an error with rc=1     <--The result of
this command becomes the return code of the job step.
  echo "Step end."
#-adsh_step_error
  echo "step error block"  #Command that results in rc=0  <--No effect on
the job step's rc.
#-adsh_step_end    <--The error result of the cd command is applied and the
job step terminates with rc=1.
```

(6) The relationship of shell functions and the trap command

Regardless of where a shell function or action of a trap command was defined, whether the shell function or action is executed inside or outside of a job step is determined by where the shell function or action was executed.

5.8.4 Defining commands that terminate normally

(1) Defining commands that terminate normally, even when the return code is not 0

You can specify a return code threshold for normal termination so that nonzero values as a command's return code will be treated as a normal termination. Any return code that does not exceed the specified threshold will cause the command to be treated as having terminated normally.

You can use the following environment setting parameters to specify a return code threshold.

- `CMDRC_THRESHOLD_USE_PRESET` parameter

  This parameter specifies a return code threshold for all UNIX-compatible commands. The permitted threshold specifications are `0` and `1`.

  - `cmp` command
  - `diff` command
  - `egrep` command
  - `expr` command
  - `grep` command

    - `sort` command

  For details, see *CMDRC_THRESHOLD_USE_PRESET parameter (defines a threshold for the return code of a UNIX-compatible command)*.

- `CMDRC_THRESHOLD_DEFINE` parameter

  This parameter specifies target commands and a return code threshold. You can specify the following commands:

    - External commands
    - UNIX-compatible commands
    - Shell scripts
    - Shell operation command
    - child jobs

  For details, see *CMDRC_THRESHOLD_DEFINE parameter (defines a return code threshold for a command)*.

(2) Defining commands that always terminate normally

When the `#-adsh_rc_ignore` command is used, a command whose name is defined always terminates normally regardless of its return code. In such a case, the return code of the target command has no effect on the evaluation of the result of the job step (success or fail).

However, if the command is terminated by signal, the command always terminates with an error regardless of this specification.

The commands shown below will not result in an error even if their return code is not zero, which means that the return code is ignored regardless of the specification of this command:

- `true` and `false` commands

The `#-adsh_rc_ignore` command definition takes effect on the execution of a job definition script starting at the location immediately following where this definition is specified. If this definition is specified outside a job step, it takes effect on the entire job definition script. If the definition is specified inside a job step, it takes effect only inside that job step. When the definition is specified inside a job step, it takes effect from the location immediately following the location of its specification through the end of the job step, and any value specified outside the job step becomes invalid temporarily. Until a value is specified inside the job step, a value specified outside the job step becomes valid.

The following shows an example specification:

```
01: #!/opt/jp1as/bin/adshexec
02: #-adsh_job JOB0001
03:
04: uap01
05: uap02
06: #-adsh_rc_ignore uap03,uap04    <--1. Specified outside the job step.
07: uap03                           <--Scope of 1 is from line 07 through
14.
08:
09: #-adsh_step_start STEP1
10:   uap04
11: #-adsh_step_end
12:
13: #-adsh_step_start STEP2
14:   uap05
15:   #-adsh_rc_ignore uap06,uap07  <--2. Specified inside the job step.
```

```
16:   uap06                     <--Scope of 2 is from line 16 through
17.
17:   uap07
18: #-adsh_step_end
19:
20: #-adsh_step_start STEP4         <--Scope of 1 is from line 20 through
22.
21:   uap08
22: #-adsh_step_end
```

5.8.5 Defining shell variables that handle path names

The `#-adsh_path_var` command enables you to define shell variables that handle path names. When you use shell variables that handle path names, you can convert the path and directory separators in character strings containing path names according to the environment being used, such as between Windows and UNIX.

JP1/Advanced Shell converts the path and directory separators in character strings that satisfy all the conditions listed below. For details about path conversion settings 1 and 2, see *PATH_CONV_RULE parameter (defines a rule for converting file paths) (Windows only)*.

- Character string enclosed in double quotation marks (`"`) (applicable to path conversion setting 1)
- Character string not enclosed in single quotation marks (`'`) (applicable to path conversion setting 2)
- Character string separated by the path separator defined in the `PATH_CONV_ENABLE` parameter in the environment file, whose leading part matches the character string `$`*name-of-shell-variable-that-handles-paths* or `${`*name-of-shell-variable-that-handles-paths*`}`

You can use the `#-adsh_path_var` command only in either of the following cases:

- Line following `#!`*any-character-string* on the first line
- Line following a line containing the `#-adsh_job` command

You can specify a continuation line as shown below. Note that no shell variable name can be specified after the `#-adsh_path_var` command on the first line.

```
Line 1: #-adsh_path_var
Line 2: #-adsh  var001,var002,var003,var004
```

(1) Examples

This example defines shell variables `PATH`, `DIR`, and `DIR3` that handle path names in the `#-adsh_path_var` command.

(a) In Windows

- Specification of the environment file

```
#-adsh_conf PATH_CONV_ENABLE / :    <--Enable path conversion.
#-adsh_conf PATH_CONV_RULE 1                      <--Select path
conversion setting 1.
#-adsh_conf PATH_CONV /home/hitachi "C:\\hitachi"  <--Path string
substitution 1.
```

```
#-adsh_conf PATH_CONV /tmp/jp1as "D:\\jp1as_tmp"   <--Path string
substitution 2.
#-adsh_conf PATH_CONV /tmp "C:\\temp"              <--Path string
substitution 3.
#-adsh_conf PATH_CONV_ACCESS /dev/null nul    <--Convert file paths
during file input and output.
```

- Specification of the job definition script

```
#-adsh_path_var PATH,DIR,DIR3    <--Shell variable definition 1.

DIR="/home/hitachi/bin"    <--Convert to DIR="C:\\hitachi\\bin" by path
string substitution 1.

"$DIR/myprog"    <--Convert to "$DIR\\myprog" by shell variable
definition 1.

"${DIR}/myprog"    <--Convert to "${DIR}\\myprog" by shell variable
definition 1.

DIR2=$DIR
"$DIR2/myprog"    <--$DIR2 is not converted because it is not in shell
variable definition 1.

$DIR/myprog    <--Not converted because this is not enclosed in double
quotation marks (").

FILE1="/tmp/jp1as/file"    <--Convert to "D:\\jp1as_tmp\\file" by path
string substitution 2.

DIR3=""
ls "$DIR3../bin"    <--Convert to "..\\bin" by shell variable definition
1. Relative paths are also converted.

DIR4="/home/hitachi/sbin:$DIR2"    <--Convert to C:\\hitachi\\sbin;$DIR2
by path string substitution 1. Path separators are also converted.

PATH="../bin/:$DIR"    <--Convert to ..\\bin\\;$DIR by shell variable
definition 1. Path separators are also converted.

"$DIR2/myprog" > /dev/null    <--Convert to nul by file path conversion
during file input and output.
```

(b) In UNIX

- Specification of the environment file

```
#-adsh_conf PATH_CONV_ENABLE \\ ;    <--Enable path conversion.
#-adsh_conf PATH_CONV "C:\\hitachi" /home/hitachi  <--Path string
substitution 1.
#-adsh_conf PATH_CONV "D:\\jp1as_tmp" /tmp/jp1as   <--Path string
substitution 2.
#-adsh_conf PATH_CONV "C:\\temp" /tmp     <--Path string substitution 3.
#-adsh_conf PATH_CONV_ACCESS nul /dev/null    <--Convert file paths
during file input and output.
```

- Specification of the job definition script

```
#-adsh_path_var PATH,DIR,DIR3    <--Shell variable definition 1.

DIR="C:\\hitachi\\bin"    <--Convert to DIR="/home/hitachi/bin" by path
string substitution 1.
"$DIR\\myprog"    <--Convert to "$DIR/myprog" by shell variable
definition 1.

"${DIR}\\myprog"    <--Convert to "${DIR}/myprog" by shell variable
definition 1.

DIR2=$DIR
"$DIR2\\myprog"    <--$DIR2 is not converted because it is not in shell
variable definition 1.

$DIR\\myprog    <--Not converted because this is not enclosed in double
quotation marks (").

FILE1="D:\\jp1as_tmp\\file"    <--Convert to "/tmp/jp1as/file" by path
string substitution 2.

DIR3=""
ls "$DIR3..\\bin"    <--Convert to "../bin" by shell variable definition
1. Relative paths are also converted.

DIR4="C:\\hitachi\\sbin;$DIR2"    <--Convert to /home/hitachi/sbin:$DIR2
by path string substitution 1. Path separators are also converted.

PATH="..\\bin\\;$DIR"    <--Convert to ../bin/:$DIR by shell variable
definition 1. Path separators are also converted.

"$DIR2\\myprog" > nul    <--Convert to /dev/null by file path conversion
during file input and output.
```

(2) Example output to job definition script images

JP1/Advanced Shell outputs a job definition script before path conversion and the lines after conversion to job execution logs. JP1/Advanced Shell also outputs as messages the conversion rules that were satisfied in the job definition script, job definition script name, and line numbers.

```
********  JOB CONTROLLER MESSAGE  ********
(omitted)
10:48:48 000007 KNAX6803-I The access path matched the conversion rule.
filename="D:\home\user001\path_conv.ash" line=4 path converted="./
local.log":".\\mylog"
(omitted)

********   Script IMAGE    ********

***** D:\home\user001\path_conv.ash *****
0001 : #-adsh_job JOB001
0002 : #-adsh_path_var DIR
0003 : DIR="/home/hitachi/bin"; DIR2="/tmp/tmpfile"
0004 : "$DIR/myprog" > ./local.log
0005 : exit

***** Converted lines in "D:\home\user001\path_conv.ash" *****
```

```
0003 : DIR="c:\\Program Files"; DIR2="c:\\temp\\tmpfile"
0004 : "$DIR\\myprog" > ./local.log

***** CONVERSION INFORMATION *****
KNAX6800-I The path matched the conversion rule. filename="D:\home
\user001\path_conv.ash" line=3 path converted="/home/hitachi/bin":"c:\
\Program Files"
KNAX6800-I The path matched the conversion rule. filename="D:\home
\user001\path_conv.ash" line=3 path converted="/tmp":"c:\\temp"
KNAX6801-I The path matched the conversion rule. filename="D:\home
\user001\path_conv.ash" line=4 shell variable handling path="DIR"
```

5.8.6 Calling an external job definition script file from an executing job definition script

You use the `#-adsh_script` command to insert into the currently executing job definition script file the contents of an external job definition script file that is in existence at the start of JP1/Advanced Shell.

Unlike with the `.` (dot) command, this standard shell command expands into the calling job definition script the contents of a specified external script that exists at the time JP1/Advanced Shell starts. JP1/Advanced Shell treats the calling job definition script containing the expanded job definition script as a single job definition script and performs syntax analyses on it.

The following shows an example of an external script definition and a calling job definition script.

- `/scripts/exScript.ash` (contents when the job starts)

```
#-adsh_step_start exS1
  exUap01
#-adsh_step_end

#-adsh_step_start exS2
  exUap01
#-adsh_step_end
```

- `script.ash`

```
#!/opt/jp1as/bin/adshexec
#-adsh_job JOB001

uap01

#-adsh_script /scripts/exScript.ash

uap2
```

The contents of `script.ash` are equivalent to the following job definition script:

```
#!/opt/jp1as/bin/adshexec
#-adsh_job JOB001

uap01

#-adsh_step_start exS1    <--The six lines starting with this line
```

```
constitute the expanded exScript.ash.
  exUap01
#-adsh_step_end

#-adsh_step_start exS2
  exUap01
#-adsh_step_end

uap2
```

The following table describes the differences between the `.` (dot) command and the `#-adsh_script` command.

Table 5–36: Differences between the . (dot) command and the #-adsh_script command

No.	Comparison item		. (dot) command	#-adsh_script command
1	Handling of extended script commands inside the external job definition script		Handled as comments.	Handled as extended script commands.
2	Output to script images		Not output.	Output
3	Operation when relative paths are specified		Allows specification of relative paths. However, this command resolves the paths by referencing the value of the `PATH` environment variable.	Allows specification of relative paths. However, this command interprets the specified paths as being relative to the current directory used when `adshexec` started; it does not reference the value of the `PATH` environment variable.
4	Maximum number of commands permitted in a job		No limit	Maximum of 4,095
5	Command execution from within the external job definition script		Executable	Executable. However, the command cannot be executed by calling the same external job definition script recursively.
6	Specification of arguments for the external job definition script		Permitted	Not permitted
7	CUI debugger	Setting breakpoints in the external job definition script by using the `break` command	Cannot set breakpoints.	Can set breakpoints.
8		Displaying information about the functions defined in the external job definition script by using the `info function` command	Cannot display information. However, the information can be displayed once the function definition is completed within the external job definition script.	Can display information.
9		Displaying the contents of the external job definition script by using the `list` command	Cannot display the contents.	Can display the contents.
10		Information displayed when execution of the job definition script is stopped	Line numbers: Can be displayed. Lines in the source file: Cannot be displayed. Command strings: Can be displayed.	Line numbers: Can be displayed. Lines in the source file: Can be displayed. Command strings: Can be displayed.

(1) Specifying relative paths

If the relative path of an external script is specified, JP1/Advanced Shell assumes the current directory used when adshexec started, regardless of the processing of the previous job definition script.

A path relative to any other directory cannot be specified. To use such a path, you must specify its absolute path.

The following show a specification example:

Current directory when the adshexec command starts: /scripts

```
#/opt/jp1as/bin/adshexec
cd /work
#-adsh_script ex_script.ash      <-- /scripts/ex_script.ash is executed.
```

This example uses the #-adsh_script command to execute the external script file /scripts/ex_script.ash. The current directory was changed by the cd command on the immediately preceding line, but this change has no effect on the path of the external script file that is called.

(2) Specifying absolute paths

If you want to execute /work/ex_script.ash, specify an absolute path as shown in the following.

Current directory when the adshexec command starts: /scripts

```
#/opt/jp1as/bin/adshexec
cd /work
#-adsh_script /work/ex_script.ash      <-- /work/ex_script.ash is
executed.
```

5.8.7 Return codes of extended script commands and handling of errors

The table below lists and describes the return codes for the extended script commands. You can define these return codes by using the environment setting parameters.

Table 5–37: Return codes of the extended script commands

Extended script commands	Execution result	Default value for return code	Environment setting parameter for specifying return codes
#-adsh_file #-adsh_file_temp #-adsh_job #-adsh_job_stop #-adsh_path_var #-adsh_rc_ignore #-adsh_spoolfile #-adsh_step_start #-adsh_step_error	Normal termination	0	ADSHCMD_RC_SUCCESS
	Error termination	1	ADSHCMD_RC_ERROR
#-adsh_step_end	Job step normal termination	Return code of the last command executed in the job step normal block	--
	Job step error termination		

Extended script commands	Execution result	Default value for return code	Environment setting parameter for specifying return codes
#-adsh_step_end	The job step was terminated by executing the exit command with arguments specified inside the job step error block.	Argument of the exit command	--
	Error termination of #-adsh_step_end	1	ADSHCMD_RC_ERROR
#-adsh_script	Normal termination	Return code of the last command executed in the called external script	--
	Error termination	1	ADSHCMD_RC_ERROR

Legend:

--: Not applicable

If execution of an extended script command results in error termination or job step error termination, the following occurs:

- If abnormal or always is specified in the run attribute, the job step is executed.
- If the run attribute is omitted or normal is specified in the run attribute, the job step is not executed.
- None of the commands outside the job step is executed.

The following shows an execution example:

```
#-adsh_job EXCMD_ERROR

echo "Job start."

#-adsh_file ERRFILE file01 -chk exist  <-- This extended script command
results in an error.

#-adsh_step_start STEP01 -run normal    <-- Do not execute a job step for
which normal is specified in the run attribute.
  echo "command in step"
#-adsh_step_end

#-adsh_step_start STEP02 -run abnormal  <-- Execute a job step for which
abnormal is specified in the run attribute.
  echo "command in step"
#-adsh_step_end

#-adsh_step_start STEP03 -run always    <-- Execute a job step for which
always is specified in the run attribute.
  echo "command in step"
#-adsh_step_end

echo "Job end."                         <-- Do not execute any command
outside the job step.
```

5.8.8 Return codes of jobs, job steps, and commands

This subsection explains the return codes and the normal and abnormal execution results.

(1) Return codes of jobs

The return code of a job is the return code of the last job definition script that executed.

JP1/Advanced Shell does not identify whether job execution results are normal or abnormal. It returns the return codes to other programs such as JP1/AJS as is.

If a job is executed from JP1/AJS, the JP1/AJS job's return code that can be checked by using a program such as JP1/AJS - View might be the JP1/AJS-defined return code, not the `adshexec` command's return code. In such a case, the JP1/AJS job's return code might differ from the JP1/Advanced Shell job's return code that is output to job execution logs.

Example:

If the `adshexec` command of a UNIX edition started from JP1/AJS receives `SIGINT`, JP1/Advanced Shell job's return code is `130`, but JP1/AJS job's return code is `-1`.

If an error occurs in a job, an error message is output.

(2) Return codes of job steps

The return code of a job step is the return code of the last command executed in the job step normal block. If you execute the `exit` command with the argument specified in the job step error block, you can set the argument of the `exit` command as the job step's return code. The return codes of other commands executed in the job step error block have no effect on the job step's return code. The following explains normal termination and error termination of job steps:

- Job step normal termination
  The last command that executed in the job step normal block terminated normally.
- Job step error termination
  The last command that executed in the job step normal block terminated with an error.

(3) Return codes of external commands

The return codes are predefined for each external command.

Although the range of values that the external commands can return depends on the platform and the programming language specifications used for the external commands, we recommend that you use a range from 0 through 255. If the value is outside this range, JP1/Advanced Shell uses the following value as the external command's return code:

UNIX

The trailing eight bits of the value returned by the external command

Windows

- If the value returned by the external command is `0` or greater: the trailing eight bits of the value
- If the value returned by the external command is less than `0`: `255`
- If the external command terminated due to an exception:# the trailing eight bits of the exception code

#

The following table explains the exception codes that are treated as exceptions.

Table 5–38: Exception codes treated as exceptions and their meaning

No.	Exception code	Meaning
1	`0xC0000005`	The thread attempted to access a virtual address for which access permissions are not granted.
2	`0x80000003`	A breakpoint has been reached.
3	`0x80000002`	An attempt was made to access misaligned data on hardware with memory access-related alignment rules (for example, a 16-bit value spanning a two-byte boundary and a 32-bit value spanning a four-byte boundary are not permitted).
4	`0x80000004`	This indicates execution of a single instruction at a time by using a trace or single-step method.
5	`0xC000008C`	Hardware detected an attempt made by the thread to access data outside the range of an array.
6	`0xC000008D`	In a floating-point operation, at least one of the operands is an unnormalized number (the value is too small to express in the normal floating-point format).
7	`0xC000008E`	The thread attempted to perform division by zero during a floating-point operation.
8	`0xC000008F`	The accurate value could not be obtained from a floating-point operation.
9	`0xC0000030`	An exception of to a floating-point operation other than an exception listed in this table occurred.
10	`0xC0000091`	As a result of a floating-point operation, the value of the exponent part was above the permitted range.
11	`0xC0000032`	As a result of a floating-point operation, a stack resulted in overflow or underflow.
12	`0xC0000033`	As a result of a floating-point operation, the value of the exponent part was below the permitted range.
13	`0xC0000094`	The thread attempted to perform division by zero during an integer arithmetic operation.
14	`0xC0000035`	An integer arithmetic operation resulted in overflow.
15	`0xC0000096`	An attempt was made to execute an instruction (privileged instruction) that cannot be executed in the current machine mode.
16	`0xC0000025`	An attempt was made to re-execute a command that resulted in an unresumable exception.

If the external command execution result is any of the ones listed in the following table, JP1/Advanced Shell assumes that the corresponding external command terminated with an error.

External command execution result	Return code
The external command's return code is not 0 (this value can be changed with the `successRC` attribute and with the `CMDRC_THRESHOLD_USE_PRESET` and `CMDRC_THRESHOLD_DEFINE` parameters)	External command's return code
The external command was terminated by signal.	Return code for signal termination that is predefined by the external command
The specified external command could not be executed due to a lack of execution permissions.	`126`
The specified external command could not be executed because it did not exist.	`127`

The external commands specified in the `#-adsh_rc_ignore` command will not result in an error regardless of the return code.

(4) Termination codes of built-in commands

The termination codes of built-in commands depend on the command. Whether a built-in command terminated normally or resulted in an error is determined based on any error event that occurred during command execution, not on the value of the termination code. Note that a built-in command specified in the `#-adsh_rc_ignore` command always terminates normally regardless of the command's execution results.

For details about the conditions that determine whether each command has terminated normally or resulted in an error, see the termination codes for each command described in *9. Job Definition Script Commands and Control Statements*.

(5) Notes

The return code of UNIX-compatible commands and user-created commands might not be `0` even if they terminate normally. For example, the `diff` command terminates with code `1` when the compared files differ.

To correctly determine whether such commands terminated normally or with an error, use the methods described below to code your job definition scripts. For details about the return codes of UNIX-compatible commands, see *8.4 UNIX-compatible commands*.

(a) Specifying in the environment setting parameters

- To correctly determine whether UNIX-compatible commands terminated normally or with an error, specify `ENABLE` in the `CMDRC_THRESHOLD_USE_PRESET` parameter.
- To correctly determine whether OS-provided commands or user-created commands terminated normally or with an error, specify the `CMDRC_THRESHOLD_DEFINE` parameter.

(b) Specifying in job definition scripts

- To set a command to always terminate normally, specify the name of that command in the argument of the `#-adsh_rc_ignore` command.
- To determine whether a command terminated normally or with an error according to the command specifications, define the corresponding command in a job step and specify the return code for normal termination with the `successRC` attribute. The specified `successRC` attribute takes effect on all commands that are executed within the corresponding job step.

5.8.9 Job cancellation by the standard shell commands

When a standard shell command is executed, job execution might be canceled depending on the type of standard shell command and the execution result of the standard shell command. If this happens, the `KNAX6584-I` message is issued and the subsequent job steps and job definition scripts are not executed. Also, the job steps whose `run` attribute is `bnormal` or `always` are not executed.

In this case, the specification of the `#-adsh_rc_ignore` command and the `successRC` attribute of the `#-adsh_step_start` command do not take effect on the executed commands.

(1) Executing a command that immediately terminates a job definition script

If a standard shell command that immediately terminates a job definition script is executed, job execution is canceled. The following commands immediately terminate a job definition script:

- `exit` command
- `return` command#
- `exec` command with an executable command specified in its argument

#

A job definition script is not terminated in the following cases:

- The command was executed within a function.
- The command was executed within an external script.

(2) Unresumable errors

If a standard shell command is executed, an error (such as a syntax error) that disables operation of the job definition script itself might occur. If this happens, job execution is canceled. Syntax errors occur in the following cases:

- Executing the `unset` command with no argument specified
- Specifying a character string in the argument of the `return` command that is specified outside the function as well as outside an external script[#]
- In the Windows edition, executing an unsupported conditional expression while `ERR` is set in the `UNSUPPORT_TEST` parameter

#

Job execution is not cancelled in the following cases:

- The command was executed within a function.
- The command was executed within an external script.

5.8.10 Processing in the event of an error during job execution

This subsection explains how commands and control statements are handled when an error occurs during job execution. The following types of errors can occur:

- Errors in extended script commands
  This type of error occurs when file allocation by the `#-adsh_file` command fails.
- Errors in standard shell commands
  - If the processing is resumable
    This type of error occurs when a specified command name cannot be found or a regular built-in command results in an error.
  - If the processing is not resumable
    This type of error occurs when a special built-in command results in an error.

(1) Errors outside job steps

The following table explains the processing in the event of an error outside the job step.

Table 5–39: Processing in the event of an error outside the job step

Type of error	Processing of subsequent commands and control statements
Error in an extended script command	• Executes a job step if its `run` attribute is `abnormal` or `always`. • Does not execute any commands or control statements other than the above.
Error in a standard shell command (resumable)	• Does not execute a job step if its `run` attribute is `normal`. • Executes all commands and control statements other than the above.

Type of error	Processing of subsequent commands and control statements
Error in a standard shell command (not resumable)	Terminates execution of the job definition script.

The following table provides examples of errors outside job steps.

Table 5–40: Examples of errors outside job steps

Example of coding in the job definition script	Error in extended script command	Error in standard shell command	
		Resumable	Not resumable
`#-adsh_file JOBFILE jobfile`	E	--	--
`cmdA`	N	E	--
`shift $n`	N	Y	E
`cmdB`	N	Y	N
	--	--	--
`#-adsh_step_start NO -run normal`	N	N	N
`  echo "run normal step"`	N	N	N
`  cmdNormal`	N	N	N
`#-adsh_step_end`	N	N	N
	--	--	--
`#-adsh_step_start AB -run abnormal`	Y	Y	N
`  echo "run abnormal step"`	Y	Y	N
`  cmdAbnormal`	Y	Y	N
`#-adsh_step_end`	Y	Y	N
	--	--	--
`#-adsh_step_start AL -run always`	Y	Y	N
`  echo "run always step"`	Y	Y	N
`  cmdAlways`	Y	Y	N
`#-adsh_step_end`	Y	Y	N

Legend:

Y: Executed.

N: Not executed.

E: Results in an error.

--: Not applicable.

Note:

A row that is blank in the *Example of coding in the job definition script* column means that nothing is specified.

(2) Errors inside the job step normal block

The following table explains the processing in the event of an error inside the job step normal block.

Table 5–41: Processing in the event of an error inside the job step normal block

Type of error	Processing of subsequent commands and control statements
Error in an extended script command Error in a standard shell command (resumable)#	• Does not execute any commands or control statement inside the job step normal block. • Executes the job step error block, if defined. • The job step terminates with an error. The processing of subsequent commands and control statements is the same as when an error occurs in an extended script command outside the job step.
Error in a standard shell command (not resumable)	Terminates execution of the job definition script.

#

If the `onError` attribute is `cont` and the error (resumable) resulted from the last standard shell command inside the job step, JP1/Advanced Shell assumes that an error occurred inside the job step normal block.

The following table provides examples of errors inside the job step normal block.

Table 5–42: Examples of errors inside the job step normal block

Example of coding in the job definition script	Error in extended script command	Error in standard shell command		
		Resumable	Resumable (onError attribute is cont)	Not resumable
`#-adsh_step_start S1 -onError stop`	--	--	--	--
`  #-adsh_file JOBFILE jobfile`	E	--	--	--
`  cmdA`	N	E	E	--
`  shift $n`	N	N	Y	E
`  cmdB`	N	N	Y	N
`#-adsh_step_error`	Y	Y	N	N
`  echo "step error block"`	Y	Y	N	N
`#-adsh_step_end`	Y	Y	Y	N
	--	--	--	--
`#-adsh_step_start NO -run normal`	N	N	Y	N
`  echo "run normal step"`	N	N	Y	N

Example of coding in the job definition script	Error in extended script command	Error in standard shell command		
		Resumable	Resumable (onError attribute is cont)	Not resumable
`  cmdNormal`	N	N	Y	N
`#-adsh_step_end`	N	N	Y	N
	--	--	--	--
`#-adsh_step_start AB -run abnormal`	Y	Y	N	N
`  echo "run abnormal step"`	Y	Y	N	N
`  cmdAbnormal`	Y	Y	N	N
`#-adsh_step_end`	Y	Y	N	N
	--	--	--	--
`#-adsh_step_start AL -run always`	Y	Y	Y	N
`  echo "run always step"`	Y	Y	Y	N
`  cmdAlways`	Y	Y	Y	N
`#-adsh_step_end`	Y	Y	Y	N

Legend:

Y: Executed.

N: Not executed.

E: Results in an error.

--: Not applicable.

Note:

A row that is blank in the *Example of coding in the job definition script* column means that nothing is specified.

(3) Errors inside the job step error block

The following table explains the processing of subsequent commands and control statements in the event of an error in the job step error block.

Table 5–43: Processing in the event of an error inside the job step error block

Type of error	Processing of subsequent commands and control statements
Error in an extended script command	• Does not execute any command or control statement inside the job step error block. • The job step terminates with an error. The processing of subsequent commands and control statements is the same as when an error occurs in an extended script command outside the job step.
Error in a standard shell command (resumable)	• Executes all commands and control statements inside the job step error block. • The job step terminates with an error. The processing of subsequent commands and control statements is the same as when an error occurs in an extended script command outside the job step.

Type of error	Processing of subsequent commands and control statements
Error in a standard shell command (not resumable)	Terminates execution of the job definition script.

The following table shows examples of errors inside the job step error block.

Table 5–44: Examples of errors inside the job step error block

Example of coding in the job definition script	Error in extended script command	Error in standard shell command	
		Resumable	Not resumable
`#-adsh_step_start S1 -onError stop`	--	--	--
`  echo "step normal block"`	--	--	--
`#-adsh_step_error`	--	--	--
`  #-adsh_file JOBFILE jobfile`	E	--	--
`  cmdA`	N	E	--
`  shift $n`	N	Y	E
`  cmdB`	N	Y	N
`#-adsh_step_end`	Y	Y	N
	--	--	--
`#-adsh_step_start NO -run normal`	N	N	N
`  echo "run normal step"`	N	N	N
`  cmdNormal`	N	N	N
`#-adsh_step_end`	N	N	N
	--	--	--
`#-adsh_step_start AB -run abnormal`	Y	Y	N
`  echo "run abnormal step"`	Y	Y	N
`  cmdAbnormal`	Y	Y	N
`#-adsh_step_end`	Y	Y	N
	--	--	--
`#-adsh_step_start AL -run always`	Y	Y	N

Example of coding in the job definition script	Error in extended script command	Error in standard shell command	
		Resumable	Not resumable
`  echo "run always step"`	Y	Y	N
`  cmdAlways`	Y	Y	N
`#-adsh_step_end`	Y	Y	N

Legend:

Y: Executed.

N: Not executed.

E: Results in an error.

--: Not applicable.

Note:

A row that is blank in the *Example of coding in the job definition script* column means that nothing is specified.

(4) Notes

If a resumable error occurs in a standard shell command outside the job step, all subsequent commands and control statements are executed except for a job step whose `run` attribute is `normal`. In this case, the job's return code is overwritten by the return codes of the subsequent commands and control statements. If you want to apply as the job's return code the return code that caused the error (to treat the job as having resulted in an error in JP1/AJS), specify the commands and control statements in the job step with `stop` as the `onError` attribute.

5.8.11 Notes about output of command execution results

The following notes apply to checking command execution results that are output to a job execution log file.

(1) Output of command execution results when commands are grouped by a separate program

A group of commands enclosed in parentheses (`(` and `)`) is executed as a single job definition script in a separate process. For the command execution results, one of the following messages is output as the execution result of a single job definition script:

`KNAX6540-I`, `KNAX6541-E`, `KNAX6542-E`, `KNAX6560-I`, `KNAX6561-E`, `KNAX6562-E`

(2) Notes about background execution

The following notes apply to output of the termination message for a command that is executed in the background by specifying `&` or `|&`:

- A termination message is always output because the job terminates after all the commands executed in the background have terminated.
- A command started within a job step might not terminate until after the job step has terminated. In such a case, the termination message for the command is output after the termination message for the job step that started the command. The return code of a command executed in the background by specifying `&` or `|&` has no effect on the return code of the job step and job.

- The order in which information about commands executed in the background is output to the job execution logs is undefined regardless of the actual order in which the processes started and terminated. The same applies to a group of commands linked with the vertical bar (`|`).

(3) Notes about the builtin, command, eval, time, . (dot), and exec commands

The following notes apply to the execution results of the `builtin`, `command`, `eval`, `time`, `.` (dot), and `exec` commands.

- Built-in `builtin`, `command`, `eval`, `time`, and `exec` commands

  JP1/Advanced Shell outputs only the execution results of commands executed as arguments. The execution results of the `builtin`, `command`, `eval`, `time`, and `exec` commands are not actually output. The execution results of these commands are not used to evaluate whether the job or job step resulted in normal termination or error termination.

  If an option that is not supported by the platform being used is specified for the `command` command, JP1/Advanced Shell outputs the execution results of the `command` command and then evaluates whether the job and job step resulted in normal termination or error termination.

- Built-in `.` (dot) command

  JP1/Advanced Shell outputs only the execution results of each command in a specified external script.

  The `.` (dot) command itself terminates normally, but its execution results are not output. The termination results of the `.` (dot) command are not used to evaluate whether the job or job step resulted in normal termination or error termination.

  If the specified external script did not exist and the `.` (dot) command terminated with an error, JP1/Advanced Shell outputs the execution results of the `.` (dot) command and then evaluates whether the job or job step resulted in normal termination or error termination.

(4) Differences in the output of command execution results depending on the command format

If you execute commands using the following formats, the command names before conversion are output to the command execution results:

- Execution of separate processes by using pipes (`|`)
- Execution of separate processes by using command substitution (`$()`, ` `` `)
- Execution of background processes by using `|&`
- Subshell execution by using command grouping
- Background execution by using `&`

The following shows examples.

- If a variable is executed as a command, the character string before the variable is expanded is output as the command name.

  Definition in the job definition script:

  ```
  01: $CMD &
  ```

  Output execution results:

  ```
  KNAX6116-I Execution of the command $CMD (line=1) finished successfully.
  exit status=0 execution time=0.001s CPU time=0.000s
  ```

- If a character string that satisfies the rule defined in the CHILDJOB_PGM parameter in the environment file is executed as a command, the character string before it was replaced by the CHILDJOB_PGM parameter is output as the command name.

  Definition in the environment file:

  ```
  #-adsh_conf CHILDJOB_PGM /bin/sh
  ```

  Definition in the job definition script:

  ```
  01: /bin/sh ./test.ash &
  ```

  Output execution results:

  ```
  KNAX6116-I Execution of the command /bin/sh (line=1) finished
  successfully. exit status=0 execution time=0.008s CPU time=0.000s
  ```

5.9 Allocating files and performing postprocessing

You can use extended script commands or the `adshfile` command to allocate regular files, temporary files, and program output data files and to perform postprocessing.

File allocation is an operation that occurs each time a command executes. It includes registering the handling of each file when job steps and jobs terminate and creating file names and file entities.

Postprocessing is an operation that occurs when job steps and jobs terminate. It includes deleting and retaining each file according to the definition specified during file allocation.

For details about the extended script commands, see *9.5 Extended script commands*. For details about the `adshfile` command, see *adshfile command (specifies the allocation and postprocessing of regular files)*.

5.9.1 Allocating regular files and performing postprocessing

Use the `#-adsh_file` command (extended script command) or the `adshfile` command (shell operation command) to do the following:

- Performing postprocessing on the allocated files according to the results of the corresponding job step or job.
- If the `#-adsh_file` command is used, assigning to shell variables and environment variables the file paths of regular files that will be used in the job or job step and the commands that will be started.

The table below describes the functional differences between the `#-adsh_file` and `adshfile` commands. For details about the `#-adsh_file` command, see *#-adsh_file command (specifies assignment and postprocessing of regular files)*. For details about the `adshfile` command, see *adshfile command (specifies the allocation and postprocessing of regular files)*.

Item	#-adsh_file command	adshfile command
Specifying file paths in environment variables	Y	N
Timing of file postprocessing	• Outside the step When the job terminates • Inside the step When the step terminates	Specifies either job termination or step termination.
Whether specification is permitted in external scripts specified in `.` (dot) commands	N	Y
Whether specification is permitted within iteration processing	N	Y
Whether specification is permitted within functions	N	Y
Postprocessing of allocated regular files in the event of a file allocation error	Assumes `keep` and does not delete files. For details, see *5.9.1(2)(a) #-adsh_file command*.	Depends on the specification of the `-a` argument. For details, see *5.9.1(2)(b) adshfile command*.

Legend:

Y: Supported.

N: Not supported.

Important note

Regular files allocated with the `adshfile` command are managed separately from regular files allocated with the `#-adsh_file` command. Postprocessing is performed on regular files allocated by the `adshfile` command before it is performed on regular files allocated by the `#-adsh_file` command. If the same file is allocated with both commands, it will be postprocessed twice, which might result in an error.

(1) Allocating regular files

The `#-adsh_file` command assigns the file path of a regular file to a shell variable and an environment variable that have the same name as the file environment variable definition name.

The `#-adsh_file` and `adshfile` commands do not create the entities of the specified regular files.

To check whether a file exists when the file is to be allocated, specify `exist` for `-chk` in the `#-adsh_file` command and for `-c` in the `adshfile` command. When `exist` is specified and the specified file does not exist at the time of allocation, an error results.

To allocate a file regardless of whether the file exists, specify `no` for `-chk` in the `#-adsh_file` command and for `-c` in the `adshfile` command. When `no` is specified, the command allocates the specified file, even if the file does not exist without treating it as an error.

The following examples allocate regular file `test1` to `FILE` by using the `#-adsh_file` command:

- Windows

```
#-adsh_file FILE 'C:\home\test\test1' -chk exist -normal keep -abnormal
keep
```

- UNIX

```
#-adsh_file FILE /home/test/test1 -chk exist -normal keep -abnormal keep
```

The following shows a usage example of the `chk` attribute in UNIX:

```
#-adsh_job FILE_CHK
#-adsh_step_start STEP01
  #-adsh_file FILE01 /home/test/test1 -chk no -normal keep -abnormal keep
  #-adsh_file FILE02 /home/test/test2 -chk exist -normal keep -abnormal keep
  cmdA ${FILE01} ${FILE02}
#-adsh_step_end
```

When it allocates `FILE01`, the command does not check whether the file exists. Therefore, the file is always allocated even if `/home/test/test1` does not exist.

When it allocates `FILE02`, the command checks whether the file exists. Therefore, if `/home/test/test2` does not exist, the file allocation processing results in an error.

(2) Postprocessing regular files

Regular files are postprocessed when the job step or job that allocated the files terminates. Postprocessing by the `#-adsh_file` command includes resetting the shell and environment variables in which the file paths are set to their previous values before the file paths were set. The processing described below is also performed according to the termination status of the job step or job and values specified in the command.

If the job step and job terminated normally, the #-adsh_file command performs postprocessing according to the value specified for -normal, and the adshfile command performs postprocessing according to the value specified for -n. The following explains the postprocessing of regular files, depending on the specified value:

- If del is specified, a regular file is deleted when the job step or job terminates.
- If keep is specified, a regular file is not deleted when the job step or job terminates.

If the job step or job terminates with an error, postprocessing is performed according to the value specified for -abnormal in the #-adsh_file command, or the value specified for -a in the -adshfile command. The following explains the postprocessing of regular files, depending on the specified value.

- If del is specified, a regular file is deleted when the job step or job terminates.
- If keep is specified, a regular file is not deleted when the job step or job terminates.

The following table explains postprocessing when -abnormal is specified in the #-adsh_file command or the -a option is specified in the adshfile command:

Cause of an error in the job step or job	Postprocessing of regular files allocated with the #-adsh_file command	Postprocessing of regular files allocated with the adshfile command
An error occurred while a regular file was being allocated by the subsequent #-adsh_file command	keep is assumed unconditionally.	Depends on the -a option of the adshfile command.# (keep is not assumed.)
An error occurred while a regular file was being allocated by the subsequent adshfile command	Depends on the -abnormal option of each #-adsh_file command.	Depends on the -a option of the adshfile command.
An error occurred while a temporary file was being allocated by the subsequent #-adsh_temp command	keep is assumed unconditionally.	Depends on the -a option of the adshfile command.# (keep is not assumed.)
An error occurred while a program output data file was being allocated by the subsequent #-adsh_spoolfile command	keep is assumed unconditionally.	Depends on the -a option of the adshfile command.# (keep is not assumed.)
Error in a subsequent command or program other than the above	Depends on the -abnormal option of each #-adsh_file command.	Depends on the -a option of the adshfile command.

#
If you want to retain files unconditionally in the event of an allocation error, use the #-adsh_file command.

Because the actual file is not created when a regular file is allocated, any file that does not exist when postprocessing is performed remains nonexistent.

(a) #-adsh_file command

The following shows an example of postprocessing in Windows:

```
#-adsh_job JOB
#-adsh_file FILE01 'C:\user\file01' -chk exist -normal keep -abnormal del

#-adsh_step_start STEP
    #-adsh_file FILE02 'C:\user\file02' -chk exist -normal del -abnormal del
    #-adsh_file FILE03 'C:\user\file03' -chk exist -normal keep -abnormal 
del
    uap
#-adsh_step_end

#-adsh_file FILE04 'C:\user\file04' -chk exist -normal del -abnormal del
```

Example 1:

Allocation of `FILE03` resulted in an error.

- The regular file allocated to `FILE01` is postprocessed when the job terminates. The regular file is deleted according to the specified `abnormal` attribute.
- The regular file allocated to `FILE02` is postprocessed when the job step terminates. The command assumes `keep` regardless of the specified `normal` and `abnormal` attributes and does not delete the regular file.
- The regular file whose allocation to `FILE03` was attempted is not postprocessed because allocation is not completed.
- Allocation of `FILE04` is not performed.

Example 2:

Allocation of `FILE04` resulted in an error.

- The regular file allocated to `FILE01` is postprocessed when the job terminates. The command assumes `keep` regardless of the specified `normal` and `abnormal` attributes and does not delete the regular file.
- The regular file allocated to `FILE02` is postprocessed when the job step terminates. The command deletes the regular file according to the specified `normal` attribute.
- The regular file allocated to `FILE03` is postprocessed when the job step terminates. The command does not delete the regular file according to the specified `normal` attribute.
- The regular file whose allocation to `FILE04` was attempted is not postprocessed because allocation is not completed.

Example 3:

UAP resulted in an error.

- The regular file allocated to `FILE01` is postprocessed when the job terminates. The regular file is deleted according to the specified `abnormal` attribute.
- The regular file allocated to `FILE02` is postprocessed when the job step terminates. The command deletes the regular file according to the specified `abnormal` attribute.
- The regular file allocated to `FILE03` is postprocessed when the job step terminates. The command deletes the regular file according to the specified `abnormal` attribute.
- Allocation of `FILE04` is not performed.

Example 4:

File allocation terminated normally.

- The regular file allocated to `FILE01` is postprocessed when the job terminates. The command does not delete the regular file according to the specified `normal` attribute.
- The regular file allocated to `FILE02` is postprocessed when the job step terminates. The command deletes the regular file according to the specified `normal` attribute.
- The regular file allocated to `FILE03` is postprocessed when the job step terminates. The command does not delete the regular file according to the specified `normal` attribute.

- The regular file allocated to `FILE04` is postprocessed when the job terminates. The command deletes the regular file according to the specified `normal` attribute.

(b) adshfile command

The following shows an example of postprocessing. In this example, the names of the regular files are `file01_`*execution-date* and `file02_`*execution-date*, where *execution-date* is replaced with the execution time of the job.

```
#-adsh_job JOB

VAL01=file01_`date +%y%m%d`
VAL02=file02_`date +%y%m%d`
VAL03=file03_`date +%y%m%d`
VAL04=file04_`date +%y%m%d`
VAL05=file05_`date +%y%m%d`


adshfile -s job -n keep -a del ${VAL01}

#-adsh_step_start STEP
    adshfile -s step -n del -a del ${VAL02}
    adshfile -s step -n keep -a keep ${VAL03}
    adshfile -s step -n keep -a del ${VAL04}
    uap
#-adsh_step_end

adshfile -s job -n del -a del ${VAL05}
```

Example 1:

Allocation of `VAL04` resulted in an error.

- The regular file allocated to `VAL01` is postprocessed when the job terminates. The regular file is deleted according to the specified `-a` option.
- The regular file allocated to `VAL02` is postprocessed when the job step terminates. The regular file is deleted according to the specified `-a` option (unlike the `#-adsh_file` command, postprocessing is performed according to the specified `-a` option; `keep` is not assumed).
- The regular file allocated to `VAL03` is postprocessed when the job step terminates. The command does not delete the regular file according to the specified `-a` option.
- `VAL04` results in a command error, and the regular file whose allocation to `VAL04` was attempted is not postprocessed because allocation was not completed.
- Allocation of `VAL05` is not performed.

Example 2:

Allocation of `VAL05` resulted in an error.

- The regular file allocated to `VAL01` is postprocessed when the job terminates. The regular file is deleted according to the specified `-a` option (unlike the `#-adsh_file` command, postprocessing is performed according to the specified `-a` option; `keep` is not assumed).
- The regular file allocated to `VAL02` is postprocessed when the job step terminates. The command deletes the regular file according to the specified `-n` option.

- The regular file allocated to `VAL03` is postprocessed when the job step terminates. The command does not delete the regular file according to the specified `-n` option.
- The regular file allocated to `VAL04` is postprocessed when the job step terminates. The command does not delete the regular file according to the specified `-n` option.
- `VAL05` results in a command error, and the regular file whose allocation to `VAL05` was attempted is not postprocessed because allocation was not completed.

Example 3:

`uap` resulted in an error.

- The regular file allocated to `VAL01` is postprocessed when the job terminates. The regular file is deleted according to the specified `-a` option.
- The regular file allocated to `VAL02` is postprocessed when the job step terminates. The regular file is deleted according to the specified `-a` option.
- The regular file allocated to `VAL03` is postprocessed when the job step terminates. The command does not delete the regular file according to the specified `-a` option.
- The regular file allocated to `VAL04` is postprocessed when the job step terminates. The regular file is deleted according to the specified `-a` option.
- Allocation of `VAL05` is not performed.

Example 4:

File allocation terminated normally.

- The regular file allocated to `VAL01` is postprocessed when the job terminates. The command does not delete the regular file according to the specified `-n` option.
- The regular file allocated to `VAL02` is postprocessed when the job step terminates. The command deletes the regular file according to the specified `-n` option.
- The regular file allocated to `VAL03` is postprocessed when the job step terminates. The command does not delete the regular file according to the specified `-n` option.
- The regular file allocated to `VAL04` is postprocessed when the job step terminates. The command does not delete the regular file according to the specified `-n` option.
- The regular file allocated to `VAL05` is postprocessed when the job terminates. The command deletes the regular file according to the specified `-n` option.

(3) Example of adshfile command specification (within iteration processing)

The following example specifies the `adshfile` command within iteration processing in UNIX:

```
[user001@hosta user001]$ /opt/jp1as/bin/adshexec Tloop.ash
KNAX7901-I The adshexec command will wait for all asynchronous processes at
the end of the job.
KNAX0724-I The job ID was assigned. job ID=000500
---------------------------------------------------------------
 Advanced Shell 10-10

 [Information]
   Job ID          : 000500
   Spool directory : /var/opt/jp1as/spool/000500/
   Date            : 2013/12/06
   EnvFile(system) :
   EnvFile(job)    :
   Host name       : hosta
 [Environment variable from Automatic Job Management System]
---------------------------------------------------------------
********  JOB CONTROLLER MESSAGE  ********
15:23:22 000500 KNAX0091-I LOOP The job started.
15:23:22 000500 KNAX7901-I The adshexec command will wait for all
asynchronous processes at the end of the job.
15:23:22 000500 KNAX7902-I The adshexec command will run in tty stdin mode.
15:23:22 000500 KNAX6112-I Execution of the command echo (line=3) finished
successfully. exit status=0 execution time=0.000s CPU time=0.000s
15:23:22 000500 KNAX6116-I Execution of the command /bin/ls (line=4)
finished successfully. exit status=0 execution time=0.001s CPU time=0.000s
15:23:22 000500 KNAX6116-I Execution of the command /opt/jp1as/bin/adshfile
(line=8) finished successfully. exit status=0 execution time=0.011s CPU
time=0.000s
15:23:22 000500 KNAX6116-I Execution of the command /opt/jp1as/bin/adshfile
(line=9) finished successfully. exit status=0 execution time=0.009s CPU
time=0.010s
15:23:22 000500 KNAX6112-I Execution of the command echo (line=10) finished
successfully. exit status=0 execution time=0.000s CPU time=0.000s
15:23:22 000500 KNAX6112-I Execution of the command echo (line=11) finished
successfully. exit status=0 execution time=0.000s CPU time=0.000s
15:23:22 000500 KNAX6116-I Execution of the command /bin/cat (line=12)
finished successfully. exit status=0 execution time=0.001s CPU time=0.000s
15:23:22 000500 KNAX6116-I Execution of the command /opt/jp1as/bin/adshfile
(line=8) finished successfully. exit status=0 execution time=0.006s CPU
time=0.000s
15:23:22 000500 KNAX6116-I Execution of the command /opt/jp1as/bin/adshfile
(line=9) finished successfully. exit status=0 execution time=0.009s CPU
time=0.010s
15:23:22 000500 KNAX6112-I Execution of the command echo (line=10) finished
successfully. exit status=0 execution time=0.000s CPU time=0.000s
15:23:22 000500 KNAX6112-I Execution of the command echo (line=11) finished
successfully. exit status=0 execution time=0.000s CPU time=0.000s
15:23:22 000500 KNAX6116-I Execution of the command /bin/cat (line=12)
finished successfully. exit status=0 execution time=0.001s CPU time=0.000s
15:23:22 000500 KNAX6116-I Execution of the command /opt/jp1as/bin/adshfile
(line=8) finished successfully. exit status=0 execution time=0.006s CPU
time=0.000s
15:23:22 000500 KNAX6116-I Execution of the command /opt/jp1as/bin/adshfile
(line=9) finished successfully. exit status=0 execution time=0.009s CPU
time=0.010s
15:23:22 000500 KNAX6112-I Execution of the command echo (line=10) finished
successfully. exit status=0 execution time=0.000s CPU time=0.000s
15:23:22 000500 KNAX6112-I Execution of the command echo (line=11) finished
successfully. exit status=0 execution time=0.000s CPU time=0.000s
15:23:22 000500 KNAX6116-I Execution of the command /bin/cat (line=12)
finished successfully. exit status=0 execution time=0.001s CPU time=0.000s
15:23:22 000500 KNAX1890-I The file was deallocated as "keep". path=/home/
user001/test.txt
15:23:22 000500 KNAX1890-I The file was deallocated as "del". path=/home/
user001/output_test.txt
15:23:22 000500 KNAX1604-I The file /home/user001/output_test.txt was
deleted.
15:23:22 000500 KNAX1890-I The file was deallocated as "keep". path=/home/
```

```
user001/test01.txt
15:23:22 000500 KNAX1890-I The file was deallocated as "del". path=/home/
user001/output_test01.txt
15:23:22 000500 KNAX1604-I The file /home/user001/output_test01.txt was
deleted.
15:23:22 000500 KNAX1890-I The file was deallocated as "keep". path=/home/
user001/test02.txt
15:23:22 000500 KNAX1890-I The file was deallocated as "del". path=/home/
user001/output_test02.txt
15:23:22 000500 KNAX1604-I The file /home/user001/output_test02.txt was
deleted.
15:23:22 000500 KNAX0098-I LOOP The job ended. exit status=0 execution
time=0.126s CPU time=0.050s

********   Script IMAGE    ********

***** /home/user001/Tloop.ash *****
0001 : #-adsh_job LOOP
0002 :
0003 :   echo -E "<<< list >>>" >&2
0004 :   ls test* >&2
0005 :
0006 :   for MEMBER in test*
0007 :   do
0008 :     ${ADSH_DIR_BIN}adshfile -s job -n keep -a keep $MEMBER
0009 :     ${ADSH_DIR_BIN}adshfile -s job -n del -a del output_$MEMBER
0010 :     echo -E "MEMBER=$MEMBER" >output_$MEMBER
0011 :     echo -E  "<<< output_$MEMBER >>>" >&2
0012 :     cat output_$MEMBER >&2
0013 :   done
0014 :

***** CONVERSION INFORMATION *****

********   JOB SCOPE STDERR    ********
<<< list >>>
test.txt
test01.txt
test02.txt
<<< output_test.txt >>>
MEMBER=test.txt
<<< output_test01.txt >>>
MEMBER=test01.txt
<<< output_test02.txt >>>
MEMBER=test02.txt
KNAX0098-I LOOP The job ended. exit status=0 execution time=0.126s CPU
time=0.050s

******** JOBSTEP OUTPUT ********
KNAX6380-I A job name will be added to the spool job directory of the root
job. spool job directory="/var/opt/jp1as/spool/000500-LOOP/"
KNAX7999-I Advanced Shell ended. exit status=0
[user001@hosta user001]$
```

(4) Example of adshfile command specification (within a function)

The following example specifies the `adshfile` command within a function in UNIX:

```
[user001@hosta user001]$ /opt/jp1as/bin/adshexec Tfunction.ash
KNAX7901-I The adshexec command will wait for all asynchronous processes at
the end of the job.
KNAX0724-I The job ID was assigned. job ID=000503
---------------------------------------------------------------
 Advanced Shell 10-10

 [Information]
   Job ID          : 000503
   Spool directory : /var/opt/jp1as/spool/000503/
   Date            : 2013/12/06
   EnvFile(system) :
   EnvFile(job)    :
   Host name       : hosta
 [Environment variable from Automatic Job Management System]
---------------------------------------------------------------
********  JOB CONTROLLER MESSAGE  ********
15:42:00 000503 KNAX0091-I FUNCTION The job started.
15:42:00 000503 KNAX7901-I The adshexec command will wait for all
asynchronous processes at the end of the job.
15:42:00 000503 KNAX7902-I The adshexec command will run in tty stdin mode.
15:42:00 000503 KNAX0092-I FUNCTION.STEP001 step started.
15:42:00 000503 KNAX6112-I Execution of the command echo (line=15) finished
successfully. exit status=0 execution time=0.000s CPU time=0.000s
15:42:00 000503 KNAX6116-I Execution of the command /opt/jp1as/bin/adshfile
(line=4) finished successfully. exit status=0 execution time=0.014s CPU
time=0.000s
15:42:00 000503 KNAX6116-I Execution of the command /opt/jp1as/bin/adshfile
(line=5) finished successfully. exit status=0 execution time=0.009s CPU
time=0.000s
15:42:00 000503 KNAX6116-I Execution of the command /opt/jp1as/cmd/cp
(line=6) finished successfully. exit status=0 execution time=0.001s CPU
time=0.000s
15:42:00 000503 KNAX6112-I Execution of the command echo (line=7) finished
successfully. exit status=0 execution time=0.000s CPU time=0.000s
15:42:00 000503 KNAX6116-I Execution of the command /opt/jp1as/cmd/cat
(line=8) finished successfully. exit status=0 execution time=0.001s CPU
time=0.000s
15:42:00 000503 KNAX6112-I Execution of the command echo (line=15) finished
successfully. exit status=0 execution time=0.000s CPU time=0.000s
15:42:00 000503 KNAX6116-I Execution of the command /opt/jp1as/bin/adshfile
(line=4) finished successfully. exit status=0 execution time=0.013s CPU
time=0.000s
15:42:00 000503 KNAX6116-I Execution of the command /opt/jp1as/bin/adshfile
(line=5) finished successfully. exit status=0 execution time=0.009s CPU
time=0.000s
15:42:00 000503 KNAX6116-I Execution of the command /opt/jp1as/cmd/cp
(line=6) finished successfully. exit status=0 execution time=0.001s CPU
time=0.000s
15:42:00 000503 KNAX6112-I Execution of the command echo (line=7) finished
successfully. exit status=0 execution time=0.000s CPU time=0.000s
15:42:00 000503 KNAX6116-I Execution of the command /opt/jp1as/cmd/cat
(line=8) finished successfully. exit status=0 execution time=0.001s CPU
time=0.000s
15:42:00 000503 KNAX6112-I Execution of the command echo (line=15) finished
successfully. exit status=0 execution time=0.000s CPU time=0.000s
15:42:00 000503 KNAX6116-I Execution of the command /opt/jp1as/bin/adshfile
```

```
(line=4) finished successfully. exit status=0 execution time=0.014s CPU
time=0.000s
15:42:00 000503 KNAX6116-I Execution of the command /opt/jp1as/bin/adshfile
(line=5) finished successfully. exit status=0 execution time=0.009s CPU
time=0.000s
15:42:00 000503 KNAX6116-I Execution of the command /opt/jp1as/cmd/cp
(line=6) finished successfully. exit status=0 execution time=0.001s CPU
time=0.000s
15:42:00 000503 KNAX6112-I Execution of the command echo (line=7) finished
successfully. exit status=0 execution time=0.000s CPU time=0.000s
15:42:00 000503 KNAX6116-I Execution of the command /opt/jp1as/cmd/cat
(line=8) finished successfully. exit status=0 execution time=0.001s CPU
time=0.000s
15:42:00 000503 KNAX6112-I Execution of the command echo (line=15) finished
successfully. exit status=0 execution time=0.000s CPU time=0.000s
15:42:00 000503 KNAX6116-I Execution of the command /opt/jp1as/bin/adshfile
(line=4) finished successfully. exit status=0 execution time=0.014s CPU
time=0.000s
15:42:00 000503 KNAX6116-I Execution of the command /opt/jp1as/bin/adshfile
(line=5) finished successfully. exit status=0 execution time=0.009s CPU
time=0.000s
15:42:00 000503 KNAX6116-I Execution of the command /opt/jp1as/cmd/cp
(line=6) finished successfully. exit status=0 execution time=0.001s CPU
time=0.000s
15:42:00 000503 KNAX6112-I Execution of the command echo (line=7) finished
successfully. exit status=0 execution time=0.000s CPU time=0.000s
15:42:00 000503 KNAX6116-I Execution of the command /opt/jp1as/cmd/cat
(line=8) finished successfully. exit status=0 execution time=0.001s CPU
time=0.000s
15:42:00 000503 KNAX1890-I The file was deallocated as "del". path=/home/
user001/infile1
15:42:00 000503 KNAX1604-I The file /home/user001/infile1 was deleted.
15:42:00 000503 KNAX1890-I The file was deallocated as "del". path=/home/
user001/outfile1
15:42:00 000503 KNAX1604-I The file /home/user001/outfile1 was deleted.
15:42:00 000503 KNAX1890-I The file was deallocated as "del". path=/home/
user001/infile2
15:42:00 000503 KNAX1604-I The file /home/user001/infile2 was deleted.
15:42:00 000503 KNAX1890-I The file was deallocated as "del". path=/home/
user001/outfile2
15:42:00 000503 KNAX1604-I The file /home/user001/outfile2 was deleted.
15:42:00 000503 KNAX1890-I The file was deallocated as "del". path=/home/
user001/infile3
15:42:00 000503 KNAX1604-I The file /home/user001/infile3 was deleted.
15:42:00 000503 KNAX1890-I The file was deallocated as "del". path=/home/
user001/outfile3
15:42:00 000503 KNAX1604-I The file /home/user001/outfile3 was deleted.
15:42:00 000503 KNAX1890-I The file was deallocated as "del". path=/home/
user001/infile4
15:42:00 000503 KNAX1604-I The file /home/user001/infile4 was deleted.
15:42:00 000503 KNAX1890-I The file was deallocated as "del". path=/home/
user001/outfile4
15:42:00 000503 KNAX1604-I The file /home/user001/outfile4 was deleted.
15:42:00 000503 KNAX6597-I FUNCTION.STEP001 step succeeded. exit status=0
execution time=0.204s CPU time=0.000s
15:42:00 000503 KNAX6112-I Execution of the command echo (line=21) finished
successfully. exit status=0 execution time=0.000s CPU time=0.000s
15:42:00 000503 KNAX0098-I FUNCTION The job ended. exit status=0 execution
```

```
time=0.207s CPU time=0.000s

********   Script IMAGE    ********

***** /home/user001/Tfunction.ash *****
0001 : #-adsh_job FUNCTION
0002 :
0003 : myfunc(){
0004 :    ${ADSH_DIR_BIN}adshfile -s step -n del -a del -c exist $1
0005 :    ${ADSH_DIR_BIN}adshfile -s step -n del -a del $2
0006 :    ${ADSH_DIR_CMD}cp $1 $2
0007 :    echo -E "<<< $2 >>>" >&2
0008 :    ${ADSH_DIR_CMD}cat  $2 >&2
0009 : }
0010 :
0011 : #-adsh_step_start STEP001
0012 :
0013 :     for CNT in 1 2 3 4
0014 :     do
0015 :       echo -E "dddd$CNT" >infile$CNT
0016 :       myfunc infile$CNT outfile$CNT
0017 :     done
0018 :
0019 : #-adsh_step_end
0020 :
0021 : echo -E "JOB_FUNCTION_END" >&2

***** CONVERSION INFORMATION *****

********   JOB SCOPE STDERR    ********
KNAX6597-I FUNCTION.STEP001 step succeeded. exit status=0 execution
time=0.204s CPU time=0.000s
JOB_FUNCTION_END
KNAX0098-I FUNCTION The job ended. exit status=0 execution time=0.207s CPU
time=0.000s

******** JOBSTEP OUTPUT ********
KNAX0719-I STEP. step number=0001 step name=STEP001 output
destination=STDERR
<<< outfile1 >>>
dddd1
<<< outfile2 >>>
dddd2
<<< outfile3 >>>
dddd3
<<< outfile4 >>>
dddd4

KNAX6380-I A job name will be added to the spool job directory of the root
job. spool job directory="/var/opt/jp1as/spool/000503-FUNCTION/"
KNAX7999-I Advanced Shell ended. exit status=0
[user001@hosta user001]$
```

5.9.2 Allocating temporary files and performing postprocessing

You use the `#-adsh_file_temp` command to create a file that will be used temporarily within the job definition script and then assign its file path to a shell variable or environment variable. The allocated temporary file is deleted when the job terminates.

(1) Allocating temporary files

Create a file that will be used temporarily within the job definition script and then allocate its file path to a shell variable or environment variable that has the same name as the specified file environment variable definition name.

There are two ways to allocate temporary files:

- Creating and allocating temporary files
- Allocating existing temporary files

(a) Creating and allocating temporary files

Specify `create` for the `chk` attribute. The size of the file to be created is zero bytes. In UNIX, the permission for a created temporary file depends on the `umask` value for the file owner (creator) part and is always `0` for the group and other users' access permission part. In Windows, no file permission is specified.

To use a temporary file allocated within a job step also in subsequent job steps, specify a temporary file identifier and `keep` for the `normal` attribute.

If a temporary file allocated within a job step is not to be used in subsequent job steps or is to be specified outside the job step, specify `del` for the `normal` attribute.

To allocate a temporary file outside the job step, neither a temporary file identifier nor `normal keep` can be specified.

(b) Allocating existing temporary files

Specify `exist` in the `chk` attribute and the temporary file identifier specified in the earlier job step.

It is not permissible to specify the name of a temporary file that was not created in a preceding job step or a temporary file that was deleted during postprocessing of the preceding job step.

To use an allocated temporary file in a subsequent job step, specify `keep` for the `normal` attribute.

To not use an allocated temporary file in subsequent job steps, specify `del` for the `normal` attribute.

(2) Postprocessing of temporary files

Temporary files are postprocessed when the job step or job that allocated the files terminates. Postprocessing involves resetting the shell and environment variables in which the file path is set to their previous values before the file path was set. Also, the processing described below is performed according to the termination status of the job step or job and the `normal` attribute value.

If the job step and job terminated normally:

- If `keep` is specified for the `normal` attribute, the temporary files are not deleted when the job step terminates. If `keep` is specified for the `normal` attribute, but the temporary files were not used in subsequent job steps, the temporary files are deleted when the job terminates.
- If `del` is specified for the `normal` attribute, the temporary files are deleted when the job step or job terminates.

If the job step and job terminated with an error:

- If keep is specified for the normal attribute, the temporary files are deleted when the job terminates, not when the job step terminates.
- If del is specified for the normal attribute, the temporary files are deleted when the job step or job terminates.

(3) Names of temporary files

The names of temporary files are different in Windows and in UNIX. The following shows the file names used in these OSs.

In Windows

The name of a temporary file consists of the system-specific character string ASH and a name unique in the directory.

```
ASHunique-name.tmp
```

In UNIX

The name of a temporary file consists of the character string TEMP indicating a temporary file, the job name, a temporary file identifier, and a name unique in the directory.

- Temporary file name with a temporary file identifier specified
  TEMP_*job-sequence-number*_*job-name*_*temporary-file-identifier*_*unique-name*
- Temporary file name with a temporary file identifier omitted
  TEMP_*job-sequence-number*_*job-name*_*unique-name*

(4) Storage directory

You use the TEMP_FILE_DIR environment setting parameter to specify the directory in which temporary files are to be created. If the environment setting parameter is omitted, the default value for the TEMP_FILE_DIR parameter is used. For details about the TEMP_FILE_DIR parameter, see *TEMP_FILE_DIR parameter (defines the path name of the directory for storing temporary files)* in *7. Parameters Specified in the Environment Files*.

(5) Examples of usage of temporary files

The following presents examples of the usage of temporary files when temporary files are allocated.

- This example does not use a temporary file allocated within a job step in subsequent job steps, nor does it allocate a temporary file outside the job step

```
#-adsh_file_temp TEMP1 -chk create -normal del
     echo "test" > ${TEMP1}
```

- These examples use temporary files created in a preceding job step also in subsequent job steps

```
#-adsh_step_start STEP1
#-adsh_file_temp TEMP1 -id TEST1 -chk create -normal keep    -->1.
echo "test1" > ${TEMP1}
#-adsh_step_end

#-adsh_step_start STEP2
#-adsh_file_temp TEMP2 -id TEST1 -chk exist -normal keep     -->2.
echo "test2" >> ${TEMP2}
#-adsh_step_end

#-adsh_step_start STEP3
```

```
#-adsh_file_temp TEMP3 -id TEST2 -chk create -normal keep    -->3.
echo "test3" >> ${TEMP3}
#-adsh_step_end

#-adsh_step_start STEP4
#-adsh_file_temp TEMP4 -id TEST1 -chk exist -normal del      -->4.
echo "test4" >> ${TEMP4}
#-adsh_step_end

#-adsh_step_start STEP5
#-adsh_file_temp TEMP5 -id TEST2 -chk exist -normal del      -->5.
echo "test5" >> ${TEMP5}
#-adsh_step_end
```

1. Creates and allocates a temporary file that can be used in subsequent job steps. This example specifies `TEST1` as the temporary file identifier.
2. Allocates the temporary file created in step 1 (identifier: `TEST1`). This example sets the allocated temporary file to be available to subsequent job steps.
3. Creates and allocates a temporary file that can be used in subsequent job steps. This example specifies `TEST2` as the temporary file identifier.
4. Allocates the temporary file used in step 2 (identifier: `TEST2`). This example deletes the allocated temporary file when the job step (job step name: `STEP4`) terminates.
5. Allocates the temporary file created in step 3 (identifier: `TEST2`). This example deletes the allocated temporary file when the job step (job step name: `STEP5`) terminates.

(6) Using temporary files to code input files for a program in a job definition script

If you specify a user program's parameters in a temporary file and use the temporary file as the standard input, you can directly specify the parameters in the job definition script and automatically create and delete the temporary file. The following shows an example:

```
#-adsh_step_start
#-adsh_file_temp SYSIN -id parmfile -chk create -normal keep
cat << @@@ > ${SYSIN}
-in /files/indata
-out /files/outdata
-work /tmp
@@@
uap  ${SYSIN}
#-adsh_step_end
```

This example creates a temporary file and loads multiple lines of character strings specified in the job definition script by using a here document. The user program uses the temporary file as the standard input during execution and deletes the temporary file once execution is completed.

5.9.3 Allocating program output data files and performing postprocessing

JP1/Advanced Shell automatically creates execution results output files for the purpose of providing centralized management of the output results from user programs as well as of the system execution logs. These files are called *program output data files*.

The `#-adsh_spoolfile` command automatically generates the file paths of the program output data files that acquire the execution results output by user programs, and assigns the required shell variables and environment variables.

(1) Allocating program output data files

Automatically create the file path of a program output data file and then allocate it to the shell variable or environment variable that has the same name as the specified file environment variable definition name. These variables are reset to their initial values when the job step or job terminates. No file entity is created.

(2) Names of program output data files

The name of a program output data file consists of such elements as the job name or job step name, a number, and a file environment variable definition name specified in the `#-adsh_spoolfile` command. Such a name is unique for each file definition in the job definition script.

An allocated program output data file is stored in the directory for the corresponding job in the spool root directory specified in the `SPOOL_DIR` environment setting parameter. If no spool root directory is specified in the environment setting parameter, the default value of the `SPOOL_DIR` parameter is used as the spool root directory. For details about the `SPOOL_DIR` parameter, see *7. Parameters Specified in the Environment Files*.

The format of program output data file names is shown below. In Windows, the file name is appended with the extension `.sysout`.

- Name of a program output data file that is allocated outside the job step

```
0000_job-name_sequence-number-of-file-environment-variable-definition-
name_file-environment-variable-definition-name
```

- Name of a program output data file that is allocated inside the job step

```
step-number_step-name_sequence-number-of-file-environment-variable-
definition-name_file-environment-variable-definition-name
```

If the child job was executed with `MERGE` (merging a child job's spool job into the root job's spool job) specified in the `SPOOLJOB_CHILDJOB` parameter in the root job's environment file, the following file name is used:

- When allocated outside the job step

```
Cnumber-giving-the-order-in-which-a-child-job-starts_0000_job-
name_sequence-number-of-file-environment-variable-definition-name_file-
environment-variable-definition-name
```

- When allocated within the job step

```
Cnumber-giving-the-order-in-which-a-child-job-starts_step-number_step-
name_sequence-number-of-file-environment-variable-definition-name_file-
environment-variable-definition-name
```

The variable parts of the program output data file name are replaced with the following information:

step-number

Four-digit decimal number assigned sequentially to each job step. The step number of the first job step is 1.

Examples: `0001`, `0034`, `4095`

job-name

Job name specified with the `#-adsh_job` command. The length of a job name is variable with a maximum length of eight bytes. If a specified character string exceeds eight bytes, the first eight bytes are used as the job name.

step-name

Job step name specified with the `#-adsh_step_start` command. The length of a job step name is variable with a maximum length of eight bytes. If a specified character string exceeds eight bytes, the first eight bytes are used as the job step name.

sequence-number-of-file-environment-variable-definition-name

Sequential number of a program output data file that was allocated outside the job step or that was allocated inside each job step. The sequence number of a file environment variable definition name is a three-digit decimal number. The value range is from `1` through `255` outside the job step and inside each job step.

Examples: `001`, `034`, `255`

file-environment-variable-definition-name

File environment variable definition name specified with the `#-adsh_spoolfile` command.

number-giving-the-order-in-which-a-child-job-starts

Sequential number indicating the order in which a child job started within the root job. It consists of seven decimal digits in the range of `0000001` to `9999999`.

(3) Example of usage of program output data files

This subsection explains the results of program output data file creation when the following job definition script is executed.

```
#-adsh_job JOBSAMPLE001

#-adsh_spoolfile SYS001                        -->1.
#-adsh_spoolfile SYS002                        -->2.
echo "----- job01 --------" 1>&2
echo "SYS001" > $SYS001
echo "SYS002" > $SYS002

#-adsh_step_start STEP01
    #-adsh_spoolfile SYS001                    -->3.
    #-adsh_spoolfile SYS002                    -->4.
    echo "----- Step001 --------" 1>&2
    echo "SYS001" > $SYS001
    echo "SYS002" > $SYS002
#-adsh_step_end

#-adsh_spoolfile SYS001                         -->5.
#-adsh_spoolfile SYS002                         -->6.
echo "----- job02 --------" 1>&2
echo "SYS001" > $SYS001
echo "SYS002" > $SYS002

#-adsh_step_start STEP02
    #-adsh_spoolfile SYS001                      -->7.
    #-adsh_spoolfile SYS002                      -->8.
    echo "----- Step002 --------" 1>&2
    echo "SYS001" > $SYS001
    echo "SYS002" > $SYS002
#-adsh_step_end

#-adsh_spoolfile SYS001                              -->9.
#-adsh_spoolfile SYS002                              -->10.
echo "----- job03 --------" 1>&2
echo "SYS001" > $SYS001
echo "SYS002" > $SYS002
```

When this job definition script is executed, the program output data files with the file names shown below are created. The numbers at the right end of the job definition script lines correspond to the numbers assigned to the program output data files that are created.

In Windows

```
1.      0000_JOBSAMPL_001_SYS001.sysout
2.      0000_JOBSAMPL_002_SYS002.sysout
3.      0001_STEP01_001_SYS001.sysout
4.      0001_STEP01_002_SYS002.sysout
5.      0000_JOBSAMPL_003_SYS001.sysout
6.      0000_JOBSAMPL_004_SYS002.sysout
7.      0002_STEP02_001_SYS001.sysout
8.      0002_STEP02_002_SYS002.sysout
9.      0000_JOBSAMPL_005_SYS001.sysout
10.     0000_JOBSAMPL_006_SYS002.sysout
```

In UNIX

```
1.      0000_JOBSAMPL_001_SYS001
2.      0000_JOBSAMPL_002_SYS002
3.      0001_STEP01_001_SYS001
4.      0001_STEP01_002_SYS002
5.      0000_JOBSAMPL_003_SYS001
6.      0000_JOBSAMPL_004_SYS002
7.      0002_STEP02_001_SYS001
8.      0002_STEP02_002_SYS002
9.      0000_JOBSAMPL_005_SYS001
10.     0000_JOBSAMPL_006_SYS002
```

5.10 Example coding of a job definition script file

The following shows an example coding of a job definition script file.

```
#!/opt/jp1as/bin/adshexec        # Control of the entire job # -->1.
#-adsh_job SAMPLE_JOB
####################
# If the job step return code is 8 or greater, cancel the job
#-adsh_job_stop 8:
# Temporary file used inside the job
#-adsh_file_temp JOBTEMP
###################
# Job step 1 #                                   -->2.
###################
#-adsh_step_start S1
# Definition of input and output files
#-adsh_file INFILE /files/infile -chk exist
#-adsh_file OUTFILE /files/outfile
#-adsh -chk no -normal keep -abnormal del
# Definition of parameter files
#-adsh_file_temp PARMFILE -id param
cat<<@@@>${PARMFILE}
-in ${INFILE}
-out ${OUTFILE}
-work /tmp
@@@
# User program execution
s1uap ${PARMFILE}
#-adsh_step_error
# Program error handling procedure
recovery_uap ${JTMP}
#-adsh_step_end
###################
# Job step 2 #                                    -->3.
###################
if [[ $ADSH_STEPRC_S1= -eq 0 ]]; then
# Execute only if the preceding job step terminates normally
#-adsh_step_start S2 -onError cont -stepVar PATH
PATH=/s2bin:$PATH
#-adsh_rc_ignore s2uap
echo "s2uap1"
echo "s2uap2 parm1"
#-adsh_step_end
fi
```

The numbers shown at the right in the above example correspond to the numbers in the following explanation:

1. Control of the entire job
   Performs processing such as passing control to job steps and terminating the job.
2. Processing of job step 1
   - Definition of input and output files
   - Definition of parameter files
   - Execution of user program

- Error handling procedure

3. Processing of job step 2
   Performed only if job step 1 terminates normally.

6 Debugging Job Definition Scripts

This chapter explains the debugger functions of JP1/Advanced Shell.

6.1 About the debugger

A debugger is a tool that supports debugging of programs. JP1/Advanced Shell provides a debugger for job definition script files. A GUI is provided for running the debugger in a Windows development environment, and a CUI is provided for running the debugger in a UNIX execution environment. You can debug your job definition script files interactively by using the buttons, menus, and shortcut keys provided by JP1/Advanced Shell Editor with the GUI and by obtaining replies from the debugger by using the debugger's commands with the CUI.

The debugger enables the following:

- Starting the debugger
- Terminating the debugger
- Running job definition scripts
- Terminating job definition scripts
- Stopping execution of job definition scripts
- Restarting execution of job definition scripts
- Displaying information about job definition scripts#
- Specifying and displaying variables
- Displaying back traces#
- Displaying source files
- Moving directories#
- Starting the login shell#
- Injecting errors
- Displaying Help

#

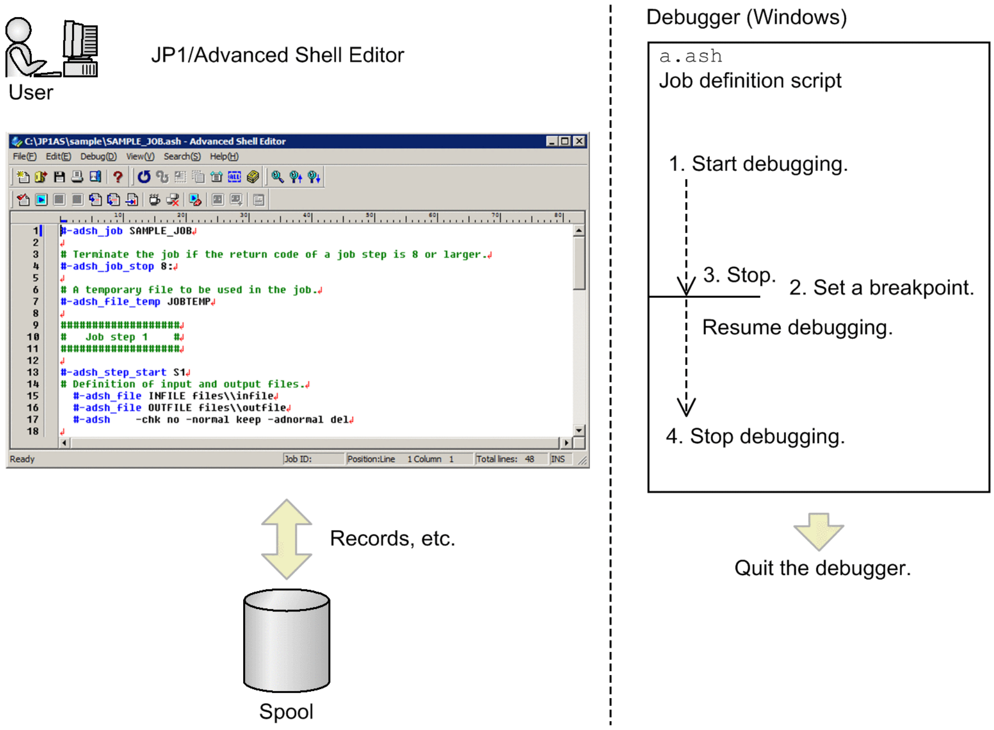
Available in the CUI debugger.

6.1.1 Debugging with the GUI (Windows only)

You can use GUI operations from the JP1/Advanced Shell Editor to debug job definition script files.

The following figure provides an overview of debugging.

Figure 6–1: Overview of debugging (GUI)



```
#-adsh_job SAMPLE_JOB

# Terminate the job if the return code of a job step is 8 or larger.
#-adsh_job_stop 8:

# A temporary file to be used in the job.
#-adsh_file_temp JOBTEMP

###################
#   Job step 1    #
###################

#-adsh_step_start S1
# Definition of input and output files.
  #-adsh_file INFILE files\\infile
  #-adsh_file OUTFILE files\\outfile
  #-adsh    -chk no -normal keep -adnormal del

```

1. Debugging of a job definition script from the editor is started.
2. A breakpoint is set.
3. When the job definition script is run, it stops at the breakpoint.
4. Debugging stops.

(1) Output

To run job definition scripts interactively, JP1/Advanced Shell displays the standard output and standard error output on the console at suitable times during debugging execution, unlike during normal execution where the standard output and standard error output are output after execution has completed. However, error messages are also displayed in the Error List window. For details about the Error List window, see *4.7.4 Error List window*. The files for the standard output and the standard error output are not created in the spool job directory.

During normal execution, job execution logs are output to the standard error output after job definition scripts have been completed. During debugging, information equivalent to the job execution logs is output to the standard error output.

(2) Initialization of information

When a job definition script is run after already having been run once, information about the shell variables and environment variables that were specified during the previous execution is initialized.

(3) Spool

Each time debug execution is performed on a job definition script, JP1/Advanced Shell creates a spool job folder and stores the following files:

- Script image: Contents of the script that was run
- Job execution logs: JP1/Advanced Shell messages

- Output files: Files created by executing the `#-adsh_spoolfile` command
- `sysout` management file (`.sysout`)

(4) Notes

If `[[`*conditional-expression*`]]` is used for conditional, the `E-Time` entry in an execution result message might be the debugger's processing time.

6.1.2 Debugging with the CUI (UNIX only)

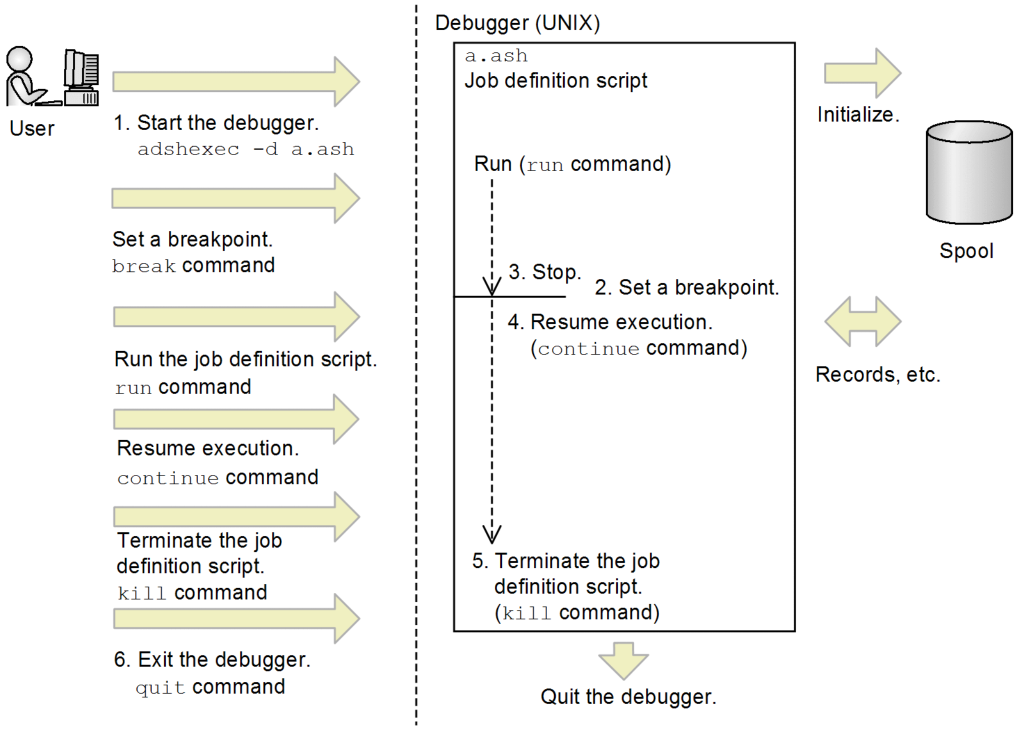
If you execute the `adshexec` command with the `-d` option specified, the job controller starts in the debugger mode, enabling you to use CUI operations to debug job definition scripts. To use commands in the execution environment to debug batch jobs, you enter the `adshexec` command as shown below. Enter the command from the UNIX shell.

```
adshexec -d /script/batchjob2.ash
```

For details about the `adshexec` command, see *adshexec command (executes a batch job)* in *8.3 Shell operation commands*.

The following figure provides an overview of debugging.

Figure 6–2: Overview of debugging (CUI)



1. The `adshexec` command with the `-d` option specified is entered to start the debugger.
2. The `break` command is entered to set a breakpoint.
3. The `run` command is entered to run the job definition script, which stops at the breakpoint.
4. The `continue` command is entered to continue execution from the breakpoint.

5. The `kill` command is entered to terminate the job definition script.
6. The `quit` command is entered to terminate the debugger.

(1) Output

To run job definition scripts interactively, JP1/Advanced Shell displays the standard output and standard error output on the console at suitable times during debugging execution, unlike during normal execution where the standard output and standard error output are output after execution has completed. The files for the standard output and the standard error output are not created in the spool job directory.

During normal execution, job execution logs are output to the standard error output after job definition scripts have been completed. During debugging, information equivalent to the job execution logs is output to the standard error output.

(2) Initialization of information

When a job definition script is run by entering the `run` command after already having been run once from execution of the `run` command, the following information that was specified during the previous execution is initialized:

- Shell variables
- Environment variables
- Fault injection mode

The following information is inherited until the debugger terminates:

- Information about breakpoints and watchpoints
- Debugger's work directory path
- Files#

  #

  Files created by using extended script commands are postprocessed appropriately.

(3) Spool

During CUI debugging, JP1/Advanced Shell creates two types of spool job directories, one type for the debugger and one type for the job definition scripts executed as a result of entering the `run` command. In each debugging execution, JP1/Advanced Shell creates one spool job directory for the debugger and one spool job directory each time the `run` command is executed. This subsection explains the spool job directories for the debugger and for job definition scripts.

(a) Debugger

Job definition scripts are not executed in the spool job directory for the debugger. JP1/Advanced Shell creates a spool job directory that stores in a management file the number of job definition scripts executed during a single debug execution and that stores files containing the debugger's internal data.

The following are the files that JP1/Advanced Shell stores in the spool job directory for the debugger:

- Script images: Contents of the scripts that were run
- Job execution logs: JP1/Advanced Shell messages (including the `pid`s of created processes)
- Breakpoint information (`.DBG`): Debugger internal data
- `sysout` management file (`.sysout`)

(b) Job definition scripts

JP1/Advanced Shell creates a spool job directory each time the `run` command is executed in which it stores the following files:

- Script image: Contents of the script image that was run
- Job execution logs: JP1/Advanced Shell messages
- Output files: Files created by executing the `#-adsh_spoolfile` command
- `sysout` management file (`.sysout`)

(4) Notes

If `[[`*conditional-expression*`]]` is used for conditional or if multiple commands are joined by a pipe, the `E-Time` entry in an execution result message might be the debugger's processing time.

6.1.3 List of functions of the GUI debugger (Windows only)

The following table lists the functions of the GUI debugger and the subsections in this manual to be referenced for details.

Table 6–1: Functions of the GUI debugger

Function	Subsection
Executing job definition scripts	*4.4.6*
Canceling debugging	*4.4.6(2)*
Stopping a job definition script	*4.4.6(2)(d)*
Setting breakpoints	*4.4.6(1)*
Releasing breakpoints	*4.4.6(1)*
Performing sequential execution	*4.4.6(2)(b)*, *4.4.6(2)(c)*
Performing continuous execution	*4.4.6(2)(a)*
Executing functions	*4.4.6(2)(d)*
Specifying and displaying the values of variables	*4.4.6(3)*, *4.7.6*
Displaying the status	*4.3.1(1)*
Changing the fault injection mode	*4.3.1(1)*

6.1.4 List of debugger commands (UNIX only)

The table below lists and describes the debugger commands in alphabetical order. The table also shows the abbreviated form for each command the subsection in this manual to be referenced for details.

Table 6–2: List of debugger commands

Command name	Command execution	Abbreviation	Subsection
`break`	Sets a breakpoint.	`b`	*6.2.4*

Command name	Command execution	Abbreviation	Subsection
cd	Changes the directory.	cd	*6.2.25*
continue	Performs continuous execution.	c	*6.2.9*
delete	Deletes breakpoints and watchpoints.	d	*6.2.6*
exec	Starts the login shell.	ex	*6.2.26*
finish	Executes a function.	f	*6.2.10*
help	Displays Help.	h	*6.2.27*
info breakpoints	Displays information about breakpoints and watchpoints.	i b	*6.2.13*
info coverage	Displays coverage information during debugging.	i c	*6.2.14*
info functions	Displays information about functions.	i f	*6.2.15*
info jobsteps	Displays information about job steps.	i j	*6.2.16*
info signals	Displays information about signals.	i si	*6.2.17*
info status	Displays the status.	i st	*6.2.18*
info variables	Displays information about variables.	i v	*6.2.19*
joberrmode	Enables or disables the fault injection mode.	jem	*6.2.20*
kill	Terminates the job definition script.	k	*6.2.3*
list	Displays the source file.	l	*6.2.24*
next	Performs sequential execution without stopping within a function.	n	*6.2.8*
print	Displays the value of a variable.	p	*6.2.22*
quit	Terminates the debugger.	q	*6.2.1*
return	Terminates a function.	ret	*6.2.11*
run	Executes the job definition script.	r	*6.2.2*
set	Specifies a value for a variable.	set	*6.2.21*
signal	Sends a signal.	si	*6.2.12*
step	Performs sequential execution, including sequential execution within a function.	s	*6.2.8*
watch	Sets a watchpoint.	wa	*6.2.5*
where	Displays a backtrace.	whe	*6.2.23*

The following notes apply to using debugger commands:

- Do not enter for a command any command name or command abbreviation that is not listed in the table above.
- In the case of a command that allows argument values to be specified, check that the number of argument values specified does not exceed the permitted maximum.
- Do not specify an argument in a command that does not allow arguments.
- The maximum number of characters that can be entered to the debugger's standard input is 4,094 bytes. Make sure that the number characters entered does not exceed this limit.

6.1.5 Whether execution can be stopped at the elements of a job definition script

The table below shows whether execution can be stopped at each element of a job definition script. For the elements listed below at which execution can be stopped, you use a breakpoint or sequential execution to stop execution.

Table 6–3: Whether execution can be stopped at each element of a job definition script

Element of a job definition script	Description	Whether stoppable
`for` statement	Indicates a `for` statement.	Y
`case` statement	Indicates a `case` statement.	Y
`if` statement	Indicates an `if` statement.	Y
`while` statement	Indicates a `while` statement.	Y
`until` statement	Indicates an `until` statement.	Y
`elif` statement	Indicates an `elif` statement.	Y
`else` statement	Indicates an `else` statement.	N
`case` pattern statement	Indicates a pattern statement specified in a `case` statement. This does not apply to the internal processing of the pattern statement.	N
Termination of a conditional statement	Indicates the termination of a conditional statement, such as `then` for the `if` and `elif` statements or `do` for the `for`, `while`, and `until` statements.	N
Termination of a block	Indicates the termination of a block, such as `done` for the `for`, `while`, and `until` statements, `esac` for the `case` statement, or `fi` for the `if` statement.	N
Start of function definition	Indicates the start of a function definition.	N
End of function definition	Indicates the end of a function definition.	N
Start of function execution	Indicates that a function begins.	Y
End of function execution	Indicates that a function ends.	N
Extended script command	Indicates an extended script command beginning with `#-adsh`.	Y
Standard shell command	Indicates a command provided by the shell.	Y
Extended shell command	Indicates an extended shell command.	Y
External command	Indicates an executable external command.	Y
Assignment and arithmetic operations	Indicates an assignment or arithmetic operation.	Y
End of job definition script (EOF)	Indicates the end of the main job definition script.	Y
Comment and space	Indicates a comment or space.	N

Legend:

Y: Execution can be stopped.

N: Execution cannot be stopped.

Notes:

- Execution is not stopped at a command that is executed by another process.

  Example:

```
1: funcA(){
2:  a=100
3:  echo $a
4: }
5: funcA &
6: pwd
```

In this example job definition script, if the `step` command is executed while execution is stopped before `funcA &` at line 5, execution stops before `pwd` at line 6. Execution does not stop within the `funcA` function because its internal commands are executed by another process according to the specification of `&`.

- Execution is not stopped at a command specified in an argument of a shell command.
- In Windows, if you specify a command substitution in the argument of a shell command, you can stop execution once before the argument part is executed and again before the actual command is executed. You can also stop execution twice in this manner when you execute a command substitution as a command, rather than as an argument.

  In UNIX, if you specify a command substitution in the argument of a shell command, you can stop execution once before the entire command is executed. The following shows examples.

  Examples:

  In Windows:

```
  echo `pwd` ← Execution can be stopped once before `pwd` is executed
and once before echo executed.
  `echo pwd` ← Execution can be stopped twice before `echo pwd` is
executed.
```

  In UNIX:

```
  echo `pwd` ← Execution can be stopped once before echo `pwd` is
executed.
  `echo pwd` ← Execution can be stopped once before `echo pwd` is
executed.
```

- If you specify an extended script command onto a continuation line, you can stop execution on each line. Note that the extended script command itself executes when the last continuation line is executed.
- If you specify a command after an assignment operator delimited by a space, use the debugger's `set` command to set values in variables when execution is stopped at the first assignment operator. When execution is stopped at each command, the character string obtained after variables have been expanded is displayed as the command name.
- A breakpoint cannot be set at the end of a job definition script (EOF). Note that execution can be stopped at the EOF by using a watchpoint or in sequential execution, as well as by receiving a stop signal.
- Execution will not stop at a breakpoint while `action` of the `trap` shell command is running. In UNIX, note that execution can be stopped in such a case by using a watchpoint or in sequential execution, as well as by receiving a stop signal.
- If the `time` command is executed with a function call specified in its argument and the stop evaluation condition is satisfied by a watchpoint or by receipt of a stop signal, execution stops before the first command following the `time` command that allows execution to be stopped.
- If commands are grouped (executed in child processes), execution can be stopped at the location of the right parenthesis `()`). The command group will not execute while execution is stopped at the right parenthesis. You set the breakpoint at the lines that contains the right parenthesis. If you attempt to set a breakpoint at a line that does not contain the right parenthesis, the breakpoint is set automatically at the line where the right parenthesis is located. The following shows an example.

  Example:

```
1: (          ← Cannot be stopped.
2: pwd        ← Cannot be stopped.
3: date       ← Cannot be stopped.
4: )          ← Can be stopped.
```

- If you group multiple commands by enclosing them in curly brackets ({}) and you specify their execution in separate processes by means such as adding an ampersand (&), specify the grouping on a single line in order to set a breakpoint. If the grouping is specified over multiple lines, execution cannot be stopped at breakpoints that might be set. The following shows an example.

  Example:

  Specifying on a single line

  ```
  1: { pwd; date;} &
  ```

  Execution cannot be stopped at the set breakpoints

  ```
  1: echo "test"
  2: {          ← Cannot be set or execution cannot be stopped.
  3: pwd        ← Can be set, but execution cannot be stopped.
  4: date       ← Can be set, but execution cannot be stopped.
  5: } &        ← Cannot be set, but execution can be stopped.
  ```

  In the above example, breakpoints can be set on lines 3 and 4, but execution cannot be stopped at those locations. Execution can actually be stopped on line 5. For example, if sequential execution is performed from the command immediately preceding the group of commands (in this example, `echo` on line 1), execution can be stopped on line 5. When execution is stopped at the location of the right curly bracket, the group of commands will not have executed yet.

- If only a command substitution is specified as a command, execution can be stopped at the line containing the termination symbol for the command substitution. When execution is stopped at the location of the termination symbol, the command substitution will not have executed yet. You set a breakpoint at the lines that contains the termination symbol. If you attempt to set a breakpoint at a line that does not contain the termination symbol, the breakpoint is set automatically at the line where the termination symbol is located. The following shows examples.

  Example 1:

  ```
  1: $(          ← Cannot be stopped.
  2: echo pwd    ← Cannot be stopped.
  3: )           ← Can be stopped.
  ```

  Example 2:

  ```
  1: `echo pwd`  ← Can be stopped.
  ```

- If you specify only a command substitution as a command or in the argument of the `builtin`, `command`, or `time` command and you want to stop execution before execution of the command immediately following the command substitution, make sure that the result of the command substitution is not `NULL`, spaces, or a comment.

- If a group of commands connected by pipes is stopped while executing or a command running in the background is stopped, multiple consecutive prompt character strings `(adshdb)` might be displayed.

6.2 CUI debugger (UNIX only)

In a UNIX execution environment, you use commands to run the debugger. The following shows the command specification format for the CUI debugger:

```
Δ0-command-name[Δ1-option]...[Δ1-option][Δ1-operand]
```

- First specify options, and then specify operands. Operands includes option names, option values, and any arguments that can be specified in the command. If an operand is specified before an option, the command assumes that all items specified are operands.
- Specify an option in the format -*option-name*[Δ1-*value*]. Multiple options can be specified in any order.
- Options with no value can be specified consecutively (example: `-a -b -c` and `-abc` are treated as being the same). If you specify options consecutively, you can specify a value for the last option (example: in `-abc xyz`, `xyz` is the value of the `-c` option).
- If an invalid option is specified or a specified value is outside the permitted range of values, an error results.

Starting the debugger

You start the debugger by specifying the `-d` option and the path name of a job definition script file in the command for executing batch jobs (`adshexec` command).

The started debugger outputs a prompt character string (`adshdb`) and then goes onto input wait status. When the debugger accepts a command entered by the user, it executes that command's processing. When the command's processing terminates, the debugger outputs the prompt character string and again goes onto input wait status. The debugger repeats this process until the debugger is terminated.

The debugger also outputs a prompt character string `(adshdb)` when it goes back into input wait status after receiving a signal while in input wait status.

The format used to start the debugger is shown below. For details about the command for executing batch jobs, see *adshexec command (executes a batch job)* in *8.3 Shell operation commands*.

```
adshexec -d path-name-of-job-definition-script-file
```

Pausing a job definition script

You can pause a job definition script by entering the **Ctrl** + **C** keys. This method of stopping a job definition script is useful when the job definition script has entered an infinite loop.

Notes:

- If the **Ctrl** + **C** keys are used during execution of an extended script command, extended shell command, standard shell command, or reserved script command, execution stops before the next stoppable instruction after execution of the current command has been completed.
- If the **Ctrl** + **C** keys are used while an external command is executing in the foreground, execution depends on the external command's processing.

6.2.1 Terminating the debugger (quit command)

The `quit` command terminates the debugger. The abbreviation for the `quit` command is `q`. The following shows the format of the `quit` command:

```
quit
```

The following describes the `quit` command's processing.

When no argument is specified in the quit command

When the `quit` command is executed while the job definition script is running, a confirmation message is output asking whether the debugger is to be terminated. To terminate the debugger, enter `y` or `Y`.

If no job definition script is running, the command terminates the debugger.

When an argument is specified in the quit command

Executing the command results in an error.

Notes:

A job definition script terminated by the `quit` command results in termination with an error. The return code for the job and job step is `128`.

6.2.2 Running the job definition script (run command)

The `run` command executes the job definition script to be debugged. The `run` command imports environment variables when the `adshexec` command starts, and then starts running the job definition script. The abbreviation for the `run` command is `r`. The following shows the format of the `run` command:

```
run[ arguments]
```

When the `run` command is executed while the job definition script is running, a confirmation message is output asking whether the job definition script is to be re-executed.

When the run command argument is omitted

Enter `y` or `Y` to re-execute the executing job definition script.

If the job definition script is not running, the command outputs an execution start message and then executes the job definition script.

When the run command argument is specified

Enter `y` or `Y` to define the specified arguments as run-time parameters and re-execute the executing job definition script.

If job definition script is not running, the command outputs an execution start message, defines the specified arguments as run-time parameters, and then executes the job definition script.

Notes:

- When the job definition script is re-executed by the `run` command, the instance of the job definition script that was already running terminates with an error. The return code for the job and job step is `128`.

6.2.3 Terminating the job definition script (kill command)

The `kill` command terminates the job definition script that is being debugged. The debugger itself does not terminate when the job definition script is terminated. The abbreviation for the `kill` command is `k`. The following shows the format of the `kill` command:

```
kill
```

When the `kill` command is executed while the job definition script is running, a confirmation message is output asking whether the job definition script is to be terminated. Enter `y` or `Y` to terminate the job definition script.

If the job definition script is not running or if an argument is specified in the `kill` command, an error message is output.

Notes:

The job definition script terminated by the `kill` command results in termination with an error. The return code for the job and job step is `128`.

6.2.4 Setting a breakpoint (break command)

The `break` command sets a breakpoint. The abbreviation for the `break` command is `b`.

Numbers are assigned sequentially to the breakpoints as they are set, beginning with `1`. Watchpoints that are set are numbered in the same sequence (that is, point number *n* could be a breakpoint or a watchpoint). When a running job definition script reaches a breakpoint, the command displays information about the breakpoint at the stop location. The following shows the format of the `break` command:

Specifying a line number

```
break[ [job-definition-script-file-name:]line-number]
```

When a line number is specified in the argument, the `break` command sets a breakpoint at the specified line.

Specifying a function name

```
break function-name
```

When a function name is specified in the argument, the `break` command sets a breakpoint at the specified function.

Specifying a job step name

```
break -s [job-definition-script-file-name:]job-step-name
```

When a job step name is specified in the `-s` option, the command sets a breakpoint at the line on which the specified job step name is defined.

By using a colon (`:`) in the argument, you can specify a job definition script file name for setting a breakpoint. If you specify a function name, there is no need to specify a file name because only one function that is enabled at that point becomes the target.

The following describes the `break` command's processing.

When the break command argument is omitted

The command sets a breakpoint at the line at which execution is currently stopped. If the job definition script is not running, the command sets a breakpoint at the first line after line 1 at which execution can stop. The command displays information about the set breakpoint.

When the break command argument is specified

The command processing depends on the specification of the argument. The following provides the details.

- *line-number*

The command sets a breakpoint at the line specified in the argument. The command displays information about the set breakpoint. If the specified line does not exist, an error results.

- *function-name*

  The command sets a breakpoint at the function specified in the argument. The command displays information about the set breakpoint. The location of the breakpoint is the first line in the function definition at which execution can stop. If the specified function does not exist, an error results.

- `-s` *job-step-name*

  The command sets a breakpoint at the job step specified in the argument. The command displays information about the set breakpoint. If the specified job step does not exist, an error results.

- *job-definition-script-file-name*

  The file name you specify must be the name of a file specified in the command used to execute the batch job or of a file specified in the `#-adsh_script` extended script command. The command treats the specified file as the target for setting a breakpoint. If no file name is specified, the command assumes that the current file is the target for setting a breakpoint.

Notes:

- The character string up to the last colon (`:`) is treated as a job definition script file name.
- A line number must be specified as `0` or a greater integer. Do not specify a plus sign (`+`) at the beginning.
- If the specified line number is outside the permitted range for the `int` type, the command rounds it to the maximum value for the `int` type.
- If there are multiple job steps with the same name, the command sets a breakpoint at all those job steps.
- The total number of breakpoints and watchpoints combined cannot exceed 999. If this limit has been reached and you want to set a new breakpoint or watchpoint, terminate and restart the debugger. Once the maximum value has been reached, no new breakpoint or watchpoint can be set even if you use the `delete` command to delete existing breakpoints or watchpoints.
- You can set a breakpoint at any line that contains at least one command at which execution can stop. If a line contains multiple commands at which execution can stop, execution stops for each command (before each command is executed).
- If the line specified by its line number contains only commands at which execution cannot be stopped, the `break` command outputs a warning message and sets a breakpoint at the next line that contains a command at which execution can stop.

  If there is no such line at which execution can stop, the command outputs an error message.
- If a function name is specified but the specified function contains no command at which execution can stop, the command sets a breakpoint at the first line following the end of the function definition at which execution can stop. If the line containing the function definition also contains a command at which execution can stop, the `break` command sets a breakpoint at the line containing the function definition, regardless of whether that command is inside the function.
- If an extended script command defining a job step consists of multiple lines and the job step name is specified for setting a breakpoint, the `break` command sets a breakpoint at the line number where the first extended script command is specified.
- You can set a breakpoint in an external script only if the external script is specified in the `#-adsh_script` extended script command in the job definition script that calls the external script. Execution cannot be stopped within an external script that is not specified in `#-adsh_script`.
- Only one breakpoint can be set on any one line. Once a breakpoint is set, no more breakpoints can be set on the same line.

- If the job definition script that is under debug execution is in one of the following statuses, do not execute the `break` command without an argument specified:
  - Execution is stopped at the end of the job definition script (EOF)
  - Execution is stopped while `action` of the `trap` command is executing

Example

This example sets a breakpoint on line 8 in the job definition script and then executes the job definition script. Execution stops before `funcA` on line 8 is executed.

```
1: funcA(){
2:   echo "funcA"
3:   num=10
4: }
5:
6: val=1
7: num=2
8: funcA
9: echo $num

KNAX7018-I Breakpoint "1": filename="test.ash" line=8
(adshdb) run
KNAX7007-I Execution of the following script will now start: /home/test/
test.ash
...
KNAX7018-I Breakpoint "1": filename="test.ash" line=8
KNAX7032-I The script "test.ash" stopped running.
8: funcA
Current: funcA
(adshdb)
```

6.2.5 Setting a watchpoint (watch command)

The `watch` command sets a watchpoint. The abbreviation for the `watch` command is `wa`.

Numbers are assigned sequentially to the watchpoints as they are set, beginning with `1`. Breakpoints that are set are numbered in the same sequence (that is, point number *n* could be a watchpoint or a breakpoint). The following shows the format of the `watch` command:

```
watch variable-name
```

You specify a variable name as the argument of the `watch` command. Whenever the value of the specified variable is updated, execution of the job definition script stops at the next command where execution can be stopped and watchpoint information is displayed.

The following describes the `watch` command's processing.

When the watch command argument is specified

The command sets a watchpoint at the specified variable. The command also displays information about the set watchpoint.

When the watch command argument is omitted

The command outputs an error message.

When execution of a job definition script is stopped by a watchpoint, the `watch` command displays as the watchpoint information the pre-update value, the post-update value, and the line number of the line that updated the variable. The following shows the display format:

```
Old value = pre-update-value
New value = post-update-value
Line = line-number
```

- *pre-update-value*: This is the watched variable's value before updating. If there is no value, `<No value>` is displayed.
- *post-update-value*: This is the watched variable's value after updating. If there is no value, `<No value>` is displayed.
- *line-number*: This is the line number of the line that updated the variable. If the `trap` command's `action` is running, `<Trap action>` is displayed. If the line is the end of the job definition script, `<EOF>` is displayed.

Notes:

- To specify an array for the variable name, you must specify an array element.

  Examples:

  Normal variable specification: `aaa`

  Array specification: `aaa[1]`
- A variable and its array 0 (example: `aaa` and `aaa[0]`) are the same. However, you can set a watchpoint for each of them.
- A variable name with a dollar sign (`$`) attached is not recognized as a variable name.
- The command does not check at the time the `watch` command is entered whether the specified variable actually exists.
- If the variable name specification violates a naming convention, the command outputs an error message.
- Execution stops at the watchpoint even if the new value assigned to the variable is the same as its pre-update value.
- When the `typeset` standard shell command is used to change the value of the variable from character string type to integer type, or vice versa, execution stops at the location as a watchpoint.
- Because the same naming conventions apply to both functions and variables, you can set a watchpoint by specifying a function name as the argument. However, execution of the job definition script will not stop unless the value of the variable whose name is the same as the specified function name is updated.
- If one line contains multiple commands and the value of a watched variable is updated, execution stops before the next command at which execution can be stopped even on the same line.
- The total number of breakpoints and watchpoints combined cannot exceed 999. If this limit has been reached and you want to set a new breakpoint or watchpoint, terminate and restart the debugger. Once the maximum value has been reached, no new breakpoint or watchpoint can be set even if you use the `delete` command to delete existing breakpoints or watchpoints.
- If the `set` command is used to change the value of a watched variable while the job definition script is stopped and then execution of the job definition script is restarted, execution will stop before the next command at which execution can be stopped.
- Only one watchpoint can be set for the same variable. Once a watchpoint is set for a variable, no more watchpoints can be set for that variable.
- Execution of a job definition script does not stop when the value of a watched variable is changed by any of the following commands:

- Commands executed in the background (`&` or `|&`)
- Commands joined by a pipe
- Group of commands enclosed in parentheses (`()`)
- Commands that are executed as separate processes, such as external commands

Example:

```
1: a=1 &
2: b=2
3: c=3
```

In the above example, if a watchpoint is set for variable `a` and then the job definition script is executed, execution of the job definition script does not stop because the assignment expression `a=1` is executed in the background.

Example

In the following job definition script, specifying `watch b` will set a watchpoint for variable `b`:

```
1: echo "start"
2: a=1
3: b=5
4: c=10
5: echo "end"
```

When the job definition script is executed and then the assignment expression `b=5` is executed, the watchpoint information will be displayed and execution will stop before execution of `c=10` on line 4:

```
KNAX7023-I Watchpoint "1": variable="b"
Old value = <No value>
New value = 5
Line = 3
KNAX7032-I The script "test.ash" stopped running.
4: c=10
Current: c=10
(adshdb)
```

6.2.6 Deleting breakpoints and watchpoints (delete command)

The `delete` command deletes breakpoints and watchpoints. The abbreviation for the `delete` command is `d`. The following shows the format of the `delete` command:

```
delete[ breakpoint-or-watchpoint-number[-breakpoint-or-watchpoint-number]]
```

You can delete any desired breakpoint or watchpoint by specifying its number in the argument of the `delete` command. You can use the hyphen (`-`) to specify a range of numbers. For example, to delete point numbers 1 through 5, specify `1-5`. If the argument is omitted, the command deletes all breakpoints and watchpoints.

The following describes the `delete` command processing.

When the delete command argument is omitted

If at least one breakpoint or watchpoint has been set, the command displays a confirmation message asking whether all breakpoints and watchpoints are to be deleted. To delete all breakpoints and watchpoints, enter `y` or `Y`.

If no breakpoint or watchpoint has been set, the command outputs an error message.

When the delete command argument is specified

- *breakpoint-or-watchpoint-number*

  The command deletes the breakpoint or watchpoint with the specified number. If the specified number does not exist, the command outputs an error message.

- *number*−*number*

  The command deletes the breakpoints and watchpoints whose numbers are in the range from the value specified to the left of the hyphen (`-`) to the value specified to the right of the hyphen. If the specified range contains no breakpoints or watchpoints, the command outputs an error message.

- Other

  The command outputs an error message.

Notes:

- A number must be specified in the argument as `0` or a greater integer. Do not specify a plus sign (`+`) at the beginning.
- When a range of numbers is specified and the beginning number is the same as the ending number, the command deletes only the breakpoint or watchpoint with the specified number.
- When a range of numbers is specified and the ending number is smaller than the beginning number, the command outputs an error message.
- If a specified number is outside the permitted value range for the `int` type, the command rounds it to the maximum value for the `int` type.

6.2.7 Commands for restarting execution of the job definition script

There are three different ways to restart execution of the job definition script:

- Sequential execution

  This method executes one command from the location at which the job definition script is stopped. The `step` and `next` commands are used to perform sequential execution.

- Continuous execution

  This method resumes execution of a job definition script that is stopped. The `continue` command is used to perform continuous execution.

- Executing a function

  When execution of a job definition script is stopped within a function, this method executes the job definition script until the function returns control. The `finish` command is used to execute a function.

The `return` command is used to terminate a function. The `signal` command is used to send a signal to the job definition script.

When execution of the job definition script stops after execution of a command, a message, the line number of the next line that is scheduled to be executed, and the line in the source file are displayed in one of the formats shown in the following.

For a job definition script that is specified in the command for executing a batch job or a job definition script that is specified in the #-adsh_script extended script command

```
line-number: line-contents-in-source-file
Current: command-string
```

- *line-number*: Line number of the next command to be executed
- *line-contents-in-source-file*: Contents of the line in the source file that correspond to the line number
- *command-string*: Next command string to be executed

For an external script that is not specified in the #-adsh_script extended script command

```
Line: line-number
Current: command-string
```

- *line-number*: Line number of the next command to be executed
- *command-string*: Command string of the next command to be executed

Notes:

- In the case of a job definition script that is executed in another process, `<Another process script>` is displayed as the command string.
- If the job definition script being debugged is executing the `trap` command's `action`, the following is displayed:

```
   Line: <Trap action>
   Current: command-string
```

- If the end-of-job definition script (EOF) has been reached, the following is displayed:

```
   Current: <EOF>
```

Example of output

This example displays the next line number to be executed and the contents of the line in the source file.

- **For a job definition script that is specified in the command for executing a batch job or a job definition script that is specified in the #-adsh_script extended script command**

  `100: echo "aaa"` ← *The next process to be performed is echo "aaa" on line 100.*

  `Current: echo` ← *The command to be executed then is echo.*

- **For an external script that is not specified in the #-adsh_script extended script command**

  `Line: 50` ← *The next process to be performed is line 50 in the external script.*

  `Current: num=1` ← *The process to be performed then is num=1.*

6.2.8 Performing sequential execution (step and next commands)

The `step` and `next` commands are used to execute the first command from the location at which execution of the job definition script is stopped. If a function is called by the command where execution has stopped, the `step` command enters the function (performs sequential execution within the function), while the `next` command executes the processing without stopping inside the function. The abbreviations for the `step` and `next` commands are `s` and `n`, respectively.

(1) step command

The following shows the format of the `step` command:

```
step
```

The following describes the step command's processing.

When no argument is specified in the step command

When the step command is executed while execution of the job definition script is stopped, the step command executes the first command from the location where execution of the job definition script has stopped. If a function is called, the step command enters the function (and performs sequential execution within the function).

If the job definition script is not running, the command outputs an error message.

When an argument is specified in the step command

Executing the command results in an error.

Notes:

If a function call is specified in the argument of the eval standard shell command and sequential execution is performed on the eval command, the stop location after execution of the function call depends on the step command's processing.

Example

If the step command is executed while execution is stopped before val=1 on line 6, execution stops before num=2 on line 7.

```
1: funcA(){
2:   echo "funcA"
3:   num=10
4: }
5:
6: val=1
7: num=2
8: funcA
9: echo $num

6: val=1
Current: val=1
(adshdb) step
...
KNAX7032-I The script "test.ash" stopped running.
7: num=2
Current: num=2
(adshdb)
```

If the step command is executed while execution is stopped before funcA on line 8, execution stops before echo on line 2.

```
8: funcA
Current: funcA
(adshdb) step
KNAX7032-I The script "test.ash" stopped running.
2:   echo "funcA"
Current: echo
(adshdb)
```

(2) next command

The following shows the format of the next command:

```
next
```

The following describes the `next` command's processing.

When no argument is specified in the next command

When the `next` command is executed while execution of the job definition script is stopped, the `next` command executes the first command from the location where execution of the job definition script has stopped. If a function is called, the `next` command does not stop execution inside the function.

If the job definition script is not running, the command outputs an error message.

When an argument is specified in the next command

Executing the command results in an error.

Notes:

- When the `next` command is executed and a function is called by the subsequent command, execution of the job definition script is stopped if a stop evaluation condition is satisfied by a breakpoint, a watchpoint, or a signal within that function.
- If a function call is specified in the argument of the `eval` standard shell command and sequential execution is performed on the `eval` command, the stop location after execution of the function call depends on the `next` command's processing.

Example

If the `next` command is executed while execution is stopped before `val=1` on line 6, execution stops before `num=2` on line 7.

```
1: funcA(){
2:   echo "funcA"
3:   num=10
4: }
5:
6: val=1
7: num=2
8: funcA
9: echo $num

6: val=1
Current: val=1
(adshdb) next
...
KNAX7032-I The script "test.ash" stopped running.
7: num=2
Current: num=2
(adshdb)
```

If the `next` command is executed while execution is stopped before `funcA` on line 8, execution stops before `echo` on line 9.

```
8: funcA
Current: funcA
(adshdb) next
funcA
...
KNAX7032-I The script "test.ash" stopped running.
9: echo $num
Current: echo
(adshdb)
```

6.2.9 Performing continuous execution (continue command)

The `continue` command continues (resumes) execution of the job definition script from where it is stopped. The abbreviation for the `continue` command is `c`. The following shows the format of the `continue` command:

```
continue
```

The following describes the `continue` command's processing.

When no argument is specified in the continue command

When the `continue` command is executed while execution of the job definition script is stopped, the command outputs a message indicating resumption of execution and restarts execution of the job definition script.

If the job definition script is not running, the command outputs an error message.

When an argument is specified in the continue command

Executing the command results in an error.

6.2.10 Executing a function (finish command)

The `finish` command executes the job definition script until control is returned from a function. The abbreviation for the `finish` command is `f`. The following shows the format of the `finish` command:

```
finish
```

The following describes the `finish` command's processing.

When no argument is specified in the finish command

When execution is stopped within a function, this command outputs a message indicating that execution will resume through the end of the current function, and then resumes execution of the job definition script through the end of the function. When execution of the job definition script stops, the command displays the frame information for the stop location, and then displays the next line number scheduled to be executed and the line in the source file. The following shows the display format for the frame information at the stop location.

Frame information at the stop location

```
Num  Function  File:Line
frame-number  function-name  file-name:line-number
```

- *frame-number*: Indicates the frame number (`0` is always displayed).
- *function-name*: Indicates the name of the function corresponding to the frame information to which the name of the job definition script calling the function is attached. If no function has been called, `<main>` is displayed as the function name. A maximum of 63 bytes can be displayed.
- *file-name*: Indicates the name of the file in which execution is currently stopped.
- *line-number*: Indicates the line number at which execution is currently stopped.

  If execution is stopped at the end of the job definition script, `<EOF>` is displayed. If execution is stopped while the `trap` command's `action` is executing, `<Trap action>` is displayed.

If execution is stopped without having entered a function or if the job definition script is not running, the command outputs an error message.

When an argument is specified in the finish command

Executing the command results in an error.

Notes:

If a stop evaluation condition is satisfied by a breakpoint, a watchpoint, or a signal at a subsequent line in the function, the command stops execution of the job definition script.

Example

If the `finish` command is executed while execution is stopped before `echo` on line 2, the command stops execution before `echo` on line 9 and displays the frame information.

```
1: funcA(){
2:   echo "funcA"
3:   num=10
4: }
5:
6: val=1
7: num=2
8: funcA
9: echo $num

2:   echo "funcA"
Current: echo
(adshdb) finish
KNAX7036-I Execution will continue until the end of the current function.
...
Num  Function  File:Line
0    <main>    test.ash:9
KNAX7032-I The script "test.ash" stopped running.
9: echo $num
Current: echo
(adshdb)
```

6.2.11 Terminating a function (return command)

The `return` command terminates commands. The `return` command returns control to the source of a function call without executing the lines following the current location in the function. The abbreviation for the `return` command is `ret`. The following shows the format of the `return` command:

```
return
```

The following describes the `return` command's processing.

When no argument is specified in the return command

When execution is stopped within a function, the command outputs a confirmation message asking whether the current function is to be terminated. To terminate the current function and return to the source of the function call, enter `y` or `Y`. The command displays the frame information for the source of the function call, the next line number scheduled to be executed, and the line in the source file. The following shows the display format for the frame information for the source of the function call.

Frame information for the source of the function call

```
Num  Function  File:Line
frame-number  function-name  file-name:line-number
```

- *frame-number*: Indicates the frame number (`0` is always displayed).
- *function-name*: Indicates the name of the function corresponding to the frame information to which the name of the job definition script calling the function is attached. If no function has been called, `<main>` is displayed as the function name. A maximum of 6 bytes can be displayed.
- *file-name*: Indicates the name of the file in which execution is currently stopped.
- *line-number*: Indicates the line number at which execution is currently stopped.

  If execution is stopped at the end of the job definition script, `<EOF>` is displayed. If execution is stopped while the `trap` command's `action` is executing, `<Trap action>` is displayed.

If execution is stopped without having entered a function or if the job definition script is not running, the command outputs an error message.

When an argument is specified in the return command

Executing the command results in an error.

Notes:

The command terminates the function without stopping execution of the job definition script even if a stop evaluation condition is satisfied by a breakpoint, a watchpoint, or a signal on a subsequent line in the function.

Example

If the `return` command is executed while execution is stopped before `echo` on line 2, the command stops execution before `echo` on line 9 and displays the frame information. The command skips `num=10` on line 3.

```
1: funcA(){
2:   echo "funcA"
3:   num=10
4: }
5:
6: val=1
7: num=2
8: funcA
9: echo $num

2:   echo "funcA"
Current: echo
(adshdb) return
KNAX7037-I Are you sure you want to exit the current function? (y or n)
y
KNAX7068-I Commands will be skipped until the end of the function.
...
Num  Function  File:Line
0    <main>    test.ash:9
KNAX7032-I The script "test.ash" stopped running.
9: echo $num
Current: echo
(adshdb)
```

6.2.12 Sending a signal (signal command)

The `signal` command sends a signal to the job definition script. The abbreviation for the `signal` command is `si`. The following shows the format of the `signal` command:

```
signal {signal-name|signal-number}
```

When a signal name or a signal number is specified in the argument, the `signal` command sends the corresponding signal and then executes the job definition script continuously. You use the `info signals` command to display information about the signals that can be specified in the argument. For details about the processing when a signal is received, see *3.10.2 Processing when signals are received (UNIX only)*.

The following describes the `signal` command's processing.

When the signal command argument is specified

When the `signal` command is executed while the job definition script is running, the following occurs:

Signal number or signal name

The command outputs a message indicating that the specified signal will be sent, sends the specified signal to the job definition script, and then executes the job definition script continuously.

If the specified signal does not exist or the job definition script is not running, the command outputs an error message.

When the signal command argument is omitted

When the `signal` command is executed while the running job definition script is stopped, an error message is output.

The command also outputs an error message if the job definition script is not running.

Notes:

- A signal number that is specified in the argument must be `0` or a greater integer. Do not specify a plus sign (+) at the beginning.
- If the specified number is outside the permitted value range for the `int` type, the command rounds it up to the maximum value for the `int` type.
- To send any of the following signals in AIX, specify the signal number or another signal name with the same signal number:

  To send `SIGLOST` or `SIGIOT`: Specify signal number `6` or `SIGABRT`.

  To send `SIGPOLL`: Specify signal number `23` or `SIGIO`.
- To send any of the following signals in HP-UX, specify the signal number or another signal name with the same signal number:

  To send `SIGIOT`: Specify signal number `6` or `SIGABRT`.

  To send `SIGPOLL`: Specify signal number `22` or `SIGIO`.
- To send any of the following signals in Solaris, specify the signal number or another signal name with the same signal number:

  To send `SIGIOT`: Specify signal number `6` or `SIGABRT`.

  To send `SIGPOLL`: Specify signal number `22` or `SIGIO`.

6.2.13 Displaying breakpoint and watchpoint information (info breakpoints command)

The `info breakpoints` command displays information about the breakpoints and watchpoints that have been set. The abbreviation for the `info breakpoints` command is `i b`. The following shows the format of the `info breakpoints` command:

```
info breakpoints[ breakpoint-or-watchpoint-number]
```

The `info breakpoints` command with a breakpoint or watchpoint number specified in the argument displays information about the specified breakpoint or watchpoint. If the argument is omitted, the command displays information about all breakpoints and watchpoints. The following shows the display format:

```
Num  Type        What
number  breakpoint/watchpoint  file-name:line-number/variable-name
...
```

- *number*: Sequence number of the breakpoint or watchpoint. The command displays a maximum of three digits left-justified.
- *file-name*: File name of the job definition script.
- *line-number*: For a breakpoint, the line number at which the breakpoint is set.
- *variable-name*: For a watchpoint, the variable name specified in the `watch` command.

The following describes the `info breakpoints` command's processing.

When the info breakpoints command argument is omitted

If at least one breakpoint or watchpoint has been set, the command displays information about all breakpoints and watchpoints.

If no breakpoint or watchpoint has been set, the command outputs a message.

When the info breakpoints command argument is specified

- *breakpoint-or-watchpoint-number*

  If the specified number exists, the command displays information about the breakpoint or watchpoint corresponding to the specified number.

  If the specified number does not exist, the command outputs an error message.

- Other

  The command outputs an error message.

Notes:

- A number that is specified in the argument must be `0` or a greater integer. Do not specify a plus sign (+) at the beginning.
- If the specified number is outside the permitted value range for the `int` type, the command rounds it to the maximum value for the `int` type.

Example of output

This example displays a breakpoint and a watchpoint.

```
Num  Type         What
1    breakpoint   sample.ash:100
2    watchpoint   rc
```

6.2.14 Displaying coverage information (info coverage command)

The `info coverage` command displays coverage information during debugging. The abbreviation for the `info coverage` command is `i c`. The following shows the format of the `info coverage` command:

```
info coverage[ n1[-[n2]]
                [,n3[-[n4]]]...]
```

Specify line numbers in the arguments *n1*, *n2*, *n3*, *n4*, and so on. The command displays coverage information for the range of lines specified in the arguments. If the argument is not specified, the command displays all coverage information.

Example

This example displays coverage information for lines 1 through 10, 15, and 21 through the end.

```
info coverage 1-10,15,21-
```

For details about the display of coverage information, see *3.9 Acquiring coverage information*.

6.2.15 Displaying function information (info functions command)

The `info functions` command displays information about functions. The abbreviation for the `info functions` command is `i f`. The following shows the format of the `info functions` command:

```
info functions[ function-name]
```

When a function name is specified in the argument, the command displays that function name and the corresponding file name and line number. If the argument is not specified, the command displays all function names and their file names and line numbers. The following shows the display format:

```
Function  File:Line
function-name  file-name:line-number
...
```

- *function-name*: Name of a defined function. When all functions are displayed, this information is displayed in ASCII code order of the function names. A maximum of 31 bytes can be displayed for each function name. If a function name consists of more than 31 byes, only the first 31 bytes are displayed. The command adjusts the column based on the length of the function names.
- *file-name*: Name of the job definition script file in which the function is defined.
- *line-number*: Line number at which the function is displayed.

The following describes the `info functions` command's processing.

When the info functions command argument is omitted

The command displays all function names and the corresponding file names and line numbers.

When the info functions command argument is specified

- Name of an existing function

  The command displays the specified function's name and the corresponding file name and line number.
- Other

  The command outputs an error message.

Notes:

Information about functions specified in the job definition script or in an external script specified in the `#-adsh_script` extended script command can be referenced even before the job definition script executes because this information is loaded automatically when the `adshexec` command starts. However, information about functions that are not specified in the `#-adsh_script` extended script command is not displayed while the job definition script is being run by the `run` command and until the processing defining that function is completed.

Example of output

This example displays function names and the corresponding file names and line numbers:

```
Function  File:Line
funcA     script1.ash:100
funcB     script2.ash:10
funcXXX   script1.ash:50
```

6.2.16 Displaying job step information (info jobsteps command)

The `info jobsteps` command displays job step information. The abbreviation for the `info jobsteps` command is `i j`. The following shows the format of the `info jobsteps` command:

```
info jobsteps[ job-step-name]
```

When a job step name is specified in the argument, the command displays that job step name and the corresponding file name and line number. If the argument is not specified, the command displays all job step names and the corresponding file names and line numbers. The following shows the display format:

```
Jobstep  File:Line
job-step-name  file-name:line-number
...
```

- *job-step-name*: Name of a defined job step. When all job steps are displayed, the command displays this information in ASCII code order of the job step names. A maximum of 31 bytes can be displayed for each job step name. The command adjusts the column based on the length of the job step names. If a job step name was omitted, the command displays `<No name>` for that job step.
- *file-name*: Name of the job definition script file in which the job step is defined.
- *line-number*: Line number at which the job step is displayed.

The following describes the `info jobsteps` command's processing.

When the info jobsteps command argument is omitted

The command displays all job step names and the corresponding file names and line numbers.

When the info jobsteps command argument is specified

If the specified job step name exists, the command displays the specified job step name and the corresponding file name and line number.

If a nonexistent job step name is specified, the command outputs an error message.

Notes:

You can display job step information regardless of whether `run` has been executed because this information is loaded when the `adshexec` command starts.

Example of output

This example displays job step names and the corresponding file names and line numbers:

```
Jobstep  File:Line
step1    script1.ash:10
step2    script1.ash:30
step3    script2.ash:10
```

6.2.17 Displaying signal information (info signals command)

The `info signals` command displays signal information. The abbreviation for the `info signals` command is `I si`. The following shows the format of the `info signals` command:

```
info signals[ signal-name| signal-number]
```

When a signal name or signal number is specified in the argument, the command displays information about the corresponding signal. If the argument is not specified, the command displays information about all signals. The following shows the display format:

```
Num  Signal       Stop  Print
signal-number   signal-name  Yes/No   Yes/No
...
```

- *signal-number*: Number of a signal. The command displays the signal numbers in ascending order. A maximum of two digits can be displayed left-justified.
- *signal-name*: The signal's name. A maximum of 11 bytes can be displayed left-justified.

The following describes the `info signals` command's processing.

When the info signals command argument is omitted

The command displays information about all signals.

When the info signals command argument is specified

- *signal-number*

  If the specified signal number exists, the command displays information about the signal with the specified signal number.

  If a nonexistent signal number is specified, the command outputs an error message.
- *signal-name*

If the specified signal name exists, the command displays information about the specified signal.

If a nonexistent signal name is specified, the command outputs an error message.

- `Stop`

  `Yes`: Indicates that when the signal indicated by `Signal` is received, the command stops the job definition script that is running.

  `No`: Indicates that the command does not stop the job definition script that is running when the signal indicated by `Signal` is received.

  For details about the processing when signals are received, see *3.10.2 Processing when signals are received (UNIX only)*.

- `Print`

  `Yes`: Indicates that when the signal indicated by `Signal` is received, the command displays a signal received message.

  `No`: Indicates that the command does not display a signal received message when the signal shown indicated by `Signal` is received.

Notes:

- A number that is specified in the argument must be `0` or a greater integer. Do not specify a plus sign (+) at the beginning. If any other value is entered, the command outputs an error message.
- If the specified number is outside the permitted value range for the `int` type, the command rounds it to the maximum value for the `int` type.

Example of output

This example displays signal information:

```
Num  Signal        Stop  Print
1    SIGHUP        No    No
2    SIGINT        Yes   Yes
```

6.2.18 Displaying the status (info status command)

The `info status` command displays the status of the job definition script that is being debugged. The abbreviation of the `info status` command is `i st`. The following shows the format of the `info status` command:

```
info status[ joberrmode]
```

If `joberrmode` or nothing is specified as the argument, the command displays the job definition script's fault injection mode. The abbreviation of `joberrmode` is `jem`.

Example of output

This example displays the status of the job definition script (when the fault injection mode is enabled):

```
joberrmode:on
```

This example displays the status of the job definition script (when the fault injection mode is disabled):

```
joberrmode:off
```

6.2.19 Displaying shell variable information (info variables command)

The `info variables` command displays information about all types of shell variables. The abbreviation for the `info variables` command is `i v`. The following shows the format of the `info variables` command:

```
info variables[ variable-name]
```

When a variable name is specified in the argument, the command displays information about the specified shell variable. If the argument is not specified, the command displays information about all shell variables. The following shows the display format:

```
variable-name = variable-value [(Step local)]
...
```

- *variable-name*: Name of a shell variable. When all shell variables are displayed, the command displays the information in ASCII code order of the shell variable names.
- *variable-value*: Value of the shell variable. For a shell variable that is valid only within a job step, the command adds the supplementary information `Step local`. A shell variable that is valid only within a job step is one that is specified in the `stepVar` attribute of the `#-adsh_step_start` extended script command.

For a variable that has no value, the command displays neither the equal sign (=) nor a variable value. The following shows the display format:

```
variable-name[(Step local)]
```

The following describes the `info variables` command's processing.

When the info variables command argument is omitted

The command displays information about all shell variables.

When the info variables command argument is specified

If the specified variable name exists, the command displays information about the specified shell variable.

If a nonexistent variable name is specified, the command outputs an error message.

Notes:

- If the job definition script is not being run by the `run` command, variables cannot be displayed because the variables are not defined.
- If you specify an array for the variable name, specify an array element.
- A variable and its array 0 (example: `aaa` and `aaa[0]`) are the same. If a variable is an array within the job definition script, the variable is represented with a subscript; if a variable is not an array, it is represented without a subscript.
- When an array is created in the shell, array element zero is created automatically. Therefore, when information about all shell variables is displayed, information about element zero is also included.

Example of output

This example displays information about all shell variables:

```
SHELL = /bin/sh
TEMPFILE = /tmp/file01
num = 1 (Step local variable)
val = 100
```

6.2.20 Enabling and disabling the fault injection mode (joberrmode command)

You execute the `joberrmode` command to enable and disable the fault injection mode for the job definition script being debugged. The abbreviation of `joberrmode` is `jem`. The following shows the format of the `joberrmode` command:

```
joberrmode{on|off}
```

When `on` is selected:

The fault injection mode is enabled, which enables you to test cases where errors have occurred in the job. This mode is used to simulate errors when the C1 execution percentage rate is not 100% even though all execution paths have executed. For details about the case where the C1 execution percentage rate is not 100%, see *3.9.4(5) Case where the C1 execution percentage rate is not 100%*.

When the fault injection mode is enabled, the debugger behaves as described in the following (the return code is not changed):

- When `abnormal` or `always` is specified in the `run` attribute, the debugger executes the job step.
- When the `run` attribute is omitted or `normal` is specified, the debugger does not execute the job step.
- The debugger does not execute a command outside the job step.
- If an error is injected within a step normal block, the debugger executes the commands within that step's step error block. The debugger executes the step error block without executing any subsequent command in that step even if `cont` is specified in the `onError` attribute.

When the `run` command is executed while the fault injection mode is `on`, the fault injection mode is reset to `off`.

When `off` is selected:

The fault injection mode is disabled.

Execution example 1

This example executes the job definition script shown in the following (the line numbers on the left correspond to the line numbers displayed in the execution results):

```
001  #!/bin/sh
002
003  #-adsh_step_start STEP001                <--Start of STEP001
004
005  ./cmd1                                  <--Step normal block
006
007  #-adsh_step_error                       <--Step error block
008
009  ./cmd2
010
011  #-adsh_step_end
012
013  ./cmd3
014
015  #-adsh_step_start STEP002 -run abnormal   <--Start of STEP002
(executed in the event of an error)
016
017  ./cmd4                                  <--Step normal block
018
019  #-adsh_step_error                       <--Step error block
020
```

```
021  ./cmd5
022
023  #-adsh_step_end
024
025  ./cmd6
026
027  echo JOB01-ended
```

Execution results 1

This example stops execution outside the step and uses the `joberrmode` command to enable the fault injection mode (the numbers at the right correspond to the numbers in the explanation provided below):

```
[jobuser1@HOST01 joberrmode]$ adshexec -d joberrmode.ash       <- 1.
KNAX7901-I The adshexec command will wait for all asynchronous
processes at the end of the job.
KNAX0724-I The job ID was assigned. job ID=000037
(adshdb) b 2                                                   <- 2.
KNAX7015-W The breakpoint cannot be set at line "2". The breakpoint
will be set at the next available line.
KNAX7018-I Breakpoint "1": filename="joberrmode.ash" line=3
(adshdb) b 17                                                  <- 3.
KNAX7018-I Breakpoint "2": filename="joberrmode.ash" line=17
(adshdb) run                                                   <- 4.
KNAX7007-I Execution of the following script will now start: /home/
jobuser1/joberrmode/joberrmode.ash

KNAX0724-I The job ID was assigned. job ID=000038
KNAX0091-I ADSH000038 The job started.
KNAX7902-I The adshexec command will run in tty stdin mode.

KNAX7018-I Breakpoint "1": filename="joberrmode.ash" line=3    <- 5.
KNAX7032-I The script "joberrmode.ash" stopped running.
3: #-adsh_step_start STEP001
Current: #-adsh_step_start STEP001
(adshdb) info status                                           <- 6.
joberrmode:off                                                 <- 7.
(adshdb) joberrmode on                                         <- 8.
KNAX7126-I Fault injection mode is set to "on".
(adshdb) info status                                           <- 9.
joberrmode:on                                                 <- 10.
(adshdb) c                                                    <- 11.
KNAX7034-I The script will continue.
KNAX6508-I ADSH000038.STEP001 step was skipped because a previous step
or command ended abnormally.
KNAX0092-I ADSH000038.STEP002 step started.

KNAX7018-I Breakpoint "2": filename="joberrmode.ash" line=17  <- 12.
KNAX7032-I The script "joberrmode.ash" stopped running.
17: ./cmd4
Current: ./cmd4
(adshdb) joberrmode off
KNAX7127-E Fault injection mode could not be modified.        <- 13.
(adshdb) c                                                    <- 14.
KNAX7034-I The script will continue.
cmd4 start
cmd4 end
KNAX6116-I Execution of the command ./cmd4 (line=17) finished
successfully. exit status=0 execution time=0.001s CPU time=0.000s
```

```
KNAX6597-I ADSH000038.STEP002 step succeeded. exit status=0 execution
time=3.081s CPU time=0.000s
KNAX0101-E ADSH000038 An error occurred during execution of the job.
KNAX0098-I ADSH000038 The job ended. exit status=0 execution
time=12.162s CPU time=0.000s
KNAX6380-I A job name will be added to the spool job directory of the
root job. spool job directory="/home/jobuser1/test6/spool/000038-
ADSH000038/"

(adshdb) quit
KNAX6380-I A job name will be added to the spool job directory of the
root job. spool job directory="/home/jobuser1/test6/spool/000037-
ADSH000037/"
KNAX7999-I Advanced Shell ended. exit status=0
```

The numbers below correspond to the numbers added at the right in the execution results above:

1. Starts debugging.
2. Stops execution inside the script.
3. Stops execution inside the step that is executed in the event of an error.
4. Executes the script.
5. Stops at the first breakpoint.
6. Displays the status.
7. The fault injection mode is disabled.
8. Enables the fault injection mode.
9. Displays the status.
10. The fault injection mode is enabled.
11. Restarts debugging.
12. Stops execution inside the step that is executed in the event of an error.
13. The fault injection mode could not be changed.
14. Restarts debugging.

Execution results 2

This example stops execution inside the step and uses the `joberrmode` command to enable the fault injection mode (the numbers added at the right correspond to the numbers in the explanation provided below):

```
[jobuser1@HOST01 joberrmode]$ adshexec -d joberrmode.ash       <- 1.
KNAX7901-I The adshexec command will wait for all asynchronous
processes at the end of the job.
KNAX0724-I The job ID was assigned. job ID=000043
(adshdb) b 5                                                    <- 2.
KNAX7018-I Breakpoint "1": filename="joberrmode.ash" line=5
(adshdb) b 17                                                   <- 3.
KNAX7018-I Breakpoint "2": filename="joberrmode.ash" line=17
(adshdb) run                                                    <- 4.
KNAX7007-I Execution of the following script will now start: /home/
jobuser1/joberrmode/joberrmode.ash

KNAX0724-I The job ID was assigned. job ID=000044
KNAX0091-I ADSH000044 The job started.
KNAX7902-I The adshexec command will run in tty stdin mode.
KNAX0092-I ADSH000044.STEP001 step started.
```

```
KNAX7018-I Breakpoint "1": filename="joberrmode.ash" line=5     <- 5.
KNAX7032-I The script "joberrmode.ash" stopped running.
5: ./cmd1
Current: ./cmd1
(adshdb) info status                                            <- 6.
joberrmode:off                                                  <- 7.
(adshdb) joberrmode on                                          <- 8.
KNAX7126-I Fault injection mode is set to "on".
(adshdb) info status                                            <- 9.
joberrmode:on                                                  <- 10.
(adshdb) c                                                     <- 11.
KNAX7034-I The script will continue.
cmd2 start
cmd2 end
KNAX6116-I Execution of the command ./cmd2 (line=9) finished
successfully. exit status=0 execution time=0.001s CPU time=0.000s
KNAX6596-E ADSH000044.STEP001 step failed. exit status=0 execution
time=62.384s CPU time=0.000s
KNAX0092-I ADSH000044.STEP002 step started.

KNAX7018-I Breakpoint "2": filename="joberrmode.ash" line=17  <- 12.
KNAX7032-I The script "joberrmode.ash" stopped running.
17: ./cmd4
Current: ./cmd4
(adshdb) joberrmode off
KNAX7127-E Fault injection mode could not be modified.        <- 13.
(adshdb) c                                                    <- 14.
KNAX7034-I The script will continue.
cmd4 start
cmd4 end
KNAX6116-I Execution of the command ./cmd4 (line=17) finished
successfully. exit status=0 execution time=0.001s CPU time=0.000s
KNAX6597-I ADSH000044.STEP002 step succeeded. exit status=0 execution
time=11.865s CPU time=0.000s
KNAX0101-E ADSH000044 An error occurred during execution of the job.
KNAX0098-I ADSH000044 The job ended. exit status=0 execution
time=74.272s CPU time=0.000s
KNAX6380-I A job name will be added to the spool job directory of the
root job. spool job directory="/home/jobuser1/test6/spool/000044-
ADSH000044/"
```

The numbers below correspond to the numbers added at the right in the execution results above:

1. Starts debugging.
2. Stops execution inside the script within the step.
3. Stops execution inside the step that is executed in the event of an error.
4. Executes the script.
5. Stops at the first breakpoint.
6. Displays the status.
7. The fault injection mode is disabled.
8. Enables the fault injection mode.
9. Displays the status.

10. The fault injection mode is enabled.
11. Restarts debugging.
12. Stops execution inside the step that is executed in the event of an error.
13. The fault injection mode could not be changed.
14. Restarts debugging.

Execution example 2

This example executes the job definition script shown in the following (the line numbers on the left correspond to the line numbers displayed in the execution results):

```
001  #-adsh_job JOB001
002  #-adsh_step_start STEP001 -onError cont  <--Start of STEP001 with -
onError cont specified
003   ./cmd1                                  <--Step normal block
004   ./cmd2
005   ./cmd3
006  #-adsh_step_error
007   ./cmd4                                  <--Step error block
008   ./cmd5
009  #-adsh_step_end
010   ./cmd6                                  <--Command outside the step
```

Execution results

This example stops execution within the step (line 4) and then enters the `joberrmode` command (the numbers added at the right correspond to the numbers in the explanation provided below):

```
[jobuser1@HOST01 joberrmode]$ adshexec -d test_cont.ash       <- 1.
KNAX7901-I The adshexec command will wait for all asynchronous
processes at the end of the job.
KNAX0724-I The job ID was assigned. job ID=000045
(adshdb) b 4                                                  <- 2.
KNAX7018-I Breakpoint "1": filename="test_cont.ash" line=4
(adshdb) r                                                    <- 3.
KNAX7007-I Execution of the following script will now start: /home/
jobuser1/joberrmode/test_cont.ash

KNAX0724-I The job ID was assigned. job ID=000046
KNAX0091-I JOB001 The job started.
KNAX7902-I The adshexec command will run in tty stdin mode.
KNAX0092-I JOB001.STEP001 step started.
cmd1 start
cmd1 end
KNAX6116-I Execution of the command ./cmd1 (line=3) finished
successfully. exit status=0 execution time=0.001s CPU time=0.000s

KNAX7018-I Breakpoint "1": filename="test_cont.ash" line=4    <- 4.
KNAX7032-I The script "test_cont.ash" stopped running.
4:  ./cmd2
Current: ./cmd2
(adshdb) info status                                          <- 5.
joberrmode:off                                                <- 6.
(adshdb) jem on                                               <- 7.
KNAX7126-I Fault injection mode is set to "on".
(adshdb) info status                                          <- 8.
joberrmode:on                                                 <- 9.
(adshdb) c                                                   <- 10.
```

```
KNAX7034-I The script will continue.
cmd4 start                                                   <- 11.
cmd4 end
KNAX6116-I Execution of the command ./cmd4 (line=7) finished
successfully. exit status=0 execution time=0.001s CPU time=0.000s
cmd5 start
cmd5 end
KNAX6116-I Execution of the command ./cmd5 (line=8) finished
successfully. exit status=0 execution time=0.001s CPU time=0.000s
KNAX6596-E JOB001.STEP001 step failed. exit status=0 execution
time=42.531s CPU time=0.000s
KNAX0101-E JOB001 An error occurred during execution of the job.
KNAX0098-I JOB001 The job ended. exit status=0 execution time=42.533s
CPU time=0.000s
KNAX6380-I A job name will be added to the spool job directory of the
root job. spool job directory="/home/jobuser1/test6/spool/000046-
JOB001/"
```

The numbers below correspond to the numbers added at the right in the execution results above:

1. Starts debugging.
2. Stops execution inside the script within the step.
3. Executes the script.
4. Stops at the first breakpoint.
5. Displays the status.
6. The fault injection mode is disabled.
7. Enables the fault injection mode.
8. Displays the status.
9. The fault injection mode is enabled.
10. Restarts debugging.
11. Executes the command in the step error block.

6.2.21 Setting a variable value (set command)

The `set` command sets a value for a shell variable. By specifying an assignment expression in the argument, you can evaluate the expression and then set a value in the variable. There is no abbreviation for the `set` command. The following shows the format of the `set` command:

```
set assignment-expression
```

The following shows the format of assignment expression:

```
variable-name={variable-name|numeric-value|"character-string"}
```

- *variable-name* (left-hand term): Specifies the name of a shell variable. The value of the right-hand term is assigned to the specified variable.
- *variable-name* (right-hand term): Specifies the name of a shell variable. The value of this specified variable is assigned to the variable of the left-hand term.

- *numeric-value*: Specifies an integer value. The specified integer value is assigned to the variable of the left-hand term.
- *character-string*: Specifies a character string. The specified character string is assigned to the variable of the left-hand term.

The following describes the `set` command's processing.

When the set command argument is specified

- Assignment expression
  The command sets a variable value according to the specified assignment expression.
- Assignment expression containing a variable name that has not been created
  The command outputs an error message.
- Other than an assignment expression
  The command outputs an error message.

When the set command argument is omitted

The command outputs an error message.

Notes:

- If the job definition script is not being run by the `run` command, an error results because variable information has not been specified.
- If you specify a character string, you must enclose it in double quotation marks (`"`). To use a double quotation mark as part of the character string, specify it as a backslash followed by a double quotation mark (`\"`). To specify `\"`, specify `\\\"`.
- The command processes the argument following the first equal sign (=) in the assignment expression as a variable name and a numeric value or a character string.
- If you specify an array for the variable name, specify an array element.
- Do not attach a dollar sign (`$`) to a variable name.
- If a specified numeric value is outside the permitted value range for the `signed long` type, it is rounded to the maximum or minimum value permitted for the `signed long` type.
- If execution is stopped before a `for` script control statement, a variable specified in `wordlists` in the `for` statement has a fixed value. If you want to change the value of the variable that is assigned by the `for` statement, use the `set` command at the first stop location following `do` in the `for` statement to change the variable value, or use the `set` command before execution reaches the `for` statement to change the value of the variable in `wordlists`. The following shows examples.

  Example 1:

```
  1: a=1
  2: b=2
  3: date
  4: for num in $a $b
  5: do
  6:  echo $num <--To change the value of num, execute the set command
before line 6 executes.
  7:  pwd
  8: done
```

  Example 2:

```
  1: a=1
  2: b=2
  3: date       <--To apply the values of $a and $b to the variable that
is assigned by the for statement, execute the set command before line 3
executes.
  4: for num in $a $b
  5: do
  6:  echo $num
  7:  pwd
  8: done
```

- The command assigns to the left-hand term a numeric value stored in the variable specified in the right-hand term of an assignment expression as is (without rounding).

Examples

To assign a numeric value, you must use the `-i` attribute of the `typeset` command to declare the variable's type as integer.

Assign numeric value 10 to variable a

```
(adshdb) set a=10
```

Assign character string test to variable b

```
(adshdb) set b="test"
```

Assign the value of variable a to variable c

```
(adshdb) set c=a
```

Assign numeric value 1 to variable d[5](array)

```
(adshdb) set d[5]=1
```

6.2.22 Displaying a variable's value (print command)

The `print` command displays the value of a variable in the job definition script. You display a variable's value by specifying its variable name in the argument. The abbreviation for the `print` command is `p`. The following shows the format of the `print` command:

```
print variable-name
```

The following describes the `print` command's processing.

When the print command argument is specified

If the specified variable is defined, the command displays its value. The following shows the display format:

```
variable-value
```

- *variable-value*: Value of the specified variable. If there is no value, `<No value>` is displayed.

If the variable is undefined, the command outputs an error message.

When the print command argument is omitted

The command outputs an error message.

Notes:

- If the job definition script is not being run by the `run` command, an error results because variable information has not been specified.
- If you specify an array for the variable name, specify an array element.
- Do not attach a dollar sign (`$`) to a variable name.

Examples

Display the value of variable a

```
(adshdb) print a
```

Display the value of variable b[1] (array)

```
(adshdb) print b[1]
```

6.2.23 Displaying a backtrace (where command)

A backtrace consists of information that shows how the executing job definition script reached the current location where execution has stopped. A backtrace is represented by frames. A frame is data related to a single call to a function. When a function is called, one frame is added. When a function terminates, its frame is deleted. A sequential number, beginning with zero, is assigned to each frame starting from the innermost frame. The innermost frame indicates the currently executing function. The `where` command displays a backtrace. The abbreviation for the `where` command is `whe`. The following shows the format of the `where` command:

```
where[ frame-number]
```

When a frame number is specified in the argument, the `where` command displays information about the innermost frame through the frame with the specified number. If the argument is omitted, the command displays information about all frames in order starting from the innermost frame. The following shows the display format:

```
Num  Function  File:Line
frame-number  function-name  file-name:line-number
...
```

- *frame-number*: Number assigned to a frame. The frame numbers are displayed in ascending order starting with `0`. Up to three left-justified digits can be displayed.
- *function-name*: Name of the function corresponding to the frame to which the name of the job definition script that called the function is attached. If no function has been called, `<main>` is displayed as the function name. A maximum of 63 bytes can be displayed.
- *file-name*: Name of the job definition script file corresponding to the frame. If the frame number is zero, this is the name of the current file whose execution is stopped. If the frame number is 1 or greater, this is the file name used when a new function was called.
- *line-number*: Line number corresponding to the frame. If the frame number is zero, this is the current line number at which execution has stopped. If the frame number is 1 or greater, this is the line number used to call a new function.

  If execution has stopped at the end of the job definition script, `<EOF>` is displayed. If execution has stopped while the `trap` command's `action` is executing, `<Trap action>` is displayed.

The following describes the `where` command's processing.

When the where command argument is omitted

If the `where` command is executed while execution of the job definition script is stopped, the command displays information about all frames in order starting from the innermost frame.

If the job definition script is not running, the command outputs an error message.

When the where command argument is specified

If the `where` command is executed while the job definition script is running and a valid frame number is specified, the command displays information about the innermost frame through the frame with the specified frame number.

If a non-numeric value is specified or the job definition script is not running, the command outputs an error message.

Notes:

- Specify `0` or a greater integer for a frame number. Do not specify a plus sign (+) at the beginning.
- If the specified frame number is outside the permitted value range for the `int` type, it is rounded to the maximum value permitted for the `int` type.
- A maximum of 255 frames can be displayed (frame numbers `0` through `254`). Specifying `255` or a greater value in the argument does not display a frame with the specified number. If there are more than 255 frames, a message is displayed following the frame information.

Example

This example executes the `where` command when `funcA` was called on line 12 of `sample.ash`, `funcB` was called on line 9, and then execution stopped on line 12 of `test.ash`.

`sample.ash`

```
5: #-adsh_script test.ash
6:
7: funcA(){
8:   num=10
9:   funcB
10: }
11:
12: funcA
```

`test.ash`

```
10: funcB(){
11:   val=5
12:   num=20
13: }

Num  Function               File:Line
0    funcB (in sample.ash)  test.ash:12
1    funcA (in sample.ash)  sample.ash:9
2    <main>                 sample.ash:12
```

6.2.24 Displaying the source file (list command)

The `list` command displays the source file. The abbreviation for the `list` command is `l`. The following shows the format of the `list` command:

When no file name is specified

```
list[ line-number]
```

When this is the first `list` command executed since execution stopped on the current line and no argument is specified in the command, the command displays a total of 11 numbered lines starting with the fifth line preceding the current line where execution has stopped. If this is the second or a subsequent `list` command entered since execution stopped, the command displays a total of 11 numbered lines starting from the line immediately following the last line displayed by the previous `list` command.

When a file name is specified

```
list job-definition-script-file-name:line-number
```

You can specify a job definition script file name by using a colon (`:`) in the argument.

If a line number is specified, the command displays 11 numbered lines starting with the fifth line preceding the specified line. The following shows the display format:

```
line-number: line-contents-in-source-file
...
```

- *line-number*: Line number in the source file.
- *Line-contents-in-source-file*: Contents of the indicated line in the source file.

The following describes the `list` command's processing.

When the list command argument is omitted

When this is the first `list` command executed since execution stopped on the current line, the command displays a total of 11 numbered lines in the source file starting with the fifth line preceding the current line where execution has stopped.

When this is the second or subsequent `list` command entered since execution stopped, the command displays a total of 11 numbered lines starting with the line immediately following the last line displayed by the previous `list` command.

When the list command argument is specified

If an existing line number is specified, the command displays 11 numbered lines starting with the fifth line preceding the specified line.

If a nonexistent line number or any other type of value is specified, the command outputs an error message.

Notes:

- If you specify a file name, make sure that you specify a file that was specified in the command for executing batch jobs or in the `#-adsh_script` extended script command.
- The command treats the entire character string up to the colon (`:`) as a file name.
- If the source file display range would include a line with a line number less than 1, the command displays lines 1 through 11, regardless of whether the argument was specified.
- If the source file display range would include a line beyond the last line, the command displays the last 11 lines starting with the tenth line preceding the last line, regardless of whether the argument was specified.
- Specify `0` or a greater integer for a line number. Do not specify a plus sign (+) at the beginning.
- If the specified line number is outside the permitted value range for `int` type, it is rounded to the maximum value permitted for the `int` type.
- If the job definition script under debug execution is in either of the statuses listed below and the `list` command is executed with the argument omitted, an error results. However, once you execute the `list` command with the argument specified, you can display the remaining lines.
    - Execution has stopped at the end of the job definition script (EOF).
    - Execution has stopped while the `trap` command's `action` is executing.

Example of output

This example executes the `list` command with the argument omitted while execution is stopped on line 20.

```
15: echo "start"
16:
17: a=1
18: while [[ $a -ne 10 ]]
19: do
20:   echo $a
21:   let a+=1
22: done
23:
24: pwd
25: echo "end"
```

6.2.25 Changing the directory (cd command)

The `cd` command changes the debugger's work directory. There is no abbreviation for the `cd` command. The following shows the format of the `cd` command:

```
cd directory-path-name
```

You change the work directory by specifying a directory path name in the argument.

The following describes the `cd` command's processing.

When the cd command argument is specified

The command changes the debugger's work directory to the specified directory. The command displays the absolute path of the new directory.

If the job definition script is executing, the `cd` command changes only the debugger's work directory; it does not change the current directory of the executing job definition script.

If the `cd` command is not able to change the debugger's work directory because a directory path name without execution permissions or a nonexistent directory path name is specified, the command outputs an error message.

When the cd command argument is omitted

The command outputs an error message.

Example

This example uses the `cd` command to change the work directory. The example then uses the `exec` command to execute an external command and outputs the contents of a file located at the destination directory.

```
(adshdb) cd work
KNAX7048-I Working directory: /home/xxx/work
(adshdb) exec cat test.txt
aaa bbb ccc
(adshdb)
```

6.2.26 Starting the login shell (exec command)

The `exec` command starts the login shell during debugging. The login shell is the shell specified in the `SHELL` shell variable. The abbreviation for the `exec` command is `ex`. The following shows the format of the `exec` command:

```
exec[ arguments-to-be-passed-to-login-shell]...
```

If arguments to be passed to the login shell are specified in the argument, the `exec` command passes the specified arguments to the login shell and then starts the login shell. If the argument is omitted, the `exec` command starts the login shell.

Notes:

- If the argument contains an ampersand (`&`), the command outputs an error message.
- To use an ampersand (`&`) for other purposes than background execution, specify `\&`.
- There is no limit to the number of arguments.

Example

This example uses the `exec` command to execute the `ls` shell command.

```
(adshdb) exec ls
aaa.txt  bbb.log  bin
```

6.2.27 Displaying Help (help command)

The `help` command displays the debugger's commands Help. The abbreviation for the `help` command is `h`. The following shows the format of the `help` command:

```
help[ command-name]
```

When a command name is specified in the argument, the `help` command displays a description of that command. If the argument is omitted, the `help` command displays the names of all commands. You can specify in the argument a command's full name or its abbreviation.

The following describes the `help` command's processing.

When the help command argument is omitted

The command displays the names of all commands. The following shows the display format:

```
Available commands:
break      cd         continue   delete     exec
finish     help       info       joberrmode kill
list       next       print      quit       return
run        set        signal     step       watch
where
```

When the help command argument is specified

When a command name is specified, the `help` command displays the usage of the specified command.

If a nonexistent command name is specified, the `help` command outputs an error message.

7 Parameters Specified in the Environment Files

You define in environment files various information such as return codes, coverage, system execution logs, directory paths, and environment variables. You use parameters to specify this information.

This chapter explains the formats of the parameters and provides the details of the parameters.

7.1 Specification format of environment files

This section explains the specification formats of parameters that are specified in the environment files.

There are three types of parameters:

Parameter type	Definition contents
Environment setting parameters	Define items such as return codes, coverage, system execution logs, and directory paths.
`export` parameter	Defines environment variables.
Conditional parameters	Specify environment setting parameters that are effective only at physical hosts or specific logical hosts or the `export` parameter.

The maximum length of one line in an environment file is 4,092 bytes including comments and separators. If a line exceeds 4,092 bytes, a parsing error will occur. You must not specify the ampersand (`&`) in an environment file, including within a comment.

The following notes apply to the parameters that are specified in the environment files:

- If a line contains `NULL` (`0x00`, or `\0` in C language), the job controller treats everything on the line from the beginning of the line up to that `NULL` character as the line, and ignores all characters following that `NULL` character. To prevent invalid execution results and run-time errors, do not specify `NULL`.
- Make sure that the encoding used for environment files is consistent with the value of the `LANG` environment variable in the environment in which job definition scripts are to be run.
- Text to the right of a hash mark (`#`) not followed by `-adsh_conf` Δ_1 is treated as a comment.

7.1.1 Formats of parameters

(1) Format of environment setting parameters

The following shows the format of environment setting parameters:

```
Δ0#-adsh_conf Δ1parameter Δ1value
```

- Specify a parameter following `#-adsh_conf` on one line.
- Specify nothing after the parameter value.
- If a parameter value contains a space, enclose the entire value in double quotation marks (`"`). No other escape character is permitted.

(2) Format of the export parameter

The following shows the format of the `export` parameter:

```
Δ0export Δ1environment-variable-name=environment-variable-value
```

- In the `export` parameter, specify one environment variable per line.

- The only environment variable that can be referenced is `PATH`.

  For example, the following specification sets in the `NEWHOME` environment variable the character string `${HOME}` as is, not the contents of the `HOME` environment variable,:

  ```
     export NEWHOME=${HOME}
  ```

- To specify a value containing a space, enclose the value in double quotation marks (`"`) or single quotation marks (`'`).
- A backslash (`\`) is treated as an escape character (a backslash enclosed in single quotation marks (`'`) is treated as a normal character).

(3) Format of conditional parameters

The following shows the format of the conditional parameters:

```
Δ0#-adsh_confΔ1 [phost_start | lhost_startΔ1 host-name]
environment-setting-parameter-or-export-parameter
     :
Δ0#-adsh_confΔ1 [phost_end | lhost_end]
```

- You can specify multiple environment setting parameters or `export` parameters.
- To specify `export` and environment setting parameters that are to be valid only on the physical host or only on a specified logical host, you must enclose them within conditional parameters specified on the preceding and following lines.
- Specify nothing after a parameter value.
- You can specify multiple conditional parameters. However, nesting conditional parameters is not permitted (see example 2).

Example 1: Specifying multiple conditional parameters

```
#-adsh_conf phost_start
export HOME=/home/phost
#-adsh_conf phost_end

#-adsh_conf phost_start
export TEMP=/tmp
#-adsh_conf phost_end
```

Example 2: Nesting conditional parameters (results in an error)

```
#-adsh_conf phost_start
export HOME=/home/phost
#-adsh_conf phost_start
export TEMP=/tmp
#-adsh_conf phost_end
#-adsh_conf phost_end
```

7.1.2 Specification format of comments

The following shows the specification format of a comment:

```
Δ0#any-character-string-not-beginning-with--adsh_conf
```

- Any character string that does not begin with `-adsh_conf` Δ1 that is specified following a hash mark (`#`) is treated as a comment (through the end of the line).

7.2 Lists of parameters

7.2.1 List of environment setting parameters

(1) Defining the environment setting parameters

You define the environment setting parameters in system environment files and job environment files. For details about these files, see *2.6.1 Specifying the environment files*.

The parameters that can be specified in a system environment files and a job environment file are the same. The following table shows the specification values that take effect depending on whether the system environment file and job environment file are specified:

System environment file	Job environment file	
	Omitted	Specified
Omitted	Each parameter's default value	Values specified in the job environment file
Specified	Values specified in the system environment file	Depends on the parameter specifications#

#

If the same parameter is specified in both, it is handled as described in the following:

- Environment setting parameters that can be specified only once in a file

  The specification in the job environment file takes effect.

- Environment setting parameters that can be specified more than once in a file

  JP1/Advanced Shell merges all the specifications of the parameter beginning with the specifications in the job environment file.

Example:

Specifications in the system environment file

```
#-adsh_conf PATH_CONV /jp1as/abc c:\\jp1as\\abc
#-adsh_conf PATH_CONV /jp1as/def c:\\jp1as\\def
```

Specifications in the job environment file

```
#-adsh_conf PATH_CONV /jp1as/abc c:\\jp1as\\ghi
#-adsh_conf PATH_CONV /jp1as/def c:\\jp1as\\def
#-adsh_conf PATH_CONV /jp1as/kkk c:\\jp1as\\kkk
```

The analysis results are equivalent to the following:

```
#-adsh_conf PATH_CONV /jp1as/abc c:\\jp1as\\ghi <--1.
#-adsh_conf PATH_CONV /jp1as/def c:\\jp1as\\def <--1.
#-adsh_conf PATH_CONV /jp1as/kkk c:\\jp1as\\kkk <--1.
#-adsh_conf PATH_CONV /jp1as/abc c:\\jp1as\\abc <--2.
#-adsh_conf PATH_CONV /jp1as/def c:\\jp1as\\def <--2.
```

Notes:

1. Specified in the job environment file
2. Specified in the system environment file

(2) List of environment setting parameters

The following table lists and describes the environment setting parameters that are specified in the environment files of JP1/Advanced Shell. Specification of these parameters is optional.

Table 7–1: Environment setting parameters specified in the environment files of JP1/Advanced Shell

Parameter name	Definition	Maximum number of times parameter can be specified[#1]	Changeability at the start of child jobs
ADSHCMD_RC_ERROR	Defines the return code to be used when an extended script command fails.	1	Y
ADSHCMD_RC_SUCCESS	Defines the return code to be used when an extended script command is successful.	1	Y
ASC_FILE	Defines a naming rule for accumulation files used in the coverage auto-acquisition functionality.	1	--
BATCH_CVR	Enables the coverage auto-acquisition functionality.	1	--
CHILDJOB_EXT	Defines an extension for job definition script files that are to be executed as child jobs.	255	Y
CHILDJOB_PGM	Defines a program path specification that is to be executed as child jobs.	255	Y
CHILDJOB_SHEBANG	Defines an executable program path for job definition script files that are to be executed as child jobs.	255	Y
CMDRC_THRESHOLD_DEFINE	Defines a return code threshold for a command. This parameter can also be defined for return codes of shell scripts and child jobs.	No limit[#3]	Y
CMDRC_THRESHOLD_USE_PRESET	Defines a return code threshold for UNIX-compatible commands.	1	Y
COMMAND_CONV_ARG	Defines a rule for converting an argument in job definition scripts during command execution.	255	Y
ESCAPE_SEQ_ECHO_DEFAULT	Defines the action of the echo command when the escape-character option is omitted.	1	Y
ESCAPE_SEQ_ECHO_HEX	Specifies whether ASCII code characters in hexadecimal notation are to be interpreted as escape characters.	1	Y
EVENT_COLLECT	Specifies whether the operation information acquisition functionality is to be enabled for job definition scripts.	1	Y
HOSTNAME_JP1IM_MANAGER[#2]	Specifies for the user-reply functionality the operation management server on which JP1/IM - Manager is running that is to be the destination of JP1 events.	1	DN
JOBEXECLOG_PRINT	Defines the job execution log contents to be output to the standard error output when a job terminates.	1	--

Parameter name	Definition	Maximum number of times parameter can be specified[#1]	Changeability at the start of child jobs
JOBLOG_SUPPRESS_MSG	Defines a message whose output to job execution logs is to be suppressed.	No limit[#3]	Y
KSH_ENV_READ	Defines whether the ENV shell variable is to be read.	1	Y
LOG_DIR[#4]	Defines the path name of the directory to which system execution logs are to be output.	1	Y
LOG_FILE_CNT[#5]	Defines the number of files to be used to back up system execution logs.	1	Y
LOG_FILE_SIZE[#5]	Defines the size of a file to which system execution logs are to be output.	1	Y
OUTPUT_MODE_CHILD	Specifies whether a job execution log is to be output to the standard error output when a child job terminates.	1	Y
OUTPUT_MODE_ROOT	Specifies whether a job execution log is to be output to the standard error output when a root job terminates.	1	--
OUTPUT_STDOUT	Defines the destination for the root job standard output.	1	--
PATH_CONV	Defines a rule for converting absolute path names.	255	Y
PATH_CONV_ACCESS	Defines a rule for converting file path names in job definition scripts when files are input and output.	255	Y
PATH_CONV_ENABLE	Enables the path conversion functionality.	1	Y
PATH_CONV_RULE (Windows only)	Defines a rule for converting file paths.	1	Y
PERMISSION_SPOOLJOB_DIR (UNIX only)	Defines permission for the spool job directory.	1	--
PERMISSION_SPOOLJOB_FILE (UNIX only)	Defines permission for the files under the spool job directory.	1	--
PIPE_CMD_LAST (UNIX only)	Defines execution processing for the last command in a pipe.	1	Y
SPOOL_DIR[#2, #4, #6]	Defines the spool root directory path name.	1	DN
SPOOLJOB_CHILDJOB	Specifies whether a child job's spool job is to be deleted or is to be merged into the spool job of the root job when the child job terminates.	1	N
SPOOLJOB_CREATE	Defines whether a spool job is to be created when a job definition script is run.	1	N
TEMP_FILE_DIR[#4]	Defines the path name of the directory for storing temporary files.	1	Y
TRACE_DIR[#4]	Defines the path name of the directory to which traces are to be output.	1	Y

Parameter name	Definition	Maximum number of times parameter can be specified[#1]	Changeability at the start of child jobs
TRACE_FILE_CNT[#7]	Defines the number of files to which traces are to be output.	1	Y
TRACE_FILE_SIZE[#7]	Defines the size of a file to which traces are to be output.	1	Y
TRACE_LEVEL	Defines a trace output level.	1	Y
TRAP_ACTION_SIGTERM	Defines the job controller's action when a forced termination request is received.	1	Y
UNSUPPORT_TEST (Windows only)	Defines the handling of an unsupported conditional expression.	1 for each condition	Y
USERREPLY_DEBUG_DESTINATION	Defines the input source and the destination of event notification and reply-request messages during debug execution when the user-reply functionality is used.	1	DN
USERREPLY_JP1EVENT_INTERVAL[#2]	Defines the minimum interval at which JP1 events are to be issued by the adshecho and adshread command in the user-reply functionality.	1	DN
USERREPLY_WAIT_MAXCOUNT[#2]	Defines the maximum number of concurrent reply-request messages that can be output by the user-reply functionality per physical or logical host.	1	DN
VAR_ENV_NAME_LOWERCASE (Windows only)	Defines whether environment variable names in lowercase letters are supported.	1	Y
VAR_SHELL_FUNCINFO	Defines arrays for managing information about shell functions.	1	Y
VAR_SHELL_GETLENGTH	Defines the unit for lengths when the lengths of variable values are referenced.	1	Y

Legend:

Y: If a change is made when a child job starts to the setting in effect when its root job started, that change will be effective (but making such a change is not recommended).

N: If a change is made when a child job starts to the setting in effect when its root job started, that change will be ignored.

DN: When a child job starts, do not make any change to the setting in effect when its root job started. If a change is made, JP1/Advanced Shell might not operate correctly.

--: Not applicable to child jobs.

#1

For a parameter that can be specified in both the system environment file and the job environment file, make sure that the total number of specifications of the parameter in both files combined does not exceed the maximum value shown in this column.

To determine whether a parameter can be specified in both the system environment file and the job environment file, see the explanations for the individual parameters.

#2

To use the user-reply functionality, you must specify these parameters in the system environment file. When you make a change to these parameters in the system environment file, you must restart the user-reply functionality's management daemon or service.

Also note the following regarding the specification of these parameters:

- Do not specify these parameters in a job environment file. If they are specified in a job environment file, the following problems might occur:

  HOSTNAME_JP1IM_MANAGER: When a reply is entered or canceled with the adshchmsg command, the reply-waiting events accumulated in JP1/IM - View are not released while the user-reply functionality management's daemon or service is stopped.

  USERREPLY_WAIT_MAXCOUNT: The information specified in the job environment file is ignored.

  USERREPLY_JP1EVENT_INTERVAL: JP1/IM - View's processing load increases.

- If different values are specified in the system environment file and the job environment file for the parameter shown below, the problem described below might occur. If you use the user-reply functionality, do not specify the following parameter in the job environment file:

  SPOOL_DIR: Output of reply-request messages fails.

#3

Limited by the available memory size.

#4

You can use multiple environments on the same host by specifying different directories in these parameters.

#5

If multiple users output system execution logs to the same file, the values of LOG_FILE_CNT and LOG_FILE_SIZE specified by the last user that started output of system execution logs become effective.

#6

To inherit information to a host in a standby system during cluster operation, you must share the directory to be inherited among the multiple hosts. In this case, share at least the directory specified in this parameter among the hosts.

#7

If multiple users output trace logs to the same file, the largest values specified for TRACE_FILE_CNT and TRACE_FILE_SIZE by the users become effective.

If the values of TRACE_FILE_CNT and TRACE_FILE_SIZE are changed in the environment file, the specified values are compared with the existing values for the number of trace files and file size, and whichever are larger become effective.

To reduce the number of trace files and the file size, you must delete all files from the trace folder (do not delete files from the trace folder while a job is outputting traces to those trace files).

7.2.2 export parameter

(1) Defining the export parameter

You can define export parameters in both the system environment file and the job environment file. For details about these files, see *2.6.1 Specifying the environment files*.

If export parameters are specified in both the system environment file and the job environment file, they are handled as follows:

- Because the environment files are also analyzed whenever a child job starts, the values specified in `export` parameters at the start of the child job are set in the environment variables again.
- Both the system environment file specifications and the job environment file specifications are effective and these files are executed in this order.

Example:

System environment file

```
export A=s1
export B=s2
export A=s3
```

Job environment file

```
export C=j1
export B=j2
```

Order of execution

```
export A=s1
export B=s2
export A=s3
export C=j1
export B=j2
```

- The following are examples of adding paths to the `PATH` environment variable:

```
export   PATH='d:\user\prg;${PATH}'
export   PATH='/user/prg:${PATH}'
```

When a root job starts, paths are added to the `PATH` environment variable, and paths are also added to the `PATH` environment variable when a child job of such a root job starts. In this manner, the length of the value of the `PATH` environment variable increases. As a result, the maximum size of the `PATH` environment variable might be exceeded. When you use child jobs, make sure that the maximum size of the `PATH` environment variable will not be exceeded.

If the maximum size will be exceeded, separate the environment file for the root job from the environment file for the applicable child job and add paths to the `PATH` environment variable only in the environment file for the root job.

(2) List of export parameter

The following describes the `export` parameter definition conditions. Specification of this parameter is optional. Only the job controller uses this parameter.

Parameter name	Definition	Maximum number of times parameter can be specified
`export`	Defines an environment variable that is to take effect when the job controller that uses the environment file is started.	No limit

7.2.3 Conditional parameters

(1) Defining the conditional parameters

You use conditional parameters to enclose the definitions of environment setting parameters and `export` parameters that are to apply only to a logical host or to the physical host.

Parameters specified outside the conditional parameters apply to all hosts. Specifications of the same parameter inside and outside conditional parameters are treated as duplicates, and if the total number of such parameters causes the permitted maximum number of specifications to be exceeded, an error results.

An example definition of conditional parameter is explained below.

```
(parameter group A)
#-adsh_conf lhost_start HOST01
(parameter group B)
#-adsh_conf lhost_end
#-adsh_conf lhost_start HOST02
(parameter group C)
#-adsh_conf lhost_end
#-adsh_conf phost_start
(parameter group D)
#-adsh_conf phost_end
#-adsh_conf lhost_start HOST01
(parameter group E)
#-adsh_conf lhost_end
#-adsh_conf lhost_start HOST02
(parameter group F)
#-adsh_conf lhost_end
#-adsh_conf phost_start
(parameter group G)
#-adsh_conf phost_end
(parameter group H)
```

Based on this definition, the following parameter groups are executed on the specified hosts:

Parameter groups executed on logical host `HOST01`:

```
(parameter group A)
(parameter group B)
(parameter group E)
(parameter group H)
```

Parameter groups executed on logical host `HOST02`:

```
(parameter group A)
(parameter group C)
(parameter group F)
(parameter group H)
```

Parameter groups executed on the physical host:

```
(parameter group A)
(parameter group D)
(parameter group G)
(parameter group H)
```

You can define conditional parameters in both the system environment file and the job environment file. For details about these files, see *2.6.1 Specifying the environment files*.

The same parameters can be specified in the system environment file and the job environment file. Parameters specified in both the system environment file and the job environment file are handled as follows:

- The parameter specifications in both the system environment file and the job environment file take effect.
- The rules for individual parameters apply to those parameters that are determined to be valid because they satisfied the conditions.

Example:

System environment file

```
#-adsh_conf lhost_start host01
#-adsh_conf TEMP_FILE_DIR /jp1as/temp
#-adsh_conf PATH_CONV /jp1as/abc c:\\jp1as\\sys1
#-adsh_conf PATH_CONV /jp1as/def c:\\jp1as\\sys2
#-adsh_conf lhost_end
```

Job environment file

```
#-adsh_conf lhost_start host01
#-adsh_conf TEMP_FILE_DIR /home/temp
#-adsh_conf PATH_CONV /jp1as/abc c:\\jp1as\\job1
#-adsh_conf PATH_CONV /jp1as/def c:\\jp1as\\job2
#-adsh_conf PATH_CONV /jp1as/kkk c:\\jp1as\\job3
#-adsh_conf lhost_end
```

The results of executing the job on logical host `host01` are equivalent to the following:

```
#-adsh_conf TEMP_FILE_DIR /home/temp
#-adsh_conf PATH_CONV /jp1as/abc c:\\jp1as\\job1
#-adsh_conf PATH_CONV /jp1as/def c:\\jp1as\\job2
#-adsh_conf PATH_CONV /jp1as/kkk c:\\jp1as\\job3
#-adsh_conf PATH_CONV /jp1as/abc c:\\jp1as\\sys1
#-adsh_conf PATH_CONV /jp1as/def c:\\jp1as\\sys2
```

(2) List of conditional parameters

The following table lists and describes the conditional parameters. Specification of these parameters is optional.

Table 7–2: Conditional parameters specified in the environment files of JP1/Advanced Shell

Parameter name	Definition	Maximum number of times parameter can be specified
`lhost_start`	Starts a set of environment setting parameters or `export` parameters that are to take effect only on a specified logical host. This parameter also specifies the target logical host.	No limit
`lhost_end`	Ends the definition of a set of environment setting parameters or `export` parameters started by `lhost_start`. This parameter must always be paired with an `lhost_start` parameter.	
`phost_start`	Starts a set of environment setting parameters or `export` parameters that are to take effect only on the physical host.	

Parameter name	Definition	Maximum number of times parameter can be specified
phost_end	Ends the definition of a set of environment setting parameters or export parameters started by phost_start. This parameter must always be paired with a phost_start parameter.	No limit

(3) Examples of definitions of conditional parameters

This subsection presents definition examples of parameters that are specified in the environment files.

(a) Example definitions of a system environment file and a job environment file

This subsection explains the relationship between the system environment file and the job environment file by way of examples.

- Running a single host

  This example defines a system environment file as the system default. The example defines the following information:

  - The name of the host to which JP1 events are issued is HostJp1IM.
  - Output of the KNAX6110-I and KNAX6111-I messages is suppressed.

  The following is an example of definitions in the system environment file:

```
#-adsh_conf  JOBLOG_SUPPRESS_MSG  KNAX6110-I
#-adsh_conf  JOBLOG_SUPPRESS_MSG  KNAX6111-I
#-adsh_conf  HOSTNAME_JP1IM_MANAGER  HostJp1IM
```

  This example uses job environment files to define changes to the settings for each job.

  You can convert the arguments in a job definition script to different values for each logical host, and you can define for each job the return code to be used in the event of an error in an extended script command.

  The following is an example of definitions in the job environment file:

```
#-adsh_conf  ADSHCMD_RC_ERROR  8
#-adsh_conf  COMMAND_CONV_ARG  /var/tmp  /home/user01/tmp
```

- Running logical hosts HOST01 and HOST02 at the same time

  This example defines a system environment file as the system default. The example defines the following information:

  - Output of the KNAX6110-I and KNAX6111-I messages is suppressed.
  - So that the logical hosts can use different execution environments, the SPOOL_DIR, LOG_DIR, TRACE_DIR, TEMP_FILE_DIR, and HOSTNAME_JP1IM_MANAGER parameters are defined separately for each host.
  - The value of the ABC environment variable is specified for each logical host.

  The following is an example of definitions in the system environment file:

```
#-adsh_conf  JOBLOG_SUPPRESS_MSG  KNAX6110-I
#-adsh_conf  JOBLOG_SUPPRESS_MSG  KNAX6111-I
#-adsh_conf  lhost_start HOST01
#-adsh_conf  SPOOL_DIR  /jp1as/host01/spool
#-adsh_conf  LOG_DIR    /jp1as/host01/log
#-adsh_conf  TRACE_DIR  /jp1as/host01/trace
#-adsh_conf  TEMP_FILE_DIR /jp1as/host01/temp
#-adsh_conf  HOSTNAME_JP1IM_MANAGER  HostJp1IM01
```

```
#-adsh_conf  lhost_end
#-adsh_conf  lhost_start HOST02
#-adsh_conf  SPOOL_DIR  /jp1as/host02/spool
#-adsh_conf  LOG_DIR    /jp1as/host02/log
#-adsh_conf  TRACE_DIR  /jp1as/host02/trace
#-adsh_conf  TEMP_FILE_DIR /jp1as/host02/temp
#-adsh_conf  HOSTNAME_JP1IM_MANAGER  HostJp1IM02
#-adsh_conf  lhost_end
#-adsh_conf  lhost_start HOST01
export  ABC=/jp1as/host01/abc
#-adsh_conf  lhost_end
#-adsh_conf  lhost_start HOST02
export  ABC=/jp1as/host02/abc
#-adsh_conf  lhost_end
```

This example uses job environment files to define changes to the settings for each job.

You can convert the arguments in a job definition script to different values for each logical host, and you can define for each job the return code to be used in the event of an error in an extended script command.

The following is an example of definitions in a job environment file:

```
#-adsh_conf ADSHCMD_RC_ERROR  8
#-adsh_conf lhost_start HOST01
#-adsh_conf COMMAND_CONV_ARG  /var/tmp  /home/user01/tmp01
#-adsh_conf lhost_end
#-adsh_conf lhost_start HOST02
#-adsh_conf COMMAND_CONV_ARG  /var/tmp  /home/user01/tmp02
#-adsh_conf lhost_end
```

- Running logical host `HOST01` during normal operation and temporarily running the physical host

  This example defines a system environment file as the system default. The example defines the following information:

  - Output of the `KNAX6110-I` and `KNAX6111-I` messages is suppressed.
  - The `SPOOL_DIR`, `LOG_DIR`, `TRACE_DIR`, `TEMP_FILE_DIR`, and `HOSTNAME_JP1IM_MANAGER` parameters are defined separately for the logical host and for the physical host so that they can use different execution environments.
  - The default directory is used for the `SPOOL_DIR`, `LOG_DIR`, `TRACE_DIR`, and `TEMP_FILE_DIR` parameters for the physical host.
  - Different values are used for the `ABC` environment variable in the logical host and the physical host.

  The following is an example of definitions in the system environment file:

```
#-adsh_conf  JOBLOG_SUPPRESS_MSG  KNAX6110-I
#-adsh_conf  JOBLOG_SUPPRESS_MSG  KNAX6111-I
#-adsh_conf  lhost_start HOST01
#-adsh_conf  SPOOL_DIR  /jp1as/host01/spool
#-adsh_conf  LOG_DIR    /jp1as/host01/log
#-adsh_conf  TRACE_DIR  /jp1as/host01/trace
#-adsh_conf  TEMP_FILE_DIR /jp1as/host01/temp
#-adsh_conf  HOSTNAME_JP1IM_MANAGER  HostJp1IM01
#-adsh_conf  lhost_end
#-adsh_conf  phost_start
#-adsh_conf  HOSTNAME_JP1IM_MANAGER  HostJp1IM01
#-adsh_conf  phost_end
#-adsh_conf  lhost_start HOST01
export  ABC=/jp1as/host01/abc
```

```
#-adsh_conf  lhost_end
#-adsh_conf  phost_start
export  ABC=/jp1as/abc
#-adsh_conf  phost_end
```

This example uses job environment files to define changes to the settings for each job.

You can convert the arguments in a job definition script to different values for the logical host and the physical host, and you can define for each the return code to be used in the event of an error in an extended script command.

The following is an example of definitions in a job environment file:

```
#-adsh_conf ADSHCMD_RC_ERROR  8
#-adsh_conf lhost_start HOST01
#-adsh_conf COMMAND_CONV_ARG  /var/tmp  /home/user01/tmp01
#-adsh_conf lhost_end
#-adsh_conf phost_start
#-adsh_conf COMMAND_CONV_ARG  /var/tmp  /home/user01/tmp00
#-adsh_conf phost_end
```

(b) Example definitions of conditional parameters

This subsection explains by way of an example the definition of conditional parameters and parameters that are applied to different hosts.

This example defines the conditional parameters as follows:

```
(parameter group A)
#-adsh_conf lhost_start HOST01
(parameter group B)
#-adsh_conf lhost_end
#-adsh_conf lhost_start HOST02
(parameter group C)
#-adsh_conf lhost_end
#-adsh_conf phost_start
(parameter group D)
#-adsh_conf phost_end
#-adsh_conf lhost_start HOST01
(parameter group E)
#-adsh_conf lhost_end
#-adsh_conf lhost_start HOST02
(parameter group F)
#-adsh_conf lhost_end
#-adsh_conf phost_start
(parameter group G)
#-adsh_conf phost_end
(parameter group H)
```

Based on these definitions, the following parameter groups are executed on each host:

Parameter groups executed on logical host `HOST01`:

```
(parameter group A)
(parameter group B)
(parameter group E)
(parameter group H)
```

Parameter groups executed on logical host HOST02:

```
(parameter group A)
(parameter group C)
(parameter group F)
(parameter group H)
```

Parameter groups executed on the physical host:

```
(parameter group A)
(parameter group D)
(parameter group G)
(parameter group H)
```

7.3 Environment setting parameters

This section explains the parameters that are specified in JP1/Advanced Shell's environment files.

ADSHCMD_RC_ERROR parameter (defines the return code to be used when an extended script command fails)

Format

```
#-adsh_conf ADSHCMD_RC_ERROR return-code
```

Description

This parameter defines the return code to be used when an extended script command fails.

Operands

return-code **~<unsigned integer>((0 to 255))<<1>>**

Specifies the return code to be used when an extended script command fails.

Notes

- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect.

ADSHCMD_RC_SUCCESS parameter (defines the return code to be used when an extended script command is successful)

Format

```
#-adsh_conf ADSHCMD_RC_SUCCESS return-code
```

Description

This parameter defines the return code to be used when an extended script command is successful.

Operands

return-code **~<unsigned integer>((0 to 255))<<0>>**

Specifies the return code to be used when an extended script command is successful.

Notes

- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect.

ASC_FILE parameter (defines a naming rule for accumulation files)

Format

```
#-adsh_conf ASC_FILE file-naming-rule
```

Description

This parameter defines a naming rule for accumulation files used in the coverage auto-acquisition functionality.

Operands

file-naming-rule

Windows: **~<any character string>((1 to 247 bytes))**

UNIX: **~<any character string>((1 to 1,023 bytes))**

Specifies a path name with a substitution position.

The asterisk (`*`) specifies the substitution position. The location where the asterisk is specified is replaced with the job definition script name (without the file extension). The search begins at the beginning and substitution occurs at the location of the first asterisk that is detected. Any subsequent asterisks that are specified are not subject to substitution.

If no substitution position is specified, the specified name is used as is as the file name, and no substitution occurs.

If the resulting value is invalid as a path name regardless of whether substitution occurred, an error results. If this parameter is omitted, the file name is determined according to the rules used when the `-o` option is omitted in the `adshexec` command.

You must specify the file naming rule in such a manner that the file name obtained after conversion does not exceed the maximum length permitted for the path name of an `asc` file in the `adshexec` command. If this maximum length is exceeded, an error will occur during execution of the `adshexec` command.

Notes

- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect.
- This parameter is ignored when the `BATCH_CVR` parameter's operand value is not `YES`.
- Do not use a file name that begins with a dot (`.`).
- Do not use reserved device names (such as `CON`, `AUX`, and `NUL`) as file names. (Windows only)
- Do not use NTFS streams for file names. (Windows only)

Example

- This example enables the coverage auto-acquisition functionality and defines a naming rule for accumulation files.

```
#-adsh_conf BATCH_CVR YES
#-adsh_conf ASC_FILE ./cvrg/ver001-*
```

In this example, execution of `adshexec sample.ash` will be the same as execution of `adshexec -t -o ./cvrg/ver001-sample sample.ash`.

BATCH_CVR parameter (enables the coverage auto-acquisition functionality)

Format

```
#-adsh_conf BATCH_CVR YES
```

Description

This parameter enables the coverage auto-acquisition functionality.

Operands

`YES`

Enables the coverage auto-acquisition functionality. Enabling this functionality results in the same effects as when the `-t` option is specified in the `adshexec` command.

As is the case when the `-f` option is omitted in the `adshexec` command, if there is any difference between the job definition script file and backup information, JP1/Advanced Shell does not collect coverage information.

Notes

- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect.
- The coverage auto-acquisition functionality is disabled in the following cases:
  - The `-v` or `-c` option is specified in the `adshexec` command
  - JP1/Advanced Shell - Developer is used
- For the `ASC_FILE` parameter to take effect, this parameter must be specified.
- If the `-t` option is specified in the `adshexec` command, the command outputs an error message and terminates with `RC=1`.
- If the job definition script file is changed while there is already a coverage information file, coverage information can no longer be collected. In such a case, take one of the following actions:
  - Delete the `asc` file used to change the job definition script.
  - Use the environment variable to change the directory in which the `asc` files are created.

CHILDJOB_EXT parameter (defines an extension for job definition script files that are to be executed as child jobs)

Format

```
#-adsh_conf CHILDJOB_EXT extension
```

Description

This parameter defines an extension for job definition script files that are to be executed as child jobs. When a job definition script file with the extension specified in this parameter is specified as a command name in another job definition script, the command outputs the `KNAX6832-I` message to the job execution logs, interprets the file as a job definition script for JP1/Advanced Shell, and then executes the file as a child job. You can use the JOBLOG_SUPPRESS_MSG parameter to suppress output of the KNAX6832-I message to the job execution logs.

Operands

extension **~<path name>((1 to 245 bytes))**

Specifies an extension for job definition script files that are to be executed as child jobs. Specifying as the value only the null character enclosed in double quotation marks (") is not permitted.

Notes

- If this parameter is defined in both the system environment file and the job environment file, both definitions take effect. However, if the total number of times this parameter is specified in the system environment file and the job environment file combined exceeds 255, an error occurs.
- Do not include a dot (.) in the extension specified in the operand.
- If only a forward slash (/) is specified as the extension in the operand, the command will terminate with an error during parameter analysis.
- Do not specify exe, bat, cmd, or com, which are executable extensions in Windows, as an extension in this operand. If these extensions are specified, the corresponding files will be executed as child jobs in the same manner as when other extensions are specified. (Windows only)
- The command name output to the job execution logs is the path of the script file that was executed as a child job.
- If a file that can be executed by using this parameter is evaluated by using the operator -x for evaluating file attributes, the result will be true. (Windows only)
- Specifying the same extension more than once does not result in an error.
- Execution of a file specified in the argument of the exec, command, or eval standard shell command or the time reserved script command is also subject to this parameter.
- This parameter is not case sensitive in the Windows edition, but it is case sensitive in the UNIX edition.
- To execute a file with no extension as a child job, use the CHILDJOB_PGM or CHILDJOB_SHEBANG parameter.
- This parameter is applied to a command name following variable substitution or alias resolution.

Example

In this example, any file with the extension ash or sh specified as a command name is executed as a child job.

```
#-adsh_conf CHILDJOB_EXT ash
#-adsh_conf CHILDJOB_EXT sh
```

CHILDJOB_PGM parameter (defines a program path specification that is to be executed as descendent jobs)

Format

```
#-adsh_conf CHILDJOB_PGM program-path-name [arguments-of-executable-program]
```

Description

This parameter specifies that when the file specification *program-path-name* [*arguments-of-executable-program*] appears in a job definition script, the KNAX6830-I message is to be output to the job execution logs, the file is to be interpreted as a job definition script, and then the file is to be executed as a child job. You can use the JOBLOG_SUPPRESS_MSG parameter to suppress output of the KNAX6830-I message to the job execution logs.

The following shows an example:

Example specification of parameter:

```
#-adsh_conf CHILDJOB_PGM sh -x
```

Contents of job definition script:

```
sh -x $HOME/script/test.ash
```

This example interprets $HOME/script/test.ash as a job definition script and executes it as a child job.

The total number of *program-path-name* and *arguments-of-executable-program* operands combined that can be specified is 64. If more than 64 operands are specified, the command will terminate with an error during parameter analysis.

In the Windows edition, even if the CHILDJOB_PGM parameter is omitted from the environment file, a CHILDJOB_PGM parameter with adshscripttool -exec specified for the program path is defined. For details about the adshscripttool command, see *adshscripttool command (supports creation of job definition scripts) (Windows only)*.

Operands

program-path-name ~**<path name>((1 to 1,023 bytes))**

Defines a program path that is to be converted to a program to be executed as a child job. Specifying as a value only the null character enclosed in double quotation marks (") is not permitted. The backslash (\) is not handled as an escape character.

arguments-of-executable-program ~**<any character string>((1 to 1,023 bytes))**

Defines arguments of the program path that is to be converted to a program to be executed as a child job. You can specify multiple arguments by delimiting them with the space or tab character. The backslash (\) is not handled as an escape character.

Notes

- If this parameter is defined in both the system environment file and the job environment file, both definitions take effect. However, if the total number of times this parameter is specified in the system environment file and the job environment file combined exceeds 255, an error occurs.
- If the same operand is defined in this parameter and the PATH_CONV parameter, conversion by the PATH_CONV parameter is performed first. An example follows:

  Contents of environment file

  ```
  #-adsh_conf PATH_CONV_ENABLE / :             <- 1.
  #-adsh_conf PATH_CONV /usr/bin C:\\usr\\bin    <- 2.
  #-adsh_conf CHILDJOB_PGM /usr/bin/ksh        <- 3.
  ```

  Contents of job definition script

```
"/usr/bin/ksh" C:\\script\\test.ash
```

In this example, steps 1 and 2 are executed first to convert /usr/bin/ksh into C:\\usr\\bin\\ksh. In step 3, /usr/bin/ksh is interpreted to be the program path for executing a child job. However, because no matching character string is found in the job definition script whose path has been converted, C:\\script\\test.ash is not executed as a child job.

- If the same operand is specified in this parameter and the COMMAND_CONV_ARG parameter, conversion by the COMMAND_CONV_ARG parameter is performed first.
- This parameter is applied to a character string following variable substitution or alias resolution. An example of variable substitution follows:

  Contents of environment file

  ```
  #-adsh_conf CHILDJOB_PGM /usr/bin/ksh
  ```

  Contents of job definition script (the value of the variable SHELL is assumed to be /usr/bin/ksh)

  ```
  $SHELL /home/usr/test.ash
  ```

  In this example, because $SHELL is first resolved as /usr/bin/ksh and is then loaded according to the parameter definition, /home/usr/test.ash is executed as a child job.
- If this parameter is used to execute a child job, the command name output to the job execution logs is the JP1/Advanced Shell command (adshexec or adshexecsub).

  However, if the command is executed in any of the manners listed below, the program path name existing before this parameter is applied is output as the command name to the job execution logs:

  - Execution of a separate process by using a pipe (|)
  - Execution of a separate process by using a command substitution ($(), ``)
  - Execution of a background process by using |&
  - Execution of a subshell by grouping a single command
  - Background execution by using &
- A program path specified in the argument of the exec, command, or eval standard shell command or the time reserved script command is also subject to this parameter.
- This parameter is applied to the standard shell commands, extended shell commands, functions, and external commands in job definition scripts.
- Specify a path name and the arguments for the executable program within the permitted maximum line length for the environment file.

Example

These examples execute $HOME/script/test.ash in the job definition script as a child job. The contents of the job definition script and environment file are shown below. The information specified in the CHILDJOB_PGM parameter is underlined.

- Example 1

  Contents of environment file:

  ```
  #-adsh_conf CHILDJOB_PGM /bin/sh
  ```

  Contents of job definition script:

  ```
  /bin/sh $HOME/script/test.ash
  ```

- Example 2

  Contents of environment file:

  ```
  #-adsh_conf CHILDJOB_PGM /opt/jp1as/bin/adshexec
  ```

  Contents of job definition script:

  ```
  /opt/jp1as/bin/adshexec $HOME/script/test.ash
  ```

- Example 3

  Contents of environment file:

  ```
  #-adsh_conf CHILDJOB_PGM sh -x
  ```

  Contents of job definition script:

  ```
  sh -x $HOME/script/test.ash
  ```

- Example 4

  Contents of environment file:

  ```
  #-adsh_conf CHILDJOB_PGM /usr/bin/env ksh
  ```

  Contents of job definition script:

  ```
  /usr/bin/env ksh $HOME/script/test.ash
  ```

CHILDJOB_SHEBANG parameter (defines an executable program path for job definition script files that are to be executed as child jobs)

Format

```
#-adsh_conf CHILDJOB_SHEBANG path-name
```

Description

This parameter defines an executable program path that is specified, following `#!`, as a job definition script file to be run as a child job. The character string immediately following `#!` through the end of the line is treated as the comparison path name.

When a job definition script file is specified as a command name in another job definition script and the path name specified in this parameter appears on the first line of the job definition script file in the format `#!`*path-name*, the command outputs the `KNAX6831-I` message to the job execution logs and executes the job definition script file as a child job. You can use the `JOBLOG_SUPPRESS_MSG` parameter to suppress output of the `KNAX6831-I` message to the job execution logs.

The `CHILDJOB_SHEBANG` parameter has the following two default definitions:

Default definition	Output mode when the child job starts
`/opt/jp1as/bin/adshexec`	Operation is performed according to the specification of the `OUTPUT_MODE_CHILD` parameter.
`/opt/jp1as/bin/adshexec -mMINIMUM`	Operation is performed in the minimum output mode.

For details about the default definitions for the `CHILDJOB_SHEBANG` parameter, see *3.2.3(1)(b) Executing child jobs by using a default definition for the parameter*.

Operands

path-name **~<any character string>((1 to 1,023 bytes))**

Defines an executable program path that, when specified beginning with `#!` in a job definition script file, is to be executed as a child job.

Notes

- If this parameter is defined in both the system environment file and the job environment file, both definitions take effect. However, if the total number of times this parameter is specified in the system environment file and job environment file combined exceeds 255, an error occurs.
- Do not include the leading characters `#!` in the path name specification in this operand.
- If only a forward slash (`/`) is specified in this operand as the path name, the command will terminate with an error during parameter analysis.
- The command name output to the job execution logs becomes the path of the directly specified script file.
- If a file that can be executed by using this parameter is evaluated by using the operator `-x` for evaluating file attributes, the result will be true. (Windows only)
- Specifying the same executable program path more than once does not result in an error.
- A file that is specified in the argument of the `exec`, `command`, or `eval` standard shell commands or the `time` reserved script command is also subject to this parameter.

Example

The following examples execute a job definition script file that contains a specific code at the beginning as a child job:

- Executing a file that begins with `#!/bin/sh` or `#!/bin/ksh` as a child job

```
#-adsh_conf  CHILDJOB_SHEBANG  /bin/sh
#-adsh_conf  CHILDJOB_SHEBANG  /bin/ksh
```

- Executing a file that begins with `#!/bin/ksh -x` as a child job

```
#-adsh_conf  CHILDJOB_SHEBANG  "/bin/ksh -x"
```

- Executing a file that begins with `#!/usr/bin/env ksh` as a child job

```
#-adsh_conf  CHILDJOB_SHEBANG  "/usr/bin/env ksh"
```

CMDRC_THRESHOLD_DEFINE parameter (defines a return code threshold for a command)

Format

```
#-adsh_conf CMDRC_THRESHOLD_DEFINE command-name threshold
```

Description

This parameter defines a threshold value for the return code for a command executed from a job definition script. Normal termination will be considered to have occurred whenever the command's return code is equal to or smaller than the specified threshold value. You use this parameter when you want normal termination to be considered to have occurred even though the actual return code of the command was not 0.

If the command terminates by receiving a signal, an error termination will be considered to have occurred regardless of the specification of this parameter.

The CMDRC_THRESHOLD_DEFINE parameter can also be used to change the threshold for a UNIX-compatible command for which a threshold has been defined by specification of ENABLE in the CMDRC_THRESHOLD_USE_PRESET parameter.

Operands

command-name **~<command name>((1 to 255 bytes))**

Specifies the name of the command for which the threshold of the return code is to be defined. In Windows, a command name with an extension can also be specified. A command path cannot be specified.

The command types that can be specified are listed below. Other commands are also affected by this parameter if they are executed in a separate process (using a pipe, command substitution, |& or &).

- External command
- UNIX-compatible command
- Shell operation command
- Shell script
- Child job

If specification of a command extension is omitted in Windows, commands and batch files having the same name as the specified name become the targets for threshold control regardless of their extension.

To specify a command name containing a space in Windows, enclose the entire command name in double quotation marks (").

threshold **~<integer>((-1 to 255))**

Specifies the threshold to be used for determining that the return code represents a normal termination. If the return code is greater than the value specified here, it will be assumed that an error termination has occurred.

If -1 is specified, execution of the command will always result in an error termination.

If 255 is specified, the execution result will always be regarded as a normal termination.

Notes

- This parameter applies to the specified command following variable substitution or alias resolution.
- The setting for the successRC attribute of the #-adsh_rc_ignore and #-adsh_step_start commands of the job definition script takes precedence for the command action based on the command threshold specified by this parameter and the return code of the executed command.
- If this parameter is defined in both the system environment file and the job environment file, both definitions take effect. However, if different thresholds are defined for the same command, the last one defined in the job environment file takes effect.
- Although there is no limit to the number of times this parameter can be specified, do not define it for unnecessary commands because specifying it too many times will affect adversely the execution performance of job definition scripts.
- To define a return code threshold for a job definition script to be executed as a child job, use the CHILDJOB_EXT or CHILDJOB_SHEBANG parameter for the child job definition.

If you use the CHILDJOB_PGM parameter to define a child job, the specified return code threshold will not be applied to the child job.

- In Windows, you can specify the command name with or without an extension.

  However, if you specify a command name with an extension and the same command name without an extension, they will be not considered to be the same command.

  - If you use this parameter to specify a threshold for a command name with an extension and another threshold for the same command name without an extension, the first one specified takes effect.
  - The command names that are assumed when ENABLE is specified in the CMDRC_THRESHOLD_USE_PRESET parameter are the names with an extension. Therefore, to use this parameter to change a threshold, you must specify an extension. If the extension is omitted, the command will be registered as a different command.

- Regardless of their specification order, the CMDRC_THRESHOLD_USE_PRESET parameter is always processed first when both it and the CMDRC_THRESHOLD_DEFINE parameter are specified.

Example

- In this example, 10 is defined as the return code threshold for the CHILD_EXEC1.sh child job (normal termination whenever the return code is 10 or smaller):

```
#-adsh_conf  CHILDJOB_EXT sh
#-adsh_conf  CMDRC_THRESHOLD_DEFINE CHILD_EXEC1.sh 10
```

- In this example, 1 is generally defined as the return code threshold for UNIX-compatible commands; for certain other UNIX-compatible commands (acmd, bcmd, ccmd), 20 is defined as the threshold:

```
#-adsh_conf CMDRC_THRESHOLD_USE_PRESET ENABLE
#-adsh_conf CMDRC_THRESHOLD_DEFINE acmd 20
#-adsh_conf CMDRC_THRESHOLD_DEFINE bcmd 20
#-adsh_conf CMDRC_THRESHOLD_DEFINE ccmd 20
```

- (Windows only) In this example, 10 is defined as the return code threshold for the acmd.exe command, and 20 is defined as the threshold for the generic command name acmd:

```
#-adsh_conf  CMDRC_THRESHOLD_DEFINE acmd.exe 10
#-adsh_conf  CMDRC_THRESHOLD_DEFINE acmd 20
```

  In this definition sequence, 10 is applied as the threshold for the acmd.exe command, and 20 is applied as the threshold for acmd commands whose extension is not .exe (such as acmd.bat).

- (Windows only) In this example, 20 is defined as the threshold for the generic command name acmd, and 10 is defined as the return code threshold for the acmd.exe command.

  20 is applied as the return code threshold for commands whose name is acmd (such as acmd.exe and acmd.bat):

```
#-adsh_conf  CMDRC_THRESHOLD_DEFINE acmd 20
#-adsh_conf  CMDRC_THRESHOLD_DEFINE acmd.exe 10
```

  In this definition sequence, the definition for the acmd generic command name, which is described first, is also preferentially applied to the acmd.exe command. Therefore, the threshold for the acmd.exe command also becomes 20.

- In this example, the acmd.exe command is specified twice in the same parameter, with 30 and 20 defined as the thresholds:

```
#-adsh_conf  CMDRC_THRESHOLD_DEFINE acmd.exe 30
#-adsh_conf  CMDRC_THRESHOLD_DEFINE acmd.exe 20
```

In this case, the threshold for the last parameter specified, which is 20, is applied.

- In this example, the acmd.exe command is specified with a threshold of 10 in the system environment file and with a threshold of 20 in the job environment file:

  System environment file specification:

  ```
  #-adsh_conf CMDRC_THRESHOLD_DEFINE acmd.exe 10
  ```

  Job environment file specification:

  ```
  #-adsh_conf CMDRC_THRESHOLD_DEFINE acmd.exe 20
  ```

  In this case, the threshold of 20 specified in the job environment file is applied.

- (Windows only) In this example, 1 is generally defined as the return code threshold for UNIX-compatible commands and then the threshold for the cmp.exe command only is changed to 2:

  ```
  #-adsh_conf CMDRC_THRESHOLD_USE_PRESET ENABLE
  #-adsh_conf CMDRC_THRESHOLD_DEFINE cmp.exe 2
  ```

- (Windows only) In this example, cmp 2 is specified by the CMDRC_THRESHOLD_DEFINE parameter for the cmp.exe command for which a threshold of 1 is defined by the CMDRC_THRESHOLD_USE_PRESET parameter. In this case, however, the threshold cannot be changed from 1 to 2:

  ```
  #-adsh_conf CMDRC_THRESHOLD_USE_PRESET ENABLE
  #-adsh_conf CMDRC_THRESHOLD_DEFINE cmp 2
  ```

  In this case, cmp.exe and cmp are managed as separate commands for which different thresholds are effective.

- If the CMDRC_THRESHOLD_USE_PRESET and CMDRC_THRESHOLD_DEFINE parameters are specified in different environment files, as shown below, the threshold of 2 specified by the CMDRC_THRESHOLD_DEFINE parameter is applied to the cmp.exe command.

  System environment file specification:

  ```
  #-adsh_conf CMDRC_THRESHOLD_DEFINE cmp.exe 2
  ```

  Job environment file specification:

  ```
  #-adsh_conf CMDRC_THRESHOLD_USE_PRESET ENABLE
  ```

CMDRC_THRESHOLD_USE_PRESET parameter (defines a threshold for the return code of a UNIX-compatible command)

Format

```
#-adsh_conf CMDRC_THRESHOLD_USE_PRESET {ENABLE|DISABLE}
```

Description

This parameter defines a return code threshold for assuming normal termination of all the UNIX-compatible commands listed below. You can specify only 0 or 1 for the value.

- cmp command

- `diff` command
- `egrep` command
- `expr` command
- `grep` command
- `sort` command

You must use the `CMDRC_THRESHOLD_DEFINE` parameter to define a different threshold for a specific command.

Command names defined in this parameter are registered with an extension in Windows. For example, the `cmp` command is registered by the OS as follows:

- In Windows edition: `cmp.exe`
- In UNIX edition: `cmp`

If you intend to use the `CMDRC_THRESHOLD_DEFINE` parameter to change a threshold, you must define the command name with an extension.

Operands

`ENABLE`

Specifies `1` as the return code threshold. As a result, commands that terminate with a return code of `1` are also considered to have terminated normally.

A threshold of `1` is also set for commands with the same name as the targeted UNIX-compatible commands.

`DISABLE`

Specifies `0` as the return code threshold.

Notes

- This parameter is applied to a command name following variable substitution or alias resolution.
- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect.
- The setting for the `successRC` attribute of the `#-adsh_rc_ignore` and `#-adsh_step_start` commands of the job definition script takes precedence for the command action based on the command threshold specified by the `ENABLE` specification and the return code of the executed command.
- Regardless of their specification order, the `CMDRC_THRESHOLD_USE_PRESET` parameter is always processed first when both it and the `CMDRC_THRESHOLD_DEFINE` parameter are specified.

Example

- In this example, normal termination is assumed even when the return code of the targeted UNIX-compatible commands is `1`:

```
#-adsh_conf  CMDRC_THRESHOLD_USE_PRESET ENABLE
```

COMMAND_CONV_ARG parameter (defines a rule for converting an argument in job definition scripts during command execution)

Format

```
#-adsh_conf COMMAND_CONV_ARG command-argument-1 command-argument-2
```

Description

This parameter defines a rule for converting an argument in standard shell commands, extended shell commands, functions, extended script commands, reserved script commands, external commands, and user programs used in job definition scripts.

When a job definition script is run, a command argument that matches exactly *command-argument-1* is converted to *command-argument-2*. Both *command-argument-1* and *command-argument-2* must be specified.

If different rules are defined for the same command argument, the first rule defined takes effect.

External scripts specified in `.` (dot) commands and `#-adsh_script` commands are also subject to this argument's conversion if they satisfy the specified rule during execution.

The `KNAX6804-I` or `KNAX6806-I` message is output to the job execution logs as the conversion result.

Operands

command-argument-1 ~**<any character string>((1 to 247 bytes))**

Specifies the command argument before conversion. To specify a command argument containing a space, enclose the entire command argument in double quotation marks (`"`). A value enclosed in double quotation marks cannot consist of only a space, tab character, or null character. None of the following characters is permitted:

`* ? < > | `` (grave accent mark) `$`

If *command-argument-1* is omitted or the value specified in *command-argument-1* is invalid, the command will terminate with an error during parameter analysis.

command-argument-2 ~**<any character string>((1 to 247 bytes))**

Specifies the command argument after conversion. To specify a command argument containing a space, enclose the entire command argument in double quotation marks (`"`). A value enclosed in double quotation marks cannot consist of only a space, tab character, or null character. None of the following characters is permitted:

`* ? < > | `` (grave accent mark) `$`

If *command-argument-2* is omitted or the value specified in *command-argument-2* is invalid, the command will terminate with an error during the parameter analysis.

Notes

- If this parameter is defined in both the system environment file and the job environment file, both definitions take effect. However, if the total number of times this parameter is defined in the system environment file and the job environment file combined exceeds 255, an error occurs.
- If the command argument obtained after conversion is the same as the command argument before conversion in a subsequent parameter and the `eval` command containing the argument that satisfies the rule is executed, conversion is performed twice, once during execution of the `eval` command and once during execution of the command of that argument, as shown in an example below.

  Example:

Contents of environment file

```
#-adsh_conf COMMAND_CONV_ARG /tmp /var/tmp        <--1.
#-adsh_conf COMMAND_CONV_ARG /var/tmp /jp1as/tmp  <--2.
```

Contents of job definition script

```
eval cd /tmp
```

In this example, the following conversions occur on the corresponding lines in the environment file:

1. The `eval` command is executed and `/tmp` is converted to `/var/tmp`.
2. The `cd` command is executed and `/var/tmp` is converted to `/jp1as/tmp`.

- The argument obtained after conversion by this parameter is interpreted as a single character-string argument even if it contains characters that match the `IFS` shell variable, which is the string separator. Therefore, if the character string obtained as a result of parameter conversion contains a metacharacter and the command to be executed is not `eval`, that character is treated as a normal character, not a metacharacter.

  Contents of environment file (`IFS` shell variable is the space)

```
#-adsh_conf COMMAND_CONV_ARG "D:\JP1AS" "C:\\Documents and Settings"  <--
Argument conversion rule 1
#-adsh_conf COMMAND_CONV_ARG "A=1" "A=1 &" <-- Argument conversion rule 2
```

  Contents of job definition script

```
cd "D:\JP1AS"       <--1.
readonly A=1        <--2.
eval cd "D:\JP1AS"  <--3.
eval readonly A=1   <--4.
```

1. The argument is converted as follows according to conversion rule 1:
   `cd "C:\\Documents and Settings"`
2. The character string is separated as follows according to conversion rule 2:
   `readonly A=1 &`
3. When the `eval` command is executed, the argument is converted as follows according to conversion rule 1:
   `eval cd "C:\\Documents and Settings"`
   However, when the `cd` command is executed, the argument is separated and interpreted as follows:
   `cd C:\Documents and Settings`
4. When the `eval` command is executed, the argument is converted as follows according to conversion rule 2:
   `eval readonly A=1 &`
   However, when the `readonly` command is executed, the argument is separated and interpreted as follows:
   `readonly A=1 &`

- This parameter performs conversion in such a manner that variable substitution and file name substitution are resolved. Therefore, if a character string containing a wildcard is specified as the argument, the character string obtained after substitution by the wildcard is recognized as the command's argument.
- If this parameter is applied to the `test` command, the interpretation of the character string in the argument during command execution depends on the format used. In the `[[ ]]` format, if a variable substitution is specified for array elements, conversion by this parameter is not applied. To use this parameter to convert a command's argument by using the `test` command, we recommend that you use the `test` or `[ ]` format.

  Example:

Contents of environment file

```
#-adsh_conf PATH_CONV_ENABLE \\ :
#-adsh_conf COMMAND_CONV_ARG "/tmp2" "/tmp"  <-- Argument conversion rule
1
#-adsh_conf COMMAND_CONV_ARG "ARY[1]" "/tmp"  <-- Argument conversion
rule 1
```

Contents of job definition script

```
ARY[0]="/var"    ← Stores "/var", "/tmp2", and "/home" in the array ARY.
ARY[1]="/tmp2"
ARY[2]="/home"
id1=1
[[ -d ${ARY[$id1]} ]]  <-- Replaces ${ARY[$id1]} through "/tmp2".
Converted to "/tmp" according to argument conversion rule 1.
[[ -d   ARY[$id1] ]]  <-- "ARY[$id1]" becomes the argument because
substitution is not performed.
Not subject to conversion because "$" cannot be specified.
[ -d ${ARY[$id1]} ]  <-- Replaces ${ARY[$id1]} through "/tmp2".
The result of substitution is converted to "/tmp" according to argument
conversion rule 1.
[ -d   ARY[$id1] ]  <-- Replaces ${ARY[$id1]} through "ARY[1]"
Converted to "/tmp" according to argument conversion rule 1.
```

- If any of the commands listed below is executed, the command specified in the argument is actually executed, but this parameter performs comparison and conversion also on the command specified in the argument:
  - `builtin` command
  - `command` command
  - `eval` command
  - `exec` command
  - `time` reserved command

  Example:

  In this example, `pwd` specified in the argument of the `builtin` command is actually executed, but the `readonly` command is executed according to the definition in the environment file.

  Contents of environment file

  ```
  #-adsh_conf COMMAND_CONV_ARG pwd readonly
  ```

  Contents of job definition script

  ```
  builtin pwd
  ```

- The conversion rules are searched in the order they are defined and the first conversion rule that matches the conversion condition is applied.
- If conversions are defined for the same path name by using this parameter and the `PATH_CONV` parameter, the conversion defined by the `PATH_CONV` parameter is performed first. To convert the path name obtained after conversion by the `PATH_CONV` parameter further by the `COMMAND_CONV_ARG` parameter, specify the path name converted by the `PATH_CONV` parameter.
- Use this parameter carefully, because each time a command is executed, all arguments specified in `COMMAND_CONV_ARG` parameters are scanned. Therefore, if you specify this parameter many times, the job definition script's execution time might be affected adversely.

- To specify a character string containing a backslash (`\`), not a metacharacter, in *command-argument-2*, specify `\\` instead of `\`.

Example

- This example converts `"/tmp"` to `"C:\temp"` to run in Windows a job definition script created for UNIX:

```
#-adsh_conf PATH_CONV_ENABLE / :
#-adsh_conf COMMAND_CONV_ARG /tmp "C:\\temp"
```

- This example converts `"C:\temp"` to `"/tmp"` to run in UNIX a job definition script created for Windows:

```
#-adsh_conf PATH_CONV_ENABLE \\ ;
#-adsh_conf COMMAND_CONV_ARG "C:\temp" /tmp
```

ESCAPE_SEQ_ECHO_DEFAULT parameter (defines the action of the echo command when the escape-character option is omitted)

Format

```
#-adsh_conf ESCAPE_SEQ_ECHO_DEFAULT {YES|NO}
```

Description

This parameter defines the action the `echo` command is to take for escape characters when the escape-character option (`-E` or `-e`) is omitted in the `echo` command.

When `YES` is specified in this parameter, escape characters are interpreted, even if the `-e` option is not specified in the `echo` command. When `NO` is specified, escape characters are handled as directed by the `echo` command specification.

The specification in this parameter is ignored when the `-E` or `-e` option is specified in the `echo` command.

Operands

`YES`

Specifies that the `echo` command is to be used to interpret escape characters, even if the `-e` option is not specified in the `echo` command.

`NO`

Specifies that the `echo` command is not to be used to interpret escape characters, unless the `-e` option is specified in the `echo` command.

Notes

- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect.
- A parameter error occurs if this parameter is defined multiple times in the same environment file on the same host.
- If you want to interpret ASCII code characters expressed in 1- or 2-digit hexadecimal notation as escape characters, you must also specify `YES` in the `ESCAPE_SEQ_ECHO_HEX` parameter.

Example

- In this example, escape characters are interpreted, even though the `-e` option is not specified in the `echo` command:

  ```
  #-adsh_conf ESCAPE_SEQ_ECHO_DEFAULT YES
  ```

ESCAPE_SEQ_ECHO_HEX parameter (specifies whether ASCII code characters in hexadecimal notation are to be interpreted as escape characters)

Format

```
#-adsh_conf ESCAPE_SEQ_ECHO_HEX {YES|NO}
```

Description

This parameter specifies whether ASCII code characters in hexadecimal notation are to be interpreted as escape characters by the `echo` command. This parameter is valid when either of the following conditions is satisfied:

- The `-e` option is specified in the `echo` command.
- Neither the `-e` nor the `-E` option is specified in the `echo` command but `YES` is specified in the `ESCAPE_SEQ_ECHO_DEFAULT` parameter.

Operands

`YES`

Specifies that ASCII code characters expressed in 1- or 2-digit hexadecimal notation are to be interpreted as escape characters.

`NO`

Specifies that ASCII code characters expressed in 1- or 2-digit hexadecimal notation are not to be interpreted as escape characters.

Notes

- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect.
- A parameter error occurs if this parameter is defined multiple times in the same environment file on the same host.
- If a value that is outside the ASCII code range is specified as an argument in the `echo` command, the content output will be consistent with the character encoding specified for the terminal. Consequently, unprintable characters might produce an incorrect output.

Example

- In this example, ASCII code characters expressed in 1- or 2-digit hexadecimal notation are interpreted as escape characters by the `echo` command.

  Contents of environment file:

  ```
  #-adsh_conf ESCAPE_SEQ_ECHO_HEX YES
  ```

Contents of job definition script:

```
STR="\x48\x49\x54\x41\x43\x48\x49"
echo -e $STR
```

Contents to be output to the standard output:

```
HITACHI
```

- In this example, ASCII code characters expressed in 1- or 2-digit hexadecimal notation are not to be interpreted as escape characters by the `echo` command.

  Contents of environment file:

```
#-adsh_conf ESCAPE_SEQ_ECHO_DEFAULT YES
#-adsh_conf ESCAPE_SEQ_ECHO_HEX NO
```

  Contents of job definition script:

```
STR="\t\x48\x49\x54\x41\x43\x48\x49"
echo $STR
```

  Contents to be output to the standard output:

```
tab-character\x48\x49\x54\x41\x43\x48\x49
```

EVENT_COLLECT parameter (specifies whether the operation information acquisition functionality is to be enabled for job definition scripts)

Format

```
#-adsh_conf EVENT_COLLECT {YES|NO}
```

Description

This parameter specifies whether the operation information acquisition functionality is to be enabled for job definition scripts. When the operation information acquisition functionality is disabled for job definition scripts, event files are not created in the spool directory.

This parameter is ignored in JP1/Advanced Shell - Developer, and event files are not created.

This parameter is also ignored when a CUI debugger is used, and event files are not created.

Operands

`YES`

Enables the operation information acquisition functionality for job definition scripts.

Job definition script operation information is collected and output to event files.

`NO`

Disables the operation information acquisition functionality for job definition scripts.

No event files are created.

Notes

- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect.
- If this parameter is defined more than once for the same host in the same environment file, a parameter error results.

Example

- This example disables the operation information acquisition functionality for job definition scripts.

```
#-adsh_conf EVENT_COLLECT NO
```

export parameter (defines an environment variable)

Format

```
export environment-variable-name=environment-variable-value
```

Description

This parameter defines an environment variable that is to take effect when job definition scripts are executed.

Arguments

environment-variable-name **~<environment variable name>((1 to 255 bytes))**

Specifies a name for the environment variable being defined.

environment-variable-value **~<any character string>((0 through 1,023 bytes))**

Specifies the value to be set in the environment variable.

You can specify an environment variable value containing a space by enclosing it in double-quotation marks (`"`) or single quotation marks (`'`) or by specifying an escape character (`\`) before the space. If a character string enclosed in double quotation marks (`"`) contains the escape character (`\`), all the characters following `\` are treated as escaped characters. Therefore, to specify `\` as a part of a character string enclosed in double quotation marks, specify `\\` instead of `\`.

You can insert the current value of the `PATH` environment variable by specifying `${PATH}` in a desired character string. Specify `${PATH}` at the target location in the following format:

```
export environment-variable-name=[any-character-string]${PATH}[any-character-string]
```

`${PATH}` is replaced with the value of the `PATH` environment variable regardless of the specification of double-quotation marks (`"`), single quotation marks (`'`), and escape characters (`\`). The specified double-quotation marks (`"`), single quotation marks (`'`), or escape characters (`\`) take effect on the entire character string inserted as the value of the `PATH` environment variable, and then the value is set in the environment variable.

If you define the `PATH` environment variable in Windows, enclose the value of the `PATH` environment variable in single quotation marks (`'`) so that the resulting value containing a space or `\` is interpreted correctly.

Notes

- If this parameter is defined in both the system environment file and the job environment file, the environment variable is set in the following order:

    - Environment variable specified in the system environment file
    - Environment variable specified in the job environment file
- Even when a value for the `LANG` environment variable is specified by the `export` parameter, the locale of the process of the `adshexec` command that loaded this environment-setting parameter is not changed. (UNIX only)

Example

- This example sets the value of environment variable `BBB` in environment variable `AAA`:

```
export AAA=BBB
```

- This is an example of incorrectly setting the value of environment variable `BBB` in environment variable `AAA`:

```
AAA=BBB
export AAA
```

- This example adds `/opt/jp1as/bin` to the existing `PATH` environment variable:

```
export PATH=/opt/jp1as/bin:${PATH}
```

  The colon (`:`) is used as the path separator.

- This example adds `C:\Program Files\HITACHI\JP1AS\JP1ASE\bin` to the existing `PATH` environment variable:

```
export PATH='C:\Program Files\HITACHI\JP1AS\JP1ASE\bin;${PATH}'
```

  The semicolon (`;`) is used as the path separator.

HOSTNAME_JP1IM_MANAGER parameter (specifies the operation management server on which JP1/IM - Manager is running that is to be the destination of JP1 events)

Format

```
#-adsh_conf HOSTNAME_JP1IM_MANAGER host-name-of-operation-management-server-on-which-JP1/IM-Manager-is-running
```

Description

When the user-reply functionality is used, this parameter specifies the host name of the operation management server on which JP1/IM - Manager is running that is to be the destination of JP1 events.

Operands

host-name-of-operation-management-server-on-which-JP1/IM-Manager-is-running **~<symbolic name>((1 to 255 characters))<<physical host name of the batch operation server on which JP1/Advanced Shell is running>>**

Specifies the host name of an operation management server on which JP1/IM - Manager is running. The user will be able to perform on the specified host the following tasks related to the user-reply functionality:

- Checking JP1 events from the JP1/IM - View that is connected to JP1/IM - Manager.
- Replying to reply-waiting events.

Note the following about specifying a host name:

- If JP1/IM - Manager is not running on the specified host, JP1 events output by JP1/Integrated Management will not be displayed on JP1/IM - View, and the `adshread` command will remain in reply-waiting status. If this happens, either start JP1/IM - Manager on the host specified in this parameter or execute the `adshchmsg` command to enter replies manually.
- You specify the operation management server on which JP1/IM - Manager is running by specifying its host name. If its IP address is specified, JP1 events will be issued to JP1/IM, but they will not be treated as reply-waiting events. Also make sure that the specified host name is resolved.

If this parameter is omitted, JP1/Advanced Shell assumes the host name that is displayed when the `hostname` command is executed on the batch operation server on which JP1/Advanced Shell is running.

Notes

- Do not specify this parameter in the job environment file.
- When you specify this parameter, determine the maximum number of concurrent reply-request messages that can be output to each physical host or logical host and then specify that value in the `USERREPLY_WAIT_MAXCOUNT` parameter.
- Make sure that the host name of the batch operation server on which JP1/Advanced Shell is run can be resolved on the operation management server on which this parameter is specified.
- This parameter is ignored when debug execution is performed with the standard input and output specified as the input source and output destination, respectively, for the user-reply functionality.

JOBEXECLOG_PRINT parameter (defines the job execution log contents to be output to the standard error output when a job terminates)

Format

```
#-adsh_conf JOBEXECLOG_PRINT {JOBLOG_SCRIPT_STDERR|STDERR}
```

Description

This parameter defines the contents of the job execution log that are to be output to the standard error output when a job terminates. The contents defined by this parameter are displayed, for example, on the screen of the terminal from which the `adshexec` command was executed and in the Execution Result Details dialog box of JP1/AJS - View.

When JP1/Advanced Shell - Developer or the `adshexec -d` command executes debugging, the contents of the job execution log are not output to the standard error output when the job terminates, regardless of the specification of this parameter.

Operands

`JOBLOG SCRIPT STDERR`

Specifies that the following contents are to be output to the standard error output when the job terminates:

- `JOBLOG` contents (messages indicating the job execution status, such as command execution results and file allocation results)

  Also includes the `JOBLOG` contents of child jobs to be output to the standard error output of the root job.
- Job definition script

- Contents of the standard error output during job execution

STDERR

Specifies that the only contents to be output to the standard error output when the job terminates are the standard error output during job execution.

In addition to the standard information messages, messages other than those whose message type is I (information messages that are normally output to the JOBLOG file) are also output to the standard error output.

The JOBLOG contents of child jobs to be output to the standard error output of the root job are not output.

Notes

- If this parameter is defined in both the system environment file and the job environment file, both definitions take effect.
- If any of the following occurs with respect to this parameter in the same environment file, the job controller of JP1/Advanced Shell terminates with an error without executing the job:
  - This parameter is specified more than once.
  - No operand is specified.
  - More than three operands are specified.
  - An operand other than JOBLOG SCRIPT STDERR or STDERR is specified.
  - The specification order of the JOBLOG SCRIPT STDERR operand is invalid.
- If a job is operating in the simple output mode or the minimum output mode, no job execution log is output to the standard error output regardless of the specification of this parameter.

Example

- In this example, only the contents of the standard error output during job execution are output to the standard error output:

```
#-adsh_conf JOBEXECLOG_PRINT STDERR
```

JOBLOG_SUPPRESS_MSG parameter (defines a message that is not to be output to job execution logs)

Format

```
#-adsh_conf JOBLOG_SUPPRESS_MSG message-ID
```

Description

This parameter specifies the message ID of a message that is not to be output to the job execution logs.

To specify multiple messages, specify this parameter for each applicable message ID. The parameters can be specified in any order.

Specifying the same message ID in more than one parameter will not result in an error. JP1/Advanced Shell assumes that each message ID is specified only once.

The following table shows the relationship between this parameter and message output.

No.	Message output destination	Specification of the ID of a message to be suppressed	
		Specified	Omitted
1	Job execution log files on spool	N	Y
2	Debugger's console	N	Y
3	Editor's console	N	Y
4	System execution logs	N	Y
5	Other output destination (such as the standard output)	N	N

Legend:

Y: Message is output.

N: Message is not output.

Operands

message-ID ~**<message ID>((10 bytes))**

Specifies the message ID of a message whose output to job execution logs is to be suppressed. The specifiable message IDs are as follows:

Suppressible message content		Specifiable message IDs

Built-in commands	Updating of shell variable	`KNAX6110-I`
		`KNAX6111-I`
		`KNAX6120-I`
		`KNAX6121-I`

	Standard shell command	`KNAX6112-I`
		`KNAX6113-I`
		`KNAX6122-I`
		`KNAX6123-I`

External commands	Executable file	`KNAX6116-I`
		`KNAX6117-I`
		`KNAX6126-I`
		`KNAX6127-I`

	Script file	

Extended shell commands		`KNAX6114-I`
		`KNAX6115-I`
		`KNAX6124-I`
		`KNAX6125-I`

Parameters related to execution of child jobs	`CHILDJOB_PGM` parameter	`KNAX6830-I`
	`CHILDJOB_SHEBANG` parameter	`KNAX6831-I`
	`CHILDJOB_EXT` parameter	`KNAX6832-I`

Notes

- If this parameter is defined in both the system environment file and the job environment file, both definitions take effect.

Example

- This example suppresses output of the `KNAX6110-I` and `KNAX6111-I` messages to the job execution logs:

```
#-adsh_conf JOBLOG_SUPPRESS_MSG KNAX6110-I
#-adsh_conf JOBLOG_SUPPRESS_MSG KNAX6111-I
```

KSH_ENV_READ parameter (defines whether the ENV shell variable is to be read)

Format

In Windows and Linux

```
#-adsh_conf KSH_ENV_READ {YES | NO}
```

In AIX, HP-UX, and Solaris

```
#-adsh_conf KSH_ENV_READ {YES | NO}
```

Description

This parameter specifies whether the `ENV` shell variable is to be read when the job controller starts and the script file indicated by the variable's value is to be executed.

Operands

`YES`

Read the `ENV` shell variable when the job controller starts.

`NO`

Do not read the `ENV` shell variable when the job controller starts.

Notes

- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect.

LOG_DIR parameter (defines the path name of the directory to which system execution logs are to be output)

Format

```
#-adsh_conf LOG_DIR path-name
```

Description

Messages about the concurrently executing jobs are collected as system execution logs in a single file. This parameter defines the path name of the directory to which these system execution logs are to be output.

Operands

path-name

Windows execution environment: ~<path name>((1 to 128 bytes))<<*shared-documents-folder*`\Hitachi\JP1AS\JP1ASE\log`**>>**

Windows development environment: ~<path name>((1 to 128 bytes))<<*shared-documents-folder*`\Hitachi\JP1AS\JP1ASD\log`**>>**

UNIX: ~<path name>((1 to 512 bytes))<<`/opt/jp1as/log`**>>**

Specifies the path name of the directory to which system execution logs are to be output.

Notes

- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect.
- Do not specify in this parameter a directory in a file system that is not supported by JP1/Advanced Shell. For details about the file systems that are not supported by JP1/Advanced Shell, see *2.6.19(2) File systems*.
- Do not include any of the following characters in the path name: `& ( ) [ ] { } ^ = ; ! ' + , ` ~ # %`. If any of these characters is included, JP1/Advanced Shell will not function correctly.

Supplementary information

- You can use multiple environments on the same host by specifying separate directories in the corresponding parameters.

LOG_FILE_CNT parameter (defines the number of files to be used to back up system execution logs)

Format

```
#-adsh_conf LOG_FILE_CNT number-of-files
```

Description

This parameter defines the number of files to be used to back up system execution logs.

Operands

number-of-files **~<unsigned integer>((1 to 64))<<4>>**

Specifies the number of files to be used to back up system execution logs.

Notes

- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect. If the value of the `LOG_DIR` parameter and the value specified in the system environment file are the same, specifying a different value in this parameter results in an error (even when the system environment file's default value is assumed).

Supplementary information

- If multiple users output system execution logs to the same file, the `LOG_FILE_CNT` and `LOG_FILE_SIZE` parameter values specified by the last user that started output of system execution logs take effect. We recommend that all users who output system execution logs to the same file use the same parameter values.

LOG_FILE_SIZE parameter (defines the size of a file to which system execution logs are to be output)

Format

```
#-adsh_conf LOG_FILE_SIZE file-size
```

Description

This parameter defines the size of a file to which system execution logs are to be output.

Operands

file-size ~**<unsigned integer>((1 to 16))<<2>>**

Specifies in megabytes the size of a file to which system execution logs are to be output.

Notes

- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect. If the value of the `LOG_DIR` parameter and the value specified in the system environment file are the same, specifying a different value in this parameter results in an error (even when the system environment file's default value is assumed).

Supplementary information

- If multiple users output system execution logs to the same file, the `LOG_FILE_CNT` and `LOG_FILE_SIZE` parameter values specified by the last user that started output of system execution logs take effect. We recommend that all users who output system execution logs to the same file use the same parameter values.

OUTPUT_MODE_CHILD parameter (defines the method for outputting the execution results of a child job)

Format

```
#-adsh_conf OUTPUT_MODE_CHILD {EXTENDED|SIMPLE|MINIMUM}
```

Description

This parameter specifies one of the following modes for output of the execution results of a child job:

- Expansion output mode (default)
- Simple output mode

- Minimum output mode

By selecting the simple output mode, you can suppress information messages and output only the command execution results, making it easier for other programs to use the job execution results. By selecting the minimum output mode, you suppress output of more messages than when you use the simple output mode.

Standard output and standard error output are output to the destinations that were set when the child job was started.

When a child job is executed with the `-m` option specified in the `adshexec` command for starting the child job, the `-m` option takes precedence over the specification of this parameter.

Operands

`EXTENDED`

Specifies the extended output mode. Information is output after execution of the child job as follows:

- The job execution log is output to the standard error output when the child job terminates.
- All messages to be output to the standard error output and the standard output of JP1/Advanced Shell are output.

`SIMPLE`

Specifies the simple output mode. Information is output after execution of the child job as follows:

- The job execution log is not output to the standard error output when the child job terminates.
- Only error messages are output to the standard error output and the standard output of JP1/Advanced Shell. Error messages that are output to `JOBLOG` are output to the standard error output.

`MINIMUM`

Specifies the minimum output mode. Information is output after execution of the child job as follows:

- No job execution log is output to the standard error output when the child job terminates.
- Only error messages, excluding notifications of command, job step, and job termination codes, are output to the standard error output and the standard output of JP1/Advanced Shell. Messages notifying receipt of signals and events are also suppressed. Unlike the simple output mode, the suppressed messages are not output to `JOBLOG` under the spool job directory.

  Error messages that are output to `JOBLOG` are output to the standard error output.

JP1/Advanced Shell inherits the parent process's output mode until the analysis of environment setting parameters is finished.

Notes

- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect.

OUTPUT_MODE_ROOT parameter (specifies the method for outputting the execution results of a root job)

Format

```
#-adsh_conf OUTPUT_MODE_ROOT {EXTENDED|SIMPLE|MINIMUM}
```

Description

This parameter specifies one of the following modes for output of the execution results of a root job:

- Expansion output mode (default)
- Simple output mode
- Minimum output mode

By selecting the simple output mode, you can suppress information messages and output only the command execution results, making it easier for other programs to use the job execution results. By selecting the minimum output mode, you suppress output of more messages than when you use the simple output mode.

When a root job is executed with the `-m` option specified in the `adshexec` command for starting the root job, the `-m` option takes precedence over the specification of this parameter.

Operands

`EXTENDED`

Specifies the extended output mode. Information is output after execution of the root job as follows:

- Standard output and standard error output are redirected to a file on the spool.
- The job execution log is output to the standard error output when the job terminates.
- All messages to be output to the standard error output and the standard output of JP1/Advanced Shell are output.
- During debugging, `JOBLOG` is output to the standard error output at suitable times.

`SIMPLE`

Specifies the simple output mode. Information is output after execution of the root job as follows:

- The job execution log is not output to the standard error output when the job terminates.
- Standard output and standard error output are output to the destinations that were set when the job was started.
- Only error messages are output to the standard error output and the standard output of JP1/Advanced Shell. Error messages that are output to `JOBLOG` are output to the standard error output.
- During debugging, `JOBLOG` is not output to the standard error output, except for error messages. All messages to be output to the standard error output and the standard output of JP1/Advanced Shell are output. Messages other than the error messages at the time of debugging termination are output.

`MINIMUM`

Specifies the minimum output mode. Information is output after execution of the root job as follows:

- No job execution log is output to the standard error output when the job terminates.
- The standard output and standard error output are output to the destinations that were set when the job was started.
- Only error messages, excluding notifications of command, job step, and job termination codes, are output to the standard error output and the standard output of JP1/Advanced Shell. Messages notifying receipt of signals and events are also suppressed. Unlike the simple output mode, the suppressed messages are not output to `JOBLOG` under the spool job directory.

  Error messages that are output to `JOBLOG` are output to the standard error output.
- During debugging, `JOBLOG` is not output to the standard error output, except for error messages. All messages to be output to the standard error output and the standard output of JP1/Advanced Shell are output. Messages are not output except for the error messages at the time of debugging termination.

Notes

- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect.
- When `SIMPLE` or `MINIMUM` is specified, the standard output is not redirected to a file on the spool, even if a command is executed with `SPOOL` specified in the `OUTPUT_STDOUT` parameter or with `adshexec -s SPOOL` specified.

OUTPUT_STDOUT parameter (defines the destination for the root job standard output)

Format

```
#-adsh_conf OUTPUT_STDOUT {SPOOL | PARENT}
```

Description

This parameter defines the destination for the root job's standard output. Child jobs assume that `PARENT` is specified in this option. If this option is omitted, the root job assumes that `SPOOL` is specified in this option.

If the `adshexec` command is executed with the `-s` option specified, the `-s` option takes precedence over this parameter.

If the root job is running in the simple output mode or the minimum output mode (specified by the `OUTPUT_MODE_ROOT` parameter or the `-m` option of the `adshexec` command), the standard output that was set when the process started is inherited regardless of the specification of the `OUTPUT_STDOUT` parameter or the `-s` option of the `adshexec` command.

Operands

`{ SPOOL | PARENT }`

Specifies one of the following values as the output destination for the root job's standard output:

- `SPOOL`

  Sets the root job's standard output to a file on the spool.
- `PARENT`

  Sets the root job's standard output to the destination inherited from the parent process when the process started. If the parent process does not redirect the output destination, the output destination of the parent process is used.

Notes

- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect.

PATH_CONV parameter (defines the details of path conversion)

Format

```
#-adsh_conf PATH_CONV path-name-1 path-name-2
```

Description

This parameter defines path names before and after conversion in job definition scripts.

In character strings separated by the path separator defined in the `PATH_CONV_ENABLE` parameter, a character string's leading part that matches *path-name-1* is converted to *path-name-2* when the job definition script is executed. This conversion is also performed on the path separator and directory separator defined in the `PATH_CONV_ENABLE` parameter.

In Windows, the conversion result varies depending on the path conversion rule selected by the `PATH_CONV_RULE` parameter. For details, see *PATH_CONV_RULE parameter (defines a rule for converting file paths) (Windows only)*.

This parameter is not valid when the `PATH_CONV_ENABLE` parameter is not defined. If this parameter is defined multiple times, the parameters are searched in the order they are defined in the environment file and the first definition satisfying the conversion condition is applied.

Operands

path-name-1 **~<path name>((1 to 247 bytes))**

Specifies the path before conversion. To specify a value containing a space, enclose the value in double quotation marks (`"`).

In Windows, you can specify a path in UNIX for *path-name-1*. In UNIX, you can specify a path in Windows for *path-name-1*. If you intend to use `\`, specify `\\` because JP1/Advanced Shell handles `\` as an escape character. Note that none of the following characters is permitted:

`* ? < > |` `` ` `` (grave accent mark) `$`

A path name must contain a directory separator. Use the following for the directory separator:

- In Windows, specify a forward slash (`/`).
- In UNIX, use two consecutive backslashes (`\\`).

path-name-2 **~<path name>((1 to 247 bytes))**

Specifies the path after conversion. To specify a value containing a space, enclose the value in double quotation marks (`"`). If you intend to use `\`, specify `\\` because JP1/Advanced Shell handles `\` as an escape character. Note that none of the following characters is permitted:

`* ? < > |` `` ` `` (grave accent mark) `$`

A path name must contain a directory separator. Use the following for the directory separator:

- For execution in Windows, specify two consecutive backslashes (`\\`).
- For execution in UNIX, use a forward slash (`/`).

Notes

- If this parameter is defined in both the system environment file and the job environment file, both definitions take effect. However, if the total number of times this parameter is specified in the system environment file and the job environment file combined exceeds 255, an error occurs.

- Conversion by this parameter is performed one line at a time. Therefore, if the part of a job definition script that corresponds to a path name contains an end-of-line character, the correct conversion cannot be performed.
- Character strings in comments are also converted.
- The conversion rules are searched in the order they are defined and only the first rule satisfying the conversion condition is applied.
- If `DELETE` is specified in the `SPOOLJOB_CHILDJOB` parameter, no script image is output for a job definition script that is executed as a child job. Therefore, if this parameter is used to convert the path name of a job definition script that is executed as a child job, the conversion results will not be output.
- According to path conversion rule 2, you cannot nest single quotation marks (`'`) inside a range enclosed in double quotation marks (`"`). If they are specified, they will be subject to path conversion.

Example

- This example uses the `PATH_CONV` parameter to execute in Windows job definition scripts created for UNIX:

```
#-adsh_conf PATH_CONV /home/hitachi "C:\\hitachi"
#-adsh_conf PATH_CONV_ENABLE / :
#-adsh_conf PATH_CONV /tmp/jp1as "D:\\jp1as_tmp"
#-adsh_conf PATH_CONV /tmp "C:\\temp"
```

- This example uses the `PATH_CONV` parameter to execute in UNIX job definition scripts created for Windows:

```
#-adsh_conf PATH_CONV "C:\\hitachi" /home/hitachi
#-adsh_conf PATH_CONV_ENABLE \\ ;
#-adsh_conf PATH_CONV "D:\\jp1as_tmp" /tmp/jp1as
#-adsh_conf PATH_CONV "C:\\temp" /tmp
```

PATH_CONV_ACCESS parameter (defines path conversion details when files are input and output)

Format

```
#-adsh_conf PATH_CONV_ACCESS path-name-1 path-name-2
```

Description

This parameter defines the path names before and after conversion for converting file path names in job definition scripts when files are input and output.

If an input or output operation occurs while a job definition script is running and the path name of the file subject to the input or output operation matches the specified *path-name-1*, this parameter converts it to *path-name-2*. Both *path-name-1* and *path-name-2* must be specified.

If different rules are defined for the same file path name, the first rule defined takes effect.

External scripts specified in `.` (dot) commands and `#-adsh_script` commands are not subject to conversion by the `COMMAND_CONV_ARG` parameter. For details about the `COMMAND_CONV_ARG` parameter, see *COMMAND_CONV_ARG parameter (defines a rule for converting an argument in job definition scripts during command execution)*.

The conversion results are output to the job execution logs as the `KNAX6803-I` or `KNAX6805-I` message.

If the `PATH_CONV_ENABLE` parameter is not defined in the environment file, the `PATH_CONV_ACCESS` parameter is ignored.

Operands

path-name-1 **~<path name>((1 to 247 bytes))**

Specifies the file path before conversion. To specify a file path containing a space, enclose the entire file path in double quotation marks ("). A value enclosed in double quotation marks cannot consist of only a space, tab character, or null character. None of the following characters is permitted:

* ? < > | ` (grave accent mark) $

If *path-name-1* is omitted or the value specified in *path-name-1* is invalid, the command will terminate with an error during parameter analysis.

path-name-2 **~<path name>((1 to 247 bytes))**

Specifies the file path after conversion. To specify a file path containing a space, enclose the entire file path in double quotation marks ("). A value enclosed in double quotation marks cannot consist of only a space, tab character, or null character. None of the following characters is permitted:

* ? < > | ` (grave accent mark) $

If *path-name-2* is omitted or the value specified in *path-name-2* is invalid, the command will terminate with an error during parameter analysis.

Notes

- If this parameter is defined in both the system environment file and the job environment file, both definitions take effect. However, if the total number of times this parameters is specified in the system environment file and the job environment file combined exceeds 255, an error occurs.
- This parameter performs conversion in such a manner that a variable substitution and a file name substitution have been resolved.
- The conversion rules are searched in the order they are defined and only the first rule satisfying the conversion condition is applied.
- If conversions are defined for the same path name by applying both this parameter and the `PATH_CONV` parameter, the conversion defined by the `PATH_CONV` parameter is performed first. To convert the path name obtained after conversion by the `PATH_CONV` parameter further by the `PATH_CONV_ACCESS` parameter, specify the path name converted by the `PATH_CONV` parameter.
- To specify a character string containing a backslash (\), not a metacharacter, in *path-name-2*, specify `\\` instead of `\`.

Example

- This example converts"`/dev/null`" to "`nul`" to run in UNIX a job definition script created for Windows:

```
#-adsh_conf PATH_CONV_ENABLE / :
#-adsh_conf PATH_CONV_ACCESS /dev/null nul
```

- This example converts "`nul`" to "`/dev/null`" to run in Windows a job definition script created for UNIX:

```
#-adsh_conf PATH_CONV_ENABLE \\ ;
#-adsh_conf PATH_CONV_ACCESS nul /dev/null
```

PATH_CONV_ENABLE parameter (enables the path conversion functionality)

Format

```
#-adsh_conf PATH_CONV_ENABLE directory-separator path-separator
```

Description

This parameter enables the path conversion functionality. If the path conversion functionality has already been enabled, the parameter outputs a message and terminates.

Operands

directory-separator **~((1 or 2 bytes))**

Specifies the directory separator in path names before it is converted by the path conversion functionality. The specified value must be a forward slash (`/`) or two consecutive backslashes (`\\`) .

path-separator **~((1 byte))**

Specifies the path separator in path names before it is converted by the path conversion functionality. The specified value must be a colon (`:`) or a semicolon (`;`).

Notes

- If this parameter is defined in the both system environment file and the job environment file, the definition in the job environment file takes effect.

Example

- This example specifies a `PATH_CONV_ENABLE` parameter to run in Windows:

```
#-adsh_conf PATH_CONV_ENABLE / :
#-adsh_conf PATH_CONV /tmp "C:\\temp"
```

- This example specifies a `PATH_CONV_ENABLE` parameter to run in UNIX:

```
#-adsh_conf PATH_CONV_ENABLE \\ ;
#-adsh_conf PATH_CONV "C:\\temp" /tmp
```

PATH_CONV_RULE parameter (defines a rule for converting file paths) (Windows only)

Format

```
#-adsh_conf PATH_CONV_RULE {1|2}
```

Description

This parameter defines (selects) a rule for converting file paths. It is effective when file path conversion is defined in the `PATH_CONV` parameter or the `#-adsh_path_var` command. If this parameter is omitted, path conversion rule 1 goes into effect.

If a path separator is defined in the PATH_CONV_ENABLE parameter, the separated ranges are converted. The separator defined in the PATH_CONV_ENABLE parameter is converted into a separator to be used by the OS that executes the job definition script.

Operands

1

Specifies selection of path conversion rule 1.

Only a range enclosed in double quotation marks (") is converted.

2

Specifies selection of path conversion rule 2.

Character strings separated by the separators shown below under *Separators used in path conversion rule 2* are converted. Character strings that are separated here are further separated by a path separator (defined in the PATH_CONV_ENABLE parameter) and are then converted. Note, however, that characters enclosed in double quotation marks (") and ${ } are not converted.

If the location to be converted is not enclosed in double quotation marks ("), the conversion results will be enclosed in double quotation marks ("). Additionally, the ranges listed below will also be enclosed in double quotation marks (").

- Path separator

  Converted to ";".

- $*shell-variable-name*

  Also includes shell variables that are not specified by the #-adsh_path_var command.

  The range that begins following the dollar sign ($) and ends with the first character that is not an alphanumeric character (a letter if the leading character is a letter) or an underscore (_) is treated as a shell variable name. If there is no shell variable name, the dollar sign ($) alone is not enclosed in double quotation marks (").

- Range from ${ to }

  Also includes shell variables that are not specified by the #-adsh_path_var command.

Separators used in path conversion rule 2

The following table lists the separators used in path conversion rule 2 and the positions at which these separators are valid or invalid:

Separator	Separator position						
	Within ' '	Within " "#1	Within ` `#1	Single character following \	Within $ ( )#1	Within $ { }#1	Other
|	N	N	Y	N	Y	#2	Y
&	N	N	Y	N	Y	#2	Y
;	N	N	Y	N	Y	#2	Y
<	N	N	Y	N	Y	#2	Y
>	N	N	Y	N	Y	#2	Y
(	N	#3	Y	N	Y	#2	Y
)	N	N	Y	N	Y	#2	Y
`	N	Y	Y	N	Y	#2	Y

Separator	Separator position						
	Within ' '	Within " "#1	Within ` `#1	Single character following \	Within $ ( )#1	Within $ { }#1	Other
'	Y	N	Y	N	Y	#2	Y
"	N	Y	Y	N	Y	#2	Y
#	N	N	Y	N	Y	#2	Y
=	N	N	Y	N	Y	#2	Y
Space (including the tab symbol)	N	N	Y	N	Y	#2	Y
Linefeed code	N	N	Y	N	Y	#2	Y

Legend:

Y: The separator is valid.

N: The separator is not valid.

#1

Another range can be nested inside the enclosed range. The following table lists the combinations that can be nested:

Range	Ranges to be nested					
	' '	" "	` `	Single character following \	$( )	${ }
Within " "	N	N	Y	Y	Y	Y
Within ' '	Y	Y	N	Y	Y	Y
Within $( )	Y	Y	Y	Y	Y	Y
Within ${ }	Y	Y	Y	Y	Y	Y

Legend:

Y: Can be nested.

N: Cannot be nested.

If a character indicating the end of a range is encountered within a range to be nested, it is not treated as the end of a range. For example, if "\"" is specified, the second double quotation mark (") is included in the range following the backslash (\) and is therefore not treated as the end of the double-quotation mark enclosed range that begins with the first double quotation mark (").

If no character indicating the end of a range is found before the end of the line, the entire line including its end is treated as being within the range.

#2

Same as the range in which ${ } is nested.

For example, the vertical bar (|) is valid within a pair of grave accent marks (` `) but is not valid within a pair of double quotation marks (" ").

#3

Only a left parenthesis (() that follows a dollar sign ($() is valid. Other left parentheses are not valid.

Notes

- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect.
- Because path conversion rule 2 has a wider conversion range than path conversion rule 1, selecting path conversion rule 2 might not produce the expected conversion results. Before executing a job definition script, use the syntax check function to check the path conversion results in the generated script image. If there are conversion results that are not appropriate, switch the path conversion rule or modify and re-execute the job definition script.

- When path conversion rule 2 is selected, variables to be substituted and patterns are not subject to conversion. An example follows.

  Environment setting parameters:

  ```
  #-adsh_conf  PATH_CONV_ENABLE  / :
  #-adsh_conf  PATH_CONV_RULE  2
  #-adsh_conf  PATH_CONV  /tmp  d:\\temp
  ```

  Job definition script before conversion:

  ```
  #-adsh_path_var  DIR
  AA=${DIR:-/tmp}
  ```

  Job definition script after conversion:

  ```
  #-adsh_path_var  DIR
  AA="${DIR:-/tmp}"
  ```

  In this example, because a variable is defined by the `#-adsh_path_var` command, the part `${`*variable-name*`}` is enclosed in double quotation marks (`"`) but `/tmp` is not converted. To avoid this problem, modify the job definition script by, for example, specifying `/tmp` as a variable, as follows:

  ```
  #-adsh_path_var  DIR
  BB=/tmp
  AA=${DIR:-$BB}
  ```

- Path conversion rule 2 uses character string substitution to convert path names. Consequently, the same conversion result might not be obtained even when the same path is specified if the specified character string is different.

- Because path conversion rule 2 also converts the document portion of a here document, make sure that programs are not converted into unprocessable data. You can take the following steps:

  - Modify the conversion rule.
  - Make sure paths are not converted using variable substitution.

  If these methods do not work, switch to path conversion rule 1 or consider making the here document an external file.

  The following shows an example of here document conversion.

  Environment setting parameters:

  ```
  #-adsh_conf  PATH_CONV_ENABLE / :
  #-adsh_conf  PATH_CONV_RULE  2
  #-adsh_conf  PATH_CONV  /home/user001  d:\\home\\user001
  ```

  Job definition script before conversion:

  ```
  uap << EOF
  IN=/home/user001/infile
  FTP=/home/user001/ftp/outfile
  EOF
  ```

  Job definition script after conversion:

  ```
  uap << EOF
  IN="d:\\home\\user001"\\infile
  FTP="d:\\home\\user001"\\ftp\\outfile
  EOF
  ```

  As a result, the data in the here document is converted as follows:

```
IN="d:\\home\\user001"\\infile
FTP="d:\\home\\user001"\\ftp\\outfile
```

Because the post-conversion path is enclosed in double quotation marks (`"`), the user program will not run correctly if it cannot properly process double quotation marks (`"`). In some cases, it might be desirable not to convert the path name, such as when the `ftp` command is used to specify a path at a remote site. In this case, make a change, such as substituting a variable for the path name portion and moving it outside the here document.

For example, if you want to convert the path `IN=/home/user001/infile` but do not want to convert the path `FTP=/home/user001/ftp/outfile`, you can make the following changes:

```
VARIN=/home/user001/infile
VARFTP='/home/user001/ftp/outfile'
uap << EOF
IN=$VARIN
FTP=$VARFTP
EOF
```

- Note that the following characters will be converted if they satisfy the path conversion rule:
  - Operators
  - Command option characters that begin with a forward slash (`/`)
  - Data containing a directory separator that is not a path name, specified for a command argument
  - Ternary operator

In the following example, a ternary operator is converted.

Environment setting parameters:

```
#-adsh_conf  PATH_CONV_ENABLE  /  :
#-adsh_conf  PATH_CONV_RULE  2
#-adsh_conf  PATH_CONV  A2/A1  A2\\A1
```

Job definition script before conversion:

```
#-adsh_job JOB001
A1=10
A2=5
((BB=A1>A2?A1/A2:A2/A1))
```

Job definition script after conversion:

```
#-adsh_job JOB001
A1=10
A2=5
((BB=A1>A2?A1\\A2";""A2\\A1"))
```

In this example, the last `A2/A1` in `((BB=A1>A2?A1/A2:A2/A1))` satisfies the conversion rule and is converted. To prevent this, modify the job definition script by using one of the following methods.

- Specify the variable in `$`*shell-variable-name* format:

```
 ((BB=$A1>$A2?$A1/$A2:$A2/$A1))
```

- Rewrite the script using an `if` statement:

```
 if((A1>A2)) then
 ((BB=$A1/$A2))
 else
```

```
  ((BB=$A2/$A1))
  fi
```

- According to path conversion rule 2, if a character string satisfying the conversion rule contains a column name in ${*column-name*[*]} format, it is enclosed in double quotation marks ("). Note also that "${*column-name*[*]}", which is the conversion result, might appear differently after conversion because its individual elements are separated by the value of the IFS shell variable.
- According to path conversion rule 2, a path not containing a directory separator, such as cd work, cannot be converted. In this case, modify the description of the job definition script or do not specify conversion that changes the directory name.
- According to path conversion rule 2, when specifying a path name as an option value, as in *command* -p *path-name*, use a space to separate the option character from the path name.
- According to path conversion rule 2, even if you convert a path contained in a command by using command substitution in the grave accent (`) format, the expected action will not occur. An example follows.

  Environment setting parameters:

  ```
  #-adsh_conf  PATH_CONV_ENABLE / :
  #-adsh_conf  PATH_CONV_RULE  2
  #-adsh_conf  PATH_CONV  /home/user001  d:\\home\\user001
  ```

  Job definition script before conversion:

  ```
  cat file | grep `cat /home/user001/text`
  ```

  Job definition script after conversion:

  ```
  cat file | grep `cat "d:\\home\\user001"\\text`
  ```

  In this example, because \ used in the command for command substitution is processed as a meta character, \\ is erased in the end, and as a result the expected action does not occur.

  In this case, change to command substitution in the $() format, as shown in the following:

  ```
  cat file | grep $(cat /home/user001/text)
  ```

- According to path conversion rule 2, if you want to change /dev/null to Windows NUL device, use the COMMAND_CONV_ARG and PATH_CONV_ACCESS parameters.
- According to path conversion rule 2, if you describe a path as shown below, it cannot be converted correctly.

  Environment setting parameters:

  ```
  #-adsh_conf  PATH_CONV_ENABLE  / :
  #-adsh_conf  PATH_CONV_RULE  2
  ```

  Example of a script before conversion:

  ```
  #-adsh_path_var  homedir

  INPUT1=${homedir}"/test/data"
  INPUT2="${homedir}"/test/data
  ```

  A path is converted in units of character strings separated by separators. Therefore, even if ${homedir}/test/data is a path name, ${homedir} and /test/data are converted separately as two character strings because of the double quotation marks ("), and the following expected conversion result is not obtained.

  Expected conversion result:

```
#-adsh_path_var  homedir

INPUT1="${homedir}""\\test\\data"
INPUT2="${homedir}"\\test\\data
```

Actual conversion result:

```
#-adsh_path_var  homedir

INPUT1="${homedir}""/test/data"
INPUT2="${homedir}"/test/data
```

In this case, modify the description as follows:

```
#-adsh_path_var  homedir

INPUT1="${homedir}/test/data"
INPUT2="${homedir}/test/data"
```

Example

Parameter-setting examples are described below. For a job definition script conversion example, see *2.6.2 Converting path names*.

- Path conversion rule 1 is used to convert paths:

```
#-adsh_conf  PATH_CONV_ENABLE  / :
#-adsh_conf  PATH_CONV_RULE  1
#-adsh_conf  PATH_CONV  /home/user001  d:\\home\\user001
```

- Path conversion rule 2 is used to convert paths:

```
#-adsh_conf  PATH_CONV_ENABLE  / :
#-adsh_conf  PATH_CONV_RULE  2
#-adsh_conf  PATH_CONV /home/user01  d:\\home\\user01
#-adsh_conf  PATH_CONV BB/AA BB\\AA
```

PERMISSION_SPOOLJOB_DIR parameter (defines permission for the spool job directory) (UNIX only)

Format

```
#-adsh_conf PERMISSION_SPOOLJOB_DIR permission
```

Description

This parameter defines the new permission when the permission for the spool job directory is to be changed when the job terminates.

If this parameter is not specified, the permission for the spool job directory is `0700`.

Operands

permission **~<4-digit octal number>((0000 to 1777))**

Specifies the new permission. The spool job directory permission specified by this operand is set when the job terminates.

Notes

- If any of the following is specified for this parameter in an environment file, the job will terminate in an error without being executed:
  - This parameter is specified more than once.
  - No operand is specified.
  - More than two operands are specified.
  - A non-octal number or a value exceeding the value that can be specified for a permission is specified.
  - A value that is not four digits in length is specified.
- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect.

Example

- This example sets the spool job directory permission to `0750`:

```
#-adsh_conf PERMISSION_SPOOLJOB_DIR 0750
```

PERMISSION_SPOOLJOB_FILE parameter (defines permission for the files under the spool job directory) (UNIX only)

Format

```
#-adsh_conf PERMISSION_SPOOLJOB_FILE permission
```

Description

This parameter defines the new permission when the permission for the files under the spool job directory is to be changed when the job terminates.

Specifying this parameter also changes the permission for the directories located under the spool job directory. However, the permission for the files and directories in the directories located under the spool job directory does not change.

If this parameter is not specified, the permission for the files to be created under the spool job directory is as follows:

- `.DBG` files: The permission is `666`.
- Files allocated by the `#adsh_spoolfile` command: The permission specified by the command or program that creates the files takes effect.
- Files other than the above: The permission is `600`.

Operands

permission **~<3-digit octal number>((000 to 777))**

Specifies the new permission. The permission for the files located under the spool job directory is set to the value specified by this operand when the job terminates.

Notes

- If any of the following is specified for this parameter in an environment file, the job will terminate in an error without being executed:
  - This parameter is specified more than once.
  - No operand is specified.
  - More than two operands are specified.
  - A non-octal number or a value exceeding the value that can be specified for a permission is specified.
  - A value that is not three digits in length is specified.
- The `adshhk` and `adshevtout` commands operate on the spool job directory and the files located under the spool job directory. When changing the permission, take into consideration the permission of the users who execute these commands.
- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect.
- Specifying this parameter deletes the `setuid` and `setgid` bits of the files located under the spool job directory.

Example

- This example sets the permission for the files located under the spool job directory to `744`:

```
#-adsh_conf PERMISSION_SPOOLJOB_FILE 744
```

PIPE_CMD_LAST parameter (defines execution processing for the last command in a pipe) (UNIX only)

Format

```
#-adsh_conf PIPE_CMD_LAST {CURRENT|OTHER}
```

Description

This parameter specifies whether the last command in the pipeline in the current process is to be executed.

You specify `CURRENT` to update the contents of variables by using the last command in the pipeline and you want to use the updated contents in commands that follow the pipeline, as described in the following.

Contents of job definition script:

```
typeset -i CNT=0
cat INFILE | while read STR
do
  echo "$STR"
  let CNT=CNT+1
done
echo "Line count is $CNT."
```

As a result, when the `while` statement terminates, the shell variable `CNT` stores the number of lines loaded by the `read` command (`INFILE` line count).

On the other hand, if you want use the last command in the pipeline to update the contents of variables but you want to revert to using the contents of the pre-update variables after the pipeline has terminated, specify `OTHER` (`CBL_SYSUT1` and `CBL_SYSUT2` are assumed to be common interface variables of `CBLUAP`*x*).

Contents of job definition script:

```
CBL_SYSUT1=/file1
CBL_SYSUT2=/file2
CBLUAP1
cat INFILE | while read DIR
do
CBL_SYSUT1=`cmd1 y`
CBL_SYSUT2=`cmd2 y`
CBLUAP2
done
CBLUAP3
```

In this case, when the `while` statement terminates, the shell variable `CBL_SYSUT1` stores "`/file1`" and the shell variable `CBL_SYSUT2` stores "`/file2`".

This parameter cannot be specified more than once in the same environment file. If it is specified more than once, an error message is output and the job terminates.

Operands

`CURRENT`

Specifies that when the last command in the pipeline is one of the following, the command is to run in the current process:

- Shell standard command
- Substitution expression
- Script control statement

`OTHER`

Specifies that the last command in the pipeline is to run in another process.

Notes

- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect.

Example

The examples below show the difference in the execution result when the input file and job definition script shown in the following are used to specify the `PIPE_CMD_LAST` parameter.

- Example 1

  Contents of job definition script:

```
STR="abcdefg"
echo "ABCDEFG" | read STR
echo $STR
```

Result output to the standard output (when `CURRENT` is specified in the `PIPE_CMD_LAST` parameter):

```
ABCDEFG
```

Result output to the standard output (when `OTHER` is specified in the `PIPE_CMD_LAST` parameter):

```
abcdefg
```

- Example 2

  Contents of job definition script:

```
A=1
echo "Hello World" | A=10
echo $A
```

Result output to the standard output (when `CURRENT` is specified in the `PIPE_CMD_LAST` parameter):

```
10
```

Result output to the standard output (when `OTHER` is specified in the `PIPE_CMD_LAST` parameter):

```
1
```

- Example 3

  Contents of the `INFILE` input file:

```
user1,500,Tomato
user2,1000,Tomato
user3,300,Lettuce
user1,450,Cabbage
user1,250,Orange
```

Contents of the job definition script:

```
typeset -i cnt=1
cat INFILE | grep user1 | while read NAME
do
  if [ $cnt -ge 3 ]; then
    break
  fi
  echo "$cnt = $NAME"
  let cnt=cnt+1
done
echo $cnt
```

The character strings from `while` to `done` are treated as the last command in the pipe.

Result output to the standard output (when `CURRENT` is specified in the `PIPE_CMD_LAST` parameter):

```
1 = user1,500,Tomato
2 = user1,450,Cabbage
3
```

Result output to the standard output (when `OTHER` is specified in the `PIPE_CMD_LAST` parameter):

```
1 = user1,500,Tomato
2 = user1,450,Cabbage
1
```

SPOOL_DIR parameter (defines the spool root directory path name)

Format

```
#-adsh_conf SPOOL_DIR path-name
```

Description

This parameter defines the path name of the spool root directory that is to be created for each job for output of batch job execution results (job execution logs and the data files output by job step programs).

Operands

path-name

Windows execution environment: **~<path name>((1 to 128 bytes))<<shared-documents-folder\Hitachi\JP1AS\JP1ASE\spool>>**

Windows development environment: **~<path name>((1 to 128 bytes))<<shared-documents-folder\Hitachi\JP1AS\JP1ASD\spool>>**

UNIX: **~<path name>((1 to 128 bytes))<</var/opt/jp1as/spool>>**

Specifies the path name of the spool root directory.

Notes

- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect.
- The user-reply functionality might not operate correctly in either of the following cases:
  - This parameter is specified in the job environment file.
  - Multibyte characters are specified in path names.
- Do not specify in this parameter a directory in a file system that is not supported by JP1/Advanced Shell. For details about file systems that are not supported by JP1/Advanced Shell, see *2.6.19(2) File systems*.
- Do not include any of the following characters in the path name: `& ( ) [ ] { } ^ = ; ! ' + , ` ~ #` `%`. If any of these characters is included, JP1/Advanced Shell will not function correctly.

Supplementary information

- You can use multiple environments on the same host by specifying separate directories in the corresponding parameters.
- To inherit information to a standby host during cluster operation, the directories to be inherited are shared among the hosts. Share at least the directory specified in this parameter among the hosts.

SPOOLJOB_CHILDJOB parameter (defines how a spool job of a child job is to be handled)

Format

```
#-adsh_conf SPOOLJOB_CHILDJOB {DELETE|MERGE}
```

Description

This parameter specifies whether the spool job of a child job is to be deleted or is to be merged into the spool job of the root job when the child job terminates.

Operands

`DELETE`

Specifies that the spool job of a child job is to be deleted when the child job terminates.

Of the child job's job execution logs, only the contents of `JOBLOG` are output to the standard error output.

For an example of the job execution log output when `DELETE` is specified, see *3.4.2 Examples of job execution log output*.

`MERGE`

Specifies that the spool job of a child job is to be merged into the spool job of the root job when the child job terminates. As a result, the following processing occurs:

- The job execution logs of child jobs are merged into the job execution logs of the root job and are output in the order in which the child jobs terminated.
- `JOBLOG` and `SCRIPT` of the root job are created with contents into which `JOBLOG`s and `SCRIPT`s of the child jobs have been merged.
- Job execution logs are output to the following location in a format that clearly shows whether the output contents are for the root job or child jobs:

  During normal execution: Standard error output (`STDERR`, *step-number*_*step_name*_`STDERR`)

  During debugging: Standard output and standard error output on the terminal screen
- The program output data files assigned by the `#-adsh_spoolfile` command during child job execution are created under the spool job directory of the root job with the following names:

#-adsh_spoolfile command's execution location	File name to be assigned (in Windows, the extension .sysout is added.)
Outside the job step of a child job	`C`*number-giving-the-order-in-which-a-child-job-starts*_`0000`_*job-name*_*sequence-number-of-file-environment-variable-definition-name*_*file-environment-variable-definition-name*
Inside the job step of a child job	`C`*number-giving-the-order-in-which-a-child-job-starts*_*step-number*_*step-name*_*sequence-number-of-file-environment-variable-definition-name*_*file-environment-variable-definition-name*

If the same job definition script is executed multiple times as child jobs, `SCRIPT` is output the number of times the script is executed.

For details about how to create a spool job directory, see *3.3.2 Outputting job execution results to spool*. For details about the format for outputting job execution logs, see *3.4.1(3) Merging a child job's spool job into the root job's spool job*.

When `MERGE` is specified, the maximum number of child jobs that can be started from a single root job, including child jobs started from child jobs, is 9,999,999. Child jobs that exceed this limit terminate in an error. However, if the OS-specified process count or file count limit is reached first, the OS's error processing takes precedence.

For an example of the job execution log output when MERGE is specified, see *3.4.3 Example of job execution log output (when a child job's spool job is merged into the root job's spool job)*.

Notes

- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect.
- When MERGE is specified and child jobs are executed asynchronously, the JOBLOG merging order might not match the SCRIPT merging order.

  For example, if child job A and child job B are executed asynchronously in a root job, JOBLOG might be merged for child job B first and then for child job A, while SCRIPT might be merged for child job A first and then for child job B.
- When MERGE is specified and commands in a job definition script are executed asynchronously, the standard output and standard error output of the asynchronously executed commands might be mixed among the ranges of job execution logs for the standard output and the standard error output of child jobs.
- When MERGE is specified and an initialization error occurs in the started child job before the job definition script has been parsed, merging will not be executed.
- Child jobs run under the assumption that a SPOOLJOB_CHILDJOB parameter value has been specified in the environment file loaded when the root job was started. Even if the value of the SPOOLJOB_CHILDJOB parameter differs between the root job's job environment file and the child job's job environment file, the child jobs run by ignoring the difference.
- If a job is executed in the syntax check mode, merging will not be executed.

Example

- In this example, the spool job of a child job is merged into the spool job of the root job when the child job terminates:

  ```
  #-adsh_conf SPOOLJOB_CHILDJOB MERGE
  #-adsh_conf CHILDJOB_SHEBANG /bin/sh
  ```

SPOOLJOB_CREATE parameter (selects whether a spool job is to be created)

Format

```
SPOOLJOB_CREATE {YES|NO}
```

Description

This parameter specifies whether a spool job is to be created when a job definition script is run.

This parameter is ignored for child jobs, because the specification for the root job is inherited.

Operands

YES

Specifies that a spool job directory is to be created.

`NO`

Specifies that a spool job directory is not to be created. For details about the operation, see *2.6.8(1)(a) Determining whether the spool job creation suppression functionality is to be used*.

Notes

- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect.
- When the `adshexec -r` or `adshscripttool -r` command is executing, temporary files are not created for the job definition script file regardless of this parameter's setting. In such a case, `-r CMDLINE` is displayed in the output part of the job definition script file in messages.
- When the CUI debugger is used, the DBG file *temporary-file-directory*`/ADSH_DBG_`*process-ID*`_`*job-ID* is created temporarily and deleted when the debugger terminates.

TEMP_FILE_DIR parameter (defines the path name of the directory for storing temporary files)

Format

```
#-adsh_conf TEMP_FILE_DIR path-name
```

Description

This parameter defines the path name of the directory for storing temporary files.

Temporary files are created in a batch job and are deleted when the batch job terminates.

Operands

path-name

Windows execution environment: **~<path name>((1 to 128 bytes))<<shared-documents-folder\Hitachi\JP1AS\JP1ASE\temp>>**

Windows development environment: **~<path name>((1 to 128 bytes))<<shared-documents-folder\Hitachi\JP1AS\JP1ASD\temp>>**

UNIX: **~<path name>((1 to 512 bytes))<</var/opt/jp1as/temp>>**

Specifies the path name of the directory for storing temporary files.

Notes

- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect.
- Do not specify in this parameter a directory in a file system that is not supported by JP1/Advanced Shell. For details about file systems that are not supported by JP1/Advanced Shell, see *2.6.19(2) File systems*.

Supplementary information

- You can use multiple environments on the same host by specifying separate directories in the corresponding parameters.

TRACE_DIR parameter (defines the path name of the directory to which traces are to be output)

Format

```
#-adsh_conf TRACE_DIR path-name
```

Description

This parameter defines the path name of the directory to which traces are to be output.

Operands

path-name

Windows execution environment: **~<path name>((1 to 128 bytes))<<common-application-data-folder\Hitachi\JP1AS\JP1ASE\trace>>**

Windows development environment: **~<path name>((1 to 128 bytes))<<common-application-data-folder\Hitachi\JP1AS\JP1ASD\trace>>**

UNIX: **~<path name>((1 to 512 bytes))<</opt/jp1as/trace>>**

Specifies the path name of the directory to which traces are to be output.

Notes

- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect.
- Do not specify in this parameter a directory in a file system that is not supported by JP1/Advanced Shell. For details about the file systems that are not supported by JP1/Advanced Shell, see *2.6.19(2) File systems*.
- Do not include any of the following characters in the path name: `& ( ) [ ] { } ^ = ; ! ' + , ` ~ #` `%`. If any of these characters is included, JP1/Advanced Shell will not function correctly.

Supplementary information

- You can use multiple environments on the same host by specifying separate directories in the corresponding parameters.

TRACE_FILE_CNT parameter (defines the number of files to which traces are to be output)

Format

```
#-adsh_conf TRACE_FILE_CNT number-of-files
```

Description

This parameter defines the number of files to which traces are to be output.

Operands

number-of-files **~<unsigned integer>((1 to 64))<<4>>**

Specifies the number of files to which traces are to be output. The normal specification value is `4`.

Notes

- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect. If the value of the `TRACE_DIR` parameter and the value specified in the system environment file are the same, specifying a different value in this parameter will result in an error (even when the system environment file's default value is assumed).

Supplementary information

- If multiple users output trace logs to the same file, the largest `TRACE_FILE_CNT` parameter value specified by any of the users takes effect.

  If the `TRACE_FILE_CNT` parameter value is changed in the environment file, the new value is compared with the existing value for the number of trace files and whichever is larger is used.

  To reduce the number of trace files, you must delete all files from the trace folder (make sure that no job is outputting traces to the trace files when you delete files from the trace folder).

  We recommend that all users who output traces to the same file use the same parameter values.

TRACE_FILE_SIZE parameter (defines the size of a file to which traces are output)

Format

```
#-adsh_conf TRACE_FILE_SIZE file-size
```

Description

This parameter defines the size of a file to which traces are output.

Operands

file-size **~<unsigned integer>((1 to 16))<<2>>**

Specifies in megabytes the size of a file to which traces are to be output. The normal specification value is `2`.

Notes

- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect. If the `TRACE_DIR` parameter value is the same as the value specified in the system environment file, specifying a different value in this parameter results in an error (even when the system environment file's default value is assumed).

Supplementary information

- If multiple users output trace logs to the same file, the largest `TRACE_FILE_SIZE` parameter value specified by any of the users takes effect.

If the `TRACE_FILE_SIZE` parameter value is changed in the environment file, the new value is compared with the existing value for the number of trace files and whichever is larger is used.

To reduce the file size, delete all files from the trace folder (make sure that no job is outputting traces to the trace files when you delete files from the trace folder).

We recommend that all users who output traces to the same file use the same parameter values.

TRACE_LEVEL parameter (defines a trace output level)

Format

```
#-adsh_conf TRACE_LEVEL trace-level
```

Description

This parameter defines a trace output level.

Operands

trace-level **~<unsigned integer>((0, 10, 20, 30))<<0>>**

Specifies a trace output level. As the specified value increases, the traces that are output become more detailed. The normal specification value is `0`.

Notes

- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect.

TRAP_ACTION_SIGTERM parameter (defines the job controller's action when a forced termination request is received)

Format

In UNIX edition

```
#-adsh_conf TRAP_ACTION_SIGTERM {DISABLE|TERM|CONT|AUTO}
```

In Windows edition

```
#-adsh_conf TRAP_ACTION_SIGTERM {DISABLE|TERM}
```

Description

This parameter specifies whether the `trap` command can be used to define an operation to be performed when the job controller receives a forced termination request. It also specifies an action for the job controller after the operation defined by the `trap` command has been executed. Forced termination requests include forced termination operations from JP1/AJS - View, transmission of a `SIGTERM` signal by the `kill` command in UNIX, and forced termination by the `taskkill` command in Windows (immediate process termination by a means such as `TerminateProcess`).

When a forced termination request is received while a job definition script is running, the job controller operates according to the operand specified here.

If this parameter is omitted, the job controller assumes that `DISABLE` is specified.

For details about the `trap` command, see *trap command (specifies the action when signals and forced termination requests are received) (UNIX only)* in *9.3 Standard shell commands*.

Operands

`DISABLE`

Specifies that an operation to be performed when a forced termination request is received cannot be defined by the `trap` command. If a forced termination request is received, the job controller will terminate with an error without executing the subsequent commands.

`TERM`

Specifies that an operation to be performed when a forced termination request is received can be defined by the `trap` command.

If a forced termination request is received, the job controller will perform the operation defined by the `trap` command, and then will terminate with an error without executing the subsequent commands.

In the UNIX edition, when a second forced termination request is received, the job terminates immediately without performing the job controller's postprocessing.

`CONT` (UNIX only)

Specifies that an operation to be performed when a forced termination request is received can be defined by the `trap` command.

If a forced termination request is received, the job controller will perform the operation defined by the `trap` command and is to continue processing even when subsequent forced termination requests are received.

Note that this operand is not applicable to jobs started from JP1/AJS. If this operand is specified for such a job, the job controller will issue the `KNAX0474-E` message during environment file analysis and then terminate with an error.

`AUTO` (UNIX only)

Specifies that the job controller is to assume that either `TERM` or `CONT` is specified, depending on the job start method. You specify this operand when you want to share the same environment file regardless of the job start method.

Target job	Operation
Job started from JP1/AJS	The job controller assumes that `TERM` is specified and operates accordingly.
Job started from a program other than JP1/AJS	The job controller assumes that `CONT` is specified and operates accordingly.

If the job is any of the following, the job controller assumes that the job was started from JP1/AJS:

1. Job started from JP1/Advanced Shell's custom job
2. Job started while `TERM` was set in the `AJS_BJEX_STOP` environment variable
3. Child job started from 1 or 2

For details about the operation that is performed when a forced termination request is performed for each operand, see *3.10 Forcibly terminating jobs*.

Notes

(Common to both UNIX and Windows)

- When an operation to be performed when a forced termination request is received is defined by the trap command for a job and that job receives a forced termination request, the job is not terminated until the defined operation is completed. Keep this in mind when you define a command that takes a long time to execute or for an operation that is not terminated.
- Even if an operation is defined by using the trap command within a job definition script, the defined operation does not take effect while normal postprocessing is underway during normal job termination. If a forced termination request is received during that time, the job controller assumes that no operation is defined by means of a trap command.
- If the operation defined by the trap command terminates with an error or the job is cancelled by execution of the exit command, the job is terminated with the termination code of the last command that was executed within the operation defined by the trap command. However, if TERM is specified for this parameter, the operation defined by the trap command is terminated with the termination code when the forced termination request was received. The termination code of the operation defined by the trap command is not applied to the termination code of the job or job steps.
- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect.

(UNIX only)

- If any of the conditions listed below is satisfied and a second forced termination request is received while the first termination request is still engaged in processing, such as terminating a descendant process, performing the operation defined by the trap command, or deleting temporary files, the postprocessing (including deletion of created temporary files) might not be completed. In such a case, take appropriate action, such as by deleting the temporary files manually.
    - DISABLE is specified in the operand.
    - TERM is specified in the operand.
    - CONT is specified in the operand and no operation is defined by the trap command.

UNSUPPORT_TEST parameter (defines the handling of an unsupported conditional expression) (Windows only)

Format

```
#-adsh_conf UNSUPPORT_TEST {h|G|L|O|ef} {ERR|TRUE|FALSE}
```

Description

This parameter specifies the handling of a conditional expression that is not supported by JP1/Advanced Shell in a Windows environment. The following are the conditional expressions that are not supported by JP1/Advanced Shell in a Windows environment:

- Conditional expression using the operator -h
- Conditional expression using the operator -G
- Conditional expression using the operator -L

- Conditional expression using the operator `-O`
- Conditional expression using the operator `-ef`

If a conditional expression that is not supported by JP1/Advanced Shell is to be evaluated but its handling is not specified in this parameter, the conditional expression will be handled in the same manner as when `ERR` is specified.

For details about the conditional expressions, see *5.2.2 Conditional expressions*.

Operands

`{ h | G | L | O | ef }`

Specifies a conditional expression that is not supported by JP1/Advanced Shell in a Windows environment.

- `h`

  Indicates a conditional expression that uses the operator `-h`.

- `G`

  Indicates a conditional expression that uses the operator `-G`.

- `L`

  Indicates a conditional expression that uses the operator `-L`.

- `O`

  Indicates a conditional expression that uses the operator `-O`.

- `ef`

  Indicates a conditional expression that uses the operator `-ef`.

`{ ERR | TRUE | FALSE }`

Specifies the handling of the unsupported conditional expression in a Windows environment.

- `ERR`

  Output an error message and terminate the job.

- `TRUE`

  Output an information message and assume that the conditional expression is correct.

- `FALSE`

  Output an information message and assume that the conditional expression is not correct.

Notes

- If this parameter is defined for a given conditional expression in both the system environment file and the job environment file, the definition in the job environment file takes effect.

USERREPLY_DEBUG_DESTINATION parameter (specifies the input source and the destination of event notification and reply-request messages during debug execution)

Format

```
#-adsh_conf USERREPLY_DEBUG_DESTINATION [ JP1EVENT | CONSOLE ]
```

Description

This parameter specifies the input source and the destination of event notification and reply-request messages when a job definition script that uses the `adshecho` and `adshread` commands is debugged using the user-reply functionality.

Operands

`JP1EVENT`

Specifies that event notification messages and reply-request messages are to be issued as JP1 events.

`CONSOLE`

Specifies that the input source and the destination of event notification messages and reply-request messages are to be set to the standard input and output, respectively.

Notes

- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect.
- If `CONSOLE` is specified in this parameter during an execution other than a debug execution, the specification is ignored. In such a case, event notification messages and reply-request messages are issued as JP1 events.

USERREPLY_JP1EVENT_INTERVAL parameter (specifies the minimum interval at which JP1 events are to be issued)

Format

```
#-adsh_conf USERREPLY_JP1EVENT_INTERVAL minimum-event-issuance-interval
```

Description

This parameter specifies the minimum interval at which JP1 events are to be issued by the user-reply functionality.

The purpose of this parameter is to manage the flow of JP1 events by not issuing a JP1 event until the amount of time defined here has elapsed since the previous JP1 event was issued.

Operands

minimum-event-issuance-interval **~<unsigned integer>((100 to 100000))<<500>>**

Specifies in milliseconds the amount of time to wait since the previous JP1 event was issued. To prevent excessive workload on JP1/IM - Manager, the normal specification value is 500 or greater.

Notes

- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect.
- This parameter is ignored when debug execution is performed with the standard input and output specified as the input source and output destination for the user-reply functionality.

USERREPLY_WAIT_MAXCOUNT parameter (specifies the maximum number of concurrent reply-request messages that can be output for a physical or logical host)

Format

```
#-adsh_conf USERREPLY_WAIT_MAXCOUNT maximum-number-of-concurrent-reply-
request-messages-to-be-output
```

Description

This parameter specifies the maximum number of concurrent reply-request messages to be output for each physical or logical host when the user-reply functionality is used.

Operands

maximum-number-of-concurrent-reply-request-messages-to-be-output ~**<unsigned integer>((1 to 100))<<5>>**

Specifies the maximum number of concurrent reply-request messages to be output for each physical host or logical host. This specification limits the number of reply-request messages that can exist for each physical host and logical host.

Because a maximum of 2,000 reply-waiting events can be accumulated in JP1/IM - View, specify a value (to be applied to each host that sends reply-waiting events) that ensures that the following condition is satisfied:

Maximum number of concurrent reply-request messages that can be output by each host times the number of hosts that send reply-waiting events to the same JP1 event destination (`HOSTNAME_JP1IM_MANAGER` parameter value)

+ *total number of other products' reply-waiting events that can be accumulated*

< 2,000

Notes

- Do not specify this parameter in the job environment file. If this parameter is specified in both the system environment file and the job environment file, the specification in the job environment file will be ignored.
- If output of more reply-waiting events than is specified in this parameter is attempted, the processing will be placed on wait status until enough space becomes available in the shared memory. Available space in the shared memory is checked every three minutes for up to three times. If there is still not enough space after three attempts, an error will result. Therefore, if multiple jobs output reply-request messages concurrently, specify an appropriate value so that the number of reply-waiting events will not exceed that value.
- This parameter is ignored when debug execution is performed with the standard input and output specified as the input source and output destination, respectively, of the user-reply functionality.

VAR_ENV_NAME_LOWERCASE parameter (specifies whether environment variable names in lowercase letters are supported) (Windows only)

Format

```
VAR_ENV_NAME_LOWERCASE {ENABLE|DISABLE}
```

Description

This parameter specifies whether environment variable names in lowercase letters are supported.

Operands

`ENABLE`

Specifies that environment variable names in lowercase letters are supported.

Environment variable names with the same spelling but different capitalization are not identified as different environment variables in Windows, but they are recognized as separate shell variables. To avoid confusion, we recommend that you use the same capitalization for shell variables that have the same spelling.

`DISABLE`

Specifies that environment variable names in lowercase letters are not supported.

Notes

- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect.

Examples

This example defines shell variable names `SAMPLE01` and `sample01` in the job definition script and a batch file, as shown in the following.

Contents of job definition script envsample.ash

```
export SAMPLE01=large
export sample01=small
.\\envsample.bat | "${ADSH_DIR_CMD}grep" -i "SAMPLE01" 1>&2
echo "*** Shell variables ***" >&2
echo "SAMPLE01=$SAMPLE01" >&2
echo "sample01=$sample01" >&2
```

Contents of batch file envsample.bat

```
set
```

- Specifying `DISABLE` (environment variable names in lowercase letters are not supported)

  Contents of the environment variable:

```
#-adsh_conf VAR_ENV_NAME_LOWERCASE DISABLE
#-adsh_conf OUTPUT_MODE_ROOT  SIMPLE
#-adsh_conf OUTPUT_MODE_CHILD SIMPLE
```

Execution results

```
D:\home>"C:\Program Files\HITACHI\JP1AS\JP1ASE\bin\adshexec" envsample.ash
KNAX6712-E Specified variable "sample01" cannot be exported because the
name is not in all capital letters on the current platform. filename="D:
\home\envsample.ash" line=2
KNAX6521-E Command export(line=2) failed. rc=1 E-Time=0.005s C-Time=0.000s
KNAX0101-E ADSH001002 An error occurred during execution of job.

D:\home>
```

In this example, the export operation for `sample01` that was executed after `SAMPLE01` has failed.

- Specifying ENABLE (environment variable names in lowercase letters are supported)

  Contents of the environment variable:

```
#-adsh_conf VAR_ENV_NAME_LOWERCASE ENABLE
#-adsh_conf OUTPUT_MODE_ROOT  SIMPLE
#-adsh_conf OUTPUT_MODE_CHILD SIMPLE
```

  Execution results

```
D:\home>"C:\Program Files\HITACHI\JP1AS\JP1ASE\bin\adshexec" envsample.ash
sample01=small
*** Shell variables ***
SAMPLE01=large
sample01=small

D:\home>
```

  In this example, shell variables whose names are in lowercase letters can be exported. As a result, sample01, the last shell variable exported, is exported to the environment variable. Shell variables SAMPLE01 and sample01 have different values.

VAR_SHELL_FUNCINFO parameter (selects whether function information arrays are used)

Format

```
#-adsh_conf VAR_SHELL_FUNCINFO {TYPE_A|TYPE_B|NONE}
```

Description

This parameter defines whether function information arrays are used. Arrays of function information are single-dimensional arrays for storing information about the function being executed by the adshexec command.

There are three types of function information arrays, as listed below. For details about the function information arrays, see *5.5.3 Arrays of function information*.

- Called function name array
- Function call line number array
- Function definition script file name array

Operands

TYPE_A

Specifies that function information arrays whose names begin with ADSH_ are to be used, as shown in the following:

Array type	Array name
Called function name array	ADSH_FUNCNAME
Function call line number array	ADSH_LINENO
Function definition script file name array	ADSH_SOURCE

TYPE_B

Specifies that function information arrays whose names are the same as the following are to be used:

Array type	Array name
Called function name array	FUNCNAME
Function call line number array	BASH_LINENO
Function definition script file name array	BASH_SOURCE

NONE

Specifies that function information arrays are not to be created.

This operand enables you to use the arrays that are created when TYPE_A or TYPE_B is specified as normal arrays.

Notes

- If this parameter is defined in both the system environment file and the job environment file, the definition in the job environment file takes effect.
- A parameter error occurs if this parameter is defined more than once in the same environment file on the same host.
- Arrays of function information are read-only. This read-only attribute cannot be released, the values cannot be updated, and the arrays cannot be disabled by using unset.
- Arrays of function information cannot be defined as local variables in a function. Do not change their attributes in a function or define them as local functions in a function.
- Arrays of function information cannot be specified for the stepVar attribute in the #-adsh_step_start command.
- If the parameter value differs between a root job and its child job, the parameter value that is in effect when the job starts applies to the arrays that are created.

Examples

This example runs the following job definition script in an environment where TYPE_A is specified in the VAR_SHELL_FUNCINFO parameter:

/home/user/script/adsh_func.ash

```
func1(){               # Define function func1
 function func1_2 {    # Define function func1_2
   echo "in func1_2"
   func2               # Call function func2 from function func1_2
 }
 echo "in func1"
 . ./func2.ash    # Load the external script that defines function func2 by using the . (dot) command
 func1_2           # Call function func1_2 from function func1
}

echo "main_script start"
export FPATH=`pwd`    # Store the current work directory in shell variable FPATH
#-adsh_script ./func3.ash # Load the external script that defines function func3 by using #-adsh_script
autoload func4        # Enable the function preload functionality for function func4

```

```
0016 : func1                 # Call function func1
0017 : echo "main_script end"
```

/home/user/script/func2.ash

```
0001 : func2(){
0002 :   echo "in func2"
0003 :   func3
0004 : }
```

/home/user/script/func3.ash

```
0001 : func3(){
0002 :   echo "in func3"
0003 :   func4
0004 : }
```

/home/user/script/func4

```
0001 : func4(){
0002 :   echo "in func4"
0003 :   cnt=0
0004 :   for cnt in 0 1 2 3 4 5
0005 :   do
0006 :     echo "ADSH_FUNCNAME[$cnt] = ${ADSH_FUNCNAME[$cnt]}"
0007 :     echo "ADSH_LINENO[$cnt]   = ${ADSH_LINENO[$cnt]}"
0008 :     echo "ADSH_SOURCE[$cnt]   = ${ADSH_SOURCE[$cnt]}"
0009 :   done
0010 : }
```

Stack information for one function is output on each `echo` command line.

The following table lists the values in the function information arrays when function `func4()` is executing:

Element no.	Array name		
	ADSH_FUNCNAME	ADSH_LINENO	ADSH_SOURCE
0	`func4`	`3`	`/home/user/script/func4`
1	`func3`	`3`	`/home/user/script/func3.ash`
2	`func2`	`4`	`/home/user/script/func2.ash`
3	`func1_2`	`8`	`/home/user/script/adsh_func.ash`
4	`func1`	`16`	`/home/user/script/adsh_func.ash`
5	`main`	`0`	`/home/user/script/adsh_func.ash`

If you use an environment where `TYPE_B` is specified in the `VAR_SHELL_FUNCINFO` parameter, change the contents of job definition script `/home/user/script/func4` as follows:

```
0001 : func4(){
0002 :   echo "in func4"
0003 :   cnt=0
0004 :   for cnt in 0 1 2 3 4 5
0005 :   do
0006 :     echo "FUNCNAME[$cnt]       = ${FUNCNAME[$cnt]}"
0007 :     echo "BASH_LINENO[$cnt]    = ${BASH_LINENO[$cnt]}"
0008 :     echo "BASH_SOURCE[$cnt]    = ${BASH_SOURCE[$cnt]}"
0009 :   done
0010 : }
```

VAR_SHELL_GETLENGTH parameter (defines the unit for the lengths of variable values that are replaced in format ${#variable})

Format

```
VAR_SHELL_GETLENGTH {BYTE|CHARACTER}
```

Description

This parameter defines the unit for the lengths of variable values that are replaced in the format `${#variable}`.

Operands

`BYTE`

Specifies that the length of a value stored in `variable` in the format `${#variable}` is to be replaced by the length in bytes.

`CHARACTER`

Specified that the length of a value stored in `variable` in the format `${#variable}` is to be replaced by the lengths in characters.

Examples

This example executes `echo ${#CVAL}` to obtain the length of the value of variable `CVAL` in which character string `abcdef英字` is set.

- Specifying `BYTE` (replace by the length in bytes) in the `VAR_SHELL_GETLENGTH` parameter or omitting the `VAR_SHELL_GETLENGTH` parameter

  The lengths of multibyte characters vary according to the execution environment. The following example assumes that a Linux UTF-8 environment is used:

  ```
  CVAL=abcdef英字
  echo ${#CVAL}
  ```

  Because `abcdef英字` is interpreted as being 12 bytes, `12` is output to the standard output.

- Specifying `CHARACTER` (replace by the length in characters) in the `VAR_SHELL_GETLENGTH` parameter

  ```
  CVAL=abcdef英字
  echo ${#CVAL}
  ```

  Because `abcdef英字` consists of eight characters, `8` is output to the standard output.

7.4 Conditional parameters

To specify environment setting parameters or `export` parameters that are to apply only on a logical or physical host, enclose the specifications between conditional parameters that are specified on the immediately preceding and following lines. This section explains the conditional parameters.

lhost_start and lhost_end parameters (define a set of parameters applicable only to a specified logical host)

Format

```
#-adsh_conf lhost_start logical-host-name
environment-setting-parameter-or-export-parameter-to-be-applicable-only-on-
the-specified-logical-host
               :
#-adsh_conf lhost_end
```

Description

If there are environment setting parameters or `export` parameters that are to take effect only on a specific logical host, define those parameters enclosed between an `lhost_start` parameter and an `lhost_end` parameter. Make sure that each `lhost_end` parameter is paired with an `lhost_start` parameter.

- `lhost_start` parameter

  Begins the specification of environment setting parameters or `export` parameters that are to take effect only on the specified logical host. This parameter also specifies the name of the target logical host.

- `lhost_end` parameter

  Ends the specification of environment setting parameters or `export` parameters that are to take effect only on the specified logical host.

Specifying multiple conditional parameters for the same logical host does not result in an error. All sets of parameters that are specified are effective.

Operands

logical-host-name **~<any character string>((1 to 255 bytes))**

Specifies the name of the applicable logical host. Specifying multiple definitions for the same logical host does not result in an error. All sets of parameters specified are effective.

In Windows, you cannot specify a logical host name exceeding 196 bytes. Because the parameter might not work in some cases when the logical host name exceeds 63 bytes, we recommend that you always specify logical host names that do not exceed 63 bytes.

phost_start and phost_end parameters (define a set of parameters applicable only to the physical host)

Format

```
#-adsh_conf phost_start
environment-setting-parameter-or-export-parameter-to-be-applicable-only-on-
the-physical-host
         :
#-adsh_conf phost_end
```

Description

If there are environment setting parameters or `export` parameters that are to take effect only on the physical host, define those parameters by enclosing them between a `phost_start` parameter and a `phost_end` parameter. Make sure that each `phost_end` parameter is paired with a `phost_start` parameter.

- `phost_start` parameter

  Specify this parameter on the line that immediately precedes the line on which begins the specification of a set of environment setting parameters or `export` parameters that are to take effect only on the physical host.

- `phost_end` parameter

  Specify this parameter on the line that immediately follows the line on which ends the specification of a set of environment setting parameters or `export` parameters that are to take effect only on the physical host.

Specifying multiple definitions for the physical host does not result in an error. All sets of parameters specified are effective.

8 Commands Used During Operations

This chapter describes the shell operation commands and UNIX-compatible commands in JP1/Advanced Shell.

8.1 Command description format

The following shows the command description format for shell operation commands and UNIX-compatible commands:

Δ_0*command-name*[Δ_1*option*]... [Δ_1*option*][Δ_1*operand*]

Following the command name, you specify options and then operands. Operands includes option names, option values, and other arguments that can be specified in commands. If an operand is specified before an option, all specified items are treated as operands.

- When there is more than one option, they can be specified in any order.
- An error results if you specify an invalid option or an option's value is outside the permissible range of values.
- Multibyte characters are not permitted in *option-name*.

For details about the specification of file path names in commands, see *8.1.3 File path names*.

The following explains how to execute a command that you enter to the standard input from the terminal keyboard.

For input terminated by an EOF

- In Windows, press **Enter** followed by **Ctrl+Z**, and then press **Enter** again.
- In UNIX, press **Ctrl+D**.

For a single line of input

- Press **Enter**.

8.1.1 Command description format for shell operation commands and UNIX-compatible command (script format) (Windows only)

The following specification rule applies to shell operation commands and UNIX-compatible commands (script format) (Windows only):

- Options specified without a value can be grouped together as a block (for example, `-a -b -c` can also be specified as `-abc`). In this case, a value can be specified for the last option (for example, `xyz` in the specification `-abc xyz`, is the value of option `-c`).

8.1.2 Command description format for UNIX-compatible commands

The following specification rules apply to UNIX-compatible commands except for those in the script format:

- Options are classified as either short options or long options.
- Two consecutive hyphens (`--`) indicates the end of the specification of options. All character strings (including options) that follow double hyphens are processed as operands.

The following subsections explain the description formats for short options and long options.

(1) Specification format for short options

A short option consists of one hyphen (-) followed by one predefined character.

The specification format for short options is shown below. Whether the option value can be omitted depends on the option.

```
-short-option-name[Δ0option-value]
```

The rules for specifying short options are as follows:

- Option names specified without a value can be grouped together as a block (for example, `-a -b -c` can be specified as `-abc`). In this case, a value can be specified for the last option (for example, in the specification `-abc xyz`, `xyz` is the value of option `-c`).
- Some options do not allow a space before the option value. If a space is specified before the option value for such an option, the option value will be treated as an operand.

  In the *Format* section in the descriptions of the individual commands in this chapter, an option that is shown without any space before its option value is an option in which a space cannot be specified.

(2) Specification format for long options

A long option consists of two consecutive hyphens (--) followed by a predefined character string.

The specification format for long options is shown below. Whether the option value can be omitted depends on the option.

```
--long-option-name[=option-value]
```

An option with an option value can also be specified in the format shown below. Note that in the case of an option whose option value can be omitted, the long option name must be separated from the option value by the equal sign (=). If a space is specified instead of an equal sign, the option value will be treated as an operand.

```
--long-option-nameΔ1option-value
```

Long option names and their option values cannot be abbreviated. For example, an error results if you specify `--char` *list* for the `--characters` option of the `cut` command or `--format=l` for the `--format` option of the `ls` command.

Important note

For the options that have both short and long option formats, only the short option is explained in this manual (although both are shown in the *Arguments* section of the command descriptions). For the options that do not have a short options, the long option is explained. In the *Format* section in the command descriptions, the long option names are provided after the short option names for the long options that have no relationship with other options.

For example, the `--side-by-side` long option of the `-y` option of the `diff` command is not provided in the *Format* section of the command description. On the other hand, the `--suppress-common-lines` long option is provided in the *Format* section because it does not have a corresponding short option.

(3) Notes about commands

- For the `cut`, `date`, `diff`, `ls`, `expand`, and `stat` commands, you can specify options and operands in any order. Options specified after operands will still be processed as options.

  If you specify one of the environment variables listed below, you must specify operands after all the options have been specified on the command line. Options specified after the operands will be treated as operands. For details about these environment variables, see *2.5 Specifying environment variables*.

  - `POSIXLY_CORRECT` environment variable
  - `ADSH_CMD_ARGORDER=seq` environment variable

  Note that the `POSIXLY_CORRECT` environment variable cannot be defined in JP1/Advanced Shell job definition scripts.
- If you specify options and operands in the `awk`, `find`, or `getopt` command, first specify all options and then specify operands.
- If you specify options and operands in the following commands, you can specify options and operands in any order in Linux, but you must specify options first and then specify operands in AIX, HP-UX, Solaris, and Windows:

  `cat`, `cmp`, `cp`, `egrep`, `grep`, `head`, `mkdir`, `mv`, `paste`, `rm`, `sed`, `sort`, `split`, `tail`, `touch`, `uniq`, `wc`, `which`

  If you specify the `POSIXLY_CORRECT` environment variable in Linux, the same operation as the AIX, HP-UX, Solaris, or Windows operation can be performed.

  Note that the `POSIXLY_CORRECT` environment variable cannot be specified in JP1/Advanced Shell job definition scripts. For details about the `POSIXLY_CORRECT` environment variable, see *2.5 Specifying environment variables*.

8.1.3 File path names

Files can be specified using absolute path names or relative path names.

Some commands require relative path names to be converted to absolute path names, which means that the length of the path name for a file specified in a command must sometimes be evaluated in terms of the equivalent absolute path name.

The following examples illustrate use of the absolute path name for a file specified in a command.

(1) When the file name specification is an absolute path name

The file name is specified using the absolute path name.

Example

```
/dir1/test1.asc
```

(2) When the file name specification is a relative path name

The absolute path name is the concatenation of the current directory at the time the command is executed with the path name of the specified file.

Example

```
Current directory when command is executed: /home/user1
Path name of file: dir2/test2.asc
Absolute path name of file: /home/user1/dir2/test2.asc
```

8.2 List of commands

8.2.1 List of shell operation commands

The shell operation commands can be executed from shells and from the command prompt and include the `adshexec` command (that executes a batch job).

The following table lists and describes the shell operation commands in JP1/Advanced Shell.

Table 8–1: Shell operation commands

Command name	Description	Location of command
`adshchmsg`	Replies manually to a reply-request message when a failure occurs.	Windows execution environment: *installation-folder*`\JP1ASE\bin` Windows development environment: *installation-folder*`\JP1ASD\bin` UNIX: `/opt/jp1as/sbin`
`adshcvmerg`	Merges coverage information from two `asc` files as the input and sends the output to the file at a specified path.	Windows execution environment: *installation-folder*`\JP1ASE\bin` Windows development environment: *installation-folder*`\JP1ASD\bin` UNIX: `/opt/jp1as/bin`
`adshcvshow`	Inputs and displays coverage information.	
`adshevtout`#	Outputs job definition script operation information.	
`adshexec`#	Starts a job controller process that reads a job definition script file as the input and runs a batch job in accordance with the job definition script.	
`adshfile`	Allocates a file that is to be postprocessed at the end of a job step or job.	
`adshhk`	Deletes a spool job as specified by a spool root directory name and the number of days until deletion.	
`adshlsmsg`	Displays a list of reply-request messages when a failure occurs.	Windows execution environment: *installation-folder*`\JP1ASE\bin` Windows development environment: *installation-folder*`\JP1ASD\bin` UNIX: `/opt/jp1as/sbin`
`adshmdctl` (UNIX only)	Starts and stops the daemon that manages shared memory for the user-reply functionality.	`/opt/jp1as/sbin`
`adshmsvcd` (Windows development environment only)	Service program that manages shared memory for the user-reply functionality. Used in a development environment.	*installation-folder*`\JP1ASD\bin`
`adshmsvce` (Windows execution environment only)	Service program that manages shared memory for the user-reply functionality. Used in an execution environment.	*installation-folder*`\JP1ASE\bin`

#

Can be used only in a Windows execution environment and in UNIX.

8.2.2 List of UNIX-compatible commands

Some of the UNIX-compatible commands are provided in the executable file format and some in the script format.

- Commands provided in the executable file format

  The same JP1/Advanced Shell commands can be used in both Windows and UNIX.

  For details about how to specify these commands, see *8.4 UNIX-compatible commands*.

- Commands provided in the script format (Windows only)

  Commands that depend on UNIX functionality are supported in Windows by being provided in the script format, thus enabling you to achieve some of the standard UNIX OS functionality while using the Windows functionality.

  For details about how to specify these commands, see *8.5 UNIX-compatible commands (script format) (Windows only)*.

(1) Commands provided in the executable file format

Of the UNIX-compatible commands, those provided in the executable file format can be executed within job definition scripts. You can also execute them from the Windows command prompt and UNIX shell.

The UNIX-compatible commands provided in the executable file format are stored at the following locations:

- Windows execution environment:

  installation-folder`\JP1ASE\cmd`

- Windows development environment:

  installation-folder`\JP1ASD\cmd`

- UNIX:

  `/opt/jp1as/cmd`

Some UNIX-compatible commands have limitations that reflect significant differences in OS control over resources such as the file system. In addition, there are Windows-specific limitations concerning owners and groups, access permissions, and symbolic links.

The following table describes the limitations on the supported UNIX-compatible commands provided in the executable file format.

Table 8–2: UNIX-compatible commands (executable file format)

Command name	Overview	Location of command
`awk`	Performs text processing and pattern matching.	• In Windows, if you specify a file or directory name that contains wildcards in an argument to a command that executes the `system` function, the wildcards are not expanded. • In Windows, if you specify a file or directory name that contains wildcards in an argument to a command connected by a pipe to the `getline`, `print`, or `printf` function, the wildcards are not expanded.

Command name	Overview	Location of command
basename	Obtains a file name from a path name, and then outputs it to the standard output.	No limitations
cat	Outputs files to the standard output.	No limitations
cmp	Compares binary files.	No limitations
cp	Copies files or directories.	• In Windows, if you specify the -H, -L, or -P option and set a symbolic link in an argument, or if there is a symbolic link in a directory, the link cannot be followed because symbolic links are not supported. Note also that the -r and -R options function identically. • In Windows, the -p option preserves only the modification date and file access time of the source file; directory information is not preserved.
cut	Displays selected parts of lines to the standard output.	No limitations
date	Displays the system date and time (cannot be used to set the date and time).	The -a option (set time) cannot be used.
diff	Compares two files.	No limitations
dirname	Retrieves a directory path name excluding any file name from a character string that satisfies the file path naming conventions, and then outputs the result to the standard output.	No limitations
egrep	Searches for characters in files. A specified pattern is treated as an extended regular expression. This is the same processing as when the -E option is specified in the grep command.	In Windows, symbolic links are not supported.
expand	Replaces the tab character with spaces in a line in which tab stops are set and then outputs the result to the standard output.	No limitations
expr	Evaluates an expression.	No limitations
find	Searches for files in directories.	No limitations
getopt	Analyzes command line options for easy syntax analysis of shell scripts.	No limitations
grep	Searches for characters in files.	In Windows, symbolic links are not supported.
head	Shows the first part of files.	No limitations
hostname	Displays the host name (cannot be used to set the host name).	No limitations
ls	Lists the contents of files or directories.	• In Windows, symbolic links are not supported. • In Windows, the -l option does not display the group, link count, or access

Command name	Overview	Location of command
`ls`	Lists the contents of files or directories.	permissions for users other than the owner of the file. • In Windows, the `TZ` environment variable does not apply to output. The time zone set in the **Date and Time** control panel is used.
`mkdir`	Creates directories.	In Windows, the `-m` option for setting the mode is ignored.
`mv`	Moves files or directories; changes the name of a file or directory.	• In Windows, only the access permissions of the owner are visible when overwriting. • In Windows, symbolic link are not supported. • In Windows, changing the owner is not supported in the same cases as for the `cp` command. Owner, group, `setuid` bit, and mode are not preserved.
`paste`	Concatenates multiple files in lines, and then outputs them to the standard output.	No limitations
`rm`	Removes files or directories.	• In Windows, symbolic links are not supported. • In Windows, only the access permissions of the owner are visible when overwriting.
`rmdir`	Removes empty directories.	No limitations
`sed`	Replaces character strings in text.	No limitations
`sleep`	Stops for a specified period of time.	No limitations
`sort`	Sorts text files	No limitations
`split`	Splits a file.	No limitations
`stat`	Outputs the statuses of files and directories to the standard output.	• In Windows, the statuses of the destinations of symbolic link files cannot be displayed. • In Windows, permissions cannot be displayed other than for file owners. • In Windows, `0` is always displayed as the information about the number of file blocks and the block size. • In Windows, drive numbers are displayed as device numbers. • In Windows, `0` is always displayed for an owner's user ID and for a group ID. • In Windows, `...` is always displayed for an owner's group name.

Command name	Overview	Location of command
`stat`	Outputs the statuses of files and directories to the standard output.	• In Windows, `0` is always displayed as the number of hard links. • In Windows, `0` is always displayed for the inode number. • In Windows, the destinations of symbolic link files cannot be displayed. • In Windows, `0` is always displayed for the total size of directories. • In Windows, `0` is always displayed for the major and minor device numbers. • In Windows, a file's most recent modification date and time is displayed for the file's most recent access date and time and for the file's most recent change date and time information. • In Windows, file types other than regular files or directories cannot be displayed.
`tail`	Outputs the last part of files.	No limitations
`touch`	Changes the most recent access date and time or the most recent modification date and time for a file.	• In Windows, the most recent access date and time cannot be changed. • In Windows, the most recent modification date and time cannot be changed for a directory. • In Windows, the time zone set in the `TZ` environment variable must match the time zone set in the **Date and Time** control panel. • In Windows, the precision of the modification time set in actual files depends on file system specifications.
`uname`	Displays information about the OS or hardware.	No limitations
`uniq`	Removes duplicated lines from a sorted file.	No limitations
`wc`	Counts the number of bytes, lines, characters, or words in a file.	No limitations
`which`	Obtains the paths of external commands to be executed from the command search path set in the `PATH` environment variable.	In Windows, this command supports only those external commands that satisfy the path search rules provided in the description of the `which` command.

#

The following limitations apply to all UNIX-compatible commands that are provided in the executable file format:

- In Windows, wildcards are not expanded when UNIX-compatible commands are executed from the command prompt. However, they are expanded when used in a job definition script file.
- The messages that are output can vary depending on the platform on which the command is executed.
- In Windows, you must use double quotation marks when executing commands from the command prompt.
- There are limitations to the supported files. For details, see *2.2.3 Files used in JP1/Advanced Shell*.
- The path conversion functionality is not applicable to the path names generated by commands or to the file names specified in job definition script files.
- If you use functionality that executes a program (such as the `system` function in the `awk` command or the `-exec` or `-ok` primaries in the `find` command) to execute an application with a GUI interface, the application might terminate when a batch job is executed. Note also that new jobs are generated when the `adshexec` command is executed.

(2) Commands provided in the script format (Windows only)

The UNIX-compatible commands provided in the script format can be executed only within job definition scripts.

Sample script files for the UNIX-compatible commands provided in the script format are stored at the following locations:

- Windows execution environment:
  installation-folder`\JP1ASE\sample`
- Windows development environment:
  installation-folder`\JP1ASD\sample`

Before using the UNIX-compatible commands provided in the script format, you must complete the preparations described in *2.6.6(2) Preparations for using the script-format UNIX-compatible commands (Windows only)* on the sample script files provided by JP1/Advanced Shell.

The table below lists and describes the supported UNIX-compatible commands in the script format. The provided sample script file is for Windows only. To use these commands in UNIX, use the OS-provided commands.

Table 8–3: UNIX-compatible commands (script format)

Command name	Overview
`chmod`	Changes the access permissions of files and folders.
`su`	Executes programs specified in the `su` command.
`who`	Displays information about the users who are currently logged in.

8.3 Shell operation commands

adshchmsg command (replies manually to a reply-request message when a failure occurs)

Format

```
adshchmsg [-h logical-host-name] -n reply-request-message-number {-r reply|-
d}
```

Description

This command enters a reply to a reply-request message in shared memory when a failure occurs. It can also be used to cancel the reply-waiting status of a reply-request message.

You specify the reply-request message number of the reply-request message in the `-n` option and the reply in the `-r` option. To cancel the reply-waiting status of a reply-request message, you specify the reply-request message number of the reply-request message in the `-n` option and you specify the `-d` option (which requests cancellation).

In an execution environment, this command can be executed by the root or an Administrator with an administrator role on the machine where JP1/Advanced Shell is installed. In a development environment, a general user can run this command.

Arguments

-h logical-host-name ~<logical host name>((1 to 255 bytes))

When you are operating in a logical host environment, specifies the host name of the logical host where the command will be executed.

In Windows, the length of the logical host name must not exceed 196 bytes, and it is recommended that it not exceed 63 bytes. If you specify a name that exceeds 63 bytes, the command might not work.

-n reply-request-message-number ~<unsigned integer>((1 to 2147483647))

Specifies the reply-request message number of the reply-request message that you want to reply to or whose reply-waiting status you want to cancel. Specify the reply-request message's number as displayed by the `adshlsmsg` command.

Either the `-r` option or the `-d` option must also be specified or an error will result.

If no reply request message number is specified, the option that is specified next will be treated as the argument, resulting in an error.

-r reply ~<ASCII character string>((0 to 512 bytes))

Specifies the reply to be read by the issuer of the reply-waiting event. If the reply contains a space, enclose the reply string in double quotation marks (`"`).

An error results if the `-n` option is not specified.

If you specify a character string that exceeds 512 bytes, only the first 512 bytes are treated as the reply. If a character string that contains an end-of-line code is specified, the portion of the character string following the end-of-line code will be ignored.

-d

Specifies that the reply-waiting status of the reply-request message specified in the `-n` option is to be canceled. An error results if the `-n` option is not also specified.

Return codes

Return code	Meaning
0	Normal termination
Other than 0	Error termination

Notes

- In the following cases, the item that is specified last will be the one that takes effect:
    - The -r and the -d option are both specified.
    - The -r option is specified more than once.
    - The -n option is specified more than once.

adshcvmerg command (merges coverage information)

Format

```
adshcvmerg -o output-asc-file-path-name base-asc-file-path-name merge-asc-file-path-name
```

Description

This command merges coverage information from a base asc file and a merge asc file, and then outputs the merged contents to an output asc file.

A command error results if the input files to be merged are the same file. Whether the input files are the same is determined by whether they have the same file name, including the absolute path name. It is not based on whether the contents of the input files are the same.

Arguments

In Windows, arguments are not case sensitive; in UNIX, arguments are case sensitive.

-o output-asc-file-path-name

Windows: ~<path name>((1 to 229 bytes))

UNIX: ~<path name>((1 to 1,005 bytes))

Specifies the path name of the output file to which the merge results are to be output.

base-asc-file-path-name

Windows: ~<path name>((1 to 229 bytes))

UNIX: ~<path name>((1 to 1,005 bytes))

Specifies the path name of the base asc file.

merge-asc-file-path-name

Windows: ~<path name>((1 to 229 bytes))

UNIX: ~<path name>((1 to 1,005 bytes))

Specifies the path name of the asc file to be merged with the base asc file.

Return codes

Return code	Meaning
0	Normal termination
1	The end of the file was detected in the middle of reading an asc file. The asc file is corrupted.
2	An error occurred during unlocking of a file.
3	There is an error in a command line specification.
4	There is an error in an environment variable setting. • The character encoding set in the LANG environment variable is not supported.
5	Different job definition scripts were used for the following: • Storing coverage information in the base asc file • Storing coverage information in the merge asc file
6	An error occurred during opening of a file. • This error also occurs if the file is the wrong type.
7	An error occurred during locking of a file.
8	An error occurred while the name of a file was being changed.
9	An error occurred during file input or output.
10	A memory shortage occurred.
11	An error occurred during message output processing.
12	An error occurred during processing of output to the standard error output.
13	An inconsistency in internal processing was detected.
14	An error was detected in the data format of an asc file. The asc file is invalid.
15	An error occurred during acquisition of the date and time.
16	An error occurred during acquisition of the job definition script file information.
17	An asc file cannot be processed by the command. An asc file was created by a different version.
19	An error occurred during interaction with the OS while command processing was ongoing.

Notes

- Merging is possible only when the same job definition script was used to collect the coverage information for the base asc file and the merge asc file. If different job definition scripts were used, a command error results.
- A command error results if you specify the base asc file or the merge asc file for the output asc file. The output asc file must be different from the base asc file and the merge asc file.
- A command error results if you specify the same file as the base asc file and the merge asc file.
- Whether the files specified as the base asc file and the merge asc file are the same is determined by the absolute path names of the specified files. They are considered the same file if their absolute path names are identical.

Usage example

- Merge the coverage information in JOB_user1.asc and JOB_user2.asc and send the output to JOB_user3.asc.

```
adshcvmerg -o JOB_user3.asc JOB_user1.asc JOB_user2.asc
```

adshcvshow command (displays coverage information)

Format

```
adshcvshow { [ -l n1 [- [n2]] [, n3 [- [n4]]]]...]|-s} asc-file-path-name
```

Description

This command displays coverage information for the `asc` file specified in the argument.

Arguments

-l n1 [- [n2]] [, n3 [- [n4]]]...

Specifies the range of coverage information to be displayed in terms of the line numbers of the job definition script.

Specify a range of line numbers in the format `n1 [- [n2]]`. `n1-` denotes the range beginning with line number `n1` through the last line. You can specify multiple ranges separated by the comma (`,`).

- `n1`: Line number of the starting line of the range to be displayed.
- `n2`: Line number of the final line of the range to be displayed.

If this option is not specified, all lines in the job definition script are displayed.

In the specification format for lines, a range is specified with a hyphen (`-`), and multiple ranges are separated by the comma. If you do not specify a number after the hyphen, the range will extend from the line whose line number is specified before the hyphen through the final line.

-s

Specifies that the display is to be based on the contents of a job definition script file that is being backed up.

This option is used to determine whether the `asc` file corresponds to the job definition script file or if there are differences.

asc-file-path-name

Windows: ~<path name>((1 to 229 bytes))

UNIX: ~<path name>((1 to 1,005 bytes))

Specifies the path name of the `asc` file containing the coverage information to be displayed.

Return codes

Return code	Meaning
`0`	Normal termination
`1`	The end of the file was detected in the middle of reading an `asc` file. The `asc` file is corrupted.
`2`	An error occurred during unlocking of a file.
`3`	There is an error in a command line specification.
`4`	There is an error in an environment variable setting. • The character encoding set in the `LANG` environment variable is not supported.

Return code	Meaning
6	An error occurred during opening of a file. • This error also occurs if the file is the wrong type.
7	An error occurred during locking of a file.
8	An error occurred while the name of a file was being changed.
9	An error occurred during file input or output.
10	A memory shortage occurred.
11	An error occurred during message output processing.
12	An error occurred while output was being sent to the standard error output.
13	An inconsistency in internal processing was detected.
14	An error was detected in the data format of an `asc` file. The `asc` file is invalid.
15	An error occurred during acquisition of the date and time.
16	An error occurred during acquisition of the job definition script file information.
17	The `asc` file cannot be processed by the command. The `asc` file was created by a different version.
19	An error occurred during interaction with the OS while command processing was ongoing.

Notes

- An error results if the `-s` option and `-l` option are both specified at the same time.
- An error results if the line number of the final line is less than the line number of the starting line in a specified range of lines.
- If the line number of the starting line is equal to the line number of the final line, the range will consist of only that line.
- If the line number of the starting line is greater than the number of lines in the job definition script file, the specification is ignored.
- If the line number of the final line is greater than the number of lines in the job definition script file, the final line will be the last line.
- Overlapping ranges are interpreted as the sum of the ranges. For example, `-l 1-10,5-20` is equivalent to `-l 1-20`.
- An error results if the format of a range is incorrect (example: `1-10-20`).
- An error results if `0` is specified as a line number.
- When the `-l` option is specified, the lines from `Total information` and after are not output.

Usage examples

- Display coverage information for lines 1 through 10, line 15, and lines 21 through the final line.

```
adshcvshow -l 1-10,15,21- JOB_user1.asc
```

- Display coverage information for lines 2 through 8.

```
adshcvshow -l 2-6,4-8 JOB_user1.asc
```

- If the job definition script file has 9 lines, nothing is displayed.

```
adshcvshow -l 10-15 JOB_user1.asc
```

- If the job definition script file has 9 lines, coverage information for lines 2 through 4 is displayed.

```
adshcvshow -l 10-15,2-4 JOB_user1.asc
```

adshevtout command (outputs job definition script operation information)

Format

```
adshevtout  [-s job-execution-start-time-lower-bound]
            [-e job-execution-start-time-upper-bound]
            [-c JP1/AJS-schedule-service-name]
            [-r JP1/AJS-root-jobnet-name]
            [-k JP1/AJS-job-execution-ID]
            [-n JP1/AJS-job-number]
            [-g JP1/AJS-job-name]
            [-u JP1/Advanced-Shell-execution-user-name]
            [-p job-definition-script-file-path-name]
            [-i JP1/Advanced-Shell-job-ID]
            [-j JP1/Advanced-Shell-job-name]
            [-t]
            [-d]
            [-m]
            [-z]
            [-h logical-host-name]
```

Description

This command searches the event file for job definition script operation information for jobs meeting specified conditions, and output the results in CSV format. The output destination is the standard output (`stdout`).

This command cannot be used in JP1/Advanced Shell - Developer.

Specification of output conditions

You use the following arguments to specify the jobs whose job definition script operation information is to be output:

```
 [-s job-execution-start-time-lower-bound]
 [-e job-execution-start-time-upper-bound]
 [-c JP1/AJS-schedule-service-name]
 [-r JP1/AJS-root-jobnet-name]
 [-k JP1/AJS-job-execution-ID]
 [-n JP1/AJS-job-number]
 [-g JP1/AJS-job-name]
 [-u JP1/Advanced-Shell-execution-user-name]
 [-p job-definition-script-file-path-name]
 [-i JP1/Advanced-Shell-job-ID]
 [-j JP1/Advanced-Shell-job-name]
```

If multiple output conditions are specified, the command outputs job definition script operation information only for jobs that satisfy all the specified conditions.

If no output conditions are specified, the command outputs job definition script operation information for all jobs on the physical host or the specified logical host.

If you specify job attributes (such as start date, job ID, and job name) as output conditions for the jobs whose information is to be output, the job attributes are determined by the attribute values of the root job.

The command outputs the job definition script operation information for the root job that satisfies all the output conditions and for all its child jobs. You cannot output job definition script operation information that is limited only to the root job or only to specific child jobs.

What job definition script operation information is output

Normally, job definition script header information is output on the first line, and operation information and messages are output on the second and subsequent lines.

You select the output information by specifying the following arguments to the `adshevtout` command:

```
[-t]
[-d]
[-m]
[-z]
```

For examples of the output of operation information, see *3.6.9 Job definition script operation information that is output*.

When the following argument is specified, the command outputs operation information for jobs executed on the specified logical host:

```
 [-h logical-host-name]
```

When this argument is not specified, the command outputs operation information for jobs executed on the physical host.

Arguments

-s job-execution-start-time-lower-bound

Specifies a lower boundary for the job execution start date and time as a condition for determining the jobs whose job definition script operation information is to be output.

For details about the specification format of the date and time, see *Job execution start date and time* below.

If this argument is omitted, there is no limitation on the lower boundary for the execution start date of the jobs to be output.

-e job-execution-start-time-upper-bound

Specifies an upper boundary for the job execution start date and time as a condition for determining the jobs whose job definition script operation information is to be output.

For details about the specification format of the date and time, see *Job execution start date and time* below.

If this argument is omitted, there is no limitation on the upper boundary for the execution start date of the jobs to be output.

-c JP1/AJS-schedule-service-name

Specifies a JP1/AJS schedule service name as a condition for the jobs whose job definition script operation information is to be output.

You can output jobs that match either of the following as character strings:

- The scheduler service name specified in this argument

- The name of the JP1/AJS scheduler service that started the job (the value of the `AJS_AJSCONF` environment variable that was specified when JP1/AJS started the job)

-r JP1/AJS-root-jobnet-name

Specifies a JP1/AJS root jobnet name as a condition for the jobs whose job definition script operation information is to be output.

You can output jobs that match either of the following as character strings:

- The root jobnet name specified in this argument
- The JP1/AJS root jobnet name at the time the job was started (the value of the `AJSNETNAME` environment variable that was specified when JP1/AJS started the job)

-k JP1/AJS-job-execution-ID

Specifies a JP1/AJS job execution ID as a condition for the jobs whose job definition script operation information is to be output.

You can output jobs that match either of the following as character strings:

- The job execution ID specified in this argument
- The JP1/AJS job execution ID at the time the job was started (the value of the `AJSEXECID` environment variable that was specified when JP1/AJS started the job)

-n JP1/AJS-job-number

Specifies a JP1/AJS job number as a condition for the jobs whose job definition script operation information is to be output.

You can output jobs that match either of the following as character strings:

- The job number specified in this argument
- The JP1/AJS job number at the time the job was started (the value of the `JP1JobID` environment variable that was specified when JP1/AJS started the job)

For example, to specify the job with job number 0000012345, you must specify `-n 0000012345`, with the leading zeros. If you specify `-n 12345`, it is not considered a match to the job number.

Note that job number formats might differ between platforms.

For details about job numbers, see the applicable JP1/AJS manual.

-g JP1/AJS-job-name

Specifies a JP1/AJS job name as a condition for the jobs whose job definition script operation information is to be output.

You can output jobs that match either of the following as character strings:

- The job name specified in this argument
- The JP1/AJS job name at the time the job was started (the value of the `AJSJOBNAME` environment variable that was specified when JP1/AJS started the job)

-u JP1/Advanced-Shell-execution-user-name

Specifies the execution user name of the process running the `adshexec` command that executed the job as a condition for the jobs whose job definition script operation information is to be output.

You can output jobs that match either of the following as character strings:

- The user name specified in this argument
- The user name for the process running the `adshexec` command that executed the job

-p job-definition-script-file-path-name

Specifies the path name of the job definition script file specified in the `adshexec` command when the job was executed as a condition for the jobs whose job definition script operation information is to be output.

You can output jobs that match either of the following as character strings:

- The path name specified in this argument
- The path name of the job definition script file specified in the `adshexec` command

Even if the path name can be interpreted as being for a job's job definition script file, if there is no match as a character string, the job will not be considered as a job whose job definition script operation information is to be output. The following is an example:

Example of when there is not a path name match:

Current directory when the `adshexec` command is executed: `/home/user1`

Path name specified in the `adshexec` command: `./test1.ash`

Path name specified in the `adshevtout` command: `/home/user1/test1.ash`

-i JP1/Advanced-Shell-job-ID

Specifies a JP1/Advanced Shell job ID as a condition for the jobs whose job definition script operation information is to be output.

You can output jobs that match either of the following as character strings:

- The job ID specified in this argument
- The JP1/Advanced Shell job ID of the job (the value of the `ADSH_JOBID` environment variable)

For example, to specify the job whose job ID is 000001, you must specify `-i 000001`. The leading zeros cannot be omitted.

-j JP1/Advanced-Shell-job-name

Specifies a JP1/Advanced Shell job name as a condition for the jobs whose job definition script operation information is to be output.

You can output jobs that match either of the following as character strings:

- The job name specified in this argument
- The JP1/Advanced Shell job name (the value of the `ADSH_JOB_NAME` environment variable)

-t

Specifies that output of the header information is to be suppressed when the job definition script operation information is output.

-d

Specifies that output of the job definition script operation information is to be suppressed.

You use this argument when you want to output the header information only.

-m

Specifies that only the messages are to be output as the job definition script operation information.

-z

Specify that output of information about environment variables is to be suppressed when the job definition script operation information is output.

-h logical-host-name

Specifies the name of a logical host that is executing jobs whose job definition script operation information is to be output.

The `adshevtout` command outputs job definition script operation information from the event files in the spool corresponding to the specified logical host.

In Windows, the length of the logical host name must not exceed 196 bytes, and it is recommended that it not exceed 63 bytes. If you specify a name that exceeds 63 bytes, the command might not function correctly.

This argument is for specifying an execution environment. It is not for specifying a condition for the jobs whose job definition script operation information is to be output.

The -h specification is ignored if the logical host name specified in this argument is not defined in an environment file.

Return codes

Return code	Meaning
0	Normal termination
1	There is an error in a command line specification.
2	There is an error in an environment variable setting.
3	The character encoding set in the LANG environment variable is not supported.
4	There is an event file for which processing was skipped. Check the messages output by the command to determine the cause.
5	An I/O error occurred in referencing the spool, and the spool could not be referenced.
6	The spool could not be referenced because it is being accessed by another command.
7	An error occurred during message output processing.
8	An error occurred during processing of output to the standard output.
10	An error occurred during acquisition of the date and time.
11	A memory shortage occurred.
12	An inconsistency in internal processing was detected.
13	An error occurred during initialization processing.

The return code will generally be the maximum value among the return codes for all events that occurred during command execution, except for return code 4, which is output only when no other events occurred.

Specifying the same argument more than once

If the same argument is specified multiple times, the last specification takes effect.

Example:

The following case is treated as specifying -s 19900401:

```
adshevtout -s 20120411  -s 19900401
```

Combinations of different arguments

When multiple arguments for determining the information to be output are specified, the arguments are interpreted according to the priority order shown in the following table:

Argument priority#	Argument	Description
1	-d	The operation information is not output.
2	-m	Only messages are output.
3	-z	Information about environment variables is not output.

#

The smallest number is the highest priority.

When a higher-priority argument is specified, any specified lower-priority arguments will be ignored.

If the format of a specified argument is invalid, a command error results, regardless of the argument's priority.

Job execution start date and time

The lower and upper boundaries for the job execution start date and time can be specified in any of the three formats listed and described in the table below.

The range of years that can be specified in `YYYY` is 1970 through 2038.

Table 8–4: Specification and interpretation of the lower and upper boundaries for job execution start date and time

Specification format of the date and time	Interpretation of the specification
YYYYMMDD`,`*hhmmss*	Specifies the year, month, date, hour, minute, and second. An upper bound is interpreted as the specified time plus 1 second. If the time is specified as `235959`, it is interpreted as 00:00:00 on the date following the specified date.
YYYYMMDD	Specifies the year, month, and date only. The hour, minute, and second are interpreted as follows. Lower bound: Interpreted as 00:00:00 on the specified date. Upper bound: Interpreted as 00:00:00 on the date following the specified date.
`,` *hhmmss*	Specifies the hour, minute, and second only. The year, month, and date are interpreted as the execution date of the command. An upper bound is interpreted as the specified time plus 1 second. If the time is specified as `235959`, it is interpreted as 00:00:00 on the date following the specified date.

Examples of interpreting the date and time are shown in the table below. For Nos. 5 through 7 in the table, it is assumed that the `adshevtout` command was executed on October 23, 2012.

No.	Specification in the command	Date and time interpreted by the command	
		Lower bound for the execution start date and time	Upper bound for the execution start date and time
1	`20120501,000000`	May 1, 2012, 00:00:00	May 1, 2012, 00:00:01
2	`20120501,100000`	May 1, 2012, 10:00:00	May 1, 2012, 10:00:01
3	`20120501,235959`	May 1, 2012, 23:59:59	May 2, 2012, 00:00:00
4	`20120501`	May 1, 2012, 00:00:00	May 2, 2012, 00:00:00
5	`,000000`	October 23, 2012, 00:00:00	October 23, 2012, 00:00:01
6	`,100000`	October 23, 2012, 10:00:00	October 23, 2012, 10:00:01
7	`,235959`	October 23, 2012, 23:59:59	October 24, 2012, 00:00:00

The date and time specified in the command are interpreted on the basis of the time zone set in the `TZ` environment variable at the time the command executes.

It is important to note that even if the same date and time character string is specified in two different commands, if those commands are executed in different time zones (according to the values set in the respective `TZ` environment variables), the dates and times will be interpreted differently.

The dates and times that can be specified are as follows:

- Expressed in Coordinated Universal Time (UTC):
  January 1, 1970, 00:00:00 through January 19, 2038, 03:14:07
- Expressed in Japan Standard Time (UTC+9):
  January 1, 1970, 09:00:00 through January 19, 2038, 12:14:07

For time zones not listed above, the representation of the specifiable range of dates and times depends on the time zone being used.

Note that there are time-related implementation differences among OSs, so errors might result even for dates that fall within the ranges listed above. The following circumstances will generate an error:

- An invalid date or time is specified.
- The lower bound for the execution start date and time is later than the upper bound.

TZ Environment variable

The time zone specified in the `TZ` environment variable is referred to for the following purposes:

- Interpretation of the date and time specified in an argument of a command when the command executes
- Representation of a date and time in job definition script operation information

For details about the relationship between time zones and the representation of dates and times, see *3.6.3 Relationship between dates and times and time zones in the operation information*.

The `TZ` environment variable is specified in POSIX format as illustrated below. Note the use of the sign.

Example:

```
export TZ=JST-9
```

The `TZ` environment variable cannot be specified in Time Zone Database format, as illustrated below.

Examples of incorrect usage:

```
export TZ=Asia/Tokyo
export TZ=Japan
```

Do not execute the `adshevtout` command with the `TZ` environment variable set to daylight saving time. The `adshevtout` command does not support daylight saving time.

For details about setting the `TZ` environment variable, see the specifications for the OS being used.

Lower bound of the job execution start date and time

If you specify a lower bound for the job execution start date and time with `-s`, the jobs whose job definition script operation information will be output must satisfy the following constraint:

$ts \leq tj$

tj: Job execution start date and time

ts: Lower bound for the job execution start date and time interpreted by the `adshevtout` command

Upper bound of the job execution start date and time

If you specify an upper bound for the job execution start date and time with `-e`, the jobs whose job definition script operation information will be output must satisfy the following constraint:

$tj < te$

tj: Job execution start date and time

te: Upper bound for the job execution start date and time interpreted by the `adshevtout` command

Job definition script operation information that can be output

- The job definition script operation information that can be output is what is stored in the event files that are accessible to the user who executed the command. If an applicable event file is not accessible to the user who executed the command, a message is output and job definition script operation information in the inaccessible event file is not output.
- Job definition script operation information is not output for jobs that are executing or for jobs for which the `adshexec` command terminated with an error. This means that job definition script operation information is not output for a job whose name in the spool job directory name is in one of the following forms:
  - *job-ID*
  - *job-ID*-
- Job definition script operation information is not output for deleted jobs.
  A job is considered to have been deleted if the spool job management file indicated below does not exist immediately below the spool job directory:
  - UNIX: `.sysout`
  - Windows: `sysout.ini`

Concurrent execution with the adshhk command

- The `adshevtout` command and the `adshhk` command (for deleting spool jobs) apply exclusive control to spool directories.
- The `adshevtout` command and the `adshhk` command cannot run at the same time on the same spool directory.
- More than one `adshevtout` command can be executed on the same spool directory at the same time.
- If the `adshevtout` command cannot be executed because the spool directory is under exclusive control, an error message is output and processing terminates.

Notes

- If there is a large quantity of job definition script operation information to be output, you can split it up by executing separate commands with output conditions that divide the information on the basis of dates or other criteria.

Usage example

- Output to the `out.csv` file the job definition script operation information for JP1/Advanced Shell job ID 000100.

```
adshevtout -i 000100  >  out.csv
```

adshexec command (executes a batch job)

Format

To run normally

```
adshexec [-v] [-c] [-m {EXTENDED|SIMPLE|MINIMUM}]
[-t [-f] [-o asc-file-path-name]] [-h logical-host-name]
[-s {SPOOL|PARENT}] [-x]
{-r command-line|job-definition-script-file-path-name}
[run-time-parameters]
```

To start in the debugger mode (UNIX only)

```
adshexec -d [-v] [-c] [-m {EXTENDED|SIMPLE|MINIMUM}]
[-t [-f] [-o asc-file-path-name]] [-h logical-host-name]
[-x] job-definition-script-file-path-name
```

Description

This command launches the job controller to execute the batch job in the job definition script file specified in an argument. You can also execute commands that would be specified in a job definition script file by directly specifying those commands in the `-r` option.

The command's arguments are specified before the values to be passed to the job definition script's positional parameters.

Arguments

-d (UNIX only)

Specifies that the job controller is to be started in the debugger mode. This option can be used in a UNIX environment. In the debugger mode, coverage information is collected in memory and continues to be collected each time the `run` command is executed. The coverage information stored in memory can be displayed by the `info coverage` command.

If the `-t` option is not specified, the coverage information collected in memory is discarded when you terminate the debugger with the `quit` command.

Job definition script operation information is not collected when the `-d` option is specified.

-v

Specifies that version information is to be displayed; no batch job is executed.

-c

Specifies that the job definition script file's syntax is to be checked.

Only syntax checking is performed; no batch job is executed.

-m {**EXTENDED**|**SIMPLE** | **MINIMUM**}

Specifies how to output the standard output and the standard error output of the job that is to be started. For details about the output modes, see *3.3.4 Suppressing output of information and warning messages to job execution logs*.

- `EXTENDED`

  Use the expansion output mode.

- `SIMPLE`

  Use the simple output mode.

- `MINIMUM`

Use the minimum output mode.

If this option is omitted, the specifications of the `OUTPUT_MODE_ROOT` and `OUTPUT_MODE_CHILD` parameters take effect.

This option applies only to the job that is started with the `adshexec` command and is not inherited by jobs that are started from that job. For a job that is started separately, specify the `-m` option again in the `adshexec` command that you use to start the job.

If the root job is run in the simple output mode or the minimum output mode, the standard output is not redirected to a spool file even when the `-s` option is specified or `SPOOL` is specified in the `OUTPUT_STDOUT` environment setting parameter.

If you use the `-r` option, also specify `-m SIMPLE` or `-m MINIMUM` to avoid job execution logs from being mixed together with the output results.

-t

Specifies that coverage information is to be collected while the batch job executes and the collected coverage information is to be output to an `asc` file when the command terminates.

If the `-t` option is not specified when the debugger mode is started in UNIX, the coverage information is collected in memory only (however, it can be displayed with the debugger's `info coverage` command). When you terminate the debugger with the `quit` command, the collected coverage information is discarded.

-f

Specifies that if coverage information has already been collected, the existing `asc` file is to be overwritten when differences are detected between the job definition script file to be executed and the backup information. The `-f` option can be specified only when the `-t` option is also specified.

When this option is specified

If coverage information has already been collected and differences are detected between the job definition script file to be executed and the backup information, the backup information is discarded and new coverage information is collected.

If no coverage information has been collected, the newly collected coverage information is output to an `asc` file.

When this option is not specified

If coverage information has already been collected and differences are detected between the job definition script file to be executed and the backup information, a command error occurs and the job definition script file is not executed. The `asc` file is not updated.

-o asc-file-path-name

Windows: ~<path name>((1 to 229 bytes))

UNIX: ~<path name>((1 to 1,005 bytes))

Specifies the path name for the `asc` file to be used for collection of coverage information, when you want to use a different file from the existing `asc` file. The `-o` option can be specified only when the `-t` option is also specified.

Omitting this option is equivalent to specifying the `asc` file in the current directory when the `adshexec` command is executed. The following shows the format of an `asc` file name:

name-of-job-definition-script-without-extension_user-name`.asc`

In the example below, the `adshexec` command is executed under the following conditions:

- Current directory when the `adshexec` command is executed: `/home/user1/test`
- User name: `user1`
- Name of job definition script: `script1.ash`

The name of the `asc` file when the `-o` option is omitted is as follows:

```
/home/user1/test/script1_user1.asc
```

-h logical-host-name ~<logical host name>((1 to 255 bytes))

Specifies the name of the logical host to be used for execution on a logical host. In Windows, the length of the logical host name must not exceed 196 bytes, and it is recommended that it not exceed 63 bytes. If you specify a name that exceeds 63 bytes, the command might not function correctly.

If an empty string is specified for *logical-host-name*, the value specified in the `JP1_HOSTNAME` environment variable is assumed. If the `JP1_HOSTNAME` environment variable is not set, the `KNAX0220-E` message is output and the command terminates. For details about the `JP1_HOSTNAME` environment variable, see the *Job Management Partner 1/Base User's Guide*.

Do not specify this option when running on the physical host.

-s {SPOOL|PARENT}

Specifies the destination for the standard output from the root job. child jobs assume that `PARENT` was specified in this option.

If this option is omitted, the root job assumes the value specified in the `OUTPUT_STDOUT` parameter. If the root job is run in the simple output mode or the minimum output mode, the standard output is not redirected to a spool file, regardless of this option.

- `SPOOL`

  Output the standard output from the root job to a file on the spool.

- `PARENT`

  Output the standard output from the root job to the destination inherited from the parent process when the process starts. Assuming the destination is not later redirected in the parent process, output goes to the same destination as for the parent process.

-x

Enables the `xtrace` shell option.

You can disable the `xtrace` shell option by executing `set +x` or `set +o xtrace` in the job definition script.

-r *command-line*

Specifies what is to be executed from the command line. In *command-line*, you can specify any commands that can be specified in a job definition script, such as standard shell commands and UNIX-compatible commands. Make sure that the *command-line* specification does not exceed the maximum length permitted for a line of a job definition script (8,191 bytes).

If this option is specified together with the `-v` option, the `-v` option takes effect.

In UNIX:

This argument cannot be specified together with the `-c`, `-d`, or `-t` option. An error results if this argument is specified together with any of these options.

In Windows:

This argument cannot be specified together with the `-c` or `-t` option. An error results if this argument is specified together with either of these options.

For details, see *3.2.4 Specifying what is to be executed by a job from the command line*.

job-definition-script-file-path-name

Windows: ~<path name>((1 to 247 bytes))

UNIX: ~<path name>((1 to 1,023 bytes))

Specifies the path name of the job definition script file that is to be run.

run-time-parameters ~<any character string>((1 to 1,022 bytes))

Specifies values to be stored in the positional parameters of the job definition script. If a run-time parameter includes a space, you must enclose that string in double quotation marks (").

Return codes

Trigger	Return code	
	-c option specified	-c option not specified
Normal execution of the exit command or of the return command from an outside function	Return code specified in the command	--
Normal execution of an external command by specifying it in the argument to the exec command	Return code of the external command specified in the argument	--
Normal execution of the job definition script through the end of the job definition script file	Return code of the most recently executed standard shell command or extended script command	--
No errors in the job controller in the debugger mode	0	--
No syntax errors in the job definition script file	The job definition script is executed, with the following return code: • Return code specified in a command • Return code of the most recently executed standard shell command or extended script command	0
Syntax error in the job definition script file	1 or the value set in the ADSH_JOBRC_FATAL environment variable	1 or the value set in the ADSH_JOBRC_FATAL environment variable
An error in the job controller, except for an error in executing the job definition script, such as an error in reading an environment file	1 or the value set in the ADSH_JOBRC_FATAL environment variable	1 or the value set in the ADSH_JOBRC_FATAL environment variable
The job controller process receives a signal and terminates (UNIX only).	128 + signal number	128 + signal number
The job controller process is forcibly terminated from outside, for example from JP1/AJS or the Windows Task Manager (Windows only).	Return code specified by the program that forcibly terminated the job controller	Return code specified by the program that forcibly terminated the job controller
Failure to start the job controller for a reason attributable to the OS (Windows only)	1 to 3	1 to 3
A parsing error occurred in the ADSH_JOBRC_FATAL environment variable.	255	255

Legend:

--: Does not apply because the job definition script is not executed.

Notes

- If the same option is specified more than once, the last specification takes effect.
- When -v and -c are specified together with any other options, they are handled in descending order of priority. The priority order is -v, then -c, and then the other options. Lower priority options are ignored.

  Example

  The -d option is ignored, only -v takes effect.

```
    $ adshexec -v -d MyShell.ash
```

- Collecting coverage information without specifying the `-o` option can result in duplication of `asc` file names. This duplication occurs in the situations listed below. To avoid duplicate `asc` file names, change the name of the job definition script file or the name of the `asc` file.
  - You collect coverage information from multiple job definition scripts whose file names differ only in their extensions

    Example: `sc.1` and `sc.2`
  - You collect coverage information from multiple job definition scripts with the same name but in different directories

    Example: `/dir1/sc1` and `/dir2/sc1`
- Do not specify file names that begin with `.` (dot).
- Do not use a reserved device name (such as `CON`, `AUX`, and `NUL`) as a file name. (Windows only)
- Do not use an NTFS stream as a file name. (Windows only)
- You cannot specify run-time parameters when the `-d` option is specified. Specify the run-time parameters in an argument of the `run` command.
- Access permissions for `asc` files are set as follows:
  - The owner (creator) of a file is granted `r` (read) and `w` (write) access permissions regardless of the umask settings. Group and general access permissions are set according to the umask settings at the time the command is invoked. (UNIX only)
  - In principle, execution users are granted Full Control access permissions over their own `asc` files. However, actual file access permissions are affected by permission inheritance in Windows (inheritance of permissions from higher-level directories). Similarly, access permissions for other users obey Windows permissions inheritance (inheritance of permissions from higher-level directories). (Windows only)
- File descriptors are closed without being inherited by `adshexec` commands that are generated as child processes. For example, an error results if the job definition script of a child process attempts to perform I/O using a file descriptor opened by a parent process without first reopening it. Standard output and standard error output do not need to be reopened. (Windows only)

Usage examples

- Start the job controller in the syntax checking mode.

```
adshexec -c /home/user/shell/JOB.ash
```

- Start the job controller in the debugger mode.

```
adshexec -d /home/user/shell/JOB.ash
```

- Display version information for the job controller without executing a batch job.

```
adshexec -v
```

- Specify run-time parameters to be passed to the positional parameters of the job definition script and start the job controller.

```
adshexec /home/user/shell/JOB.ash parm1 parm2
```

- Collect coverage information.

```
adshexec -t /home/user/shell/JOB.ash
```

- If the contents of the job definition script file have been modified, discard the coverage information collected so far and collect new coverage information.

```
adshexec -t -f /home/user/shell/JOB.ash
```

- Collect coverage information and store it in `/home/user/JOB.asc`.

```
adshexec -t -o /home/user/JOB.asc /home/user/shell/JOB.ash
```

- Enable the `xtrace` shell option and start the job controller.

```
adshexec -x /home/user/shell/JOB.ash
```

- Start the job controller with a command specified in *command-line* in the `-r` option that is to be executed in the job.

```
adshexec -r "ls *"
```

- Start the job controller with a positional parameter that is referenced in *command-line* in the `-r` option specified in the runtime parameters.

  This example specifies a command in the job definition script file. *command-line* must be enclosed in single quotation marks (`'`) because it specifies a positional parameter.

```
adshexec -r 'cat $1 | grep $2' file.txt abc
```

- Start the job controller with a command that handles file paths specified in *command-line* in the `-r` option.

  This example specifies a command in the job definition script file.

```
adshexec -r 'cat "C:\\Documents and Settings\\user001\\file.txt"'
```

adshfile command (specifies the allocation and postprocessing of regular files)

Format

```
adshfile  [-s {step|job}] [-n {del|keep}] [-a {del|keep}]
          [-c {exist|no}] file-path-name
```

Description

This shell operation command allocates a regular file, checks whether the regular file already exists, and specifies postprocessing. The command takes effect when it is specified in a job definition script executed by JP1/Advanced Shell's job controller. You can specify a maximum of 64 regular files to be allocated.

For details about the allocation and postprocessing of regular files and the difference from the `#-adsh_file` command, see *5.9.1 Allocating regular files and performing postprocessing*.

Regular files allocated with this command are managed separately from regular files allocated with the `#-adsh_file` command. Postprocessing is performed on the regular files allocated by the `adshfile` command first and then on the regular files allocated by the `#-adsh_file` command. If the same file is allocated with both commands, the file will be postprocessed twice and an error might result.

Note the following about executing this command:

- Do not execute this command asynchronously.
- In UNIX, do not execute this command as a separate process.
- This command cannot be executed if the spool job creation suppression functionality is being used.

If the same option is specified more than once, the last specification takes effect.

Arguments

-s {step|job}

Specifies the timing for performing postprocessing on the file.

- `step`

  Perform postprocessing on the file when the job step terminates.

- `job`

  Perform postprocessing on the file when the job terminates.

You can issue this command within the job or job step regardless of the specified option, but the registered file is postprocessed when the next job step or job terminates. If no job step terminates after the file is registered with `step` specified, postprocessing is performed when the job terminates.

If this option is specified in a child job, this processing is performed within that child job.

-n {del|keep}

Specifies the postprocessing to be performed when the corresponding job step or job terminates normally.

- `del`

  Delete the allocated regular file after the corresponding job step or job has terminated.

- `keep`

  Do not delete the allocated regular file after the corresponding job step or job has terminated.

-a {del|keep}

Specifies the postprocessing to be performed when the corresponding job step or job terminates with an error.

- `del`

  Delete the allocated regular file after the corresponding job step or job has terminated.

- `keep`

  Do not delete the allocated regular file after the corresponding job step or job has terminated.

If file allocation processing by the `adshfile` command results in an error, the following takes place:

- Postprocessing on the regular file specified in the corresponding `adshfile` command is not performed.
- Of the regular files allocated with the `adshfile` command beforehand, postprocessing is performed according to the `-a` options specified in the `adshfile` commands.

-c {exist|no}

Specifies whether the file path is to be checked for its existence.

- `exist`

  Check whether the file path exists.

  Postprocessing is registered only if the file path exists.

  If the file path does not exist, the command terminates with an error.

- `no`

Do not check whether the file path exists.

file-path

Windows: ~<path name>((1 to 247 bytes))

UNIX: ~<path name> ((1 to 1,023 bytes)

Specifies the path of the regular file that is to be allocated.

You can specify either a relative path or an absolute path. If a relative path is specified and its file path name exceeds the maximum permissible length after it has been converted to an absolute path name, an error results.

Return code

Return code	Meaning
`0`	Normal termination
`99`	Error termination

Notes

Unlike the `#-adsh_file` command, this command does not output to the job execution logs a message indicating the allocation results. To check the file path name specified in an argument of the `adshfile` command for purposes such as troubleshooting, do the following:

1. Collect operation information by specifying `YES` in the `EVENT_COLLECT` parameter in the environment file.
2. Execute the `adshfile` command in the job definition script.
   Operation information will be collected.
3. Execute the `adshevtout` command to output the operation information.
4. In the displayed operation information, check the lines in which `command`, indicating command information, is set in the `EvtName` column (the first column, which shows the type of operation information record).

Note that operation information for job definition scripts can be collected only in the normal mode in the execution environment. This information cannot be collected in the debugger mode in the execution environment or in the development environment.

adshhk command (deletes spool jobs)

Format

```
adshhk target-list-file-name report-file-name log-file-name [number-of-days]
```

Description

This command deletes spool jobs from the spool directories identified in the list file specified in *target-list-file-name*. The execution results are output in CSV format to the file specified in *report-file-name*. Messages output during execution, such as error messages, are output to the file specified in *log-file-name*.

This command and the `adshevtout` command apply exclusive control over spool directories. If the `adshhk` command is unable to obtain a lock on a spool directory, it skips processing of that spool directory and outputs the `KNAX4425-E` message.

Arguments

target-list-file-name

Specifies the file name of the file that specifies the spool directories that contain spool jobs that are to be targets for deletion. You specify in advance in the target list file the names of the spool root directories containing the spool directories to be deleted and for each a number-of-days value. Spool jobs that were executed more than the specified number of days prior (the count begins from the day before the `adshhk` command was executed) are deleted from the specified spool directory.

The target list file can contain multiple lines in text file format. A line cannot exceed 4,095 bytes, counting from the beginning of the line through the end-of-line code at the end. The values must be enclosed in double quotation marks (`"`).

The format of the target list file is:

```
"spool-root-directory-name"[,"number-of-days"]
```

The following explains each item.

spool-root-directory-name ~<path name>((1 to 128 bytes))

Specifies the name of a spool root directory whose spool jobs are candidates for deletion. Specifying the full path is recommended.

number-of-days ~<unsigned integer>((1 to 999))

Specifies a days-count value to be used to determine the spool jobs to be deleted from the spool directory. The spool job directory of batch jobs that were executed more than the specified number of days prior (the count begins from the day before the `adshhk` command was executed) are deleted. If this value is omitted in the list file for a spool directory, the value specified in the *number-of-days* argument in the `adshhk` command is used. If no *number-of-days* value is specified in the file or in the command argument, the line generates an error, but subsequent lines are processed.

Specifying `""` is same as omitting the *number-of-days* specification.

report-file-name

Specifies the name of the output file for the execution results. The report file is output in CSV format. If the specified file does not exist, it is created. If it already exists, its existing contents are overwritten.

The report file access permissions are set as follows:

- Windows: According to the settings for the output folder.
- UNIX: `600`

For an example of a report file's output, see *3.8 Deleting spool jobs*.

log-file-name

Specifies the name of the output file for error messages and other messages. If the specified file does not exist, it is created. If it already exists, its existing contents are overwritten.

The log file access permissions are set as follows:

- Windows: According to the settings for the output folder.
- UNIX: `600`

number-of-days ~<unsigned integer>((1 to 999))

Specifies a days-count value to be used to determine the spool jobs to be deleted from the spool directories. The spool job directory of batch jobs that were executed more than the specified number of days prior (the count begins from the day before the `adshhk` command was executed) are deleted. This argument takes precedence over the *number-of-days* values specified in the target list file. When this argument is omitted, the *number-of-days* values specified in target list file are used. If you omit this argument, you must specify *number-of-days* values in the target list file.

Return codes

Return code	Meaning
0	Normal termination
1	Error termination
2	Deletion failed because a spool directory is being used by another program (the return code might be other than 2 if other errors occurred).
253	Error in standard error output

Notes

- The only spool jobs subject to deletion are those for which the user who executes the command has deletion permissions. A failure to delete will be reported for spool jobs for which deletion permissions are lacking. If you want to delete the spool jobs of all users, you must execute the deletion as a user who has deletion permissions for all spool jobs.
- Files created under a spool job directory for which the user has deletion permissions are deleted regardless of whether they were created as batch jobs.
- Subdirectories created under a spool job directory might not be deletable.
- If a job's execution start date is unknown, deletion will not be performed (this will be treated as an error).
- The only spool jobs that are deleted are those whose spool job directories are in the format *job-ID*−*job-name* or *job-ID*−. When a spool job directory is missing the hyphen (−) following the job ID, it indicates that the batch job is still running, it was terminated improperly by means other than one described in *3.10.1 How to forcibly terminate jobs*, or some similar situation. Such directories are never deleted, regardless of the actual state of affairs.
- If an error occurs during deletion, deletion processing on the affected spool job might be in a partially completed state.
- Results are not output to the report file in number order. If necessary, a sort program can be used to sort them.
- No deletions will be performed in a specified spool directory that is being processed by the `adshevtout` command (which outputs job definition script operation information).
- Because spool jobs to be deleted are deleted from the spool job management file, if deletion processing is interrupted, they might not be deleted even if you re-run the `adshhk` command. In such a case, you must delete manually any spool jobs that failed to be deleted.
- To manually delete a spool job that the `adshhk` command failed to delete, use a command such as rm -r to delete both the spool job directory and all the files within the directory. When you do so, use the creation date of the spool job directory and of the files contained in the directory to check whether the job has completely finished. We recommend that you stop operations such as batch jobs before deleting the directory and files.

Usage example

- Delete the following batch jobs:
  - Batch jobs in the `/home/user001/jp1as/spool` directory that ran more than seven days ago
  - Batch jobs in the `/home/user999/jp1as/spool` directory that ran more than 30 days ago

  Specify the following information in advance in target list file `/home/kanrisya/hk/target`:

  ```
  "/home/user001/jp1as/spool","7"
  "/home/user999/jp1as/spool","30"
  ```

  In this case, execute the command shown below. The report is stored in the `/home/kanrisya/hk/result.csv` file.

```
adshhk /home/kanrisya/hk/target  /home/kanrisya/hk/result.csv  /home/
kanrisya/hk/result.log
```

For an example of a report that is created, see *3.8 Deleting spool jobs*.

adshlsmsg command (displays a list of reply-request messages when a failure occurs)

Format

```
adshlsmsg  [-h logical-host-name] [-n reply-request-message-number]
```

Description

This command displays a list of the reply-request messages in shared memory that are in reply-waiting status, as well as the reply-request messages in receive-waiting status and the replies to them.

In an execution environment, this command is executed by Administrators or root users with an administrator role on the machine where JP1/Advanced Shell is installed. In a development environment, general users can run this command.

Arguments

-h logical-host-name ~<logical host name>((1 to 255 bytes))

When you are operating in a logical host environment, specifies the host name of the logical host where the command will be executed.

In Windows, the length of the logical host name must not exceed 196 bytes, and it is recommended that it not exceed 63 bytes. If you specify a name that exceeds 63 bytes, the command might not function correctly.

-n reply-request-message-number ~<unsigned integer>((1 to 2147483647))

Specifies the reply-request message number of a reply-request message in reply-waiting status that you want to display, or of a reply-request message in receive-waiting status that you want to display together with the reply to it.

If only `-n` is specified and no reply-request message number is specified, any option that is specified next will be treated as the argument.

When this specification is omitted, the display is of all reply-request messages in shared memory on reply-waiting status, as well as of all reply-request messages in receive-waiting status and the replies to them.

If the `-n` option is specified more than once, the last specification takes effect.

Output items

The following describes the headers and contents that are output as a result of executing the `adshlsmsg` command.

- `MESSAGE-NO`

  The 10-digit decimal reply-request message number. This is the value that is specified in the `-n` option of this command or in the `-n` option of the `adshchmsg` command.

- `STATUS`

  The status of the reply-request message is output as follows.

  - `Wait`: The reply-request message is in reply-waiting status

    - `Set`: The reply-request message is in receive-waiting status

- `JOBID`

  The six-digit integer value representing the job ID issued by the `adshread` command for the job definition script.

- `LINENO`

  The line number in the job definition script executed by the `adshread` command.

- `DATE/TIME`

  The time the reply-request message was output (local time).

- `MESSAGE or RESPONSE`

  The following items are output:

    - `msg=`: Reply-request message body
    - `res=`: Reply contents (displayed only if in receive-waiting status)

Return codes

Return code	Meaning
`0`	Normal termination
Other than `0`	Error termination

Usage example

The following example shows the output when you view a list of reply-request messages in reply-waiting status, as well as reply-request messages in receive-waiting status and the replies to them.

```
$ adshlsmsg
 MESSAGE-NO   STATUS JOBID      LINENO DATE/TIME         MESSAGE or RESPONSE
[0000017622] [Wait] 000228          20 12/05/24 18:28:00 msg=STOP
[0000017626] [Set ] 000229         136 12/05/24 18:28:10 msg=Continue (Y/N)?
[0000017626] [Set ] 000229         136 12/05/24 18:28:10 res=Y
```

adshmdctl command (starts and stops the user-reply functionality management daemon) (UNIX only)

Format

```
adshmdctl [-h logical-host-name] {start [reuse]|stop|status|conftest
[environment-file-name]|help}
```

Description

This command starts and stops the user-reply functionality's management daemon. The user-reply functionality's management daemon manages the shared memory for the user-reply functionality.

When the user-reply functionality is used, reply-request messages are stored in shared memory. Normally, they are released when the user-reply functionality management daemon is stopped. If the user-reply functionality management daemon terminates because of a failure without releasing the shared memory, you must use this command to release the shared memory. The following is the procedure for doing this:

1. Execute this command with the `start reuse` option specified
2. Execute this command with the `stop` option specified, and then stop the user-reply functionality's management daemon

Arguments

-h logical-host-name ~<logical host name>((1 to 255 bytes))

When you are operating in a logical host environment, specifies the host name of the logical host where the command will be executed.

It is recommended that the length of the logical host name not exceed 63 bytes. If you specify a name that exceeds 63 bytes, the command might not function correctly.

If the logical host name specification is omitted, any option that is specified next will be treated as the argument.

start [reuse]

Specifies that the user-reply functionality's management daemon is to be started.

If `reuse` is specified, the information in the reply-request messages will be used as is by the user-reply functionality.

When the user-reply functionality is used, reply-request messages are stored in shared memory, and normally they are released when the user-reply functionality's management daemon is stopped. If the user-reply functionality's management daemon terminates because of a failure without releasing the shared memory, the user must release the shared memory by first starting the management daemon with the `reuse` option specified in the `adshmdctl` command, and then terminating the management daemon by specifying the `adshmdctl` command with the `stop` option specified.

stop

Specifies that the user-reply functionality's management daemon is to be stopped.

If there are reply-request messages that have not yet been replied to, the waits for replies will be cancelled.

status

Specifies that the operating status of the user-reply functionality's management daemon is to be returned, using one of the following return codes:

- `0`: The user-reply functionality's management daemon is running
- `1`: The user-reply functionality's management daemon is not running

conftest [environment-file-name]

Specifies an environment file whose parameters are to be checked. The results are output to the standard output.

If *environment-file-name* is omitted, the system environment file is checked.

help

Specifies that a help for the `adshmdctl` command is to be displayed.

Return codes

Return code	Meaning
`0`	Normal termination
Other than `0`	Error termination

Notes

- Start the user-reply functionality's management daemon as a root job.
- Do not change the file system environment while the user-reply functionality's management daemon is running.
- The `adshmdctl` command sets the `LANG` environment variable to `C` and launches the management daemon of the user-reply functionality. For this reason, subsequent messages and JP1 events are output in English.

adshmsvcd command (registers the user-reply functionality management service in a development environment) (Windows only)

Format

```
adshmsvcd [-install [-lhostname logical-host-name]]
```

Description

This command registers the user-reply functionality management service (`adshmsvcd`). The user-reply functionality management service manages shared memory for the user-reply functionality. This command can be executed only in a Windows development environment.

Arguments

-install

Specifies that the user-reply functionality management service is to be registered.

The user-reply functionality management service is registered automatically when you set up JP1/Advanced Shell. However, if the registry information is later deleted, you will need to use this option to re-register it manually.

-lhostname logical-host-name ~<logical host name>((1 to 196 bytes))

When you are operating in a logical host environment, specifies the host name of the logical host where the command is to be executed.

It is recommended that the length of the logical host name not exceed 63 bytes. If you specify a name that exceeds 63 bytes, the command might not function correctly.

Return codes

Return code	Meaning
`0`	Normal termination
Other than `0`	Error termination

Notes

- This command terminates without an error even when it is executed with invalid options or with no options specified. In such a case, the registry information is not updated.

adshmsvce command (registers the user-reply functionality management service in an execution environment) (Windows only)

Format

```
adshmsvce [-install [-lhostname logical-host-name]]
```

Description

This command registers the user-reply functionality management service (`adshmsvce`). The user-reply functionality management service manages shared memory for the user-reply functionality. This command can be executed only in a Windows execution environment.

Arguments

-install

Specifies that the user-reply functionality management service is to be registered.

The user-reply functionality management service is registered automatically when you set up JP1/Advanced Shell. However, if the registry information is later deleted, you will need to use this option to re-register it manually.

-lhostname logical-host-name ~<logical host name>((1 to 196 bytes))

When you are operating in a logical host environment, specifies the host name of the logical host where the command is to be executed.

It is recommended that the length of the logical host name not exceed 63 bytes. If you specify a name that exceeds 63 bytes, the command might not function correctly.

Return codes

Return code	Meaning
`0`	Normal termination
Other than `0`	Error termination

Notes

- This command terminates without an error even when it is executed with invalid options or with no options specified. In such a case, the registry information is not updated.

8.4 UNIX-compatible commands

This section explains how to use each of the UNIX-compatible commands. The following are general notes.

Regular expressions supported in UNIX-compatible commands

Basic regular expressions and extended regular expressions are both supported. Basic regular expressions can be used in the following commands:

- `expr`
- `grep` (when the `-G` option is specified)
- `sed` (when the `-E` option is not specified)

Extended regular expressions can be used in the following commands:

- `awk`
- `egrep`
- `grep` (when the `-E` option is specified)
- `sed` (when the `-E` option is specified)

Both types of regular expression support the use of metacharacters. The following table describes the metacharacters that can be used.

Table 8–5: Differences in the metacharacters that can be used in regular expressions

Metacharacter	Meaning	Basic regular expressions	Extended regular expressions
`*`	Zero or more repetitions	Y	Y
`\+`	One or more repetitions	Y	N
`+`	One or more repetitions	N	Y
`.`	One character	Y	Y
`\?`	The preceding regular expression	Y	N
`?`	The preceding regular expression	N	Y
`^`	Beginning of the line	Y	Y
`$`	End of the line	Y	Y
`|`	Or	Y	N
`|`	Or	N	Y
`[char-list]`	Range specification	Y	Y
`\(regexp\)`	Grouping	Y	N
`(regexp)`	Grouping	N	Y
`\{n, m\}`	Repeats at least *n* times but no more than *m* times	Y	N
`{n, m}`	Repeats at least *n* times but no more than *m* times	N	Y
`\{n\}`	*n* times	Y	N
`{n}`	*n* times	N	Y

Metacharacter	Meaning	Basic regular expressions	Extended regular expressions
`\{n, \}`	*n* or more times	Y	N
`{n, }`	*n* or more times	N	Y

Legend:

Y: Can be used.

N: Cannot be used.

About the command usage examples

- In the usage examples provided in the remainder of this chapter, the UNIX-compatible commands are executed on Windows, with a few exceptions.
- The path to the directory where the commands are installed is assumed to have been set in the `ADSH_OSCMD_DIR` environment variable.

awk command (performs text processing and pattern matching)

Format

```
awk [-F input-field-separator] [-v variable-name=variable-value]... [-f
script-file-path-name|script]
   [[target-path-name...]|[built-in-variable-name=variable-value...]]...
```

Description

This command retrieves lines (referred to hereafter as *records*) in a text file that match a particular pattern and performs specified processing on the retrieved lines.

Arguments

-F input-field-separator

Specifies the value to be used as the input field separator. The specified value becomes the value of the `awk` command's `FS` built-in variable.

-v variable-name=variable-value

Specifies a variable name and its value. The variable name and its value are passed to the script that is specified in a script file or in the argument specified in the `-f` option. Multiple variables can be specified. If you specify the same variable name more than once, the last specification takes effect.

-f script-file-path-name

Specifies the path name of a file (script file) that contains the patterns to be matched in the input files and the processing instructions for the records that match the patterns.

- If `-` is specified as the path name, the standard input is assumed for the input.
- Up to 19 `-f` options can be specified.

script

Specifies a pattern to be matched in the input files and the processing instructions for the records that match the pattern.

target-path-name

Specifies the path name of a file to be processed. Multiple path names can be specified.

If no path name is specified or - is specified as the path name, input is read from the standard input. Note that if only the BEGIN pattern is executed, no records are retrieved from the specified file or from the standard input.

built-in-variable-name=variable-value

Specifies the name of a built-in variable and its value. The variable name and its value are passed to the scripts that are specified in the script file or in the *script* argument specified in the -f option.

- If you specify a name that is not for a listed built-in variable, it is treated the same as the -v option.
- A built-in variable that is specified before all the target path names is enabled for all file processing, except for BEGIN pattern processing, and is also enabled for END pattern processing.
- A built-in variable that is specified after all the target path names is enabled for END pattern processing only.
- A built-in variable that is specified between target path names is enabled for processing of the path names specified after the variable specification and for END pattern processing.

Scripts (patterns and actions)

The following is the descriptive format of a script executed by the awk command:

```
[pattern] [{[action]}]
```

A pattern to be searched for in the input files is defined in *pattern*. For details about the *pattern* specification, see *Types of patterns* below. Processing instructions for records that match the pattern are defined in *action*.

Each successive record from the input file is compared to each specified pattern, and the action specified for a pattern is executed when a match is found for that pattern. A specified action can be performed on all records by not specifying a pattern (omitting *pattern* and specifying *action* only).

You specify for the action control statements and functions that perform desired processing on the records that match the specified pattern. The action operation can include control statements, built-in functions, user-defined functions, variables, and operators. Multiple statements, separated by an end-of-line code or semicolon, are permitted. When the entire {*action*} portion is omitted, including the braces, the matching records are output to the standard output. If you specify the empty braces without *action* ({ }), no processing is performed.

To include a comment, specify a hash mark (#) and then the comment string. Everything from the hash mark to the end of the line is treated as a comment.

Records and fields

A *record* is a unit that is obtained by using the input record separator to split up the input. In awk, an end-of-line code serves as the input record separator. In Windows, an end-of-line code is denoted by [CR] + [LF] or by [LF]. In UNIX, an end-of-line code is denoted by [LF]. Note that in UNIX, if [CR] + [LF] is used for the end-of-line code, the [CR] part will be included in the resulting record.

The input record separator can be changed by setting in the RS built-in variable any single-byte character to serve as the new record separator. If a character string is specified, only the first character in the character string becomes the input record separator.

Records are divided by field separators into units called *fields*. The default field separator is the space. The field separator can be changed by specifying in the -F option or by setting in the FS built-in variable any character string to serve as the new field separator.

The input passed to the specified action consists of the contents of the record currently being read from the input file and the value of each field in the record. The entire contents of the record are stored in field variable `$0`. The first field of the record is stored in field variable `$1`, the second field is stored in field variable `$2`, and so on.

Types of patterns

The following types of patterns can be specified.

- Character strings

  To search for a character string in a field or record, enclose it in slashes (`/`). Character strings can be specified using regular expressions. To search for a forward slash (`/`) by itself, you must use the escape character (`\`), so the specification becomes `/\//`.

  In the following example, `hitachi` is specified as the search string:

  ```
  /hitachi/{
  (action)
  }
  ```

- Relational expressions

  Relational operators (>, >=, <, <=, ==, and `!=`) can be used to perform comparisons on fields. In the following example, the specified action is performed if the second field of a record is `hitachi`:

  ```
  $2 == "hitachi"{
  (action)
  }
  ```

- Combinations of patterns

  Multiple patterns can be combined to express conditions for executing an action. The following table shows the combinations that can be used:

Format	Description
pattern1 `&&` *pattern2*	Using the logical AND operator, execute the action on records that are a match for both *pattern1* and *pattern2*.
pattern1 `||` *pattern2*	Using the logical OR operator, execute the action on records that are a match for either *pattern1* or *pattern2*.
pattern1 `?` *pattern2* `:` *pattern3*	Using the ternary operator, execute the action on records that are a match for both *pattern1* and *pattern2* or are a match for *pattern3*.
`!` *pattern*	Using the NOT operator, execute the action on records that are not a match for *pattern*.
`(`*pattern*`)`	Group multiple conditions into *pattern*.
pattern1`,` *pattern2*	Execute the action on the range of input records beginning with the record that is a match for *pattern1* and concluding with the record that is a match for *pattern2*. Note that if the search reaches the end of the input file without finding a record that matches *pattern2*, the last record in the input file will be the end of the range. However, if multiple input files are specified, the successive input files are searched for a record that matches *pattern2*. This format can be specified up to 50 times.

- `BEGIN`

  This pattern is for an action that is to be performed before processing of the input file begins. For this pattern, *action* cannot be omitted. This pattern cannot be combined with other patterns. If multiple input files are specified, the specified action is executed before input starts from the first input file.

- `END`

This pattern is for an action that is to be performed after the last record in the file has been processed or when the `exit` control statement is entered. For this pattern, *action* cannot be omitted. This pattern cannot be combined with other patterns. If multiple input files are specified, the specified action is not executed until the last record in the last file has been processed.

Control statements

The control statements that can be used are described in the table below. The `if`, `while`, `for`, `do`, `break`, `continue`, and `return` statements are subject to the C language syntax rules. An exception is the `for` statement, which is limited to a single initialization expression and a single increment expression.

Control statement	Syntax	Description
`if` statement	`if (`*conditional-expression*`)` *processing* `[else` *processing*`]`	Branch conditionally.
	`if (`*variable* `in` *array*`)` *processing* `[else` *processing*`]`	Branch based on whether the index specified in *variable* exists in *array*.
`while` statement	`while (`*conditional-expression*`)` *processing*	Repeat as long as the condition is true.
`for` statement	`for (`*initialization-expression*`;` *continuation-conditional-expression*`;` *increment-expression*`)` *processing*	Execute repeatedly.
	`for (`*variable* `in` *array*`)` *processing*	Perform *processing* while setting *variable* to successive indexes of the array. Note that the indexes are retrieved in no particular order.
`do` statement	`do` *processing* `while (`*continuation-conditional-expression*`)`	Repeat as long as the condition at the end remains true.
`break` statement	`break`	Exit immediately from a loop.
`continue` statement	`continue`	Interrupt loop processing and return to the beginning of the next cycle of the loop.
`next` statement	`next`	Stop processing the current input record after this control statement, and start processing the next input record.
`nextfile` statement	`nextfile`	Stop processing the current input file after this control statement, and start processing the next input file.
`return` statement	`return [expr]`	Exit a user-defined function. The value specified in the expression *expr* is returned to the caller. If no value is specified for the *expr* expression, the return value of the user-defined function will be `0`.
`delete` statement	`delete` *array*	Delete an array.
	`delete` *array*[*element*]	Delete an element of the array.
`exit` statement	`exit [expr]`	Stop execution of a script during processing. The value specified in the *expr* expression is returned as the return code of the command. If no value is specified for the *expr* expression, the return code of the command will be `0`. The value specified in the *expr* expression is treated as a signed four-byte numeric value. In Windows, the value specified in the expression *expr* will be the return code of the command. In UNIX, when the value specified in the *expr* expression is outside the range `0` to `255`, the low-order 8 bits of the value will be the return code of the command. If you are executing a job

Control statement	Syntax	Description
`exit` statement	`exit [expr]`	definition script in JP1/Advanced Shell, specify a value in the range `0` to `255`. If you are executing a job definition script in JP1/Advanced Shell in Windows and the value specified in the *expr* expression is outside the range `0` to `255`, the return code that is returned to the caller of the command will be different from the value specified in the *expr* expression. For details about how the return codes of commands are handled in JP1/Advanced Shell, see *5.8.8 Return codes of jobs, job steps, and commands*.

Built-in functions

The following built-in functions can be used.

- Mathematical functions

  The mathematical functions that can be used are described in the following table:

Function name	Description
`atan2(y,x)`	Returns the arctangent of *y*/*x*. The unit is radians. If an argument is missing, `1` is returned and a warning message is output.
`cos(x)`	Returns the cosine of *x*. The unit is radians.
`exp(x)`	Returns the exponential function of *x*. If the result produces an overflow or underflow, `1` is returned and a warning message is output.
`int(x)`	Returns an integer by truncating at the decimal point of *x*.
`log(x)`	Returns the natural logarithm of *x*. If *x* is zero or negative, `1` is returned and a warning message is output.
`rand()`	Returns random number *n* (in the range $0 \leq n < 1$). If you do not use the `srand` function to set a seed value, the same series of values will be generated each time `awk` is run.
`sin(x)`	Returns the sine of *x*. The unit is radians.
`sqrt(x)`	Returns the square root of *x*. If *x* is negative, `1` is returned and a warning message is output.
`srand([expr])`	Sets the *expr* expression as the seed value for the `rand` function and then returns the seed value that has been set. If *expr* is omitted, the seed value is set based on the time.

- Character string functions

  The character string functions that can be used are described in the table below. Multibyte characters are treated as single characters.

Function name	Description
`gsub(r,t[,s])`	Replaces all matches for regular expression *r* in character string *s* with *t*. If *s* is omitted, the assumed replacement target is `$0` (the field variable that stores the entire record). If `&` is specified in *t*, `&` is replaced by the matched character string. The number of replacements is returned as the return value.
`index(s,t)`	Returns the position of character string *t* in character string *s*. If character string *t* is not found, `0` is returned.
`length[([s])]`	Returns the number of characters in character string *s*. If *s* is not specified, the number of characters in `$0` (the field variable that contains the entire record) is returned.
`match(s,r)`	Returns the position at which regular expression *r* occurs in character string *s*. If regular expression *r* is not found, `0` is returned. In addition, the `RSTART` built-in variable is set to the position at which regular expression *r* matches the character string, or is set to `0` if there is no match. The `RLENGTH` built-

Function name	Description
match(s,r)	in variable is set to the length of the character string that matches regular expression *r*, or to -1 if there is no match.
sprintf(*format*, *expression*[, ...])	Returns the character string in which *expression* is formatted according to *format*. For details about *format*, see the description under *Output format*.
split(s, array[,fs])	Splits character string *s* into fields using field separator *fs* and stores the fields in array *array*. The number of elements in the array is returned as the return code. The values of the fields that were split are stored in array *array* in the order *array*[1], *array*[2], ..., *array*[*return-value*]. If field separator *fs* is not specified, the value of the FS built-in variable is used as the field separator. The *fs* field separator can be specified as a character string or regular expression. If you specify "" for the *fs* field separator to indicate no characters, *s* is split into single characters.
sub(r, t[,s])	Replaces the first match for regular expression *r* in character string *s* with *t*. If *s* is omitted, the assumed replacement target is $0 (the field variable that contains the entire record). If & is specified in *t*, & is replaced by the matched character string. The return code is 1 if regular expression *r* is found and 0 if regular expression *r* is not found.
substr(s,m[,n])	Returns a substring of up to *n* characters, beginning with the *m*th character of character string *s*. If *n* is omitted, the entire character string beginning with position *m* is returned.
tolower(str)	Returns a character string in which all uppercase characters in character string *str* have been changed to lowercase.
toupper(str)	Returns a character string in which all lowercase characters in character string *str* have been changed to uppercase.

- Bit manipulation functions

  The bit manipulation functions that can be used are described in the table below. Both *x* and *y* are treated as signed four-byte numeric values.

Function name	Description
compl(x)	Returns the one's complement of integer *x*.
and(x,y)	Returns the bitwise logical *AND* of integer *x* and integer *y*.
or(x,y)	Returns the bitwise logical *OR* of integer *x* and integer *y*.
xor(x,y)	Returns the bitwise logical *XOR* of integer *x* and integer *y*.
lshift(x,n)	Returns the value of integer *x* shifted *n* bits to the left. Because an arithmetic bit shift is performed, the sign bit is also shifted.
rshift(x,n)	Returns the value of integer *x* shifted *n* bits to the right. Because an arithmetic bit shift is performed, the sign bit is reintroduced after the right shift, preserving the sign.

- Input/output functions

  The following input/output functions can be used.

  getline [*variable-name*]

  > Reads the next record from the current input file. If *variable-name* is specified, the record is read into the variable specified in *variable-name* and is set as the value of the NR and FNR built-in variables. If *variable-name* is omitted, the record is read into field variable $0 and set as the value of the NF, NR, and FNR built-in variables. The function returns 1 if the record is successfully read, 0 if the end of the file was reached, or -1 if an error occurred.

  getline [*variable-name*] < *path-name*

  > Reads the next record from the file specified in *path-name*. The path name must be enclosed in double quotation marks ("). You can also specify in place of the path name the name of a variable assigned to the path name. If - is specified for *path-name*, input is read from the standard input.

The following is an example:

```
getline line < "file001.txt"
```

If *variable-name* is specified, the record is read into the variable specified in *variable-name*. If *variable-name* is omitted, the record is read into field variable `$0` and set as the value of the `NF` built-in variable.

The specified file is opened when the first record is read by the `getline` function and remains open until the `awk` command terminates. For this reason, you must execute the `close` function if you want to restart input from first record of the same file.

command-name `| getline [`*variable-name*`]`

Reads a record being output to a pipe by the program specified in *command-name*. The command name must be expressed in command line format, with the name of the program to be executed and the values of its arguments enclosed in double quotation marks (`"`). You can also specify in place of the command name the name of a variable that has been assigned to the command name.

If *variable-name* is specified, the record is read into the variable specified in *variable-name*. If *variable-name* is omitted, the record is read into field variable `$0` and set as the value of the `NF` built-in variable. The following are examples.

Using the output from *command-name* to the pipe as the input, read one record and assign it to field variable `$0`:

```
"cat -n file01.txt" | getline
```

Assign *command-name* to a variable, and then execute the program by connecting the variable to the `getline` function:

```
rtxt = "cat -n file01.txt"
rtxt | getline
```

Creation of a pipe to receive the output from the specified program and execution of the program are performed when you execute *command-name* `| getline [`*variable-name*`]`. If you execute the same *command-name* multiple times, pipe creation and program execution are performed only the first time the specified *command-name* is executed. The pipe that is created exists only until the `awk` command terminates. For this reason, if you want to re-run the program specified in *command-name*, you must execute the `close` function. This is illustrated in the following example:

```
"cat -n file01.txt" | getline rec -->1.
"cat -n file01.txt" | getline rec -->2.
close("cat -n file01.txt") -->3.
"cat -n file01.txt" | getline rec -->4.
```

1. The pipe is created and the `cat` command is executed. The contents of the first record that is output from the `cat` command to the pipe are set in variable *rec*.

2. The contents of the second record that is output from the `cat` command to the pipe are set in variable *rec*.

3. The pipe is closed.

4. The pipe is created and the `cat` command is executed. The contents of the first record that is output from the `cat` command to the pipe are set in variable *rec*.

Note that even if this *command-name* is identical to the *command-name* specified in a `print` or `printf` function, they are considered to be different commands.

`print [`*expression*`[, ... ]]`

Prints *expression* to the standard output. If *expression* is omitted, the current input record is printed to the standard output. If you specify multiple expressions delimited by the comma (`,`), the expressions will be separated by the value of the `OFS` built-in variable. The value of the `ORS` built-in variable is output at the end of the output record.

`print [`*expression*`[, ... ]] >` *path-name*

Outputs *expression* to the file specified in *path-name*. *path-name* must be enclosed in double quotation marks (`"`). You can also specify in place of the path name the name of a variable assigned to the path name. If *expression* is omitted, the current input record is output to the file specified in *path-name*. If you specify multiple expressions delimited by the comma (`,`), the expressions will be separated by the value of the `OFS` built-in variable. The value of the `ORS` built-in variable is output at the end of the output record. To append the output to an existing file, specify >>*path-name*.

The specified file will be opened when you first produce output using the `print` function and will remain open until the `awk` command terminates. For this reason, if you want to output from the beginning of the same file, you must execute the `close` function.

`print [`*expression*`[, ... ]] |` *command-name*

Outputs *expression* to a pipe and passes it to the program specified in *command-name*.

The command name must be expressed in command line format, with the name of the program to be executed and the values of its arguments enclosed in double quotation marks (`"`). You can also specify in place of the command name the name of a variable assigned to the command name.

If *expression* is omitted, the current input record is output to the pipe. If you specify multiple expressions delimited by the comma (`,`), the expressions will be separated by the value of the `OFS` built-in variable. The value of the `ORS` built-in variable is output at the end of the output record.

The creation of a pipe to receive the output from the specified program and execution of the program are performed when you execute `print [`*expression*`[, ... ]] |` *command-name*. If you execute the same command multiple times, pipe creation and program execution are performed only the first time the specified *command-name* is executed. The pipe that is created exists only until the `awk` command terminates. For this reason, if you want to re-run the program specified in *command-name*, you must execute the `close` function. Note that even if this *command-name* is identical to the *command-name* specified in a `getline` function, they are considered to be different commands.

`printf` *format*`[,` *expression*`[, ... ]]`

Prints to the standard output according to the format in *format*. For details about *format*, see the description under *Output format*. For details about the other parts of the command, see `print [`*expression*`[, ... ]]`.

In Windows, when you use `\n` to represent a new line, it is output as `[CR] + [LF]`.

`printf` *format*`[,` *expression*`[, ... ]]` >*path-name*

Prints to a file according to the format in *format*. For details about *format*, see the description under *Output format*. For details about the other parts of the command, see `print [`*expression*`[, ... ]]` >*path-name*.

`printf` *format*`[,` *expression*`[, ... ]] |` *command-name*

Outputs to a pipe according to the format in *format*. For details about *format*, see the description under *Output format*. For details about the other parts of the command, see `print [`*expression*`[, ... ]] |` *command-name*. If this *command-name* is identical to the *command-name* specified in a `print` function, their output is considered to be from the same program.

`close (`*path-name*`|`*command-name*`)`

Closes a file that was used by the `getline`, `print`, or `printf` function, or a pipe that was created when the `getline`, `print`, or `printf` function was executed.

When the `close` function is successful, `0` is returned. When the `close` function fails, the return value of the OS's `close` function is returned in the case of a file, or the return value of the `pclose` function is returned in the case of a pipe.

The argument specifies a path name or command that was specified in a `getline`, `print`, or `printf` function. *path-name* must be enclosed in double quotation marks (`"`). *command-name* must be expressed in command line format, with the name of the program to be executed and the values of its arguments enclosed in double quotation marks ("). Alternatively, you can specify a variable that contains the path name or command that was specified in the `getline`, `print`, or `printf` function. Note that the *path-name* or *command-name* argument to the `close` function must be identical, including the number of characters, to the path name or command used in the `getline`, `print`, or `printf` function. If *command-name* or *path-name* specifies a different character string, it will be considered a different path name or command. Note that in addition to closing *command-name*, the `close` function also closes any pipes to *command-name* from the `getline`, `print`, or `printf` functions. The following is an example:

```
print "hitachi" | "cat -n"
close("cat -n")
```

`fflush([`*path-name*`|`*command-name*`])`

Flushes a file used by the `getline`, `print`, or `printf` function, or a pipe created when a `getline`, `print`, or `printf` function is executed. When the flush is successful, `0` is returned. When the flush fails, the return value of the OS's `fflush` function is returned.

The argument specifies a path name or command that was specified in a `getline`, `print`, or `printf` function. *path-name* must be enclosed in double quotation marks ("). *command-name* must be expressed in command line format, with the name of the program to be executed and the values of its arguments enclosed in double quotation marks ("). Alternatively, you can specify a variable that contains the path name or command that was specified in the `getline`, `print`, or `printf` function. Note that the *path-name* or *command-name* argument to the `fflush` function must be identical, including the number of characters, to the path name or command used in the `getline`, `print`, or `printf` function. If *command-name* or *path-name* specifies a different character string, it will be considered a different path name or command.

If no argument is specified, all files and pipes are flushed.

- General functions

  The general function that can be used is described in the following table:

Function name	Description
`system(`*command-name*`)`	Executes the program specified in *command-name* and returns the status of the executed program. *command-name* must be expressed in command line format, with the name of the program to be executed and the values of its arguments enclosed in double quotation marks ("). You can also specify in place of the command name the name of a variable assigned to the command name. In UNIX, the status of the executed program will be the value of the low-order 8 bits of the return code that is returned by the program.

User-defined functions

In addition to the built-in functions, you can also define your own functions. The syntax of a user-defined function is as follows:

```
function | func name([param[, ...]]) { statements }
```

The function name *name* must be specified in alphanumeric characters and the underscore (_), and the first character must be non-numeric.

You can specify in *param* arguments to the function using the names of user-defined variables or arrays. An argument is passed to the function by its value in the case of a user-defined variable or as a reference in the case of an array.

A parsing error does not occur if the number of arguments specified in the function definition differs from the number of arguments specified when the function is called. However, if the number of arguments specified when the function is called is greater than the number of arguments specified in the function definition, a warning message is output. The arguments specified in the function definition are considered local variables, but if the number of arguments specified when the function is called is greater than the number of arguments specified in the function definition, the extra arguments at the time of the function call are considered global variables.

A maximum of 50 arguments can be specified in a function definition. Similarly, a maximum of 50 arguments can be specified when the function is called. A check is performed at the time the function is called to confirm that the number of arguments does not exceed 50.

Variables

The types of variables used in scripts include user-defined variables, field variables, built-in variables, and arrays. User-defined variables and arrays are generated the first time they are used in a script. Note that the initial value stored in an uninitialized variable (one that has not been used in an arithmetic or assignment expression) is 0 in the case of a numeric value and `NULL` in the case of a character string.

The type of the value of a variable changes to numeric or character string depending on the situation in which it is used. However, non-numeric character strings have a numeric value of `0`. For example, in the following example, the two print functions both produce `7` as the output:

```
x = "3" + "4"
y = 3 + 4
print x
print y
```

The following describes each type of variable.

- User-defined variable

  A variable name must consist of only alphanumeric characters and the underscore (_). The first character must be non-numeric.
- Array

  An array name is represented with an index enclosed in square brackets (`[]`) after the variable name. An index can be a character string enclosed in double quotation marks (`"`) instead of a number. Indexes written as a comma-separated list represent a multidimensional array.

  Multidimensional arrays are implemented in `awk` by connecting the index values at each dimension with the value of the `SUBSEP` built-in variable, in order to treat them as a single character string index. In other words, a multidimensional array is handled internally as a one-dimensional array. In the example below, the value of the `SUBSEP` built-in variable is set to `#`, and then an array is created as shown. The two print functions both output the same value, which in this case is `Hitachi`.

  ```
  SUBSEP = "#"
  arry["a", "b", "c"] = "Hitachi"
  print arry["a", "b", "c"]
  print arry["a#b#c"]
  ```
- Field variables

  The field variables for referring to the contents of input records are described in the following table:

Variable	Description
`$0`	Set to the entire contents of the record currently being read from the input file.
`$1,$2,...`	Set sequentially to the field contents of the record that has just been read. The record is split into fields by the value of the `FS` built-in variable, and then variable `$1` is set to the record's first field, variable `$2` is set to the record's second field, and so on.

Variable	Description
$*variable-name*	By assigning field numbers to variables, you can refer to fields in the same way as if you had written the field numbers directly as `$0`, `$1`, and so on.# When you do this in a function or control statement, the variable must be set to the field number before it is used.
$(*expression*)	By setting *expression* to an expression that evaluates to a field number (such as *variable-name*+1), you can refer to fields in the same way as if you had written the field numbers directly as `$0`, `$1`, and so on.

#

For example, in the following case, `print $a` and `print $1` produce the same output:

```
    a = 1
    print $a
    print $1
```

- Built-in variables

  The built-in variables are described in the following table:

Variable	Description
`ARGC`	Stores the number of command line arguments. This number does not include option values and script specification values. The value of `ARGC` can be changed by a script or by the `-v` option. If you set `ARGC` to `0` with the `-v` option, target path names specified in arguments will be ignored.
`ARGV`	Stores an array of command line arguments. The array elements can be modified by a script. For example, if you set to `NULL` an element storing one of the target path names specified in the arguments, that input file will be ignored and no records will be read from it. This will overwrite values specified with the `-v` option.
`CONVFMT`#	Stores the conversion format to use when converting numeric values. The default is `%.6g`. If the numeric value has a fractional part, it is also included.
`ENVIRON`	Stores an array of run-time environment variables. The indexes are the names of environment variables. The name of an environment variable must be enclosed in double quotation marks ("). You can also specify in place of the name of an environment variable the name of a variable that has been assigned the name of the environment variable.
`FILENAME`	Stores the name of the current input file. The file name is set to `-` when the input is from the standard input.
`FNR`	Stores the number of input records that have been read from the current input file. This value is updated each time a record is read from the file specified in the target path name. It is also updated when a record is read by `getline [`*variable-name*`]`. When more than one target path name is specified, it is initialized to `0` when input starts to be read from the next input file.
`FS`	Stores the field separator. The default is a one-byte space. `FS` can be set to a regular expression. If you wish to change the field separator, specify a character string of no more than 99 bytes. If the value of the field separator is a one-byte space, it will split fields at both spaces and tabs (`\t`). If no value is set for the field separator, each individual character will become a field. If you specify more than one character, the backslash (`\`) will be regarded as the regular expression escape character. If the `-F` option is also specified, this variable takes precedence.
`NF`	Stores the number of fields in the current input record. `NF` is set each time an input record is read from the file specified in the target path name. It is also set when an input record is read into the `$0` field variable by the `getline` function. You can reference the value of the last field of the current input record by specifying `$NF`.
`NR`	Stores the number of input records detected so far. The value is updated each time a record is read from a file specified by the target path name. The value is also updated when a record is read by `getline [`*variable-name*`]`. If more than one target path name is specified, the number of records includes the records from all files from which input has been performed.

Variable	Description
OFMT[#]	Stores the output format for numeric values. The default is %.6g. If the numeric value has a fractional part, it is also included.
OFS	Stores the output field separator. The default is a one-byte space.
ORS	Stores the output record separator. The default is the newline character (\n). In Windows, the newline character (\n) is output as [CR] + [LF].
RLENGTH	Stores the length of a substring matched by the match function. The value is set to -1 if no match was found.
RS	Stores the input record separator. The default is the end-of-line character. If you change its value, use single-byte characters only. If you specify a character string or a multibyte character, the initial byte only will be used. When RS is set to the newline character (\n), this will match either [CR] + [LF] or [LF] in an input file in Windows, and it will match [LF] in an input file in UNIX. If the value of RS is set to something other than the newline character, the end-of-line character included in the input records will be [LF] in the case of Windows. On a UNIX system, if the input file uses [CR] + [LF] as the end-of-line code, the [CR] part only will be included with the input records.
RSTART	Stores the starting position of a substring matched by the match function. The value is set to 0 if no match was found.
SUBSEP	Stores the separator used in multidimensional arrays. The default is 0x1C.

[#] The conversion specifiers f, e, g, E, and G are supported. Do not use any other conversion specifiers.

Operators

The operators that can be used are listed and described in the table below, in order of lowest to highest priority. For an expression with operators at the same priority level, the operators are listed from left to right in order of highest to lowest priority.

Operator	Description
=, +=, -=, *=, /=, %=, ^=, **=	Assignment operators
?:	Ternary operator
||	Logical *OR*
&&	Logical *AND*
~, !~	Operators for match (~) or fail to match (!~) a regular expression
<, <=, >, >=, !=, ==	Relational operators
space	Concatenation of character strings
+, -	Addition and subtraction
*, /, %	Multiplication, division, and modulus
+, -, !	Unary and logical negation
^, **	Exponentiation
++, --	Increment and decrement operators

Output format

The following table lists and describes the conversion specifiers that follow % to indicate conversion specifications in the printf and sprintf functions:

Character	Description
c	Output as a single-byte character.
s	Output as a character string.
d	Output as a signed decimal integer.
i	
o	Output as an unsigned octal integer.
x	Output as an unsigned hexadecimal integer. The values 10 through 15 use abcdef.
X	Output as an unsigned hexadecimal integer. The values 10 through 15 use ABCDEF.
u	Output as an unsigned decimal integer.
f	Output as a floating point number. It is converted to the format [-]dddd.dddd.
e	Output as a floating point number. It is converted to the format [-]d.dddde[+-]dd[d].
g	Output in the signed format of conversion specifier e or f, depending on which is able to represent the specified value and precision in the shortest way. Trailing zeros are not output.
E	Output as a floating point number. It is converted to the format [-]d.ddddE[+-]dd[d].
G	Output in the signed format of conversion specifier E or f, depending on which is able to represent the specified value and precision in the shortest way. Trailing zeros are not output.
%	Output as the % character.

Escape characters

You can use escape characters as follows:

- In the input field separator in the -F option
- As a variable value in the -v option
- As a built-in variable value specified in an argument
- Within character strings that are enclosed in double quotation marks (") in an assignment to a pattern or variable

The following table shows the escape characters that can be used:

Escape character	Meaning
\a	Alert character (bell)
\b	Backspace character
\f	Formfeed character (page break)
\n	Linefeed character
\r	Carriage return character
\t	Tab character
\v	Vertical tab character
\d, \dd, \ddd	Character represented by one, two, or three octal digits.[#1] You cannot specify a numeric value that denotes 0.
\xhex	Character represented by a hexadecimal value (0 to 9, a to f, A to F).[#1, #2] You cannot specify a numeric value that denotes 0.

Escape character	Meaning
`\c`	Any literal character (for example, `\"` for `"`)
`\\`	A single backslash character

#1

If you specify a pattern or regular expression enclosed in forward slashes (`/`), there are values that cannot be specified depending on the character encoding at the time of execution. The hexadecimal representations of the permissible values for each character encoding are shown below. If you specify a value that is outside these values, termination with an error will occur.

Character encoding	Permitted values (in hexadecimal)
Shift JIS	`0x01` to `0x80`, `0xA0` to `0xDF`, `0xFD` to `0xFF`
UTF-8	`0x01` to `0xBF`, `0xFE` to `0xFF`
EUC	`0x01` to `0x8D`, `0x90` to `0xA0`, `0xFF`
C	`0x01` to `0xFF`

#2

If `\x`*hex* is specified in a character string enclosed in double quotation marks (`"`), the hexadecimal digits are assumed to extend from `\x` to the first non-hexadecimal character. If the hexadecimal representation exceeds 98 characters, only the first 98 characters will be used. When the hexadecimal representation exceeds two characters, the results of converting the hexadecimal values from their hexadecimal representation are not guaranteed.

A backslash (`\`) specified in variable values in the `-v` option and in built-in variable values in arguments is treated as an escape character (except when it is enclosed in single quotation marks (`'`)). Specify path names carefully. The following shows examples.

Example 1: The correct path name, `c:\a\b\c`, cannot be passed to the `awk` command's scripts.

In this example, `\` is deleted because it is processed as an escape character. As a result, `c:\a\b\c` is set in `VAR001`.

After that, `\` is processed as an escape character again when the value is set in the variable in the `awk` command. As a result, `c:abc` is set in `VAR001`:

```
CCC01="c:\\a\\b\\c"
awk -v VAR001="${CCC01}"  -f prog01.awk
```

Similarly, the following two examples also cannot pass the correct path name:

```
awk -v VAR001=c:\\a\\b\\c  -f prog01.awk
awk -v VAR001='c:\a\b\c'  -f prog01.awk
```

Example 2: The correct path name, `c:\a\b\c`, is passed to the `awk` command's scripts.

In this example, `c:\\a\\b\\c` is stored in both `CCC01` and `CCC02`.

After that, when the value is stored in `VAR001` in the `awk` command, it becomes `c:\a\b\c`, thereby processing correctly:

```
CCC01='c:\\a\\b\\c'
CCC02="c:\\\\a\\\\b\\\\c"
awk -v VAR001="${CCC01}"  -f prog01.awk
awk -v VAR001="${CCC02}"  -f prog01.awk
```

Similarly, the following two examples can also pass the correct path name:

```
awk -v VAR001=c:\\\\a\\\\b\\\\c  -f prog01.awk
awk -v VAR001='c:\\a\\b\\c'  -f prog01.awk
```

Special file names

Special file names can be used to represent the input source and output destination when you use the getline function to read from the standard input or the print or printf function to output to the standard output or standard error output. The table below lists the special file names that are available. Note that attempting to apply the close function to a special file name will have no effect.

Special file name	Meaning
/dev/stdin	Standard input
/dev/stdout	Standard output
/dev/stderr	Standard error output

Return codes

Return code	Meaning
0	Normal termination
1 or greater	Error termination
Value specified in the exit statement	Command return code specified in the exit control statement

Notes

- The awk command treats numeric values as double-precision floating point numbers (8 bytes). When you output or convert a numeric value by specifying the conversion specifier d, i, o, x, X, or u in the output format of the printf or sprintf function, it is rounded to a four-byte signed integer. For this reason, an error occurs when you output or convert a numeric value that is outside the range of a four-byte signed integer by specifying the conversion specifier d, i, o, x, X, or u. The particular error depends on the OS.
- The maximum number of files that can be opened at the same time by the getline, print, or printf function is 256. Note that the number of files that can be open at the same time includes pipes generated by the commands that invoke these functions. In UNIX, the maximum number will be less than 256, depending on OS settings, such as the maximum number of files that can be open at the same time on the system as a whole, or restrictions imposed by ulimit on the number of open file descriptors for those processes.
- Input from binary files and output of binary data are not guaranteed to work.
- Execution of external programs, for example by the system function, is implemented in terms of arguments to other programs, as described below. For this reason, the execution specifications for the external program, such as the maximum length of path names, depend on the specifications of those executing programs.
    - Windows

      External programs are executed as arguments to the command processor specified in the COMSPEC environment variable, the default value for which is cmd.exe. The command processor that is used is determined by the COMSPEC and PATH environment variables.
    - UNIX

      External programs are executed as arguments to the shell. Different executing programs might be launched depending on OS specifications.
- In Windows, at the start of input or output from a command that is specified in a getline, print, or printf function, an application error might occur in the execution of the specified command due to insufficient desktop heap space. For this reason, either use the close function close the command when command input/output is no longer required or adjust the value of the desktop heap.

- In Windows, if you specify a file or directory name that contains wildcards in an argument to a command that executes the `system` function, the wildcards are not expanded.
- In Windows, if you specify file and directory names that contain wildcards as arguments to a command that is connected by a pipe to a `getline`, `print`, or `printf` function, the wildcards are not expanded.
- In Windows, if you specify the path name of a linked file in a place where you specify an external function, such as in the `system` function, the program that it has been linked to will launch. Take note of this when executing batch jobs.
- In Windows, when you specify a path name in an input/output function or a general function, observe the following constraints concerning the backslash (`\`) character:
    - The directory separator character `\` must be specified as `\\`.
    - If you specify a path name that includes spaces in the place where you specify an external function, such as in the `system` function, the entire path must be enclosed in `\"`.
    - The `\` character is treated as representing the escape character when it is specified as the value of a built-in variable specified in the input field separator in the `-F` option, or as a variable value specified in an argument or in the `-v` option.
- In Windows, file descriptors are closed without being inherited by processes that were generated by the `system` function or similar means. For example, an error results if you attempt to perform input or output on a file descriptor opened by the parent process without reopening it. This does not apply to the standard input, standard output, and standard error output, which do not need to be reopened.
- In Windows, if the command name specified in the `system` function contains no path, the command found in the path search order of a command processor such as the command prompt is executed.
- When you use the `exit` statement to return a return code for a command, depending on the value specified in the `exit` statement, the return code might sometimes be different from the return code that is returned to the caller of the command. See the description of the `exit` statement for details about specifying a command return code in the `exit` statement. Note that in the `awk` command, the value specified in the `exit` statement is rounded to a four-byte signed integer. For this reason, an error occurs if you specify a numeric value that is outside the range of a four-byte signed integer. The specific error is OS-dependent.

Usage examples

- In an argument to the command, specify that the action is to convert from lowercase to uppercase the records that match the search pattern. The input file is `file01.txt`.

  Contents of `file01.txt`:

  ```
  hitachi group01 Tokyo
  HITACHI group02 Yokohama
  hitachi group03 Fukuoka
  HITACHI group04 Hokkaido
  ```

  The results of executing the command are as follows:

  ```
  C:\TEMP>%ADSH_OSCMD_DIR%\awk "/hitachi/{print toupper($0)}" file01.txt
  HITACHI GROUP01 TOKYO
  HITACHI GROUP03 FUKUOKA
  ```

- Output the records where the second field matches the pattern in a regular expression. The input file is `file01a.txt`.

  Contents of `file01a.txt`:

```
hitachi group01 Tokyo
hitachi group02 Yokohama
hitachi grp0000 Fukuoka
hitachi group04 Hokkaido
```

The results of executing the command are as follows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\awk "$2 ~ /group/" file01a.txt
hitachi group01 Tokyo
hitachi group02 Yokohama
hitachi group04 Hokkaido
```

- Output the contents of the third field, using the `-F` option to specify `#` as the input field separator. The input file is `file02.txt`.

Contents of `file02.txt`:

```
hitachi#group01#Nagoya
HITACHI#group02#Hiroshima
hitachi#group03#Ooita
```

The results of executing the command are as follows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\awk -F"#" "{print $3}" file02.txt
Nagoya
Hiroshima
Ooita
```

- Specify the variable `padstr` and its value in an argument, in order to pass them to the action, which is specified in the `prog01.awk` script file. The input file is `file03.txt`.

Contents of `prog01.awk`:

```
# program name : prog01
{
count++
print padstr " " $0#
}
END{
    print "total record : " count
}
```

#

Outputs the value of the variable specified in the `-v` option, a space, and the contents of the input record.

Contents of `file03.txt`:

```
group01 Tokyo
group02 Yokohama
group03 Fukuoka
```

The results of executing the command are as follows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\awk -v padstr="hitachi" -f prog01.awk file03.txt
hitachi group01 Tokyo
hitachi group02 Yokohama
hitachi group03 Fukuoka
total record : 3
```

- In a command argument, specify # as the value of the FS built-in variable for input file file02.txt. The input files are file01.txt and file02.txt.

  Contents of file01.txt:

  ```
  hitachi group01 Tokyo
  HITACHI group02 Yokohama
  hitachi group03 Fukuoka
  HITACHI group04 Hokkaido
  ```

  Contents of file02.txt:

  ```
  hitachi#group01#Nagoya
  HITACHI#group02#Hiroshima
  hitachi#group03#Ooita
  ```

  The results of executing the command are as follows:

  ```
  C:\TEMP>%ADSH_OSCMD_DIR%\awk "{print $3}" file01.txt FS="#" file02.txt
  Tokyo
  Yokohama
  Fukuoka
  Hokkaido
  Nagoya
  Hiroshima
  Ooita
  ```

- Retrieve records beginning with the record that contains group03 through the record that contains group06, and output them to the file file06.txt. The script file is prog02.awk. The input files are file04.txt and file05.txt.

  Contents of prog02.awk:

  ```
  BEGIN{
      print "Extract record : group03 - group06" > "file06.txt"
  }
  /group03/, /group06/{#
      count++;
      print >> "file06.txt";

  }
  END{
      printf "total record : %03d\n",  count >> "file06.txt"
  }
  ```

  #

  The processing target begins with the record that matches group03 and extends through the record that matches group06.

  Contents of file04.txt:

  ```
  hitachi group01 Tokyo
  HITACHI group02 Yokohama
  hitachi group03 Fukuoka
  HITACHI group04 Hokkaido
  ```

  Contents of file05.txt:

```
hitachi group05 Nagoya
HITACHI group06 Hiroshima
hitachi group07 Ooita
```

The results of executing the command are as follows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\awk -f prog02.awk file04.txt file05.txt
C:\TEMP>%ADSH_OSCMD_DIR%\cat file06.txt
Extract record : group03 - group06
hitachi group03 Fukuoka
HITACHI group04 Hokkaido
hitachi group05 Nagoya
HITACHI group06 Hiroshima
total record : 004
```

- Output beginning with the first record in file `file04.txt` through the record that contains `group02`. In addition, output all records from the file `file05.txt`. The script file is `prog03.awk`. The input files are the previous `file04.txt` and `file05.txt`.

  Contents of `prog03.awk`:

```
{
    count++; print
    if ($2 == "group02") {
        nextfile#
    }
}
END{
    printf("total record : %03d\n",  count)
}
```

#

If the second field is `group02`, begin processing the next input file.

The results of executing the command are as follows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\awk -f prog03.awk file04.txt file05.txt
hitachi group01 Tokyo
HITACHI group02 Yokohama
hitachi group05 Nagoya
HITACHI group06 Hiroshima
hitachi group07 Ooita
total record : 005
```

- If the second field of the record is `group02`, terminate execution of the `awk` command. In addition, return as the return code of the command the number of records that were read before the command terminated. The script file is `prog04.awk`. The input files are `file04.txt` and `file07.txt`.

  Contents of `prog04.awk`:

```
{
    print
if ($2 == "group02") {
        exit(NR) #
    }
}
END{
    printf("total record : %03d\n",  NR)
}
```

#

Terminates the command and sets as the return code the value of the `NR` built-in variable, which contains the number of records read so far.

Contents of `file04.txt`:

```
hitachi group01 Tokyo
HITACHI group02 Yokohama
hitachi group03 Fukuoka
HITACHI group04 Hokkaido
```

Contents of `file07.txt`:

```
hitachi group05 Nagoya
HITACHI group06 Hiroshima
hitachi group07 Ooita
hitachi group03 Okinawa
```

The results of executing the command are as follows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\awk -f prog04.awk file04.txt file07.txt
hitachi group01 Tokyo
HITACHI group02 Yokohama
total record : 002
```

- Use the `sub` function to change the first occurrence of a specific character string in each record. The script file is `prog05.awk`. The input file is `file08.txt`.

Contents of `prog05.awk`:

```
{
    if (sub(/Hitachi/,  "& Corporation")#) {
        print
    } else {
           print "Unconverted record : " $0
    }
}
```

#

Replaces `Hitachi` with `Hitachi Corporation` in the records.

Contents of `file08.txt`:

```
Hitachi Yokohama Office Hitachi Group
Hitachi Tokyo Office Hitachi Group
Tanaka Okinawa Office Tanaka Group
Hitachi Fukuoka Office Hitachi Group
```

The results of executing the command are as follows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\awk -f prog05.awk file08.txt
Hitachi Corporation Yokohama Office Hitachi Group
Hitachi Corporation Tokyo Office Hitachi Group
Unconverted record : Tanaka Okinawa Office Tanaka Group
Hitachi Corporation Fukuoka Office Hitachi Group
```

- Use the `gsub` function to change every occurrence of a specific character string in each record. The script file is `prog06.awk`. The input file is `file09.txt`.

  Contents of `prog06.awk`:

```
{
    if (gsub(/Hitachi/,  "& Corporation")#) {
        print
    } else {
           print "Unconverted record : " $0
    }
}
```

  #

  Replaces every instance of `Hitachi` with `Hitachi Corporation` in the records.

  Contents of `file09.txt`:

```
Hitachi Yokohama Office Hitachi Group
Hitachi Tokyo Office Hitachi Group
Tanaka Okinawa Office Tanaka Group
Hitachi Fukuoka Office Hitachi Group
```

  The results of executing the command are as follows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\awk -f prog06.awk file09.txt
Hitachi Corporation Yokohama Office Hitachi Corporation Group
Hitachi Corporation Tokyo Office Hitachi Corporation Group
Unconverted record : Tanaka Okinawa Office Tanaka Group
Hitachi Corporation Fukuoka Office Hitachi Corporation Group
```

- Find the position of a specific character string using the `index` function. The script file is `prog07.awk`.

  Contents of `prog07.awk`:

```
BEGIN{
    str = "HI:hitachi"
    print "Column = " index(str,  "hitachi")
}
```

  The results of executing the command are as follows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\awk -f prog07.awk
Column = 4
```

- Find the length of a character string using the `length` function. The script file is `prog08.awk`.

  Contents of `prog08.awk`:

```
BEGIN{
    str = "HI:hitachi"
    print "Length = " length(str)
}
```

  The results of executing the command are as follows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\awk -f prog08.awk
Length = 10
```

- Use the `match` function to find the position and length of a specific character string. The script file is `prog09.awk`.

Contents of `prog09.awk`:

```
BEGIN{
    str = "hitachi:MOUSE"
    print "Column  = " match(str,  /U.E/)
    print "RSTART  = " RSTART
    print "RLENGTH = " RLENGTH
}
```

The results of executing the command are as follows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\awk -f prog09.awk
Column  = 11
RSTART  = 11
RLENGTH = 3
```

- Split a character string using a specific character as the delimiter, and store the results in an array. The script file is `prog10.awk`.

  Contents of `prog10.awk`:

```
BEGIN{
    str = "Hitachi#Yokohama Office#Hitachi Group"
    num = split(str,  arry, "#")
    for (i = 1; i <= num; i++ ) {
        print arry[i]
    }
}
```

  The results of executing the command are as follows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\awk -f prog10.awk
Hitachi
Yokohama Office
Hitachi Group
```

- Find the substring at a specific position. The script file is `prog11.awk`.

  Contents of `prog11.awk`:

```
BEGIN{
    str = "hitachi:MOUSE"
    rtnstr = substr(str,  11, 2)
    print "SUBSTR = " rtnstr
}
```

  The results of executing the command are as follows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\awk -f prog11.awk
SUBSTR = US
```

- Read records from an input file that is not specified as an argument. The script file is `prog12.awk`. The input file is `file10.txt`.

  Contents of `prog12.awk`:

```
BEGIN{
    while ((getline rec < "file10.txt"#) > 0) {
        print rec
    }
}
```

#

Reads a record from the specified input file `file10.txt` and stores the contents of the record in the `rec` variable.

Contents of `file10.txt`:

```
hitachi group01 Tokyo
HITACHI group02 Yokohama
hitachi group03 Fukuoka
HITACHI group04 Hokkaido
```

The results of executing the command are as follows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\awk -f prog12.awk
hitachi group01 Tokyo
HITACHI group02 Yokohama
hitachi group03 Fukuoka
HITACHI group04 Hokkaido
```

- Pass output from the `print` function to a command through a pipe. The script file is `prog13.awk`. The input file is `file11.txt`.

Contents of `prog13.awk`:

```
BEGIN{
    cmd = "sort "
}
{
    if (sub(/group01/,  $2)) {
        count++
        print | cmd#1
    }
}
END{
    close(cmd)#2
    print "total record : " count
}
```

#1

Passes records to the `sort` command specified in the `cmd` variable.

#2

Closes the pipe and terminates execution of the `sort` command by executing the `close` function.

Contents of `file11.txt`:

```
hitachi group01 003 tokyo
hitachi group02 001 yokohama
hitachi group03 001 fukuoka
hitachi group01 004 hokkaido
hitachi group01 001 nagoya
hitachi group02 001 hiroshima
hitachi group01 002 ooita
```

The results of executing the command are as follows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\awk -f prog13.awk file11.txt
hitachi group01 001 nagoya
hitachi group01 002 ooita
hitachi group01 003 tokyo
hitachi group01 004 hokkaido
total record : 4
```

- Delete an element of an array. The script file is `prog14.awk`.

  Contents of `prog14.awk`:

```
BEGIN{
    arry["Fukuoka"]   = "Fukuoka"
    arry["Hokkaido"]   = "Sapporo"
    arry["Kanagawa"] = "Yokohama"
    arry["Shimane"]   = "Matsue"
    for (key in arry) {
        printf("  %*s : %s\n",  6, key,   arry[key])
    }
    print "Deletes result of the array element"
    delete arry["Kanagawa"]
    for (key in arry) {
        printf("  %*s : %s\n",  6, key,   arry[key])
    }
}
```

  The results of executing the command are as follows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\awk -f prog14.awk
   Fukuoka : Fukuoka
  Hokkaido : Sapporo
  Kanagawa : Yokohama
   Shimane : Matsue
Deletes result of the array element
   Fukuoka : Fukuoka
  Hokkaido : Sapporo
   Shimane : Matsue
```

- When the program starts, create a directory and output the contents of the records to a file in the directory that was created. The script file is `prog15.awk`. The input file is `file12.txt`.

  Contents of `prog15.awk`:

```
BEGIN{
    if ((rc = system("mkdir dir001")#1)) {
        printf("system func error rc = %x\n",  rc) > "/dev/stderr"#2
        exit(1)
    }
}
{
    print >> "dir001\\outfile.txt"
}
```

#1

Creates a directory by executing the `mkdir` command from the `system` function.

#2

Outputs the return code of the system function to the standard error output.

Contents of `file12.txt`:

```
hitachi group01 Tokyo
HITACHI group02 Yokohama
hitachi group03 Fukuoka
HITACHI group04 Hokkaido
```

The results of executing the command are as follows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\awk -f prog15.awk file12.txt
C:\TEMP>%ADSH_OSCMD_DIR%\cat dir001\\outfile.txt
hitachi group01 Tokyo
HITACHI group02 Yokohama
hitachi group03 Fukuoka
HITACHI group04 Hokkaido
```

- Call a user-defined function and output its results. The script file is `prog16.awk`.

Contents of `prog16.awk`:

```
BEGIN{
a = 3
b = 4
result = func01(a,  b,  c)
print "func01 = " result
}
function func01(x, y){
x *= x
y *= y
return x + y
}
```

The results of executing the command are as follows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\awk -f prog16.awk
awk: warning: function func01 called with 3 args,  uses only 2#
 source line number 4
func01 = 25
```

#

A warning message is output because the number of arguments in the call to the user-defined function exceeds the number of arguments in the function definition.

- Display a message when a syntax error is detected in a control statement. The script file is `prog17.awk`.

Contents of `prog17.awk`:

```
BEGIN{
    while ((getline rec < "file10.txt") > 0)) {#
        print rec
    }
}
```

#

The number of opening and closing parentheses is mismatched in the `while` statement.

The results of executing the command are as follows:

```
C:\DIR>%ADSH_OSCMD_DIR%\awk -f prog17.awk
awk: extra ) at source line 2 source file prog17.awk
```

```
 context is
           while ((getline rec < "file10.txt") > >>>  0)) <<<
awk: syntax error at source line 2 source file prog17.awk
awk: illegal statement at source line 2 source file prog17.awk
        extra )
```

- Display a message when the syntax of a built-in function is invalid. The script file is `prog18.awk`.

  Contents of `prog18.awk`:

```
BEGIN{
    str = "Hitachi:hitachi"
    print "Column = " index(str)#
}
```

  #

  No character string is specified as an argument to the `index` function.

  The results of executing the command are as follows:

```
C:\DIR>%ADSH_OSCMD_DIR%\awk -f prog18.awk
awk: syntax error at source line 3 source file prog18.awk
 context is
           print "Column = " >>>  index(str) <<<
awk: illegal statement at source line 3 source file prog18.awk
```

- Use the temporary file functionality and path conversion functionality of JP1/Advanced Shell to perform input and output on a file. The job definition script is `adsh001.ash`. The script file is `prog19.awk`. The input file is `file12.txt`.

  Specification of the temporary file functionality and path conversion functionality for the environment files:

```
#-adsh_conf TEMP_FILE_DIR       "C:\\TEMP\\ADSH"
#-adsh_conf PATH_CONV_ENABLE / :
#-adsh_conf PATH_CONV "/var/hitachi/jp1as/perm" "C:\\hitachi\\JP1AS\\perm"
```

  Contents of `adsh001.ash`:

```
#-adsh_file_temp TEMP
#-adsh_step_start adsh001 -onError stop
"$ADSH_OSCMD_DIR/awk" -f prog19.awk "/var/hitachi/jp1as/perm/file12.txt"
#-adsh_step_error
exit 100
#-adsh_step_end
```

  Contents of `prog19.awk`:

```
{
    print FILENAME, ":", $0 > ENVIRON["TEMP"]
}
END{
while(getline var < ENVIRON["TEMP"])
    print var
}
```

  Contents of `file12.txt`:

```
001 abc
002 efgh
003 ijklmnop
```

Output of the awk command (sent to the standard output by the job):

```
C:\hitachi\JP1AS\perm\file12.txt : 001 abc
C:\hitachi\JP1AS\perm\file12.txt : 002 efgh
C:\hitachi\JP1AS\perm\file12.txt : 003 ijklmnop
```

- Display a message when no arguments are specified:

```
C:\DIR>%ADSH_OSCMD_DIR%\awk
usage: awk [-F fs] [-v var=value] [-f progfile | prog]
       [[file ...] | [built-in-var=value ...]] ...
```

- Display an option error message.

  This message might vary depending on the platform on which the command is executed. The following shows an example in Windows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\awk -z
awk: illegal option -- z
usage: awk [-F fs] [-v var=value] [-f progfile | prog]
     [[file ...] | [built-in-var=value ...]] ...
```

basename command (extracts a file name from a path)

Format

```
basename character-string[suffix]
```

Description

This command extracts a file name character string from a character string that conforms to the path naming conventions and then outputs it to the standard output.

If a suffix (a string of any characters) is specified, the command deletes the character string that matches the specified suffix from the end of the extracted character string.

The rules for extracting a file name character string are as follows:

- Of the elements separated by directory separators in the specified character string, the command extracts the rightmost element.
- In UNIX, a forward slash (`/`) is treated as a directory separator. In Windows, the `/` and the backslash (`\`) are both treated as directory separators.
- In Windows, colons (`:`) following a drive letter are treated as element separators.
- Multiple consecutive directory separators are treated as a single directory separator.
- If the specified character string ends with a directory separator, the command extracts the file name character string without the last directory separator.
- If the specified character string contains no directory separator, the command extracts the specified character string as is.

- If the specified character string consists of only directory separators, the command extracts the directory separators.

Arguments

character-string

Specifies the character string from which a file name is to be extracted.

suffix

Specifies the suffix to be deleted from the end of the extracted file name. If the he following conditions are satisfied, the command outputs the extracted file name character string as is:

- The suffix does not match the end of the file name character string.
- The specified suffix is the same as the extracted file name character string.

Return code

Return code	Meaning
0	Normal termination
1	Termination with an error

Notes

This command has no options. Any option that is specified in the argument is treated as part of the character string or suffix that is to be subject to the extraction of a file name character string.

Usage examples

- Extract file name character strings from path names.

  Example 1:

```
C:\TEMP>%ADSH_OSCMD_DIR%\basename E:\dir001\file01.txt
file01.txt
```

  Example 2:

```
C:\TEMP>%ADSH_OSCMD_DIR%\basename /dir001
dir001
```

  Example 3:

```
C:\TEMP>%ADSH_OSCMD_DIR%\basename .\file01.txt
file01.txt
```

  Example 4:

```
C:\TEMP>%ADSH_OSCMD_DIR%\basename E:\dir001\dir002\
dir002
```

  Example 5:

```
C:\TEMP>%ADSH_OSCMD_DIR%\basename E:\
E:
```

  Example 6:

```
C:\TEMP>%ADSH_OSCMD_DIR%\basename \\server01\
server01
```

Example 7:

```
C:\TEMP>%ADSH_OSCMD_DIR%\basename \\
\
```

Example 8:

```
C:\TEMP>%ADSH_OSCMD_DIR%\basename "C:\Documents and Settings\User01\My
Documents"
My Documents
```

Example 9:

```
C:\TEMP>%ADSH_OSCMD_DIR%\basename C:file01.txt
file01.txt
```

- Extract a file name from a path name without the extension:

```
C:\TEMP>%ADSH_OSCMD_DIR%\basename E:\dir001\file01.txt .txt
file01
```

cat command (outputs files to the standard output)

Format

```
cat [-b] [-n] [-s] [-u] [path-name ...]
```

Description

This command outputs one or more files to the standard output. If there are multiple files, they are concatenated and output.

Arguments

-b

Specifies that all non-blank output lines are to be assigned line numbers.

-n

Specifies that all output lines are to be assigned line numbers beginning with 1. Each line number is displayed as six digits. If the number of lines is such that line numbers cannot be accommodated in six digits, the number of digits is increased as necessary. A tab is output after each line number.

-s

Specifies that multiple consecutive blank lines are to be compressed and output as a single blank line.

-u

In UNIX, specifies that output buffering is to be suppressed.

In Windows, this option is ignored.

path-name

Specify the path name of a file to be output. Multiple path names can be specified, in which case the specified files are concatenated and then output. If no path name is specified, or if − is specified as the path name, the input is read from the standard input.

Return codes

Return code	Meaning
`0`	Normal termination
`1` or greater	Error termination

Notes

- A blank line is considered to be a line consisting of only a linefeed code `[LF]`. A line that includes `[CR]` + `[LF]` is not considered a blank line for the purposes of the `-b` and `-s` options. For this reason, files in Windows are normally not considered to include any blank lines.
- In Windows, input and output are performed in the binary mode for files and for the standard input and standard output. No conversion of linefeed codes is performed.
- If the standard output is a file and the specified path name indicates this file, the command outputs the following message and results in an error:

```
cat: file-name: input file is output file
```

Usage examples

These usage examples illustrate the results of executing the `cat` command on files `abc.txt` and `abcdex.txt`, whose contents are shown below. In the examples, Δ represents a space and ➔ represents a tab character.

- `abc.txt`

```
aaaaaaaaaaa

bbbbbbbb

ΔΔΔΔΔΔΔ
cccccccccccccccc

   ➔      ➔      ➔

ΔΔΔΔΔΔΔΔΔΔΔ
dddddddddddd
```

- `abcdex.txt`

```
aaaaaaaaaaa
ΔΔΔΔΔΔ
ΔΔΔΔΔΔ
ΔΔΔΔΔΔ
ΔΔΔΔΔΔ
ΔΔΔΔΔΔ
bbbbbbbb
```

```
cccccccccccc




dddddddddddd







eeeeeeeeeeeeeeeee
    →       →
    →       →
```

The files listed above are used as input files in the following examples, which illustrate the results of executing the `cat` command.

- Specify the `-b` option to number non-blank output lines.

```
$ cat -b abc.txt
     1→aaaaaaaaaaa

     2→bbbbbbbb

     3→ ΔΔΔΔΔΔΔ
     4→cccccccccccccccc

     5→ →       →       →

     6→ ΔΔΔΔΔΔΔΔΔΔΔ
     7→dddddddddddd
```

- Specify the `-n` option to number all output lines.

```
$ cat -n abc.txt
     1→aaaaaaaaaaa
     2→
     3→bbbbbbbb
     4→
     5→
     6→cccccccccccccccc
     7→
     8→
     9→
    10→
    11→dddddddddddd
```

- Specify the `-s` option to squeeze consecutive blank lines and output them as a single blank line.

```
$ cat -s abcdex.txt
aaaaaaaaaaa
ΔΔΔΔΔΔ
ΔΔΔΔΔΔ
```

```
ΔΔΔΔΔΔ
ΔΔΔΔΔΔ
ΔΔΔΔΔΔ
bbbbbbbb

cccccccccccc

dddddddddddd

eeeeeeeeeeeeeeeee
    ➔       ➔
    ➔       ➔
```

- Display an option error message.

  This message might vary depending on the platform on which the command is executed. The following shows an example in Windows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\cat -w
cat: illegal option -- w
usage: cat [-bnsu] [file ...]
```

- Display an error message because a file does not exist:

```
C:\TEMP>%ADSH_OSCMD_DIR%\cat file99
cat: file99: No such file or directory
```

cmp command (compares binary files)

Format

```
cmp [-l|-s] path-name-1 path-name-2 [skip-1 [skip-2]]
```

Description

This command compares binary files. Differences in byte values and the locations where the differences occur can be displayed.

Arguments

If the `-l` and `-s` options are both omitted, the command displays only the location where the first difference is detected. An error results if the `-l` and `-s` options are both specified.

-l

Specifies that each set of a difference in byte values (octal) and their offset in bytes (decimal) are to be displayed.

-s

Specifies that only a termination status indicating whether the files are different is to be returned.

path-name-1

Specifies the path name of the comparison source file. If you specify - for *path-name-1*, the contents for comparison are read from the standard input.

path-name-2

Specifies the path name of the comparison target file. If you specify - for *path-name-2*, the contents for comparison are read from the standard input.

skip-1

Specifies the position (in bytes) in the file in *path-name-1* that is to be the beginning point of the comparison processing.

skip-2

Specifies the position (in bytes) in the file in *path-name-2* that is to be the beginning point of the comparison processing.

Return codes

Return code	Meaning
`0`	Normal termination. The files are identical.
`1`	Normal termination. The files are different. Or, the end-of-file (`EOF`) was reached earlier in one file than in the other, in which case a message (`cmp: EOF on` *file-name*) is output.
`2` or greater	Error termination

Notes

- The end-of-line code `[CR]` + `[LF]` is considered two bytes.
- In Windows, input from a file or from the standard input is performed in binary mode. No conversion of end-of-line codes is performed.

Usage examples

These usage examples illustrate the results of executing the `cmp` command on the files `abc.txt` and `abcd.txt`, whose contents are shown below. In the examples, Δ represents a space and ➔ represents a tab character.

- `abc.txt`

```
aaaaaaaaaaa

bbbbbbbb

ΔΔΔΔΔΔΔ
cccccccccccccccc

    ➔       ➔       ➔

ΔΔΔΔΔΔΔΔΔΔΔ
dddddddddddd
```

- `abcd.txt`

```
aaaaaaaaaaa

bbbbbbbb

ΔΔΔΔΔΔΔ
cccccccccccccccc
ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ
ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ
```

```
ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ
ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ
dddddddddddd
eeeeeeeeeeeeeeeee
    →       →
    →       →
```

The files listed above are the input files in the following examples, which illustrate the results of executing the `cmp` command.

- Specify the `-l` option to display the offset in bytes (decimal) and the values (octal) of each difference between `abc.txt` and `abcd.txt`.

```
C:\TEMP>%ADSH_OSCMD_DIR%\cmp -l abc.txt abcd.txt
    49  12  40
    50  11  40
    51  11  40
    52  11  40
    53  12  40
    54  12  40
    65  40  12
    66  12  40
    67 144  40
    68 144  40
    69 144  40
    70 144  40
    71 144  40
    72 144  40
    73 144  40
    74 144  40
    75 144  40
    76 144  40
    77 144  40
    78 144  40
    79  12  40
cmp: EOF on abc.txt
```

- Specify the `-s` option to return the return code without displaying the results.

```
C:\TEMP>%ADSH_OSCMD_DIR%\cmp -s abc.txt abcd.txt
```

- Specify the optional skip arguments to set the starting bytes for comparison to 3 for each file. In the first example, the skip arguments are omitted. In the second example, they are set to 3.

```
C:\TEMP>%ADSH_OSCMD_DIR%\cmp abc.txt  abcd.txt
abc.txt abcd.txt differ: char 49, line 7

C:\TEMP>%ADSH_OSCMD_DIR%\cmp abc.txt  abcd.txt  3 3
abc.txt abcd.txt differ: char 46, line 7
```

- Display an option error message.

  This message might vary depending on the platform on which the command is executed. The following shows an example in Windows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\cmp -w
cmp: illegal option -- w
usage: cmp [-l | -s] file1 file2 [skip1 [skip2]]
```

- Display an error message if the file does not exist:

```
C:\TEMP>%ADSH_OSCMD_DIR%\cmp file99 file123
cmp: file99: No such file or directory
```

cp command (copies a file or directories)

Format

```
cp [-f|-i] [-p] [-R|-r [-H|-L|-P]] copy-source-file-name copy-target-file-
name
cp [-f|-i] [-p] [-R|-r [-H|-L|-P]] copy-source ... copy-target-directory-
name
```

Description

The command copies files or directories.

Arguments

-f

Specifies that no warning is to be issued if the target file will be overwritten. When the `-f` and `-i` options are both specified, the one specified last takes effect.

-i

Specifies that if the target file will be overwritten, a warning is to be issued asking the user to confirm the copy processing. Copying will be performed if the response from the standard input begins with the letter `y` or `Y`. If the response begins with any other character, or if the standard input is not available, processing is interrupted and the command terminates with a return code of `0`.

When the `-f` and `-i` options are both specified, the one specified last takes effect.

-p

Specifies that the attributes of the copy source file are to be kept intact (preserved).

In Windows, the most recent modification date and time and the most recent access date and time of the source file are preserved, but directory information is not preserved.

In UNIX, the owner, group, access permissions, last modification date and time, and last access date and time of each source file are preserved.

-R | -r

Specifies that directories are to be copied recursively.

-H

In UNIX, when the `-R` or `-r` option is specified, specifies that symbolic links on the command line are to be followed.

Symbolic links encountered while traversing the tree are not followed.

This option is ignored if neither the `-R` nor the `-r` option is specified. If more than one of the `-H`, `-L`, and `-P` options is specified, the one specified last takes effect.

In Windows, this option is ignored.

-L

In UNIX, when the `-R` or `-r` option is specified, specifies that all symbolic links that are encountered are to be followed.

This option is ignored if neither the `-R` nor the `-r` option is specified. If more than one of the `-H`, `-L`, and `-P` options is specified, the one specified last takes effect.

In Windows, this option is ignored.

-P

In UNIX, when the `-R` or `-r` option is specified, specifies that symbolic links are not to be followed.

This option is ignored if neither the `-R` nor the `-r` option is specified. If more than one of the `-H`, `-L`, and `-P` options is specified, the one specified last takes effect.

In Windows, this option is ignored.

copy-source-file-name

Specifies the name of the file to be copied.

copy-target-file-name

Specifies the name of the target file into which the source file's contents are to be copied.

copy-source

Specifies a file or directory to be copied.

copy-target-directory-name

Specifies the directory into which the copy source's contents are to be copied.

Return codes

Return code	Meaning
`0`	Normal termination
`1` or greater	Error termination

Notes

- In Windows, a target file is created with the file name specified for the copy source. However, uppercase letters in file names are converted to lowercase when copied. For example, if the name of the target file is `A.txt` and you execute `cp a.txt tmpdir`, the name of the file in `tmpdir` will be `a.txt`.
- In Windows, file input/output is performed in the binary mode. No conversion of end-of-line codes is performed.
- In UNIX, when a general user preserves the attributes of the copy source file by specifying the `-p` option but the owner of the source file is not the user executing the `cp` command, the owner, group, and access permission information (setuid bit, setgid bit, and sticky bit) of the source file will not be preserved.

Usage examples

- Specify the `-i` option to ask the user for confirmation before overwriting the target file.

```
C:\TEMP>%ADSH_OSCMD_DIR%\cp -i  file1.txt file2.txt
overwrite file2.txt? y

C:\TEMP>%ADSH_OSCMD_DIR%\cp -i file1.txt file2.txt
overwrite file2.txt? n
```

- Display an option error message.

  This message might vary depending on the platform on which the command is executed. The following shows an example in Windows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\cp -w
cp: illegal option -- w
```

```
usage: cp [-fip] [-Rr [-H | -L | -P]] source target
       cp [-fip] [-Rr [-H | -L | -P]] source ... directory
```

- Display an error message if the file does not exist.

```
C:\TEMP>%ADSH_OSCMD_DIR%\cp file99 file123
cp: file99: No such file or directory
```

cut command (outputs selected parts of lines to the standard output)

Format

```
cut -b list [-n] [--output-delimiter=character-string] [path-name ...]
cut -c list [--output-delimiter=character-string] [path-name ...]
cut -f list [-s] [-d delimiter] [--output-delimiter=character-string] [path-
name ...]
```

Description

This command outputs selected parts of lines to the standard output. For each line in the input files (or, by default, the standard input), the command selects the portions specified in *list* and outputs them to the standard output.

Arguments

If the `-b`, `-c`, and `-f` options are all omitted, the command outputs command usage information and terminates.

-b list

--bytes=list

Specifies that what is to be cut and output is defined as a range of byte positions. The byte positions are specified in *list* (where a file's first byte is position 1). If multiple byte position ranges are specified, they are concatenated.

If this option is specified together with the `--output-delimiter` option, the character string specified in the `--output-delimiter` option is used to concatenate data.

-c list

--characters=list

Specifies that what is to be cut and output is defined as a range of character positions. The character positions are specified in *list* (where a file's first character is position 1). If multiple character position ranges are specified, they are concatenated.

If this option is specified together with the `--output-delimiter` option, the character string specified in the `--output-delimiter` option is used to concatenate data.

-f list

--fields=list

Specifies that what is to be cut and output is defined as a range of field positions. The field positions are specified in *list* (where a file's first field is position 1). If multiple field position ranges are specified, they are concatenated.

The selected fields are output separated by a delimiter. If the delimiter is not found in a line, the entire line is output, unless the `-s` option is specified, in which case lines that do not contain the delimiter are not output.

By specifying the `--output-delimiter` option, you can change the separator that is output together with the selected fields.

list

Specifies either column positions or field positions as separated by the delimiter. Column positions start from 1.

You can specify multiple selection ranges by separating them with the comma, space, or tab. If you separate them with the space or tab, each selection range must be enclosed in double quotation marks (`"`). A selection range can be specified as *n*, *x*-, -*y*, or *x*-*y*, where *n*, *x*, and *y* are either field positions or column positions. No error results if a nonexistent position is specified.

- *n*: Specifies a single position that is to be output.
- *x*`-`: Specifies that all positions beginning with position *x* through the end of the file are to be output.
- `-`*y*: Specifies that all positions from the beginning of the file through position *y* are to be output.
- *x*`-`*y*: Specifies that all positions beginning with position *x* through position *y* are to be output. In such a case, *x* < *y*. If *x* > *y*, an error message (`cut: [-bcf]` *list*`: illegal list value`) is output.

-n

Specifies that multibyte characters are not to be split. If this option is not specified, the `-b` option results in multibyte characters being split into separate bytes.

path-name

Specifies the path name of a file that is to be read as the input. If *path-name* is omitted or a hyphen (`-`) is specified, the standard input is read.

-s

--only-delimited

Specifies that lines that do not contain a delimiter are not to be output. If this option is omitted when the `-f` option is specified, the command will only display command usage information and terminate.

-d delimiter

--delimiter=delimiter

Specifies the field delimiter to be used (only the initial character of the specified value is used as the delimiter). If this option is omitted, the field delimiter is set to the tab.

If this option is omitted when the `-f` option is specified, the command will only display command usage information and terminate.

--output-delimiter=character-string

When this option is specified together with the `-f` option, specifies the replacement character string that is to be used as the separator for the fields that are to be output.

When this option is specified together with the `-b` or `-c` option, specifies the replacement character string that is to be used to concatenate fields.

Return codes

Return code	Meaning
`0`	Normal termination
`1`	Error termination

Notes

- The cut command expects text files. Input from binary files and output of binary data are not guaranteed to work.

Usage examples

These usage examples illustrate the results of executing the cut command on the file test.txt, whose contents are shown in the following:

```
123:5678:abcdef:hijkl
field1:field2:field3:filed4
sssssssssssssssssssssss
```

This file is used as the input file in the following examples, which illustrate the results of executing the cut command.

- Output the first byte and the third through fifth bytes.

```
$ cut -b 1,3-5 test.txt
13:5
feld
ssss
```

- Output the first through fourth characters.

```
$ cut -c -4 test.txt
123:
fiel
ssss
```

- Output the first and fourth fields.

```
$ cut -f 1,4 -d : test.txt
123:hijkl
field1:filed4
sssssssssssssssssssssss
```

- Output the first byte and the third through fifth bytes with character string @:/ added between fields.

```
$ cut -b 1,3-5 --output-delimiter="@:/" test.txt
1@:/3:5
f@:/eld
s@:/sss
```

- Display the first and fourth fields and replace the separator with the character string @:/.

```
$ cut -f 1,4 -d : --output-delimiter="@:/" test.txt
123@:/hijkl
field1@:/field4
sssssssssssssssssssssss
```

- Output an option error message.

  This message might vary depending on the platform on which the command is executed. The following shows an example in Windows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\cut -z
cut: illegal option -- z
usage: cut -b list [-n] [--output-delimiter=string] [file ...]
```

```
        cut -c list [--output-delimiter=string] [file ...]
        cut -f list [-s] [-d delim] [--output-delimiter=string] [file ...]
```

date command (displays the system date and time)

Format

```
date [-u] [-d date-information-string | -r elapsed-seconds] [+format]
```

Description

This command displays the system date and time.

Arguments

-u

--utc

--universal

Specifies that the date is to be displayed in UTC (Coordinated Universal Time).

-d date-information-string

--date=date-information-string

Specifies the date and time to be displayed as a date-information string. If more than one information string is specified, the last string specified takes effect. For the supported date-information strings, see *Date-information strings that can be specified in the -d option*.

-r elapsed-seconds

Specifies that the display is to be of the date and time after the amount of time specified by *elapsed-seconds* had elapsed since the beginning of the UNIX epoch (UTC January 1, 1970, 00:00:00). The value specified in *elapsed-seconds* can range from `-1009875600` to `2147440447`. The output result cannot be guaranteed if the specified value is outside this range.

+format

Specifies a display format for the date and time using format specification codes. The format specification codes for the `strftime` function (OS's API) can be used. For details about the supported format specification codes, see *Supported format specification codes*, below.

If this argument is not specified, the display format for the date and time will be `%Y/%m/%d %A %H:%M:%S %Z`.

Supported format specification codes

In the argument that begins with a plus sign (+), you can specify the format specification codes for the `strftime` function (OS's API). These format specification codes vary according to the OS. For the format specification codes supported by your OS, see the information about the `strftime` function in the OS documentation.

The following table lists and describes the typical format specification codes that are supported by all OSs:

Format specification code	Meaning
%a	Abbreviation for the day of the week
%A	Full name for the day of the week
%b	Abbreviation for the month
%B	Full name for the month
%c	Date and time display based on the locale
%d	Day of the month as a decimal number (01 to 31)
%H	Time in the 24-hour clock (00 to 23)
%I	Time in the 12-hour clock (01 to 12)
%j	Number of days since the beginning of the year as a decimal number (001 to 366)
%m	Month as a decimal number (01 to 12)
%M	Minute as a decimal number (00 to 59)
%p	AM or PM based on the current locale
%S	Second as a decimal number. The range of values displayed varies according to the OS depending on support for leap seconds.
%U	Week as a decimal number (00 to 53). The first Sunday of the year begins the weeks.
%w	Day of the week as a decimal number (0 to 6, where 0 indicates Sunday)
%W	Week as a decimal number (00 to 53). The first Monday of the year begins the weeks.
%x	Date based on the current locale
%X	Time based on the current locale
%y	Last two digits of the year as a decimal number (00 to 99)
%Y	Year as four decimal digits
%Z	Time zone name. If the time zone is unknown, this information is not displayed.
%%	Percent symbol (%)

Date-information strings that can be specified in the -d option

Specify as explained below the date-information string that you want the date command to display:

- Specifying an absolute date only

  The specified date and time are displayed.
- Specifying a relative date only

  The date and time shifted from the current date and time are displayed.
- Specifying an absolute date combined with a relative date

  The date and time obtained by shifting the date and time specified as an absolute date by the date and time specified as a relative date are displayed.

The elements of the date-information string can be specified as uppercase or lowercase letters. If a specified value is earlier than 1970-01-01 00:00:00 or later than 2038-01-19 03:14:07 in UTC, the command issues the error message date: Invalid date: *specified-value* and then terminates with an error. In AIX, if a specified value is later than

2038-01-19 03:14:07 in the local time zone, the command issues error message `date: Invalid date:` *specified-value* and then terminates with an error. If the null character is specified as the date-information string, the command displays 00:00:00 on the current date.

The following explains the permitted elements of a date-information string and the syntax.

- Specifying absolute dates
  The following table lists and describes the elements of a date-information string that specifies an absolute date.

Table 8–6: Elements of date-information strings (specifying an absolute date)

Type	Specifiable element#	Details
Year	Four-digit calendar year in decimal (*YYYY*)	Permitted values are from `1970` to `2038`.
	Last two digits of the calendar year in decimal (*YY*)	Permitted values are from `00` to `99`. For `69` to `99`, 1969 to 1999 are assumed; for `00` to `68`, 2000 to 2068 are assumed.
Month	Name of month (*MONTH*)	The following values can be specified: • `January` or `Jan` • `February` or `Feb` • `March` or `Mar` • `April` or `Apr` • `May` • `June` or `Jun` • `July` or `Jul` • `August` or `Aug` • `September`, `Sept`, or `Sep` • `October` or `Oct` • `November` or `Nov` • `December` or `Dec`
	Month in decimal (*MM*)	Permitted values are from `01` to `12`.
Day	Day of the month in decimal (*DD*)	Permitted values are from `01` to `31`.
Hour	Time in 24-hour representation (*hh*)	Permitted values are from `00` to `23`.
	Time in 12-hour representation (*hh*)	Permitted values are from `01` to `12`.
Minute	Minute in decimal (*mm*)	Permitted values are from `00` to `59`.
Second	Second in decimal (*ss*)	Permitted values are from `00` to `59`.
am/pm	am and pm (`am | a.m. | pm | p.m.`)	The following values can be specified after the time: • `am` or `a.m.` • `pm` or `p.m.`
Time zone	Time zone name (*ST*)	The following values can be specified:

Type	Specifiable element#	Details
Time zone	Time zone name (*ST*)	• `UTC` or `UT`: Coordinated Universal Time • `GMT`: Greenwich Mean Time (UTC + 0 hours) • `WET`: Western European Time (UTC + 0 hours) • `AST`: Atlantic Standard Time (UTC - 4 hours) • `EST`: Eastern Standard Time (UTC - 5 hours) • `CST`: Central Standard Time (UTC - 6 hours) • `MST`: Mountain Standard Time (UTC - 7 hours) • `PST`: Pacific Standard Time (UTC - 8 hours) • `HST`: Hawaii Standard Time (UTC - 10 hours) • `WAT`: West Africa Time (UTC + 1 hours) • `CET`: Central European Time (UTC + 1 hours) • `MET`: Middle European Time (UTC + 1 hours) • `CAT`: Central Africa Time (UTC + 2 hours) • `EET`: Eastern European Time (UTC + 2 hours) • `JST`: Japan Standard Time (UTC + 9 hours) • `GST`: Guam Standard Time (UTC + 10 hours) • `NZST`: New Zealand Standard Time (UTC + 12 hours)
	Military time zone (*ST*)	The following values can be specified: • `A`: UTC - 1 hour • `B`: UTC - 2 hours • `C`: UTC - 3 hours • `D`: UTC - 4 hours • `E`: UTC - 5 hours • `F`: UTC - 6 hours • `G`: UTC - 7 hours • `H`: UTC - 8 hours • `I`: UTC - 9 hours • `K`: UTC - 10 hours • `L`: UTC - 11 hours • `M`: UTC - 12 hours • `N`: UTC + 1 hour • `O`: UTC + 2 hours • `P`: UTC + 3 hours • `Q`: UTC + 4 hours • `R`: UTC + 5 hours • `S`: UTC + 6 hours • `T`: UTC + 7 hours • `U`: UTC + 8 hours • `V`: UTC + 9 hours • `W`: UTC + 10 hours • `X`: UTC + 11 hours • `Y`: UTC + 12 hours • `Z`: UTC
	Specification of time from UTC (+*hhmm* | −*hhmm* | +*hh*:*mm* | −*hh*:*mm*)	The time from UTC can be specified in the format +*hhmm*, −*hhmm*, +*hh*:*mm*, or −*hh*:*mm* (*mm* and :*mm* can be omitted).

Type	Specifiable element#	Details
Time zone (summer time)	Time zone (summer time) (*DT*)	The following values can be specified: • `BST`: British Summer Time (GMT + 1 hours) • `ADT`: Atlantic Daylight Time (AST + 1 hours) • `EDT`: Eastern Daylight Time (EST + 1 hours) • `CDT`: Central Daylight Time (CST + 1 hours) • `MDT`: Mountain Daylight Time (MST + 1 hours) • `PDT`: Pacific Daylight Time (PST + 1 hours) • `MEST`: Middle European Summer Time (MET + 1 hours) • `NZDT`: New Zealand Daylight Time (NZST + 1 hours)
Daylight saving time	Daylight saving time (*DST*)	You can specify `DST` for daylight saving time. When `DST` is specified together with a time zone, the specified time zone's standard time is advanced by one hour regardless of the specified date, time, and time zone. `DST` cannot be specified alone without a time zone specification.

#

Letters in parentheses, such as *YY* and *MONTH*, correspond to the syntax explained in the table below.

The following table explains the syntax for date-information strings when an absolute date is specified.

Table 8–7: Syntax for date-information strings (when specifying an absolute date)

Type#	Syntax	Details
Date specification	[*YY*]*YYMMDD*	Define year, month, and day in this order without spaces. The first two digits of the year can be omitted.
	[*YYYY*/]*MM*/*DD*	Define year, month, and day in this order separated by `/`. If the year is omitted, the current year is assumed.
	MM/*DD*[/[*YY*]*YY*]	Define month, day, and year in this order separated by `/`. If the year is omitted, the current year is assumed. The first two digits of the year can be omitted.
	[*YY*]*YY*-*MM*-*DD*	Define year, month, and day in this order separated by `-`. The first two digits of the year can be omitted.
	DD MONTH [[*YY*]*YY*]	Define day, the formal name for month (or abbreviation for month), and year in this order separated by the space. If the year is omitted, the current year is assumed. The first two digits of the year can be omitted.
	MONTH DD [,[*YY*]*YY*]	Define the formal name for month (or abbreviation for month), day, and year in this order. Separate the formal name for month (or abbreviation for month) and day with the space and separate day and year with the comma (`,`). If the year is omitted, the current year is assumed. The first two digits of the year can be omitted.
	DD[-]*MONTH*[[-][*YY*]*YY*]	Define day, the formal name for month (or abbreviation for month), and year in this order without any spaces or separator or with the `-` as the separator. If the year is omitted, the current year is assumed. The first two digits of the year can be omitted.
	MONTH-*DD*-[*YY*]*YY*	Define the formal name for month (or abbreviation for month), day, and year in this order separated by `-`. The first two digits of the year can be omitted.
	MONTH DD [[*YY*]*YY*]	Define the formal name for month (or abbreviation for month), day, and year in this order separated by the space. If the year is omitted, the current year is assumed. The first two digits of the year can be omitted.
Time specification	*hh*[*mm*] *hh* [am | a.m. | pm | p.m.]	Define the hour and minute in this order without any spaces. The value for minute can be omitted. If only a value for hour is defined, `am` (`a.m.`) or `pm` (`p.m.`) can be specified following a space.

Type#	Syntax	Details
Time specification	*hh*[*mm*] *hh* [am | a.m. | pm | p.m.]	If you are specifying am (a.m.) or pm (p.m.), use the 12-hour representation for the time.
	hh:*mm*[:*ss*] [am | a.m. | pm | p.m.]	Define hour, minute, and second in this order separated by colons (:). The value for second can be omitted. You can specify am (a.m.) or pm (p.m.) following the definition of hour, minute, and second and a space. If you are specifying am (a.m.) or pm (p.m.), make sure that the definition of hour, minute, and second does not exceed 12:59:59.
	hh:*mm*[:*ss*] [+*hh*[*mm*] | -*hh*[*mm*] | +*hh*[:*mm*] | -*hh*[:*mm*]]	Define hour, minute, and second in this order separated colons (:). The value for second can be omitted. You can specify +*hhmm*, -*hhmm*, +*hh*:*mm*, or -*hh*:*mm* following the definition of the hour, minute, and second and a space.
Time zone	*ST* [*DST*] *ST* [+*hh*[*mm*] | -*hh*[*mm*] | +*hh*[:*mm*] | -*hh*[:*mm*]]	Specify a time zone. You can specify *DST*, +*hhmm*, -*hhmm*, +*hh*:*mm*, or -*hh*:*mm* following the time zone and a space.
	DT	Define a time zone (summer time). The time zone (summer time) cannot be specified together with *DST*, +*hhmm*, -*hhmm*, +*hh*:*mm*, or -*hh*:*mm*.

#

The same type cannot be defined multiple times, but different types can be combined.

If only a date is specified, 00:00:00 is assumed for the time.

If only hour (*hh*) is specified, minute (*mm*) and second (*ss*) are set to 0 on the current date.

If only hour (*hh*) and minute (*mm*) are specified, second (*ss*) is set to 0 on the current date.

- Specifying relative dates

  The following table lists and describes the elements of a date-information string that specifies a relative date.

Table 8–8: Elements of date-information strings (specifying a relative date)

Specifiable element#	Details
Number of years and months to be shifted (*DATE*)	Specify the following values: • year or years: Number of years to be shifted • month or months: Number of months to be shifted You can specify a numeric value (*NUM*) before this value. If the numeric value (*NUM*) is omitted, 1 is assumed.
Number of days to be shifted (*DATE*)	Specify the following values: • fortnight or fortnights: Shift by two weeks (14 days) • week or weeks: Shift by one week (7 days) • day or days: Shift by day(s) • tomorrow: Tomorrow (one day later) • yesterday: Yesterday (one day ago) • today: Today (0 day) • now: Now (0 day) You can specify a numeric value (*NUM*) before this value. If the numeric value (*NUM*) is omitted, 1 is assumed.
Number of hours and minutes to be shifted (*DATE*)	Specify the following values: • hour or hours: Number of hours to be shifted • minute, min, or minutes: Number of minutes to be shifted

Specifiable element[#]	Details
Number of hours and minutes to be shifted (*DATE*)	You can specify a numeric value (*NUM*) before this value. If the numeric value (*NUM*) is omitted, `1` is assumed.
Number of seconds to be shifted (*DATE*)	Specify the following value: • `second`, `sec`, or `seconds`: Number of seconds to be shifted You can specify a numeric value (*NUM*) before this value. If the numeric value (*NUM*) is omitted, `1` is assumed.
Number of days of week to be shifted (*DAY*)	Specify the following values: • `Monday` or `Mon` • `Tuesday`, `Tue`, or `Tues` • `Wednesday`, `Wed`, or `Wednes` • `Thursday`, `Thu`, `Thur`, or `Thurs` • `Friday` or `Fri` • `Saturday` or `Sat` • `Sunday` or `Sun` You can specify a numeric value (*NUM*) before this value to specify the *NUM*-th day of the week. If the numeric value (*NUM*) is omitted, `1` is assumed. If neither `1` nor a numeric value (*NUM*) is specified, the next day of the week is assumed. Neither signs (`+|-`) nor the before and after specification (`ago`) can be specified.
Numeric value specification (character string) (*NUM*)	Specify the following values: • `last`: -1 • `this`: 0 • `next`: 1 • `first`: 1 • `third`: 3 • `fourth`: 4 • `fifth`: 5 • `sixth`: 6 • `seventh`: 7 • `eighth`: 8 • `ninth`: 9 • `tenth`: 10 • `eleventh`: 11 • `twelfth`: 12 A numeric value (*NUM*) cannot be specified before this value.
Numeric value specification (number) (*NUM*)	Permitted values are from `0` to `2147483647`.
Sign (`+|-`)	You can specify the following values before the numeric value (*NUM*): • `+`: Positive • `-`: Negative If the sign is not followed by a numeric value, the sign is ignored.
Before and after specification (`ago`)	Specify the following value: • `ago`: Ago – The positive or negative sign for the date-information string specified immediately before `ago` is reversed (positive becomes negative and negative becomes positive).

[#] Letters in parentheses, such as *NUM* and *DATE*, correspond to the syntax explained in the table below.

The table below explains the syntax for date-information strings when a relative date is specified. *Shift in date and time* can be combined with *shift in day of week*.

Table 8–9: Syntax for date-information strings (when specifying a relative date)

Type	Syntax	Details
Shift in date and time	`[[+ | -]`*NUM*`]` *DATE* `[ago]`	Specify the amount of time to be shifted from the current date and time or the date and time specified as a date-information string. Multiple elements can be specified by separating them with the space. For the elements that can be combined, see *Table 8-8 Elements of date-information strings (specifying a relative date)*.
Shift in day of week	`[`*NUM*`]` *DAY* *DAY*`[,]`	Specify a day of the week. Only one value can be specified. A day of the week immediately preceded by a numeric value (*NUM*) defines the *NUM*-th day of the week. If the numeric value (*NUM*) is omitted, the next day of the week is assumed. The specified day of the week can be followed by a comma (`,`) or a space, and then by another definition of *shift in date and time*.

- Other specification

  The following table describes other elements of date-information strings.

Table 8–10: Elements of date-information strings (other)

Specifiable element	Details
Comment	Specify any character string as a comment by enclosing it in parentheses (`( )`) within a date-information string. If `( )` is nesting within another `( )`, the character string specified in the inner `( )` is ignored. If only a left parenthesis (`(`) is specified, all characters following the left parenthesis are ignored.

The specified absolute and relative date and time values are obtained in the order shown below. In steps 2 through 4, if the value converted to seconds is outside the range from `0` to `2147483647`, an error might result regardless of the final results.

1. Obtains the current date and time or the date and time specified as an absolute date.
2. Adds the *shift in day of week* specified as a relative date to the value obtained in step 1. If *date specification* is specified as an absolute date, the *number of days of week to be shifted* (*DAY*) is not added, if specified.
3. Adds or subtracts all results obtained based on the specified *number of years and months to be shifted* (*DATE*) as a relative date to or from the value obtained in step 2.
4. Adds or subtracts all results obtained based on the specified *number of days to be shifted* (*DATE*), *number of hours and minutes to be shifted* (*DATE*), and *number of seconds to be shifted* (*DATE*) specified as a relative date to or from the value obtained in step 3.

Steps 2 through 4 obtain the amount to be shifted from the date and time subject to calculation.

For example, if the current date and time is 2014-04-30 at 10:10:10 and `date -d "Fri, 1 year 1 month 1 day 1 hour 1 min 1 sec"` is specified, the following calculation occurs:

1. Obtain the current date and time.

   ➜ The result is 2014-04-30 (Wednesday) at 10:10:10.

2. Add the number of days from 2014-04-30 (Wednesday) to the next Friday.

   ➜ The result is 2014-05-02 (Friday) at 10:10:10.

3. Add 365 days and the number of days in May 2014.

   ➜ The result is 2015-06-02 (Monday) at 10:10:10.

4. Add one day, one hour, one minute, and one second.

   ➜ The result is 2015-06-03 (Tuesday) at 11:11:11.

Return codes

Return code	Meaning
`0`	Normal termination
`1` or greater	Error termination

Notes

- If a format code specified in *+format* is outside the range of valid format codes of the `strftime` function, the specified value is output as is.
- In Windows, if you specify a mixture of valid and invalid format codes, the specified values are output as is, without any format code conversion.
- In UNIX, valid format codes are converted while the specified values are output as is for invalid format codes.
- In Windows, if you specify the `TZ` environment variable, you must ensure that the value of the `TZ` environment variable matches the time zone set in the **Date and Time** control panel. If their values do not match, the correct date and time might not be displayed.
- An unsupported argument that is specified in the command line is ignored.
- If either of the following environment variables is specified, any option following an argument that begins with a plus sign (+) is ignored:
  - `POSIXLY_CORRECT` environment variable
  - `ADSH_CMD_ARGORDER=seq` environment variable

Usage examples

- Display the default with no options specified.

```
C:\TEMP>%ADSH_OSCMD_DIR%\date
2011/05/09 Monday 02:03:05 JST
```

- Specify the `-u` option to display the date and time in UTC (Coordinated Universal Time).

```
C:\TEMP>%ADSH_OSCMD_DIR%\date -u
2011/05/08 Sunday 17:03:11 UTC
```

- Specify the `-r` option to display the date and time after a specified number of seconds had elapsed from the beginning of the UNIX epoch.

```
C:\TEMP>%ADSH_OSCMD_DIR%\date -r 1234567890
2009/02/14 Saturday 08:31:30 JST
```

- Specify a display format for the date and time in the *+operand* argument.

```
C:\TEMP>%ADSH_OSCMD_DIR%\date "+%Y-%m-%d %H.%M.%S"
2011-05-09 02.10.02
```

- Display the date of December 12 this year.

```
C:\TEMP>%ADSH_OSCMD_DIR%\date -d "12/12"
2011/12/12 Sunday 00:00:00 JST
```

- Display the date that is three months and one day from the current date.

```
C:\TEMP>%ADSH_OSCMD_DIR%\date -d "3 months 1 day"
2011/08/10 Wednesday 00:00:00 JST
```

- Display the date that was two days ago.

```
C:\TEMP>%ADSH_OSCMD_DIR%\date -d "2 days ago"
2011/05/07 Saturday 00:00:00 JST
```

- Display the date that falls 100 days after 2011-05-01. This example uses the `--date` option to specify a date-information string.

```
C:\TEMP>%ADSH_OSCMD_DIR%\date --date="1-May-2011 100 days"
2011/08/09 Tuesday 00:00:00 JST
```

- Display an option error message.

  This message might vary depending on the platform on which the command is executed. The following shows an example in Windows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\date -a
date: illegal option -- a
usage: date [-u] [-d string | -r seconds] [+format]
```

diff command (compares two files)

Format

```
diff[-a][-b][-i][-s][-w]
     [-cnum-lines|-C num-lines|-q|-unum-lines|-U num-lines|
       -y[-W output-width] [--suppress-common-lines]]
     [-L label]
      path-name-1 path-name-2
diff[-a][-b][-i][-r][-s][-w]
     [-cnum-lines|-C num-lines|-q|-unum-lines|-U num-lines|
       -y[-W output-width] [--suppress-common-lines]]
     [-L label]
      directory-name-1 directory-name-2
```

Description

This command compares two files.

Arguments

-a

--text

Specifies that the files are to be handled as text files.

-b

--ignore-space-change

Specifies that multiple consecutive spaces or tabs in a line are to be compressed and handled as a single space or tab. The `-b` and `-w` options cannot both be specified at the same time.

-i

--ignore-case

Specifies that differences in case are not to be considered (uppercase alphabetic letters are not to be distinguished from lowercase alphabetic letters).

-s

--report-identical-files

Specifies that a message (`Files` *path-name-1* `and` *path-name-2* `are identical`) is to be output if the contents of the files are identical.

-w

--ignore-all-space

Specifies that all spaces and tabs in a line are to be ignored. The `-w` and `-b` options cannot both be specified at the same time.

-cnum-lines

-C num-lines

--context[=num-lines]

Specifies that the path names being compared are to be output to the standard output with additions, deletions, and changes to lines indicated by the symbols `+`, `-`, and `!`, respectively.

The number of lines specified in *num-lines* are to be output for context before and after each difference. If *num-lines* is not specified in the `-c` and `--context` options, the default is 3 lines.

Note that when *num-lines* is specified in the `-c` option, there must not be any spaces between `-c` and *num-lines*.

-q

--brief

Specifies that if there are differences between the files, only the message `Files` *path-name-1* `and` *path-name-2* `differ` is to be output.

-unum-lines

-U num-lines

--unified[=num-lines]

Specifies that the path names being compared are to be output to the standard output with additions and deletions indicated by the symbols `+` and `-`, respectively. The old and new text are output together as a single section.

The number of lines specified in *num-lines* are to be output for context before and after each difference. If *num-lines* is not specified in the `-u` and `--unified` options, the default is 3 lines.

Note that when *num-lines* is specified in the `-c` option, there must not be any spaces between `-u` and *num-lines*.

-L label

--label=label

When the `-c`, `-u`, `-C`, or `-U` option is specified, specifies that labels are to be output in place of the path names. If one label is specified, it is output in place of *path-name-1*. If two labels are specified, they are output in place of *path-name-1* and *path-name-2*, respectively, in the order they are specified.

-y

--side-by-side

Specifies that the symbols `>`, `<`, `|`, `\`, and `/` are to be used to indicate the difference information.

The `>`, `<`, `|`, `\`, and `/` symbols are output only for lines in which there is a difference in terms of line addition, deletion, or change or whether an end-of-line code is used; no symbol is output for lines in which there are no differences. The *path-name-1* and *path-name-2* lines are combined on one line and output side by side.

If the length of one combined line exceeds 130 columns, the lengths of the *path-name-1* and *path-name-2* lines are adjusted before the lines are output.

This option can be combined with the following options to change the output:

- `-W` option
- `--suppress-common-lines` option

-W output-width

--width=output-width

Specifies a change in the output width (number of columns) to be output per line. This option take effect when it is specified together with the `-y` option.

--suppress-common-lines

Specifies that lines that have no differences are not to be output. This option takes effect when it is specified together with the `-y` option.

path-name-1

Specifies the path name of the comparison source file.

Specify `-` to read the contents to be compared from the standard input. A temporary file will be created to store input from the standard input. The temporary file is output to the following directory:

- UNIX

  Directory specified in the `TMPDIR` environment variable, or `/var/tmp` if no value has been set for `TMPDIR`.
- Windows

  common-application-data-folder`\HITACHI\JP1AS\misc`

path-name-2

Specifies the path name of the comparison target file.

Specify `-` to read the contents to be compared from the standard input. A temporary file will be created to store input from the standard input. The temporary file is output to the following directory:

- UNIX

  Directory specified in the `TMPDIR` environment variable, or `/var/tmp` if no value has been set for `TMPDIR`.
- Windows

  common-application-data-folder`\HITACHI\JP1AS\misc`

-r

--recursive

If directories are being compared, specifies that subdirectories are to be traversed recursively.

directory-name-1

Specifies a directory as the comparison source. If you specify a file path name for either *directory-name-1* or *directory-name-2*, the command searches for a file with the same name in the other directory. If no file with the same name is found, an error message (`diff:` *path-name-to-compare*`: No such file or directory`) is output.

directory-name-2

Specifies a directory as the comparison target. If you specify a file path name for either *directory-name-1* or *directory-name-2*, the command searches for a file with the same name in the other directory. If no file with the same name is found, an error message (`diff:` *path-name-to-compare*`: No such file or directory`) is output.

Display formats

The `diff` command provides the three display formats shown below for displaying the differences between files. The output format that is used depends on the specification of options.

Format	Meaning
Traditional display format	This output format is used when none of the `-c`, `-C`, `-q`, `-u`, `-U` and `-y` options is specified. This format displays the differences between the two files, as well as the start and end positions of the differences. In between the start and end positions, it displays the following symbols, which represent the differences: • `a`: Added • `d`: Deleted • `c`: Changed If a difference extends across multiple lines, the numbers of the start and end lines are shown separated by the comma (`,`). A difference from *path-name-1* and a difference from *path-name-2* are displayed in this order separated from each other by `---`. < at the beginning of a line indicates a deletion or change from the first input file, while > indicates an addition or change from the second input file. One space is output after < and >.
Context format	This output format is used when the `-c` or `-C` option is specified. In addition to displaying the differences between the two files, it also displays unchanged lines before and after each difference. You can specify the number of such context lines to be displayed. The default is three lines. Headers display the following information about the two files: • Boundaries between change blocks: a row of 15 asterisks (`*`) • The starting and ending positions of the differences between the two files, followed by the differences themselves The differences are expressed as follows: • Line starting with `+`: Added line • Line starting with `-`: Deleted line • Line starting with `!`: Changed line A single space is output after `+`, `-`, and `!`. Two spaces are output at the beginning of a line on which there are no differences. Adjacent changed lines are treated as a single change block. However, when there is a gap between changed lines, another 15 asterisks are displayed and then the differences are displayed.
Unified format	This output format is used when the `-u` or `-U` option is specified. It displays the context format output as a single section. You can specify the number of lines of context to be displayed. The default is three lines. Headers display the following information for the two files: • Lines that start with `@@`: The starting and ending positions of the differences between the two files, followed by the differences themselves The differences are displayed as follows:

Format	Meaning
Unified format	• Line starting with `+`: Added line • Line starting with `-`: Deleted line No space is output after `+` or `-`, unlike in the context format. One space is output at the beginning of an unchanged line. Only added and deleted lines represent differences. Adjacent changed lines are treated as a single change block. However, when there is a gap between changed lines, `@@` are again used to identify lines that constitute the starting and ending positions of the differences between the two files, followed by the differences themselves.
Side-by-side format	This output format is used when the `-y` option is specified. The *path-name-1* and *path-name-2* lines are combined on one line and output side by side. By default, all lines are output regardless of whether there are any differences on lines. If the length of one combined line exceeds 130 columns, the lengths of the *path-name-1* and *path-name-2* lines are adjusted so that they can be displayed side by side within 130 columns. The following symbols are displayed before the lines to indicate the difference: • `>`: Line was added • `<`: Line was deleted • `|`: Line was changed • `\` (escape character): *path-name-1* line contains no end-of-line code • `/`: *path-name-2* line contains no end-of-line code By combining the `-y` option with the `-W` and `--suppress-common-lines` options, you can change the output width per line and suppress output of lines that have no difference.

Traditional display format example

The following example illustrates the traditional display format.

Output example

```
C:\USR\JP1\oscmd\bin>diff file1 file2
1c1,2                               <--1.
< aaaaaaaaaaa                       <--2.
---                                 <--3.
> aaAAAAAaaaa                       <--4.
> bbBBBBBbbbb                       <--4.
```

Explanation

1. This shows the position of a difference between `file1` and `file2`. To represent the difference between `file1` and `file 2`, the symbol `a` means added, `d` means deleted, and `c` means changed. The line number from `file1` is displayed before the symbol, and the line number from `file2` is displayed after it. If there are differences that span multiple lines, the start line and end line numbers are separated by the comma (`,`).
2. This shows the difference line in `file1`.
3. This shows the boundary between the difference lines in `file1` and `file2`.
4. This shows the difference lines in `file2`.

Context format example

Output example

```
C:\USR\JP1\oscmd\bin>diff -c file1 file2
*** file1       Thu May 12 20:17:54 2011  <--1.
--- file2       Thu May 12 20:18:29 2011  <--2.
```

```
***************                                 <--3.
*** 1,5 ****                                    <--4.
  aaaaaaaaaaa                                   <--5.
! bbbbbbbb                                      <--5.
  cccccccccccc                                  <--5.
- ddddddddddd                                   <--5.
  eeeeeeeee                                     <--5.
--- 1,5 ----                                    <--6.
  aaaaaaaaaaa                                   <--7.
! bbbBBBbb                                      <--7.
  cccccccccccc                                  <--7.
  eeeeeeeee                                     <--7.
+ ffffffffffffffff                              <--7.
```

Explanation

1. This shows the file name and most recent modification date and time as the information about `file1`.
2. This shows the file name and most recent modification date and time as the information about `file2`.
3. This shows a row of 15 asterisks (`*`), which represents the border between change blocks. Whenever changed lines are separated by three or more lines, this border is displayed again, and then the next change block showing differences between `file1` and `file2` is output.
4. This shows the start and end positions of the `file1` differences, separated by the comma (`,`).
5. This shows the `file1` differences. `+` means added, `-` means deleted, and `!` means changed.
6. This shows the start and end positions of the `file2` differences, separated by the comma (`,`).
7. This shows the `file2` differences. `+` means added, `-` means deleted, and `!` means changed.

Unified format example

Output example

```
C:\USR\JP1\oscmd\bin>diff -u file1 file2
--- file1       Thu May 12 20:17:54 2011  <--1.
+++ file2       Thu May 12 20:18:29 2011  <--2.
@@ -1,5 +1,5 @@                           <--3.
 aaaaaaaaaaa                              <--4.
-bbbbbbbb                                 <--4.
+bbbBBBbb                                 <--4.
 cccccccccccc                             <--4.
-ddddddddddd                              <--4.
 eeeeeeeee                                <--4.
+ffffffffffffffff                         <--4.
```

Explanation

1. This shows the file name and most recent modification date and time as the information about `file1`.
2. This shows the file name and most recent modification date and time as the information about `file2`.
3. This shows the start and end positions of the `file1` changes, separated by the comma (`,`), a space, and then the start and end positions of the `file2` changes, separated by the comma (`,`). The start and end positions of the `file1` changes are prefixed with a `-`, and the start and end positions of the `file2` changes are prefixed with a `+`.

4. This displays as a single section the differences between `file1` and `file2`. + indicates a line that was added in `file2` from `file1`, and − indicates a line that was deleted in `file2` from `file1`. Deleted lines and added lines constitute the changed portions.

Side-by-side format example

Output example

```
C:\USR\JP1\oscmd\bin>diff -y file1 file2
a                                     a    <--1.
b                                   | b1   <--1.
c                                     c    <--1.
d                                   <      <--1.
e                                     e    <--1.
                                    > f    <--1.
g                                   \ g    <--1.
```

Explanation

1. Each line of `file1` and `file2` is combined into one line and then output side by side.

   > indicates a line that is not in `file1` but was added in `file2`.

   < indicates a line that is in `file1` but was deleted from `file2`.

   `|` indicates a line in `file1` that has been changed in `file2`.

   `\` indicates a line in `file1` that has no end-of-line code.

   A line with no symbol has no differences.

Return codes

Return code	Meaning
`0`	The files are identical.
`1`	The files are different.
`2` or greater	Error termination

Notes

- If more than one of the `-c`, `-C`, `-q`, `-u`, `-U`, and `-y` options is specified, the one specified last takes effect.
- A file is considered to be binary if the first 8,192 bytes of the file includes any characters other than printable single-byte characters, spaces, tabs, backspaces, and multibyte characters.
- Files whose character encoding differs from the local character encoding are considered to be binary files.
- In Windows, input and output are performed in the binary mode for files and for the standard input and standard output. No conversion of end-of-line codes is performed.
- If − was specified for a path name and you interrupt execution of the `diff` command while the standard input is being read from the terminal or while comparison processing is underway, a temporary file with the name shown below might remain. Delete such a temporary file manually.

  In Windows:

  `diff.`*XXXXXX* (*XXXXXX*: any character string consisting of six characters)

  In UNIX:

  `diff`*ppppp*`.`*XXXXXXXX* (*ppppp*: process ID consisting of five or more digits; *XXXXXXXX*: any character string consisting of eight characters)

Usage examples

These usage examples illustrate the results of executing the `diff` command on files whose contents are shown below. Δ represents a space and ➔ represents a tab character.

- `abc.txt`

```
aaaaaaaaaaa

bbbbbbbb

ΔΔΔΔΔΔΔ
cccccccccccccccc

   ➔       ➔       ➔

ΔΔΔΔΔΔΔΔΔΔΔ
dddddddddddd
ΔΔΔ eeeeeeeeeeee
```

- `abcd.txt`

```
aaAAAAAaaaa

bbBBBbbb

ΔΔΔΔΔΔΔ
cccccccccccccccc
ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ
ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ
ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ
ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ
dddddddddddd
eeeeeeeeeeee
```

- `wxy.txt`

```
aaaaaaaaaaa

bbbbbbbb
xxxxxxxxxxxxxx

cccccccccccccccc
dddddddddddd
eeeeeeeeeeee
fffffffffffffff
ggggggggg
```

- `wxyz.txt`

```
aaaaaaaaaaa

bbbBBBbb
xxxxxxxxxxxxxx

cccccccccccccccc
dddddddddddd
fffffffffffffff
```

```
ggggggggg
hhhhhhhhhhhhhhhhh
```

The following examples illustrate the results of executing the command on the files shown above.

- Display the default with no options specified.

```
C:\TEMP>%ADSH_OSCMD_DIR%\diff abc.txt abcd.txt
1c1
< aaaaaaaaaaa
---
> aaAAAAAaaaa
3c3
< bbbbbbbb
---
> bbBBBbbb
7,10c7,10
<
<     →     →      →
<
<  ΔΔΔΔΔΔΔΔΔΔΔ
---
>  ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ
>  ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ
>  ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ
>  ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ
12c12
<  ΔΔΔ eeeeeeeeeeee
---
> eeeeeeeeeeee
```

- Specify the `-b` option to ignore differences in the number of spaces or tabs.

```
C:\TEMP>%ADSH_OSCMD_DIR%\diff -b abc.txt abcd.txt
1c1
< aaaaaaaaaaa
---
> aaAAAAAaaaa
3c3
< bbbbbbbb
---
> bbBBBbbb
12c12
<  ΔΔΔ eeeeeeeeeeee
---
> eeeeeeeeeeee
```

- Specify the `-i` option to compare without distinguishing between uppercase and lowercase letters.

```
C:\TEMP>%ADSH_OSCMD_DIR%\diff -i abc.txt abcd.txt
7,10c7,10
<
<     →     →      →
<
<  ΔΔΔΔΔΔΔΔΔΔΔ
---
>  ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ
>  ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ
>  ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ
>  ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ
12c12
<  ΔΔΔ eeeeeeeeeeee
---
> eeeeeeeeeeee
```

- Specify the `-s` option to report when the contents of the files are the same.

```
C:\TEMP>%ADSH_OSCMD_DIR%\diff -s abc.txt abc.txt
Files abc.txt and abc.txt are identical
```

- Specify the `-w` option to ignore all spaces and tabs in a line.

```
C:\TEMP>%ADSH_OSCMD_DIR%\diff -w abc.txt abcd.txt
1c1
< aaaaaaaaaaa
---
> aaAAAAAaaaa
3c3
< bbbbbbbb
---
> bbBBBbbb
```

- Specify the `-q` option to only report whether the files are different, without displaying the differences.

```
C:\TEMP>%ADSH_OSCMD_DIR%\diff -q abc.txt abcd.txt
Files abc.txt and abcd.txt differ
```

- Specify the `-c` option to indicate added, deleted, and changed lines with the symbols `+, -, and !.`

```
C:\TEMP>%ADSH_OSCMD_DIR%\diff -c ..\dir1\wxy.txt ..\dir1\wxyz.txt
*** wxy.txt     Thu May 12 20:17:54 2011
--- wxyz.txt    Thu May 12 20:18:29 2011
***************
*** 1,10 ****
  aaaaaaaaaaa

! bbbbbbbb
  xxxxxxxxxxxxxx

  cccccccccccccccc
  dddddddddddd
- eeeeeeeeeeee
  fffffffffffffff
  ggggggggg
--- 1,10 ----
  aaaaaaaaaaa

! bbbBBBbb
  xxxxxxxxxxxxxx

  cccccccccccccccc
  dddddddddddd
  fffffffffffffff
  ggggggggg
+ hhhhhhhhhhhhhhhhhh
```

- Specify the `-u` option to indicate added and deleted lines with the symbols `+` and `-`. The differences are displayed as a single section.

```
C:\TEMP>%ADSH_OSCMD_DIR%\diff -u ..\dir1\wxy.txt ..\dir1\wxyz.txt
--- wxy.txt     Thu May 12 20:17:54 2011
+++ wxyz.txt    Thu May 12 20:18:29 2011
@@ -1,10 +1,10 @@
 aaaaaaaaaaa

-bbbbbbbb
+bbbBBBbb
 xxxxxxxxxxxxxx

 cccccccccccccccc
 dddddddddddd
-eeeeeeeeeeee
 fffffffffffffff
 ggggggggg
+hhhhhhhhhhhhhhhhhh
```

- Specify the `-C` option to display a single line for context before and after a difference.

```
C:\TEMP>%ADSH_OSCMD_DIR%\diff -C1 wxy.txt wxyz.txt
*** wxy.txt     Thu May 12 20:17:54 2011
--- wxyz.txt    Thu May 12 20:18:29 2011
***************
*** 2,4 ****

! bbbbbbbb
  xxxxxxxxxxxxxx
--- 2,4 ----

! bbbBBBbb
  xxxxxxxxxxxxxx
***************
*** 7,10 ****
  dddddddddddd
- eeeeeeeeeeee
  fffffffffffffff
  ggggggggg
--- 7,10 ----
  dddddddddddd
  fffffffffffffff
  ggggggggg
+ hhhhhhhhhhhhhhhhhh
```

- Specify the `-U` option to indicate added and deleted lines with the symbols `+` and `-`. The differences are displayed as a single section, with a single line of context before and after a difference.

```
C:\TEMP>%ADSH_OSCMD_DIR%\diff -U1 wxy.txt wxyz.txt
--- wxy.txt     Thu May 12 20:17:54 2011
+++ wxyz.txt    Thu May 12 20:18:29 2011
@@ -2,3 +2,3 @@

-bbbbbbbb
+bbbBBBbb
 xxxxxxxxxxxxxx
@@ -7,4 +7,4 @@
```

```
 ddddddddddd
-eeeeeeeeeeee
 fffffffffffffff
 ggggggggg
+hhhhhhhhhhhhhhhhhh
```

- Specify the `-y` option and display line additions, deletions, and changes with the >, <, and `|` symbols.

```
C:\TEMP>%ADSH_OSCMD_DIR%\diff -y wxy.txt wxyz.txt
aaaaaaaaaaa                         aaaaaaaaaaa

bbbbbbbb                          | bbbBBBbb
xxxxxxxxxxxxxx                      xxxxxxxxxxxxxx

cccccccccccccccc                    cccccccccccccccc
ddddddddddd                         ddddddddddd
eeeeeeeeeeee                      <
fffffffffffffff                     fffffffffffffff
ggggggggg                           ggggggggg
                                  > hhhhhhhhhhhhhhhhhh
```

- Specify the `-y` option to display lines in side-by-side format and also specify the `--suppress-common-lines` option to suppress output of lines that have no differences.

```
C:\TEMP>%ADSH_OSCMD_DIR%diff -y --suppress-common-lines wxy.txt wxyz.txt
bbbbbbbb                          | bbbBBBbb
eeeeeeeeeeee                      <
                                  > hhhhhhhhhhhhhhhhhh
```

- Display the comparison source file name as a label specified with the `-L` option.

```
C:\TEMP>%ADSH_OSCMD_DIR%\diff -L name1 -c abc.txt abcd.txt
*** name1
--- abcd.txt    Thu May 12 20:36:44 2011
***************
*** 1,12 ****
! aaaaaaaaaaa

! bbbbbbbb

  ΔΔΔΔΔΔΔ
  cccccccccccccccc
!
!    →     →      →
!
! ΔΔΔΔΔΔΔΔΔΔΔ
  ddddddddddd
! ΔΔΔ eeeeeeeeeeee
--- 1,12 ----
! aaAAAAAaaaa

! bbBBBbbb

  ΔΔΔΔΔΔΔ
  cccccccccccccccc
! ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ
! ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ
! ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ
!  ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ
  dddddddddddd
! eeeeeeeeeeee
```

- Compare binary files without specifying the `-a` option.

```
C:\TEMP>%ADSH_OSCMD_DIR%\diff binaryfile1 binaryfile2
Binary files binaryfile1 and binaryfile2 differ
```

- Display an option error message.

  This message might vary depending on the platform on which the command is executed. The following shows an example in Windows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\diff -z
diff: illegal option -- z
usage: diff [-abisw] [-c[number] | -C number | -q | -u[number] | -U 
number] | -y [-W columns] [--suppress-common-lines]]
           [-L label] file1 file2
       diff [-abirsw] [-c[number] | -C number | -q | -u[number] | -U 
number] | -y [-W columns] [--suppress-common-lines]]
           [-L label] dir1 dir2
```

- Display an error message if a file does not exist.

```
C:\TEMP>%ADSH_OSCMD_DIR%\diff file99 file123
diff: file99: No such file or directory
```

dirname command (retrieves character strings for directory path names from path names)

Format

```
dirname[character-string]
```

Description

This command retrieves a directory path name excluding any file name from a character string that satisfies the file path naming conventions, and then outputs the result to the standard output.

The rules for retrieving directory path names are as follows:

- The command retrieves all elements separated by the directory separator from the specified character string except for the right-most element and the directory separator that immediately precedes that element. The command retrieves a directory path name as is even if it contains multiple consecutive separators.

  In UNIX, the forward slash (`/`) is treated as the directory separator. In Windows, both the forward slash (`/`) and the backslash (`\`) are treated as directory separators.
- If the specified character string ends with a directory separator, the command retrieves all elements except that last directory separator and the right-most element.
- If the specified character string contains no directory separator or no character string is specified, the command outputs a period (`.`) meaning the current directory.
- If the specified character string consists of only directory separators, the command retrieves the directory separators.

- In Windows, if the specified character string begins with an alphabetic character that is immediately followed by a colon (`:`), that alphabetic character is treated as a drive letter. The colon following the drive letter is also treated as a separator for elements.
- In Windows, the root directory path is retrieved as described in the following regardless of the above rules:

First character string of the path name	Result retrieved by the dirname command
drive-letter`:\`	*drive-letter*`:\`
drive-letter`:`	*drive-letter*`:`
`\\`*server-name* (UNC name specification)	`\\`
`\\?` (service function disabling specification)	`\\`
`\\.` (specification of 10th or subsequent device name)	`\\`

The following table shows examples of `dirname` values and the retrieval results:

dirname command value	Retrieval result
`C:\`	`C:\`
`C:`	`C:`
`\\server01\`	`\\`
`\\server01`	`\\`
`\\?\`	`\\`
`\\?`	`\\`
`\\.\`	`\\`
`\\.`	`\\`
`\\`	`\\`
`C:file001.txt`	`C:`
`C:\file001.txt`	`C:\`

Arguments

character-string

Specifies a file path name.

Return codes

Return code	Meaning
`0`	Normal termination
`1`	Error termination

Notes

- This command has no options. If an option is specified in the argument, the command assumes that the specified option is the character string used to retrieve directory path names.

Examples

- Retrieve directory path names from path names:

  Example 1:

```
C:\TEMP>%ADSH_OSCMD_DIR%\dirname E:\dir001\file01.txt
E:\dir001
```

Example 2:

```
C:\TEMP>%ADSH_OSCMD_DIR%\dirname /dir001
/
```

Example 3:

```
C:\TEMP>%ADSH_OSCMD_DIR%\dirname .\file01.txt
.
```

Example 4:

```
C:\TEMP>%ADSH_OSCMD_DIR%\dirname E:\dir001\dir002\
E:\dir001
```

Example 5:

```
C:\TEMP>%ADSH_OSCMD_DIR%\dirname E:\
E:\
```

Example 6:

```
C:\TEMP>%ADSH_OSCMD_DIR%\dirname \\server01\
\\
```

Example 7:

```
C:\TEMP>%ADSH_OSCMD_DIR%\dirname \\server01\com
\\server01
```

Example 8:

```
C:\TEMP>%ADSH_OSCMD_DIR%\dirname \\
\\
```

Example 9:

```
C:\TEMP>%ADSH_OSCMD_DIR%\dirname "C:\Documents and Settings\User01\My
Documents"
C:\Documents and Settings\User01
```

Example 10:

```
C:\TEMP>%ADSH_OSCMD_DIR%\dirname C:file01.txt
C:
```

Example 11:

```
C:\TEMP>%ADSH_OSCMD_DIR%\dirname C:\file01.txt
C:\
```

Example 12:

```
C:\TEMP>%ADSH_OSCMD_DIR%\dirname file01.txt
.
```

- Display option error messages:

  This message might differ depending on the platform used to execute the command. This example is for Windows:

```
C:\> dirname /a/b /c/d
usage: dirname [string]
```

egrep command (searches for characters in files)

Format

```
egrep[-a][-b][-c][-E][-h][-I][-i][-L][-l][-n]
     [-q][-R][-r][-s][-U][-v][-w][-x]
     [-A number] [-B number] [-C[number]]
     [-e pattern] [-f pattern-file-path-name] [pattern] [path-name ...]
```

Description

This command searches files for specified patterns. The patterns to be retrieved are assumed to be extended regular expressions. The behavior of the `egrep` command is the same as that of the `grep` command with the `-E` option specified.

Arguments

-a

Specifies that all files are to be handled as ASCII text files.

-b

Specifies that the offset in bytes is to be displayed at the beginning of each matching line.

-c

Specifies that only the number of lines selected is to be output to the standard output.

-E

Specifies that the value specified in *pattern* is to be handled as an extended regular expression (default value).

-h

Specifies that no file name is to be displayed at the beginning of each output line when either of the following conditions is satisfied:

- The `-R` or `-r` option is specified.
- Multiple path names are specified as being subject to search.

-I

Specifies that binary files are to be ignored.

-i

Specifies that uppercase letters are not to be distinguished from lowercase letters (and vice versa).

-L

Specifies that only the names of files that do not contain a match for the value specified in *pattern* are to be output to the standard output. If the `-L` and `-l` options are both specified, the one specified last takes effect.

-l

Specifies that only the names of files that contain a match for the value specified in *pattern* are to be output to the standard output. If the `-L` and `-l` options are both specified, the one specified last takes effect.

-n

Specifies that its relative line number in the file is to be output at the beginning of each output line. This specification is ignored when any of the `-c`, `-L`, `-l`, and `-q` options is specified.

-q

Specifies that nothing is to be output to the standard output. The command returns only the return code.

-R|-r

Specifies that directories are to be searched recursively.

If the `-L`, `-l`, and `-q` options are all omitted, the file name is output at the beginning of each output line.

-s

Specifies that unreadable or nonexistent files are to be ignored, and output of error messages related to unreadable or nonexistent files is to be suppressed.

-U

Specifies that binary files are to be searched but not output.

-v

Specifies that lines that do not contain a match for the value in *pattern* are to be output.

-w

Specifies that only lines that contain the specified character string as a whole word are to be output.

A word is a character string that consists of alphanumeric characters and the underscore (_). Words must be delimited by the space, any other non-word character, or the beginning or end of the line.

-x

Specifies that the specified character string is to be compared to each line in the file, and a line is to be output only if the entire line constitutes an exact match.

-A *number*

Specifies that as many lines as specified that follow a line matching *pattern* are to be output.

-B *number*

Specifies that as many lines as specified that precede a line matching *pattern* are to be output.

-C[*number***]**

Specifies that as many lines as specified that precede and follow a line matching *pattern* are to be output. If no value is specified (*number* is omitted), two lines preceding and following a line matching *pattern* are output. This would be the equivalent of specifying `-A 2 -B 2`.

If you specify *number* in the `-C` option, do not specify any spaces between `-C` and *number*.

-e *pattern*

Used to specify a pattern that begins with a hyphen (`-`).

-f *pattern-file-path-name*

Specifies the path name for a file that contains patterns to be searched for. The specified file specifies one line per pattern. If an empty file is specified (a file in which no patterns are specified), there will be nothing to search for and no matches will be found.

pattern

Specifies a pattern to be searched for.

path-name **...**

Specifies a path name that is to be searched. Multiple path names can be specified. If no path name is specified, the contents of the standard input are searched. If you specify a directory name, you must also specify the `-R` or `-r` option.

If the `-L`, `-l`, and `-q` options are all omitted, the file name is output at the beginning of each output line.

Return code

Return code	Meaning
`0`	Normal termination. • A line was found that contains a pattern being searched for. • Or, if the `-v` option is specified, a line was found that does not contain the pattern being searched for.
`1`	Normal termination. • No lines contain the pattern being searched for. • Or, if the `-v` option is specified, all lines contain the pattern being searched for.
`2` or greater	Error termination

Notes

- In Windows, symbolic links are not output.
- If the first 8,192 bytes of the file consist of data that is other than printable single-byte characters, spaces, tabs, backspaces, and multibyte characters, the file is considered to be a binary file.
- To execute `grep` from the command prompt in Windows, you must enclose the pattern in double quotation marks (").
- Files whose character encoding differs from the local character encoding are considered to be binary files.
- In Windows, input and output are performed in the binary mode for files and for the standard input and the standard output. No conversion of end-of-line codes is performed.
- To search for any of the metacharacters listed below that are used in regular expressions, specify an escape character (\) immediately before the metacharacter.

  `+, ?, |, (, ), {, }`

Usage examples

These usage examples show searches using extended regular expressions. For examples of the options, see the usage examples for the `grep` command.

- Search for lines that contain character string `AB` or `AD` by using `|` that indicates an extended regular expression. The input file is `file01.txt`.

  Contents of `file01.txt`:

  ```
  AA
  AB
  AC
  AD
  AB|AD
  ```

  The results of executing the command are as follows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\egrep "AB|AD" file01.txt
AB
AD
AB|AD
```

- Search for lines that contain character string `AB|AD`. Because `|` is treated as an extended regular expression, specify an escape character (\) immediately before `|`. The input file is `file01.txt`.

  Contents of `file01.txt`:

  ```
  AA
  AB
  AC
  AD
  AB|AD
  ```

  The results of executing the command are as follows:

  ```
  C:\TEMP>%ADSH_OSCMD_DIR%\egrep "AB\|AD" file01.txt
  AB|AD
  ```

expand command (replaces tab characters with spaces)

Format

```
expand[-tab-stop-list)][-t tab-stop-list][path-name ...]
```

Description

This command replaces the tab character with spaces in a line in which tab stops are set and then outputs the result to the standard output. If the tab character is followed by the backspace character, the column width for tabs is reduced in the output results.

One record delimited by an end-of-line code in the input file is treated as one line. In Windows, `[CR]+[LF]` or `[LF]` is treated as the end-of-line code; in UNIX, `[LF]` is treated as the end-of-line code. In UNIX, if a record in the input file is delimited by `[CR]+[LF]`, the output result after conversion will contain `[CR]`.

Arguments

-tab-stop-list

The function of this argument is the same as for **-t tab-stop-list**.

-tab-stop-list can be specified together with **-t tab-stop-list**, in which case all the specifications are effective.

-t tab-stop-list

--tabs=tab-stop-list

Specifies a list of the locations for tab stops, each as `1` or a greater integer. If this option is omitted, default tab stop `8` is used, which is the same as when one tab stop location is specified.

The following explains the command processing when one tab stop location is specified in the tab-stop list and when multiple tab stop locations are specified in the tab-stop list:

When one tab stop location is specified:

The command uses the specified value as the character spacing between equidistant tab stops.

When the tab character contained in one tab stop is replaced with spaces, the numbers of spaces between tab stops are adjusted to obtain the specified character spacing.

When multiple tab stop locations are specified:

The command uses each specified value as the column location for a tab stop. The column locations begin from 0.

The following explains how to specify multiple tab stop locations.

- Specify multiple tab stop locations in the tab-stop list separating them with the comma or space.

  To use the space as the separator, enclose it in double-quotation marks (`"`).

  When the tab characters contained in the tab stops are replaced with spaces, the numbers of spaces between tab stops are adjusted to the specified column locations. If it is necessary to set more tab stops than specified in the tab-stop list, the tab character is replaced by one space.
- Specify one tab stop location in the tab-stop list and specify the option multiple times.
- Combine both of the above specification methods.

Tab stops are set for each input line from the first value specified. Specify tab stop locations in ascending order in the entire arguments.

path-name

Specifies the path name of the file in which tab characters are to be replaced with spaces. If no path name or `-` is specified, the path name is loaded from the standard input.

If multiple files are specified and one of the files results in an open error, the command issues an error message and continues processing.

Return codes

Return code	Meaning
`0`	Normal termination
`1`	Error termination • Path name contained a file that could not be opened.
`2`	Error termination (excluding the case resulting in termination code `1`)

Notes

- The `expand` command expects text files. Input from binary files and output of binary data are not guaranteed to work.

Examples

Specifying one tab stop location

If one tab stop is specified, the command uses the specified value as the character spacing between tab stops.

The following example specifies `6` for the tab stop for file `file1`:

Contents of file1

In the following file, ➔ indicates the tab character.

```
----+----+----+----+----+----+-----
a001 ➔ a002 ➔ a003
```

```
b001  →  b002     →    b003
c001  →  c2    →  c03
```

Command execution example

Specify 6 for the tab stop and `file1` as the file:

```
$ expand -t 6 file1
```

Execution results

```
----+----+----+----+----+----+-----
a001  a002  a003
b001  b002        b003
c001  c2    c03
```

On the first line, tab stops are set at the specified interval of 6.

On the second line, two tab stops are set between `b002` and `b003` because there are two tab characters.

On the third line, `c001`, `c2`, and `c03` that have different numbers of characters are set at a tab stop interval of 6.

Specifying multiple tab stop locations by using a tab-stop list

When multiple tab stops are specified, the tab stops are set in the order specified. If it is necessary to set more tab stops than specified in the tab-stop list, the tab character is replaced by one space.

Contents of file1

In the following file, → indicates the tab character.

```
----+----+----+----+----+----+-----
a001  →  a002  →  a003  →  a004
```

Command execution example 1

Specify `6,16` as the tab-stop list and file `file1`:

```
$ expand -t 6,16 file1
```

Execution results 1

```
----+----+----+----+----+----+-----
a001  a002      a003 a004
```

Character strings are output with the specified number of spaces placed.

Because the first value specified is 6, the command first outputs `a001`, places spaces through column 6, then outputs `a002`.

Similarly, the command outputs `a002`, places spaces through column 16, then outputs `a003`.

Because there are no more values in the tab-stop list, the command places one space, and then outputs `a004`.

Command execution example 2

Specify `2,16` as the tab-stop list and file `file1`:

```
$ expand -t "2 16" file1
```

Execution result 2

```
----+----+----+----+----+----+-----
a001            a002 a003 a004
```

Because the first value specified is 2, the command outputs `a001` and then attempts to place spaces through column 2, but the current location is already beyond column 2 because `a001` has been output. Therefore, the command ignores value 2, places spaces through column 16, which is the next value specified, and then outputs a002.

Because there are no more values in the tab-stop list, the command places one space before each of the subsequent character strings.

Specifying multiple tab stops in options

There are several ways to specify multiple tab stops by combining [-t *tab-stop list*] and [-*tab-stop list*]. All the following examples specify 2 and 16 as the tab stops and their results are the same:

```
$ expand -t 2 -t 16 file1
$ expand -t 2 -16 file1
$ expand -2 -t 16 file1
$ expand -2 -16 file1
$ expand -t 2,16 file1
$ expand -2,16 file1
```

Entering the backspace character

If the tab character is followed by the backspace character, the tab's column width is reduced.

Contents of file1

In the following file, ➔ indicates the tab character.

There is a backspace character (^H) immediately before a003:

```
----+----+----+----+----+----+-----
a001 ➔ a002 ➔ ^Ha003➔a004
b001 ➔ b002 ➔ b003 ➔ b004
```

Command execution example

Execute the command using default tab stop 8:

```
$ expand file1
```

Execution results

```
----+----+----+----+----+----+-----
a001    a002   a003     a004
b001    b002    b003    b004
```

Because there is a backspace character (^H) immediately before a003, the command saves the results during output processing. As a result, the output location of a003 is column 16, not column 17.

Input from the standard input

If no path name is specified or if - is specified, files are input from the standard input.

Contents of file1

In the following file, ➔ indicates the tab character.

```
----+----+----+----+----+----+-----
a001 ➔ a002 ➔ a003 ➔ a004
```

Command execution example

Input file1 from the standard input:

```
$ expand < file1
```

or

```
$ expand - < file1
```

Execution results

```
----+----+----+----+----+----+-----
a001    a002    a003    a004
```

- Display option error massages:

  This message might vary depending on the platform used to execute the command. The following is an example for Windows:

```
C:\> expand -z
expand: illegal option -- z
usage: expand [-tablist] [-t tablist] [file ...]
```

expr command (evaluates an expression)

Format

```
expr expression
```

Description

This command evaluates an expression and sends the results to the standard output. All elements of the expression are specified as separate arguments.

An expression is specified as a combination of numeric values, character strings, variables, expressions, and their operators. The evaluation of the expression is retained as a character string or integer.

Arguments

expression

Specifies the expression to be evaluated. The operators are shown below in increasing order of priority. Operators shown enclosed in curly brackets (`{ }`) and separated by the comma have the same priority. When an argument is invalid, the `expr` command outputs an error message and returns `2` as the return code. Expressions are denoted by *expr1* and *expr2*.

expr1 `|` *expr2*

If *expr1* is not an empty character string or zero, the evaluation of *expr1* is returned. If *expr1* is an empty character string or zero, the evaluation of *expr2* is returned. If *expr2* is also a null character string, the null character string is returned.

expr1 `&` *expr2*

If neither expression evaluates to an empty character string or zero, the evaluation of *expr1* is returned; otherwise, `0` is returned.

expr1 `{=, >, >=, <, <=, !=}` *expr2*

If both expressions evaluate to an integer, the result of comparing the integers is returned; otherwise, the result of comparing the character strings in the collating sequence defined locally is returned. The result is 1 if the specified relationship is true, and 0 if it is false.

- =: The values on the left and right are equal.

- >: The value on the left is greater than the value on the right.
- >=: The value on the left is greater than or equal to the value on the right.
- <: The value on the left is less than the value on the right.
- <=: The value on the left is less than or equal to the value on the right.
- !=: The values on the left and right are not equal.

expr1 `{+, -}` *expr2*

If both expressions evaluate to integer values, the result of the addition or subtraction is returned.

If the value is not an integer, an error message (`expr: non-numeric argument`) is output.

- `+`: Add
- `-`: Subtract

expr1 `{*, /, %}` *expr2*

If both expressions evaluate to integer values, the result of the multiplication, division, or modulo operation is returned. If the values are not integers, an error message (`expr: non-numeric argument`) is output. If the divisor is zero, an error message (`expr: division by zero`) is output.

- `*`: Multiply
- `/`: Divide
- `%`: Modulo

expr1 `:` *expr2*

Evaluates whether *expr2* matches *expr1*.

The expression *expr2* is specified as a regular expression. The regular expression is treated as if ^ were added at the beginning of the string.

- If a tagged regular expression is specified in *expr*2 (if *expr2* matches *expr1*), the first character string that matched the tagged regular expression is returned.
- If a tagged regular expression is not specified in *expr2* (if *expr2* matches *expr1*), the number of matched characters is returned.
- If *expr2* does not match *expr1*, and a regular expression is used in *expr2*, the null character is returned. If a regular expression is not used in *expr2*, `0` is returned.
- If the specification of *expr2* matches the null character, `0` is returned. To determine if *expr1* is the null character, you must evaluate it by assigning the same character to both *expr1* and *expr2*. That is, instead of `expr '' : '$'`, you must use `expr X'' : 'X$'`, or a similar variation.

`length` *character-string*

Returns the length of the specified character string. For details about the `ADSH_CMDEXPR_LENGTH` environment variable, see *2.5 Specifying environment variables*.

- If the `ADSH_CMDEXPR_LENGTH=b` environment variable is set, the command treats `length` as an operator and returns the length (in bytes) of the character string that follows.
- If the `ADSH_CMDEXPR_LENGTH=c` environment variable is set, the command treats `length` as an operator and returns the length (in characters) of the character string that follows.
- If the `ADSH_CMDEXPR_LENGTH` environment variable is not set or a value other than `b` or `c` is set in the environment variable, the command does not treat `length` as an operator.

You can specify an expression in the `length` operator. If you specify an expression, enclose the entire expression is parentheses (`()`).

Return codes

Return code	Meaning
0	Normal termination. The expression is not an empty character string or 0.
1	Normal termination. The expression is an empty character string or 0.
2	Error termination. The expression is not valid.
3 or greater	Error termination • An error, such as a memory shortage, occurred.

Notes

- Integer values are stored in the range of -2147483648 to 2147483647. If you specify a value outside this range, any overflow into the 32-bit binary digit position will be ignored.
- Characters specified in the operators and parentheses, because they include characters that are interpreted by the shell, must be properly escaped. Because the entire expression is interpreted as a character string when it is enclosed in double quotation marks ("), each individual operator must be enclosed in double quotation marks (").
- This command does not accept options. If you specify an option as an argument, the option is interpreted as an expression.

Usage examples

- Perform a calculation using variable a and variable b.

```
$ a=2
$ b=3
$ x=`expr \( $a + $b \) \* 10`
$ echo $?
0
$ echo $x
50
$
```

- Evaluate variable a | variable b.

```
$ a=""
$ b="abcdef"
$ expr "$a" \| "$b"
abcdef
$
```

- Cut the file name from the path name, without the extension.

```
$ a='d:\jp1as\test.txt'
$ expr $a : '.*\\\(.*\)\.'
test
$
```

- Determine whether a variable includes numbers. The return will be 0 if there are no numbers.

```
$ a='abcde12345kl'
$ b='abcdefg'
$ expr $a : '.*[0-9].*'
12
$ expr $b : '.*[0-9].*'
```

```
0
$
```

- Return the length of character string テスト文字列 in bytes.

```
$ export ADSH_CMDEXPR_LENGTH=b
$ echo $LANG
ja_JP.UTF-8
$ expr length "テスト文字列"
18
```

- Return the length of character string テスト文字列 in characters.

```
$ export ADSH_CMDEXPR_LENGTH=c
$ echo $LANG
ja_JP.UTF-8
$ expr length "テスト文字列"
6
```

- Return a value that is obtained by adding 2 to the length of character string `teststring` (in bytes).

```
$ export ADSH_CMDEXPR_LENGTH=b
$ echo $LANG
ja_JP.UTF-8
$ expr length teststring + 2
12
```

find command (searches for files in directories)

Format

```
find [-d] [-H] [-h] [-L] path-name [...] [search-pattern]
```

Description

This command specifies paths where you want to conduct a search and then follows the directory hierarchies searching for files. You can specify search conditions and how files that are found are to be handled.

Arguments

You specify options, path names where the search is to be conducted, and a search pattern. The path names where the search is to be conducted are specified as arguments to the `find` command.

Each option is specified as a one-letter option name preceded by a hyphen (`-`).

-d

Specifies that files inside the deepest directory are searched for first, and processing proceeds from there up to the directory specified in *path-name*.

-H

In UNIX, specifies that if a path name specified as an argument is a symbolic link, it is to be treated as if the link target were specified. If the link target does not exist, the symbolic link itself is processed. Symbolic links encountered during the search are not followed. If more than one of the `-H`, `-h`, and `-L` options is specified, the one specified last takes effect.

In Windows, the `-H` option is ignored.

-h

In UNIX, specifies that anytime a symbolic link is encountered, it is to be followed and processing is to continue. If the link target does not exist, the symbolic link itself is processed. If more than one of the `-H`, `-h`, and `-L` options is specified, the one specified last takes effect.

In Windows, the `-h` option is ignored.

-L

In UNIX, specifies that anytime a symbolic link is encountered, it is to be followed and processing is to continue. If the link target does not exist, the symbolic link itself is processed. If more than one of the `-H`, `-h`, and `-L` options is specified, the one specified last takes effect.

In Windows, the `-L` option is ignored.

path-name

Specifies a path name.

search-pattern

Specifies a search pattern (expression). A search pattern consists of primaries and operators.

- Primaries

`-amin` *time-difference*

In UNIX, specifies an evaluation that is to be performed. The evaluation is true when the number of minutes specified in *time-difference* is valid as the difference between the date and time of the last access to the file or directory and the date and time at which execution of the `find` command starts. A difference in date and time that is less than one minute is rounded up to the nearest minute.

The specified *time-difference* is interpreted as follows depending on whether no sign is specified or a numeric value with a + or - sign is specified:

- When a sign is not specified: Specified time difference
- `+` is specified before a numeric value: Greater than the specified value
- `-` is specified before a numeric value: Less than the specified value

The specified value must not exceed `2147483647` (`0x7fffffff`). If you specify a greater value, `2147483647` is assumed.

If you specify a non-numeric value, an error message (`find:` *primary*`:` *specified-character-string*`: illegal numeric value`) is output.

If *time-difference* is omitted, an error message (`find:` *primary*`: requires additional arguments`) is output.

In Windows, an error occurs if this primary is specified.

`-anewer` *path-name*

In UNIX, specifies an evaluation that is to be performed. The evaluation is true when the date and time of the most recent access to the file or directory is more recent than the date and time of the most recent access to *path-name*.

In Windows, an error occurs if this primary is specified.

`-atime` *time-difference*

In UNIX, specifies an evaluation that is to be performed. The evaluation is true when the number of days specified in *time-difference* is valid as the difference between the date and time of the last access to the file or directory and the date and time at which execution of the `find` command starts. A difference in date and time that is less than one day is rounded up to the nearest day.

The specified *time-difference* is interpreted as follows depending on whether no sign is specified or a numeric value with a + or - sign is specified:

- When a sign is not specified: Specified time difference
- + is specified before a numeric value: Greater than the specified value
- − is specified before a numeric value: Less than the specified value

The specified value must not exceed `2147483647` (`0x7fffffff`). If you specify a greater value, `2147483647` is assumed.

If you specify a non-numeric value, an error message (`find:` *primary*`:` *specified-character-string*`: illegal numeric value`) is output.

If *time-difference* is omitted, an error message (`find:` *primary*`: requires additional arguments`) is output.

In Windows, an error occurs if this primary is specified.

`-cmin` *time-difference*

In UNIX, specifies an evaluation that is to be performed. The evaluation is true when the number of minutes specified in *time-difference* is valid as the difference between the last date and time the file information was changed (for example, the date and time the file was written to, or the date and time the owner, group, link count, or mode was changed) and the date and time at which execution of the `find` command starts. A difference in date and time that is less than one minute is rounded up to the nearest minute.

The specified *time-difference* is interpreted as follows depending on whether no sign is specified or a numeric value with a + or - sign is specified:

- When a sign is not specified: Specified time difference
- + is specified before a numeric value: Greater than the specified value
- − is specified before a numeric value: Less than the specified value

The specified value must not exceed `2147483647` (`0x7fffffff`). If you specify a greater value, `2147483647` is assumed.

If you specify a non-numeric value, an error message (`find:` *primary*`:` *specified-character-string*`: illegal numeric value`) is output.

If *time-difference* is omitted, an error message (`find:` *primary*`: requires additional arguments`) is output.

In Windows, an error occurs if this primary is specified.

`-cnewer` *path-name*

In UNIX, specifies an evaluation that is to be performed. The evaluation is true when the last date and time the file information was changed (for example, the date and time the file was written to, or the date and time the owner, group, link count, or mode was changed) is more recent than the last time the file specified in *path-name* was changed.

In Windows, an error occurs if this primary is specified.

`-ctime` *time-difference*

In UNIX, specifies an evaluation that is to be performed. The evaluation is true when the number of days specified in *time-difference* is valid as the difference between the last date and time the file information was changed (for example, the date and time the file was written to, or the date and time the owner, group, link count, or mode was changed) and the date and time at which execution of the `find` command starts. A difference in date and time that is less than one day is rounded up to the nearest day.

The specified *time-difference* is interpreted as follows depending on whether no sign is specified or a numeric value with a + or - sign is specified:

- When a sign is not specified: Specified time difference
- + is specified before a numeric value: Greater than the specified value
- - is specified before a numeric value: Less than the specified value

You can specify *time-difference* without a sign or with a leading + or - sign. When + is specified, the check determines whether the actual time is greater than the specified time difference value. When - is specified, the check determines whether the actual time is less than the specified time difference value. When no sign is specified, the check determines whether they are the same. The specified value must not exceed `2147483647` (`0x7fffffff`). If you specify a greater value, `2147483647` is assumed.

If you specify a non-numeric value, an error message (`find:` *primary*`:` *specified-character-string*`: illegal numeric value`) is output.

If *time-difference* is omitted, an error message (`find:` *primary*`: requires additional arguments`) is output.

In Windows, an error occurs if this primary is specified.

`-depth`

Specifies that an evaluation is to be performed. The evaluation is true when the directories at the deepest level are searched first, and the files within a directory are processed before the directory itself. This evaluation is always true.

`-empty`

Specifies that an evaluation is to be performed. The evaluation is true when the file or directory is empty.

`-exec` *command-line* `;`

Specifies a command line command for processing the files and directories being searched that is to be evaluated.

- Depending on the shell in which the `find` command is executed, characters such as the asterisk (`*`) and semicolon (`;`) might be expanded by the shell, and so must be enclosed in double quotation marks (`"`) or single quotation marks (`'`), or must be prefixed with the escape character (`\`).
- The *command-line* specification must terminate with a semicolon (`;`).
- The program specified in *command-line* is launched with the directory from which `find` was launched set as the current directory.
- If you specify curly brackets (`{ }`) in *command-line*, they are replaced by the path name of the file or directory being searched. Their replacement is an absolute path name if you specified an absolute path name as the path for conducting the search or a relative path name if you specified a relative path name as the path for conducting the search.
- The evaluation is true if the program specified in *command-line* terminates with a return code of `0`.

`-follow`

This evaluation is always true.

In UNIX, specifies an evaluation that is to be performed. The evaluation is true if whenever a symbolic link is encountered it is to be followed and processing is to continue. If the link target does not exist, the symbolic link itself is processed.

`-group` *group-name*

In Windows, this evaluation is always false.

In UNIX, specifies an evaluation that is to be performed. The evaluation is true when the file belongs to the group whose name is specified. If you specify a number for the group name and no such group name exists, the specified value is interpreted as a group ID.

`-iname` *pattern*

See the description of the `-name` option. This option functions identically except that it does not distinguish between uppercase and lowercase letters.

`-inum` *number*

In Windows, this evaluation is always false.

In UNIX, specifies an evaluation that is to be performed. The evaluation is true when the relationship between the inode number of the file and the specified value is valid.

- You can specify *number* without a sign or with a leading + or - sign. When + is specified, the check determines whether the inode number is greater than the specified value. When - is specified, the check determines whether the inode number is less than the specified value. When no sign is specified, the check determines whether they are the same. The specified value must not exceed `2147483647` (`0x7fffffff`). If you specify a greater value, `2147483647` is assumed.
- If you specify a non-numeric value, an error message (`find:` *primary*`:` *specified-character-string*`: illegal numeric value`) is output.
- If *number* is omitted, an error message (`find:` *primary*`: requires additional arguments`) is output.

`-links` *link-count*

In Windows, this evaluation is always false.

In UNIX, specifies an evaluation that is to be performed. The evaluation is true when the relationship between the link count for the file and the specified value is valid.

- You can specify *link-count* without a sign or with a leading + or - sign. When + is specified, the check determines whether the link count for the file is greater than the specified value. When - is specified, the check determines whether the link count for the file is less than the specified value. When no sign is specified, the check determines whether they are the same. The specified value must not exceed `2147483647` (`0x7fffffff`). If you specify a greater value, `2147483647` is assumed.
- If you specify a non-numeric value, an error message (`find:` *primary*`:` *specified-character-string*`: illegal numeric value`) is output.
- If *link-count* is omitted, an error message (`find:` *primary*`: requires additional arguments`) is output.

`-ls`

In Windows, specifies that the file permissions, owner name, size (in bytes), most recent modification date and time, and path name are to be output to the standard output. This evaluation is always true.

In UNIX, specifies that the inode number, size (in units of 512 bytes), file permissions, hard links count, owner name, group, size (in bytes), most recent modification date and time, and path name are to be output to the standard output.

This evaluation is always true. If the file is a special file, the major and minor numbers are output instead of the size (in bytes). If the file is a symbolic link, `->` is displayed followed by the path name of the link target.

`-maxdepth` *depth*

Specifies an evaluation that is to be performed. The evaluation is true when the depth of the directory currently being searched is less than or equal to the specified value. The depth of a specified directory initially is `1`.

- The specified value must be in the range of `0` through `32767`. If `0` is specified, the search target becomes the directory only (the files stored in the directory are not searched).
- If you specify a value greater than the maximum specifiable value, an error message (`find:` *specified-value*`: maxdepth value too large`) is output.

- If you specify a non-numeric value, an error message (`find:` *specified-character-string*`:` *primary*`: value invalid`) is output.
- If *depth* is omitted, an error message (`find:` *primary*`: requires additional arguments`) is output.

`-mindepth` *depth*

Specifies an evaluation that is to be performed. The evaluation is true when the depth of the directory currently being searched is greater than or equal to the specified value.

- The specified value must be in the range of `0` through `32767`.
- No error occurs is the specified value is greater than the maximum specifiable value.
- If you specify a non-numeric value for *depth*, `0` is assumed.
- If *depth* is omitted, an error message (`find:` *primary*`: requires additional arguments`) is output.

`-mmin` *time-difference*

Specifies an evaluation that is to be performed. The evaluation is true when the number of minutes specified in *time-difference* is valid as the difference between the last date and time the file or directory was modified and the time at which execution of the `find` command starts. A difference in date and time that is less than one minute is rounded up to the nearest minute.

- You can specify *time-difference* without a sign or with a leading + or − sign. When + is specified, the check determines whether the actual time is greater than the specified time difference value. When − is specified, the check determines whether the actual time is less than the specified time difference value. When no sign is specified, the check determines whether they are the same. The specified value must not exceed `2147483647` (`0x7fffffff`). If you specify a greater value `2147483647` is assumed.
- If you specify a non-numeric value, an error message (`find:` *primary*`:` *specified-character-string*`: illegal numeric value`) is output.
- If *time-difference* is omitted, an error message (`find:` *primary*`: requires additional arguments`) is output.

`-mtime` *time-difference*

Specifies an evaluation that is to be performed. The evaluation is true when the number of days specified in *time-difference* is valid as the difference between the time of the last modification of the file or directory and the time at which execution of the `find` command starts. A time difference that is less than one day is rounded up to the nearest day.

- You can specify *time-difference* without a sign or with a leading + or − sign. When + is specified, the check determines whether the actual time is greater than the specified time difference value. When − is specified, the check determines whether the actual time is less than the specified time difference value. When no sign is specified, the check determines whether they are the same. The specified value must not exceed `2147483647` (`0x7fffffff`). If you specify a greater value, `2147483647` is assumed.
- If you specify a non-numeric value, an error message (`find:` *primary*`:` *specified-character-string*`: illegal numeric value`) is output.
- If *time-difference* is omitted, an error message (`find:` *primary*`: requires additional arguments`) is output.

`-mount`

This is always true. In UNIX, this argument specifies that a directory whose device number differs from that of the directory in which the search began is not to be searched.

-name *pattern*

Specifies an evaluation that is to be performed. The evaluation is true when the name of the file or directory to be searched for matches the value specified in *pattern*.

The *pattern* is specified as a combination of characters and wildcards. You can use the escape character (\) to specify as part of the pattern value a character that would otherwise be a wildcard. If the escape character (\) is specified before a character that is not a wildcard characters, the \ is ignored.

The following table shows the characters that can be used as wildcards:

Wildcard	Meaning
?	Matches any single character.
*	Matches a character string of zero or more characters.
[ ]	Matches any single character in the character string enclosed in [ ]. If there is an ! or a ^ at the beginning of the character string enclosed in [ ], there is a match if none of the characters in [ ] is found. If two characters are separated by a hyphen (-), the match is of any character between those two characters (including the two characters themselves).

The following examples illustrate the use of the [ ] wildcard:

Example	Meaning
[!abc]	Matches any character other than a, b, or c.
[0-9]	Matches any character from 0 through 9.
[a-z]	Matches any lowercase letter.
[A-Z]	Matches any uppercase letter.
[0-9a-zA-Z]	Matches any alphanumeric character.

-newer *path-name*

Specifies an evaluation that is to be performed. The evaluation is true when the current file or directory is newer than the time at which *path-name* was last modified.

-nogroup

In Windows, this evaluation is always false.

In UNIX, specifies an evaluation that is to be performed. The evaluation is true when the current file belongs to a non-existent group.

-nouser

In Windows, this evaluation is always false.

In UNIX, specifies an evaluation that is to be performed. The evaluation is true when the owner of the current file is a non-existent user.

-ok *command-line* ;

Specifies a command line command for processing the files and directories being searched that is to be evaluated.

- Depending on the shell in which the find command is executed, characters such as the asterisk (*) and semicolon (;) might be expanded by the shell, and so must be enclosed in double quotation marks (") or single quotation marks ('), or must be prefixed with the escape character (\).
- The *command-line* specification must terminate with a semicolon (;).
- The program specified in *command-line* is launched with the directory from which find was launched set as the current directory. Before the program is launched, a prompt is output asking for confirmation. If the response from the standard input is not y, the specified command does not execute and false is returned.

- If you specify curly brackets (`{ }`) in *command-line*, they are replaced by the path name of the file or directory being searched. Their replacement is an absolute path name if you specified an absolute path name as the path for conducting the search or a relative path name if you specified a relative path name as the path for conducting the search.
- The evaluation is true if the program specified in *command-line* terminates with a return code of `0`.

`-path` *pattern*

Specifies an evaluation that is to be performed. The evaluation is true when the name of the file or directory to be searched for matches the value specified in *pattern*.

- The *pattern* is specified as a combination of characters and wildcards. You can use the escape character (`\`) to specify as part of the pattern value a character that would otherwise be a wildcard. If the escape character (`\`) is specified before a character that is not a wildcard characters, the `\` is ignored.
- For details about specifying *pattern*, see the description under `-name` *pattern*.

`-perm [-]`*permissions*

In UNIX, specifies permissions as octal numbers or symbols. If this argument is specified in Windows, an error (`find: -perm: unknown option`) results.

If the specified permissions follow a hyphen (`-`), the evaluation is true when the specified permissions are set in the mode of the file or directory. If a hyphen (`-`) is not specified, the evaluation is true when the specified permissions match the file mode exactly.

When you specify a numeric value for permissions, an error occurs if you specify a non-octal number or you specify an octal value greater than `07777` (`4095` decimal).

If you specify symbols for permissions, do so by setting, adding, and removing permissions, starting from the state in which nothing is specified (`0` in numeric representation). The result from specifying one or more symbols is then used in the search.

A symbol consists of three parts. Specify one or more symbols, as explained below. If you specify more than one, separate them with the comma (`,`).

Order within the symbol	Permitted value
First	Specifies the items for which you want to set access permissions. You can specify more than one at the same time. The items below can be specified. If nothing is specified, `a` (all users) is assumed. • `u`: Owner • `g`: Group • `o`: Other • `a`: All users
Second	Specifies an operation on the mode. The following processing is performed on the items specified in the first part of a symbol: • `=`: Set (overwrite) access permissions • `+`: Add access permissions • `-`: Remove access permissions The value to be set, added, or removed is specified in the third part of the symbol. You can specify the second and third parts of a symbol following the third part. The third part of a symbol can be omitted.
Third	Specifies the applicable access permissions. You can specify more than one at the same time. The following values can be specified: • `r`: Read • `w`: Write • `x`: Execute • `s`: Set the user ID or group ID at run time • `t`: Sticky bit

Order within the symbol	Permitted value
Third	• `u`: Owner access permissions currently set in the mode • `g`: Group access permissions currently set in the mode • `o`: Other access permissions currently set in the mode When this part is omitted and = is specified in the second part of the symbol, the items for which access permissions are set are cleared. When this part is omitted and + or - is specified in the second part of the symbol, no processing occurs. Specifying `s` or `t` in this part will be ignored if only `o` is specified in the first part.

The following table shows examples of specifying symbols:

Value specified in -perm	Numeric equivalent	Description
`u=x,g=w`	`120`	Set `u` to `x`, and set `g` to `w`.
`u=x,g=u`	`110`	Set `u` to `x`, and set the same values for `g` and `u`.
`u=x,=u`	`111`	Set `u` to `x`, and then set `a` (the default value) to the same values as `u`.
`u=x,u=w`	`200`	Set `u` to `x`, and then set (overwrite) `u` to `w`.
`u=x,u+w`	`300`	Set `u` to `x`, and then add `w` to `u`.
`ug=x`	`110`	Set `u` and `g` to `x`.
`u=rw`	`600`	Set `u` to `r` and `w`.
`u=r+x`	`500`	Set `u` to `r`, and then add `x`.
`u=r=w`	`200`	Set `u` to `r`, and then set (overwrite) it to `w`.
`=x,u=`	`011`	Set `a` (the default value) to `x`, and clear the settings for `u`.
`=`	`000`	Clear `a` (the default value).

This argument cannot be specified in Windows; if it is specified, an error occurs (`find: -perm: unknown option`).

`-print`

Specifies an evaluation that is to be performed. The evaluation is true when the path name of the file or directory being searched is output to the standard output followed by an end-of-line code. This evaluation is always true.

`-print0`

Specifies an evaluation that is to be performed. The evaluation is true when the path name of the file or directory being searched is output to the standard output followed by `NULL` ('`\0`'). This evaluation is always true.

`-prune`

Specifies an evaluation that is to be performed. The evaluation is true when directories encountered during the search are not to be followed. This evaluation is always true. This item is not valid when the `-d` option is specified.

`-size` *size*`[c]`

Specifies an evaluation that is to be performed. The evaluation is true when the relationship between the file's size and the specified *size* value (in units of blocks, rounded up to an even increment of 512 bytes) is valid. When `c` is specified after *size*, the unit for evaluation is bytes.

The specified *size* is interpreted as follows depending on whether no sign is specified or a numeric value with a + or - sign is specified:

- When a sign is not specified: Specified time difference
- + is specified before the numeric value: Greater than the specified value

- `-` is specified before the numeric value: Less than the specified value

The specified value must not exceed `9223372036854775807` (`0x7fffffffffffffff`). If you specify a greater value, `9223372036854775807` is assumed.

If you specify a non-numeric value, an error message (`find:` *primary*`:` *specified-character-string*`: illegal numeric value`) is output.

If *size* is omitted, an error message (`find:` *primary*`: requires additional arguments`) is output.

`-type` *type*

Specifies an evaluation that is to be performed. The evaluation is true when the type of the current file is the same as the specified *type* value. The types are listed below. If you specify a type other one than the following, an error message (`find: -type:` *specified-value*`: unknown type`) is output.

- `b`: Block special file (cannot be specified in Windows)
- `c`: Character special file (cannot be specified in Windows)
- `d`: Directory
- `f`: Regular file
- `l`: Symbolic link (cannot be specified in Windows)
- `p`: FIFO (cannot be specified in Windows)
- `s`: Socket (cannot be specified in Windows)

`-user` *user-name*

In Windows, specifies an evaluation that is to be performed. The evaluation is true when the file owner's user name is the same as the specified *user-name* value.

In UNIX, specifies an evaluation that is to be performed. The evaluation is true when the file owner's user name is the same as the specified *user-name* value. If you specify a numeric value for the user name and no such owner name exists, the specified value is interpreted as a user ID.

`-xdev`

Specifies an evaluation that is to be performed. This evaluation is always true. It is true in UNIX when directories are never searched with a device number that differs from that of the directory where the search was started.

- Operators

The primaries can be used with the operators listed below. The operators are shown in descending order of priority.

`(`*search-pattern*`)`

True is when the search pattern in the parentheses satisfies the conditions.

`!` *search-pattern*

False is when the search pattern that follows the `!` operator satisfies the conditions.

search-pattern `-and` *search-pattern* | *search-pattern* `-a` *search-pattern* | *search-pattern search-pattern*

This is the logical `AND` of two search patterns connected by the `-and` or `-a` operator, or else two side-by-side search patterns. True is when both search patterns are true. If the first search pattern is false, the second search pattern is not evaluated.

search-pattern `-or` *search-pattern* | *search-pattern* `-o` *search-pattern*

This is the logical `OR` of two search patterns connected by the `-or` or `-o` operator. True is when either search pattern is true.

Return codes

Return code	Meaning
`0`	Normal termination
`1` or greater	Error termination

Notes

- Depending on the shell used to execute `find`, characters such as the semicolon and parentheses might need to be prefixed with the escape character (`\`) or enclosed in single quotation marks (`'`) or double quotation marks (`"`).
- The output order of the files and directories being searched depends on the OS and the file system. If consistency across multiple platforms is desired, the output must be sorted.
- In Windows, file descriptors are closed without being inherited by processes that were generated by the `-exec` primary or a similar means. For example, an error results if you attempt to perform input or output on a file descriptor opened by the parent process without reopening it. This does not apply to the standard input, standard output, and standard error output, which do not need to be reopened.
- In Windows, if the program name specified in the `-exec` option contains no path, the program found according to the path search order of the target Windows API is executed.

Usage examples

- Display file and directory names ending in `.c`.

```
$ find . -name '*.c'
./test/a.c
./test/b.c
./test/c.c
./test/abc.c
$
```

- Display files and directories that are older than file `ttt`, or whose owner is not `root`.

```
$ ls -l
total 0
-rw-rw-r-- 1 user1 group1 0 Oct  7 10:12 a.c
-rw-rw-r-- 1 root  group1 0 Oct  7 10:12 abc.c
-rw-rw-r-- 1 user1 group1 0 Oct  7 10:12 b.c
-rw-rw-r-- 1 user1 group1 0 Oct  7 10:10 c.c
-rw-rw-r-- 1 user1 group1 0 Oct  7 10:11 ttt
$ find . \! \( -newer ttt -user root \)
.
./ttt
./b.c
./a.c
./c.c
$
```

- Display file names under the current directory that end with a dot (`.`) followed by a single digit, skipping over the directory `command1`.

```
$ ls command1 command2
command1:
a1.txt  b1.txt  command1  command1.1  command1.c  command1.o  extern.h
obj
```

```
command2:
a2.txt  b2.txt  command2  command2.1  command2.c  command2.o  extern.h
obj
$ find . ! -path './command1/*' -name '*.[0-9]'
./command2/command2.1
$
```

- Delete all `.o` files under the current directory.

```
$ ls command1 command2
command1:
a1.txt  b1.txt  command1  command1.1  command1.c  command1.o  extern.h
obj

command2:
a2.txt  b2.txt  command2  command2.1  command2.c  command2.o  extern.h
obj
$ find . -name '*.o' -exec rm {} \;
$ ls command1 command2
command1:
a1.txt  b1.txt  command1  command1.1  command1.c  extern.h  obj

command2:
a2.txt  b2.txt  command2  command2.1  command2.c  extern.h  obj
$
```

- Display an option error message.

  This message might vary depending on the platform on which the command is executed. The following shows an example in Windows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\find -w
find: illegal option -- w
usage: find [-dHhL] path ... [expression]
```

getopt command (analyzes command line options)

Format 1

```
getopt analysis-options argument-to-be-analyzed
```

Format 2

```
getopt [option] [--] analysis-options argument-to-be-analyzed
```

Format 3

```
getopt [option] -o short-analysis-option-name [option] [--] argument-to-be-
analyzed
```

Description

This command analyzes the command line specified in *argument-to-be-analyzed* according to the specified *analysis-options* and outputs the analysis results to the standard output. The command can analyze both short and long options. This command simplifies syntax analysis of shell scripts.

If one of the following specifications is made, the command assumes that format 1 is being used:

- The first parameter of the argument does not begin with `-`.
- The `GETOPT_COMPATIBLE` environment variable is set.

For any other specifications, if the `-o` option is specified, format 3 is assumed; if the `-o` option is not specified, format 2 is assumed.

Arguments

analysis-options

Specifies a character string containing the analysis options.

When you specify short options, note the following:

- If a short option begins with `-`, an operand is output at the location where it appears. If a short option begins with +, the option must be specified before an operand.

  Note that in format 1, `-` and + at the beginning are ignored.
- If a short option begins with `:` (or if a short option begins with `-` or + and the next character is `:`), and the option for the argument to be analyzed is not specified in the analysis options, no error message is issued, but the return code will be an error. Analysis of the arguments to be analyzed will continue.

By specifying `:` or `::` after an option character or an option name, you can define the following:

Specifying : after an option character or an option name

This indicates that the option requires a value.

In this case, specify an option name and an option value for the argument to be analyzed in the following format:

- Specify a short option name and an option value without a separator.

  ```
  $ getopt "xy:z" -yarg
  -y arg --
  ```
- Specify a short option name and an option value separated by a space.

  ```
  $ getopt "xy:z" -y arg
  -y arg --
  ```
- Specify a long option name and an option value separated by a space.

  ```
  $ getopt -o "abc" -l xyz: -- --xyz nml
  --xyz 'nml' --
  ```
- Specify in the format *long-option-name=option-value*.

  ```
  $ getopt -o "abc" -l xyz: -- --xyz=nml
  --xyz 'nml' --
  ```

Specifying :: after an option character or an option name

This indicates that specifying a value for the option is optional. If you specify an argument, specify `::` immediately after an option character or an option name.

In this case, specify an option name and an option value for the argument to be analyzed in the following format:

- Short options
  Specify an option name immediately followed by an option value.
- Long options
  Specify in the format *long-option-name=option-value*.

argument-to-be-analyzed

Specifies an option name, an option value, and an operand as the argument to be analyzed. For details about specifying options, see *8.1 Command description format*.

How to specify options

Specify a short option following `-` and a long option following `--`. If no long option name follows `--`, the command terminates option analysis at that location and treats all subsequent parameters as operands.

You can abbreviate long option names. In such a case, the complete long option names are displayed in the output results. The following shows an example:

```
$ getopt -o "" -l "longZ" -- --lo
 --longZ --
```

If you abbreviate a long option name, make sure that the specified name can be distinguished from all other options. If the specified long option name cannot be identified, the command treats it as any option name defined earlier in *analysis-options*.

Specification order of options and operands

By default, there is no rule for the specification order of options and operands. The options specified after operands are still analyzed. You can specify an option specification order by using the following environment variables:

- Using the `ADSH_CMD_ARGORDER` environment variable to specify an option specification order
  If the `ADSH_CMD_ARGORDER` environment variable is specified, the options specified as the argument to be analyzed must appear before operands.
- Using the `POSIXLY_CORRECT` environment variable to specify an option specification order
  If the `POSIXLY_CORRECT` environment variable is specified, the options specified as the argument to be analyzed must appear before operands.
  This definition applies to commands in the Linux version as well as to commands specified in the `ADSH_CMD_ARGORDER` environment variable.

option

In formats 2 and 3, you can specify the following options:

-l long-option-name

-longoptions=long-option-name

Specifies the long analysis option.

To specify multiple options, separate them with the comma (`,`) or the space or specify this option multiple times.

-n program-name

--name=program-name

Specifies a command name that is to be displayed in error messages for option analysis instead of the `getopt` command name.

-q

--quiet

Specifies that output of error messages resulting from option analysis of the argument being analyzed is to be suppressed.

-Q

--quiet-output

Specifies that output of the analysis results is to be suppressed.

-u

--unquoted

Specifies that option values and operands obtained as analysis results in formats 2 and 3 are not to be enclosed in quotation marks.

-o short-analysis-option-name

--options=short-option-name

Specifies a short analysis option.

If this option is specified more than once, the last value specified takes effect.

If `W;` is specified for the short option and a long option name with `-W` for the argument to be analyzed is specified, the specified value is treated as a long option name.

Output of analysis results

Analysis results are classified into option names, option values, and operands and output to the standard output.

If format 2 or 3 was used, option values and operands are enclosed in single quotation marks (`'`). If `-u` is specified, they are not enclosed in single quotation marks (`'`).

For short options, an option name with `-` is output as one option. For long options, the complete option name with `--` is output as one option.

The separator `--` is output between an option (option name and option value) and an operand. This does not apply if `-` is specified at the beginning of short options.

The command continues processing all parameter options even if an error occurs during option analysis. The following shows an example:

```
$ getopt "xyz" -w -x
getopt: invalid option -- w
 -x --
```

Return codes

Return code	Meaning
`0`	Normal termination
`1`	Error termination • The option specified for *argument-to-be-analyzed* is not defined in *analysis-options*.
`2`	Error termination • A `getopt` command option is invalid.
`3`	Error termination • An error other than return code `1` or `2` occurred.

Examples

- Execute the command using format 1:

```
$ getopt "xy:z" -z -y arg1 arg2
 -z -y arg1 -- arg2
```

- Execute the command using format 2:

```
$ getopt -q "xy:z" -z -y arg1 arg2
 -z -y 'arg1' -- 'arg2'
```

- Execute the command using format 3:

```
$ getopt -o "xy:z" -- -z -y arg1 arg2
 -z -y 'arg1' -- 'arg2'
```

- Set analysis results in positional parameters.

  Executable file

```
OPT=`getopt -o abc:d: -- -a -b`
eval set -- "$OPT"
echo $1
echo $2
echo $3
```

  Execution results

```
-a
-b
--
```

- Display option error messages:

  This message might vary depending on the platform used to execute the command. The following is an example for Windows:

```
C:\> getopt -z
getopt: illegal option -- z
usage: getopt optstring parameters
       getopt [options] [--] optstring parameters
       getopt [options] -o optstring [options] [--] parameters
```

grep command (searches for characters in files)

Format

```
grep [-a] [-b] [-c] [-E] [-G] [-h] [-I] [-i] [-L] [-l] [-n]
    [-q] [-R] [-r] [-s] [-U] [-v] [-w] [-x]
    [-A number] [-B number] [-C[number]]
    [-e pattern] [-f pattern-file-path-name] [pattern] [path-name ...]
```

Description

This command searches files for characters (specified in *pattern*).

Arguments

-a

Specifies that all files are to be handled as ASCII text files.

-b

Specifies that the offset in bytes is to be displayed at the beginning of each matching line.

-c

Specifies that only the number of lines selected is to be output to the standard output.

-E

Specifies that the value specified in *pattern* is to be handled as an extended regular expression. When the `-E` and `-G` options are both specified, the one specified last takes effect.

-G

Specifies that the value specified in *pattern* is to be handled as a regular expression. This is the default behavior. When the `-E` and `-G` options are both specified, the one specified last takes effect.

-h

Specifies that no file name is to be displayed at the beginning of each output line when either of the following conditions is satisfied:

- The `-R` or `-r` option is specified.
- Multiple path name are specified as being subject to search.

-I

Specifies that binary files are to be ignored.

-i

Specifies that uppercase letters are not to be distinguished from lowercase letters (and vice versa).

-L

Specifies that only the names of files that do not contain a match for the value specified in *pattern* are to be output to the standard output. If the `-L` and `-l` options are both specified, the one specified last takes effect.

-l

Specifies that only the names of files that contain a match for the value specified in *pattern* are to be output to the standard output. If the `-L` and `-l` options are both specified, the one specified last takes effect.

-n

Specifies that its relative line number in the file is to be output at the beginning of each output line. This specification is ignored when any of the `-c`, `-L`, `-l`, and `-q` options is specified.

-q

Specifies that nothing is to be output to the standard output.

-R|-r

Specifies that directories are to be searched recursively.

If the `-L`, `-l`, and `-q` options are all omitted, the file name is output at the beginning of each output line.

-s

Specifies that output of error messages related to unreadable or nonexistent files is to be suppressed.

-U

Specifies that binary files are to be searched but not output.

-v

Specifies that lines that do not contain a match for the value in *pattern* are to be output.

-w

Specifies that only lines that contain the specified character string as a whole word are to be output.

A word is a character string that consists of alphanumeric characters and the underscore (_). Words must be delimited by the space, any other non-word character, or the beginning or end of the line.

-x

Specifies that the specified character string is to be compared to each line in the file, and a line is to be output only if the entire line constitutes an exact match.

-A number

Specifies that as many lines as specified that follow a line matching *pattern* are to be output.

-B number

Specifies that as many lines as specified that precede a line matching *pattern* are to be output.

-C[number]

Specifies that as many lines as specified that precede and follow a line matching *pattern* are to be output. If no value is specified (*number* is omitted), the default is `2` (which would be the equivalent of specifying `-A 2 -B 2`).

If you specify *number* in the `-C` option, do not specify any spaces between `-C` and *number*.

-e pattern

Used to specify a pattern that begins with a hyphen (`-`).

-f pattern-file-path-name

Specifies the path name for a file that contains patterns to be searched for. The specified file specifies one line per pattern. If an empty file is specified (a file in which no patterns are specified), there will be nothing to search for and no matches will be found.

pattern

Specifies a pattern to be searched for.

path-name ...

Specifies a path name that is to be searched. Multiple path names can be specified. If no path name is specified, the contents of the standard input are searched. If you specify a directory name, you must also specify the `-R` or `-r` option.

If the `-L`, `-l`, and `-q` options are all omitted, the file name is output at the beginning of each output line.

Return codes

Return code	Meaning
`0`	Normal termination. • A line was found that contains a pattern being searched for. • Or, if the `-v` option is specified, a line was found that does not contain the pattern being searched for.
`1`	Normal termination. • No lines contain the pattern being searched for. • Or, if the `-v` option is specified, all lines contain the pattern being searched for.
`2` or greater	Error termination

Notes

- In Windows, symbolic links are not output.

- If the first 8,192 bytes of the file include data that is other than printable single-byte characters, spaces, tabs, backspaces, and multibyte characters, the file is considered to be a binary file.
- To execute `grep` from the command prompt in Windows, you must enclose the pattern in double quotation marks (`"`).
- Files whose character encoding differs from the local character encoding are considered to be binary files.
- In Windows, input and output are performed in the binary mode for files and for the standard input and standard output. No conversion of end-of-line codes is performed.

Usage examples

- Display the default output with no options specified.

```
C:\TEMP>%ADSH_OSCMD_DIR%\grep ABCD test1.txt
ABCDEFGHIJKLMNOPQRSTUVWXYZ
77777777[ABCD]ccccccccc
555555555:ABCD:111111111
ABCD
ABCD_XYZ
0000<ABCD>0000
/* ABCD */
```

- Display the default output from multiple files with no options specified.

```
C:\TEMP>%ADSH_OSCMD_DIR%\grep ABCD_ test1.txt test2.txt test3.txt
test4.txt
test1.txt:ABCD_XYZ
test2.txt:ABCD_XYZ
test3.txt:ABCD_XYZ
test4.txt:ABCD_XYZ
```

- Specify the `-h` option to display the matching lines without adding a file name when multiple files are searched:

```
C:\TEMP>%ADSH_OSCMD_DIR%\grep -h ABCD test1.txt test2.txt test3.txt
test4.txt
ABCD_XYZ
ABCD_XYZ
ABCD_XYZ
ABCD_XYZ
```

- Specify the `-b` option to show the offset in bytes at the beginning of matching lines.

```
C:\TEMP>%ADSH_OSCMD_DIR%\grep -b ABCD test1.txt
77:ABCDEFGHIJKLMNOPQRSTUVWXYZ
104:77777777[ABCD]ccccccccc
133:555555555:ABCD:111111111
212:ABCD
256:ABCD_XYZ
301:0000<ABCD>0000
316:/* ABCD */
```

- Specify the `-c` option to display only a count of the number of matching lines.

```
C:\TEMP>%ADSH_OSCMD_DIR%\grep -c ABCD test1.txt
7
```

- Specify the `-i` option to not distinguish between upper and lowercase.

```
C:\TEMP>%ADSH_OSCMD_DIR%\grep -i AbCd  test1.txt
ABCDEFGHIJKLMNOPQRSTUVWXYZ
77777777[ABCD]ccccccccc
555555555:ABCD:111111111
abcdefghijklmnopqrstuvwxyz
ABCD
abcd
ABCD_XYZ
0000<ABCD>0000
/* ABCD */
```

- Specify the -L option to display only the names of files that do not contain the pattern.

```
C:\TEMP>%ADSH_OSCMD_DIR%\grep -L ABC_  test1.txt test2.txt test3.txt
test4.txt
test1.txt
test2.txt
test4.txt
```

- Specify the -l option to display only the names of files that contain the pattern.

```
C:\TEMP>%ADSH_OSCMD_DIR%\grep -l ABC_ test1.txt test2.txt test3.txt
test4.txt
test3.txt
```

- Specify the -n option to display the relative line number in the file before each output line.

```
C:\TEMP>%ADSH_OSCMD_DIR%\grep -n ABCD test1.txt
4:ABCDEFGHIJKLMNOPQRSTUVWXYZ
5:77777777[ABCD]ccccccccc
7:555555555:ABCD:111111111
10:ABCD
14:ABCD_XYZ
17:0000<ABCD>0000
18:/* ABCD */
```

- Specify the -q option to not write anything to the standard output. The first example below does not specify the -q option, while the second example does.

```
C:\TEMP>%ADSH_OSCMD_DIR%\grep ABCD_XYZ test1.txt
ABCD_XYZ

C:\TEMP>%ADSH_OSCMD_DIR%\grep -q ABCD_XYZ test1.txt
```

- Specify the -R option to search directories recursively.

```
C:\TEMP>%ADSH_OSCMD_DIR%\grep -R ABCD C:\USR\data
C:\USR\data\data_2\data_3\test3.txt:ABCDEFGHIJKLMNOPQRSTUVWXYZ
C:\USR\data\data_2\data_3\test3.txt:ABCD333
C:\USR\data\data_2\data_3\test3.txt:ABCD_AS
C:\USR\data\data_2\test2.txt:77777777[ABCD]ccccccccc
C:\USR\data\data_2\test2.txt:555555555:ABCD:111111111
C:\USR\data\data_2\test2.txt:ABCD222
C:\USR\data\data_2\test2.txt:ABCD_MM
C:\USR\data\test0.txt:ABCD_1118
C:\USR\data\test0.txt:ABCD_AS321
C:\USR\data\test0.txt:0000<ABCD>0000
C:\USR\data\test0.txt:/* ABCD */
```

- Specify the `-s` option to suppress output of error messages. The first example below does not specify the `-s` option, while the second example does.

```
C:\TEMP>%ADSH_OSCMD_DIR%\grep ABCD  test5.txt
grep: test5.txt: No such file or directory

C:\TEMP>%ADSH_OSCMD_DIR%\grep -s ABCD  test5.txt
```

- Specify the `-w` option to display only whole-word matches for the pattern.

```
C:\TEMP>%ADSH_OSCMD_DIR%\grep -w ABCD test1.txt
77777777[ABCD]ccccccccc
555555555:ABCD:111111111
ABCD
0000<ABCD>0000
/* ABCD */
```

- Specify the `-x` option to display only whole-line matches.

```
C:\TEMP>%ADSH_OSCMD_DIR%\grep -x ABCD test1.txt
ABCD
```

- The example below also specifies the `-x` option, this time to search the file `file.txt`, whose contents are as follows:

  `file.txt`

```
  ABABAB
  ACACACAC
  ABABAB
```

  - The command displays nothing because there are no whole-line matches:

```
grep -x ABA file.txt
```

  - In the following example, two whole-line matches (lines 1 and 3) are found in `file.txt`:

```
grep -x ABABAB file.txt
ABABAB
ABABAB
```

- Specify `3` with the `-A` option to display three lines following each matching line as context.

```
C:\TEMP>%ADSH_OSCMD_DIR%\grep -A 3 XYZ  test1.txt
ABCDEFGHIJKLMNOPQRSTUVWXYZ
77777777[ABCD]ccccccccc
-XYZ
555555555:ABCD:111111111
abababababababababababab
abcdefghijklmnopqrstuvwxyz
--
ABCD_XYZ
asasasasasasasasas01
ASASASASASASAS
0000<ABCD>0000
```

- Specify `3` with the `-B` option to display three lines preceding each matching line as context.

```
C:\TEMP>%ADSH_OSCMD_DIR%\grep -B 3 XYZ  test1.txt
/*-----------------------*/
ABABABABABABABABABABABABAB
012345678901234567890
ABCDEFGHIJKLMNOPQRSTUVWXYZ
77777777[ABCD]ccccccccc
-XYZ
--
JJJJJJJJJJJJJJJJ
KKKKKKKKKKKKKKKK
abcd
ABCD_XYZ
```

- Specify the `-C` option to display two lines preceding and following each matching line as context.

```
C:\TEMP>%ADSH_OSCMD_DIR%\grep -C XYZ test1.txt
ABABABABABABABABABABABABAB
012345678901234567890
ABCDEFGHIJKLMNOPQRSTUVWXYZ
77777777[ABCD]ccccccccc
-XYZ
555555555:ABCD:111111111
abababababababababababababab
--
KKKKKKKKKKKKKKKK
abcd
ABCD_XYZ
asasasasasasasasas01
ASASASASASASAS
```

- Use the `-e` option to specify a pattern that begins with `-`.

```
C:\TEMP>%ADSH_OSCMD_DIR%\grep -e "-rw-" file01.txt
-rw------- user0001    12 May 12 17:19 a.txt
-rw------- user0001    79 May 12 20:36 abc.txt
-rw------- user0001   141 May 12 20:36 abcd.txt
-rw------- user0001    12 May 12 18:05 b.txt
-rw------- user0001   133 May 12 21:49 f01.txt
-rw------- user0001     0 May 12 19:42 ff
-rw------- user0001     0 May 12 20:54 ff.txt
```

- Display an option error message.

  This message might vary depending on the platform on which the command is executed. The following shows an example in Windows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\grep -d
grep: illegal option -- d
usage: grep [-abcEGhIiLlnqRrsUvwx] [-A num] [-B num] [-C[num]]
       [-e pattern] [-f file] [pattern] [file ...]
```

- Display an error message if a file does not exist.

```
C:\TEMP>%ADSH_OSCMD_DIR%\grep CHECK  file99
grep: file99: No such file or directory
```

head command (displays the first part of files)

Format

```
head [-num-lines|-n num-lines] [path-name ...]
```

Description

This command displays the first few lines from one or more files. The specified number of lines from the beginning of the file are output to the standard output. If no file is specified, the standard input is read. If no value is specified for the number of lines to be output, 10 lines is assumed.

Arguments

-num-lines | -n num-lines ~<decimal>((1 to 2147483647))

Specifies the number of lines from the beginning of the input file that are to be sent to the standard output. If you specify a value that is less than or equal to `0` or that is greater than `2147483647`, an error message (`head: line count too small:` *specified-value* or `head: line count too large:` *specified-value*) is output.

path-name

Specifies the path name of an input file.

- The default is the standard input.
- Multiple files can be specified. If you specify more than one file, each file is identified at the beginning of the output from that file by a blank line (linefeed) and its file name in a header string in the following format:
  ==> *file-name* <==
- When you execute the command with multiple files specified, all the files are processed. If any file fails to open, the command terminates with a return code of `1`.

Return codes

Return code	Meaning
`0`	Normal termination
`1` or greater	Error termination

Usage examples

The following shows the format of the files used in the examples below to illustrate the results of executing the `head` command.

- `test1.txt`

```
0001:test1.txt
0002:test1.txt
0003:test1.txt
0004:test1.txt
0005:test1.txt
0006:test1.txt
0007:test1.txt
0008:test1.txt
0009:test1.txt
0010:test1.txt
```

- `test2.txt`

```
0001:test2.txt
0002:test2.txt
0003:test2.txt
0004:test2.txt
0005:test2.txt
0006:test2.txt
0007:test2.txt
0008:test2.txt
0009:test2.txt
0010:test2.txt
```

The examples below illustrate the results of executing the command on the files shown above.

- Display the first two lines of the files `test1.txt` and `test2.txt`.

```
$ head -2 test1.txt test2.txt
==> test1.txt <==
0001:test1.txt
0002:test1.txt

==> test2.txt <==
0001:test2.txt
0002:test2.txt
$
```

- Display an option error message.

  This message might vary depending on the platform on which the command is executed. The following shows an example in Windows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\head -d
head: illegal option -- d
usage: head [-count | -n count] [file ...]
```

hostname command (displays the host name)

Format

```
hostname
```

Description

This command displays the current host's host name.

Return codes

Return code	Meaning
`0`	Normal termination
`1` or greater	Error termination

Notes

- This command takes no arguments. Any arguments that are specified are ignored during execution of the command.

Usage example

- Display the name of the current host system.

```
C:\TEMP>%ADSH_OSCMD_DIR%\hostname
HOST01
```

ls command (lists the contents of files or directories)

Format

```
ls [-1] [-A] [-a] [-C] [-c] [-d] [-F] [-f] [-g] [-h] [-i] [-k]
  [-L] [-l] [-m] [-n] [-p] [-q] [-R] [-r] [-S] [-s] [-T] [-t]
  [-u] [-x]
  [--format=display-format][--full-time]
  [--indicator-style=file-type-style][--sort=sort-key]
  [--time=file-date-and-time-type]
  [path-name ...]
```

Description

The command lists directory contents. The contents are sent to the standard output.

In the output contents, permissions are displayed as described in the following:

- The first character indicates the type of target:
  `-`: Regular file
  `b`: Block special file
  `c`: Character special file
  `d`: Directory
  `l`: Symbolic link
  `p`: FIFO
  `s`: Socket
- The subsequent nine characters are treated as three sets of three characters which indicate the owner permissions, group permissions, and other user permissions. In Windows, only the owner permissions are displayed.

Order in permissions	Displayed character#	Permission
1	r	Read by owner
2	w	Write by owner
3	x	Execute by owner
	s	Set user ID or set group ID by owner/execute
	S	Set user ID or set group ID by owner/no execute

Order in permissions	Displayed character#	Permission
4	`r`	Read by group
5	`w`	Write by group
6	`x`	Execute by group
	`s`	Set user ID or set group ID by group/execute
	`S`	Set user ID or set group ID by group/no execute
7	`r`	Read by other users
8	`w`	Write by other users
9	`x`	Execute by other users
	`t`	Sticky bits by other users/execute
	`T`	Sticky bits by other users/no execute

#

The following table explains the characters that are displayed:

Character	Meaning
`-`	The corresponding permission is not granted.
`r`	In Windows, files or directories exist. In UNIX, read permissions are granted.
`w`	In Windows, the read-only attribute is not set. In UNIX, write permissions are granted.
`x`	In Windows, one of the following: • The extension is `.com`, `.exe`, `.cmd`, or `.bat`. • This is a directory. In UNIX, execute permissions are granted.
`s`	Set user ID or set group ID is granted and execute permissions are granted (UNIX only).
`S`	Set user ID or set group ID is granted, but execute permissions are not granted (UNIX only).
`t`	Sticky bits are granted and execute permissions are granted (UNIX only).
`T`	Sticky bits are granted, but execute permissions are not granted (UNIX only).

If the `-g`, `-l`, `-n`, or `--full-time` option is specified, results are output in the long format.

The long format means that not only file and directory names but detailed information about files and directories are output. You can change the output format for each item by combining the long format with the `-h`, `-T`, and `-u` options.

Arguments

-1

--format=single-column

Specifies that the list format is to be one entry per line (in a single column).

-A

--almost-all

Specifies that all entries are to be listed except for those from those from `.` (dot) and `..` (dot dot) files.

-a

--all

Specifies that all files and directories, including those with names starting with `.` (dot), are to be listed.

-C

--format=vertical

Specifies that entries are to be listed in multiple columns, sorted vertically. This is the default for output to the terminal.

-c

--time=ctime

--time=status

Specifies that the date and time of the last change in file information rather than the most recent modification date and time is to be used for sorting (`-t` option) and for list output (`-g`, `-l`, `-n`, and `--full-time` options).

-d

--directory

Specifies that only the directory names are to be listed, without displaying the contents of the directories.

-F

--classify

--indicator-style=classify

Specifies that a forward slash (`/`) is to be output after a directory name, an asterisk (`*`) is to be output after an executable file, an at mark (`@`) is to be output after a symbolic link, a vertical bar (`|`) is to be output after a FIFO name, and an equal sign (`=`) is to be output after a socket.

-f

--sort=none

Specifies that the list is to be output without sorting.

-g

Specifies that the list is to be output in long format, but without listing the file owners.

-h

--human-readable

Specifies that when the long format is used, file sizes are to be divided by a power of 2 and rounded off to two decimal places for display purposes. A size letter (`M` for `1048576` or `K` for `1024`) is to be added to the file size.

The `-h` option is ignored for any special files in a directory.

-i

--inode

In UNIX, specifies that each file's inode number is to be output.

In Windows, `0` is always output.

-k

In UNIX, specifies that `KB` is to be output as the units for listing file sizes with the `-s` option and as the units for listing the total number of blocks for directories with the `-l`, `-g`, `-s`, and `--full-time` options.

In Windows, specifies that `KB` is to be output as the units for listing file sizes with the `-s` option.

-L

--dereference

In UNIX, specifies that list information about a referenced file rather than the symbolic link itself is to be output.

In Windows, list information about a referenced file is always output.

-l

--format=long

--format=verbose

Specifies that the list is to be output in long format with the following items displayed. To output date and time in the complete format, specify the `--full-time` option.

- UNIX

  Access permissions, number of links, owner name, group name, size, most recent modification date and time, name of file or directory. If the target is directories, the total number of directories including `.` and `..` under that directory are displayed.

- Windows

  Access permissions for the owner of the file, owner name, size (except for directories), most recent modification date and time, name of file or directory

-m

--format=commas

Specifies that file names are to be delimited by the comma (`,`).

-n

--numeric-uid-gid

In UNIX, specifies that user IDs and group IDs are to be listed instead of user names and group names.

In Windows, `0` is listed for a user ID, and group IDs are not output.

-p

--indicator-style=slash

Specifies that a forward slash (`/`) is to be output after a directory name.

-q

--hide-control-chars

Specifies that a question mark (`?`) is to be output for any unprintable character used in a file name. This is the default for output to the terminal.

-R

--recursive

Specifies that subdirectories are to be listed recursively.

-r

--reverse

Specifies that the output is to be sorted in reverse order.

-S

--sort=size

Specifies that the entries are to be sorted by size, from largest to smallest.

-s

--size

In UNIX, specifies that the number of blocks in each file, rounded up to full blocks, is to be output. A block is 512 bytes, unless you also specify the `-k` option or have defined the `BLOCKSIZE` environment variable.

In Windows, the number of blocks is always listed as `0`.

-T

Specifies that date and time information is to be listed in the order month, date, hour, minute, second, and year. This option is specified together with the `-g`, `-l`, or `-n` option.

-t

--sort=time

Specifies that the entries are to be sorted by most recent modification date and time, starting with the most recent.

-u

--time=atime

--time=access

--time=use

Specifies that the most recent access date and time instead of the most recent modification date and time is to be used when sorting (`-t` option) or listing in the long format (`-g`, `-l`, `-n`, or `--full-time` option).

-x

--format=across

--format=horizontal

Specifies that entries are to be listed in multiple columns, as with `-C`, but sorted horizontally.

--format=display-format

Specifies the format for displaying file or directory contents.

The permitted values for display formats are listed below. If the `--format` option is specified more than once, the last option specified takes effect.

`across` or `horizontal`

Specifies that entries are to be listed in multiple columns, sorted horizontally. This is the same as the `-x` option.

`commas`

Displays file names separated by the comma (`,`). This is the same as the `-m` option.

`long` or `verbose`

Displays in the long format. This is the same as the `-l` option.

`single-column`

Displays one entry (one column) per line. This is the same as the `-1` option.

`vertical`

Displays multiple columns, sorted vertically. This is the same as the `-C` option.

--full-time

Specifies that the same items as when the `-l` option is specified are to be output. However, information about the date and time is to be output in the complete format, not in the default abbreviated format.

The output format for the date and time information is as follows:

YYYY`-`*MM*`-`*DD hh*`:`*mm*`:`*ss*`.`*nnnnnnnnn* `+/-`*hhmm*

YYYY: Calendar year

MM: Month

DD: Date

hh: Hour

mm: Minute

ss: Second

nnnnnnnnn: Date and time less than one second. `000000000` is always output.

`+/-`*hhmm*: Time zone (the time differential from UTC).

--indicator-style=file-type-style

Specifies the style to be used to display information about the file type.

The following values are supported:

`classify`

Outputs the character indicating the file type immediately after the file name. For a directory name, a forward slash (`/`) is displayed immediately after the directory name. This is the same as the `-F` option.

For details about the characters used to indicate file types, see the description of the `-F` option.

`slash`

Displays a forward slash (`/`) immediately after the directory name. This is the same as the `-p` option.

If `--indicator-style=classify` is specified together with `--indicator-style=slash`, the `classify` specification takes effect.

In Windows, `classify` is ignored, if specified.

--sort=sort-key

Specifies that when multiple files are displayed, they are to be sorted by the file information indicated by the specified sort key. If the `--sort` option is specified more than once, the last specification takes effect.

For the sort key, the following values are supported:

`size`

Sorts files by file size. This is the same as the `-S` option.

`time`

Sorts files by most recent modification date and time. This is the same as the `-t` option. You can also specify the `--time` option to sort files by the date and time each was last accessed or changed.

none

Outputs files without sorting them. This is the same as the `-f` option.

--time=file-date-and-time-type

Specifies a file date and time type that is to be applied to date and time information used for sorting (`-t`) and listing (the `-g`, `-l`, `-n`, and `--full-time` options). If the `--time` option is specified more than once, the last specification takes effect. If more than one option is specified, the last option takes effect. In Windows, `--time` is ignored, if specified.

For the file date and time type, the following values are supported:

`atime`, `access`, or `use`

Uses the last date and time files were accessed. This is the same as the `-u` option.

`ctime` or `status`

Uses the last date and time file information was changed. This is the same as the `-c` option.

path-name

Specifies the name of a file or directory that is to be listed. More than one can be specified.

Return codes

Return code	Meaning
0	Normal termination
1 or greater	Error termination

Notes

- If more than one of the `-1`, `-C`, `-l`, `-m`, `-x` and `--full-time` options is specified, the one specified last takes effect.

  However, if the `-l` option is specified together with the `--full-time` option, the `--full-time` option takes effect.
- All entries other than `.` (dot) or `..` (dot dot), including entries that start with `.` (dot), are eligible to be listed, regardless of whether the `-A` option is specified.
- The default block size is 512 bytes.
- If the date and time associated with a file is at least 182 days (about six months) distant from the time the command is executed, the year is listed instead of the date and time.

  However, this does not apply when the `--full-time` option is specified.
- In UNIX, if the user name or group name cannot be acquired, the user ID or group ID, respectively, is displayed.
- In Windows, the `-F`, `-c`, and `-u` options are ignored.
- In Windows, an ellipsis (`...`) is displayed when the user name cannot be obtained.
- In Windows, the total size of the files in the directory is displayed in bytes.
- In Windows, hidden file attributes can be displayed.
- This command is affected by the following environment variables:
  - `COLUMNS` environment variable

    Sets the output width per line when listing in multiple columns with the `-C` option. You cannot define this within a job definition script in JP1/Advanced Shell.
  - `BLOCKSIZE` environment variable

In UNIX, sets the size of a block for purposes of displaying the number of blocks with the `-s` option. The permitted value range is from 512 to 1 GB (1,024 × 1,024 × 1,024). If the specified value is outside this range, the command handles it as described below, outputs a warning message to the standard error output, and then performs the subsequent processing:

■ If a value smaller than 512 is specified in the `BLOCKSIZE` environment variable

- The block size is set to 512 bytes.

■ If a value greater than 1 GB (1,024 × 1,024 × 1,024) is specified in the `BLOCKSIZE` environment variable

- The block size is set to 1 gigabyte (1,024 × 1,024 × 1,024).

If you use the `BLOCKSIZE` environment variable to change the block size, specify a multiple of 512. If the specified value is not a multiple of 512, the remainder will be discarded. For example, if a size of 1,500 bytes is defined, the block size will be treated as being 1,024 bytes. You can specify following the numeric value a size character indicating a multiple, such as `G` (1,024 × 1,024 × 1,024), `M` (1,024 × 1,024), or `K` (1,024). If any value other than a numeric value and size character is specified, the command will assume 512 bytes as the block size, output a warning message to the standard error output, and then resume the subsequent processing.

- `TZ` environment variable

  In UNIX, sets the time zone used to display the date and time.

  In Windows, the time zone set in the **Date and Time** control panel is used to display the date and time. The value of the `TZ` environment variable is ignored.

  Note that the `--full-time` option uses the value of the `TZ` environment variable and the time zone set in the **Date and Time** control panel. For this reason, you must ensure sure that the value of the `TZ` environment variable and the time zone set in the **Date and Time** control panel are the same. If they differ, the correct time zone will not be displayed by the `--full-time` option.

- In Windows, when you specify a drive letter as the directory, depending on how it is specified, it might reference the current directory where the command is being executed.

  Examples based on the following folder organization are explained below:

```
            Current drive       Other drive
            D:\                    E:\
              |                      |
              + X                    + R
              + Y                    + S
              + Z                    + T
                |                      |
                + file1                + fileA
                + file2                + fileB
                + file3                + fileC
```

When the current drive (`D:`) is specified, the entries under the directory where the command is executed are listed (`D:\Z`):

```
      D:\Z>ls -l D:
      total 462
      -rw------- ouser001  154 Jun 02 15:23 file1
      -rw------- ouser001  154 Jun 02 15:23 file2
      -rw------- ouser001  154 Jun 02 15:23 file3

      D:\Z>
```

Specify the current drive (`D:\`) to list the entries directly under the specified drive letter (`D:\`):

```
D:\Z>ls -l D:\
total 0
drwx------ ouser001   Jun 02 15:22 X
```

```
drwx------ ouser001   Jun 02 15:23 Y
drwx------ ouser001   Jun 02 15:25 Z

D:\Z>
```

Specify another drive (`E:`) to list the entries directly under the specified drive letter (`E:\`):

```
D:\Z>ls -l E:
total 0
drwx------ ouser001   Jun 02 15:24 R
drwx------ ouser001   Jun 02 15:24 S
drwx------ ouser001   Jun 02 15:25 T

D:\Z>
```

Specify another drive (`E:\`) to list the entries directly under the specified drive letter (`E:\`):

```
D:\Z>ls -l E:\
total 0
drwx------ ouser001   Jun 02 15:24 R
drwx------ ouser001   Jun 02 15:24 S
drwx------ ouser001   Jun 02 15:25 T

D:\Z>
```

Usage examples

- Specify no option to display files in the current directory.

```
C:\TEMP>%ADSH_OSCMD_DIR%\ls
cmp.exe      grep.exe     mv.exe       sleep.exe    cp.exe
hostname.exe spool        Adshuxpl.dll date.exe     ls.exe
rm.exe       tmp          cat.exe      mkdir.exe    rmdir.exe    uname.exe
```

- Specify the `-1` option to list entries in a single column.

```
C:\TEMP>%ADSH_OSCMD_DIR%\ls -1
Adshuxpl.dll
cat.exe
cmp.exe
cp.exe
date.exe
grep.exe
hostname.exe
ls.exe
mkdir.exe
mv.exe
rm.exe
rmdir.exe
sleep.exe
spool
tmp
uname.exe
```

- Specify the `-A` option to list all entries except `.` (dot) and `..` (dot dot). In Windows, entries that begin with `.` (dot) are always listed, regardless of whether the `-A` option is specified.

```
C:\TEMP>%ADSH_OSCMD_DIR%\ls -A
abcde.txt    date.exe     hostname.exe rm.exe       uname.exe
abcdex.txt   ls.exe       rmdir.exe    Adshuxpl.dll cat.exe
file1.txt    mkdir.exe    sleep.exe    abc.txt      cmp.exe
mv.exe       spool        abcd.txt     cp.exe       grep.exe
tmp
```

- Specify the `-a` option to include directories that begin with `.` (dot).

```
C:\TEMP>%ADSH_OSCMD_DIR%\ls -a
.            cat.exe      mv.exe       spool
..           cmp.exe      grep.exe     tmp
cp.exe       hostname.exe rm.exe       uname.exe
date.exe     ls.exe       rmdir.exe    Adshuxpl.dll
mkdir.exe    sleep.exe
```

- Specify the `-C` option to list entries in multiple columns sorted vertically.

```
C:\TEMP>%ADSH_OSCMD_DIR%\ls -C
cmp.exe      mkdir.exe    rmdir.exe    uname.exe    cp.exe
grep.exe     mv.exe       sleep.exe    Adshuxpl.dll date.exe
hostname.exe spool        cat.exe      ls.exe       rm.exe
tmp
```

- Specify the `-f` option to list without sorting.

```
C:\TEMP>%ADSH_OSCMD_DIR%\ls -f
Adshuxpl.dll cmp.exe      mkdir.exe    rmdir.exe    uname.exe
cp.exe       grep.exe     mv.exe       sleep.exe    date.exe
hostname.exe spool        cat.exe      ls.exe       rm.exe
tmp
```

- Specify the `-g` option to list entries in long format, but omitting the owner. In the case of Windows, the group name is omitted.

```
C:\TEMP>%ADSH_OSCMD_DIR%\ls -g
total 1069363
-rw------- 439808 May 09 11:31 Adshuxpl.dll
-rwx------  10752 May 09 11:34 cat.exe
-rwx------  10240 May 09 11:34 cmp.exe
-rwx------  18432 May 09 11:34 cp.exe
-rwx------  10240 May 09 11:34 date.exe
-rwx------  43008 May 09 11:33 grep.exe
-rwx------   7680 May 09 11:33 hostname.exe
-rwx------  22528 May 09 16:27 ls.exe
-rwx------   8192 May 09 11:33 mkdir.exe
-rwx------  12288 May 09 11:33 mv.exe
-rwx------  16384 May 09 11:33 rm.exe
-rwx------   8192 May 09 11:32 rmdir.exe
-rwx------   8192 May 09 11:32 sleep.exe
drwx------        May 10 08:50 spool
drwx------        May 10 08:50 tmp
-rwx------   9216 May 09 11:32 uname.exe
```

- Specify the `-h` option together with the long format option to append a size letter to the file size.

```
C:\TEMP>%ADSH_OSCMD_DIR%\ls -lh
total 1069363
```

```
-rw------- 10379780    430K May 09 11:31 Adshuxpl.dll
-rwx------ 10379780   10.5K May 09 11:34 cat.exe
-rwx------ 10379780   10.0K May 09 11:34 cmp.exe
-rwx------ 10379780   18.0K May 09 11:34 cp.exe
-rwx------ 10379780   10.0K May 09 11:34 date.exe
-rwx------ 10379780   42.0K May 09 11:33 grep.exe
-rwx------ 10379780    7.5K May 09 11:33 hostname.exe
-rwx------ 10379780   22.0K May 09 16:27 ls.exe
-rwx------ 10379780    8.0K May 09 11:33 mkdir.exe
-rwx------ 10379780   12.0K May 09 11:33 mv.exe
-rwx------ 10379780   16.0K May 09 11:33 rm.exe
-rwx------ 10379780    8.0K May 09 11:32 rmdir.exe
-rwx------ 10379780    8.0K May 09 11:32 sleep.exe
drwx------ 10379780         May 10 08:50 spool
drwx------ 10379780         May 10 08:50 tmp
-rwx------ 10379780    9.0K May 09 11:32 uname.exe
```

- Specify the `-i` option to display the inode number for each file. In the case of Windows, `0` is displayed for the inode number.

```
C:\TEMP>%ADSH_OSCMD_DIR%\ls -i
0 cp.exe         0 hostname.exe  0 rm.exe         0 uname.exe
0 date.exe       0 ls.exe        0 rmdir.exe      0 Adshuxpl.dll
0 mkdir.exe      0 sleep.exe     0 cat.exe        0 mv.exe
0 spool          0 cmp.exe       0 grep.exe       0 tmp

C:\TEMP>%ADSH_OSCMD_DIR%\ls -il
total 1069363
0 -rw------- 10379780  439808 May 09 11:31 Adshuxpl.dll
0 -rwx------ 10379780   10752 May 09 11:34 cat.exe
0 -rwx------ 10379780   10240 May 09 11:34 cmp.exe
0 -rwx------ 10379780   18432 May 09 11:34 cp.exe
0 -rwx------ 10379780   10240 May 09 11:34 date.exe
0 -rwx------ 10379780   43008 May 09 11:33 grep.exe
0 -rwx------ 10379780    7680 May 09 11:33 hostname.exe
0 -rwx------ 10379780   22528 May 09 16:27 ls.exe
0 -rwx------ 10379780    8192 May 09 11:33 mkdir.exe
0 -rwx------ 10379780   12288 May 09 11:33 mv.exe
0 -rwx------ 10379780   16384 May 09 11:33 rm.exe
0 -rwx------ 10379780    8192 May 09 11:32 rmdir.exe
0 -rwx------ 10379780    8192 May 09 11:32 sleep.exe
0 drwx------ 10379780         May 10 08:50 spool
0 drwx------ 10379780         May 10 08:50 tmp
0 -rwx------ 10379780    9216 May 09 11:32 uname.exe
```

- Specify the `-l` option to list entries in long format. In Windows, only the access permissions of the owner are displayed, and the group name, link count, and directory size are not displayed.

```
C:\TEMP>%ADSH_OSCMD_DIR%\ls -l
total 1069359
-rw------- 10379780  439808 May 09 11:31 Adshuxpl.dll
-rwx------ 10379780   10752 May 09 11:34 cat.exe
-rwx------ 10379780   10240 May 09 11:34 cmp.exe
-rwx------ 10379780   18432 May 09 11:34 cp.exe
-rwx------ 10379780   10240 May 09 11:34 date.exe
-rwx------ 10379780   43008 May 09 11:33 grep.exe
-rwx------ 10379780    7680 May 09 11:33 hostname.exe
-rwx------ 10379780   22528 May 09 16:27 ls.exe
```

```
-rwx------ 10379780    8192 May 09 11:33 mkdir.exe
-rwx------ 10379780   12288 May 09 11:33 mv.exe
-rwx------ 10379780   16384 May 09 11:33 rm.exe
-rwx------ 10379780    8192 May 09 11:32 rmdir.exe
-rwx------ 10379780    8192 May 09 11:32 sleep.exe
drwx------ 10379780         May 10 08:50 spool
drwx------ 10379780         May 10 08:50 tmp
-rwx------ 10379780    9216 May 09 11:32 uname.exe
```

- Specify the `-l` option together with the `-c` option to display the date and time of the most recent change in file information instead of the most recent modification date and time. Windows ignores the `-c` option and displays the most recent modification date and time.

```
C:\TEMP>%ADSH_OSCMD_DIR%\ls -lc
total 1069363
-rw------- 10379780  439808 May 09 11:31 Adshuxpl.dll
-rwx------ 10379780   10752 May 09 11:34 cat.exe
-rwx------ 10379780   10240 May 09 11:34 cmp.exe
-rwx------ 10379780   18432 May 09 11:34 cp.exe
-rwx------ 10379780   10240 May 09 11:34 date.exe
-rwx------ 10379780   43008 May 09 11:33 grep.exe
-rwx------ 10379780    7680 May 09 11:33 hostname.exe
-rwx------ 10379780   22528 May 09 16:27 ls.exe
-rwx------ 10379780    8192 May 09 11:33 mkdir.exe
-rwx------ 10379780   12288 May 09 11:33 mv.exe
-rwx------ 10379780   16384 May 09 11:33 rm.exe
-rwx------ 10379780    8192 May 09 11:32 rmdir.exe
-rwx------ 10379780    8192 May 09 11:32 sleep.exe
drwx------ 10379780         May 10 08:50 spool
drwx------ 10379780         May 10 08:50 tmp
-rwx------ 10379780    9216 May 09 11:32 uname.exe
```

- Specify the `-l` option together with the `-u` option to display the most recent access date and time instead of the most recent modification date and time. Windows ignores the `-u` option and displays the most recent modification date and time.

```
C:\TEMP>%ADSH_OSCMD_DIR%\ls -lu
total 1069363
-rw------- 10379780  439808 May 09 11:31 Adshuxpl.dll
-rwx------ 10379780   10752 May 09 11:34 cat.exe
-rwx------ 10379780   10240 May 09 11:34 cmp.exe
-rwx------ 10379780   18432 May 09 11:34 cp.exe
-rwx------ 10379780   10240 May 09 11:34 date.exe
-rwx------ 10379780   43008 May 09 11:33 grep.exe
-rwx------ 10379780    7680 May 09 11:33 hostname.exe
-rwx------ 10379780   22528 May 09 16:27 ls.exe
-rwx------ 10379780    8192 May 09 11:33 mkdir.exe
-rwx------ 10379780   12288 May 09 11:33 mv.exe
-rwx------ 10379780   16384 May 09 11:33 rm.exe
-rwx------ 10379780    8192 May 09 11:32 rmdir.exe
-rwx------ 10379780    8192 May 09 11:32 sleep.exe
drwx------ 10379780         May 10 08:50 spool
drwx------ 10379780         May 10 08:50 tmp
-rwx------ 10379780    9216 May 09 11:32 uname.exe
```

- Specify the `-m` option to list entries in stream output format delimited by the comma.

```
C:\TEMP>%ADSH_OSCMD_DIR%\ls -m
Adshuxpl.dll,  cat.exe,  cmp.exe,  cp.exe,  date.exe,
grep.exe,  hostname.exe,  ls.exe,  mkdir.exe,  mv.exe,
rm.exe,  rmdir.exe,  sleep.exe,  spool,  tmp,  uname.exe
```

- Specify the `-n` option to display user ID and group ID instead of user name and group name. In Windows, `0` is displayed for the user ID and group IDs are not output.

```
C:\TEMP>%ADSH_OSCMD_DIR%\ls -n
total 1069363
-rw------- 0  439808 May 09 11:31 Adshuxpl.dll
-rwx------ 0   10752 May 09 11:34 cat.exe
-rwx------ 0   10240 May 09 11:34 cmp.exe
-rwx------ 0   18432 May 09 11:34 cp.exe
-rwx------ 0   10240 May 09 11:34 date.exe
-rwx------ 0   43008 May 09 11:33 grep.exe
-rwx------ 0    7680 May 09 11:33 hostname.exe
-rwx------ 0   22528 May 09 16:27 ls.exe
-rwx------ 0    8192 May 09 11:33 mkdir.exe
-rwx------ 0   12288 May 09 11:33 mv.exe
-rwx------ 0   16384 May 09 11:33 rm.exe
-rwx------ 0    8192 May 09 11:32 rmdir.exe
-rwx------ 0    8192 May 09 11:32 sleep.exe
drwx------ 0         May 10 08:50 spool
drwx------ 0         May 10 08:50 tmp
-rwx------ 0    9216 May 09 11:32 uname.exe
```

- Specify the `-p` option to display a forward slash (`/`) after a directory name.

```
C:\TEMP>%ADSH_OSCMD_DIR%\ls -p
cp.exe          hostname.exe    rm.exe          uname.exe
date.exe        ls.exe          rmdir.exe       mkdir.exe
sleep.exe       cat.exe         mv.exe          spool/
cmp.exe         grep.exe        tmp/

C:\TEMP>%ADSH_OSCMD_DIR%\ls -alp
total 1069363
drwx------ 10379780         May 10 09:45 ./
drwx------ 10379780         May 10 10:02 ../
-rw------- 10379780  439808 May 09 11:31 Adshuxpl.dll
-rwx------ 10379780   10752 May 09 11:34 cat.exe
-rwx------ 10379780   10240 May 09 11:34 cmp.exe
-rwx------ 10379780   18432 May 09 11:34 cp.exe
-rwx------ 10379780   10240 May 09 11:34 date.exe
-rwx------ 10379780   43008 May 09 11:33 grep.exe
-rwx------ 10379780    7680 May 09 11:33 hostname.exe
-rwx------ 10379780   22528 May 09 16:27 ls.exe
-rwx------ 10379780    8192 May 09 11:33 mkdir.exe
-rwx------ 10379780   12288 May 09 11:33 mv.exe
-rwx------ 10379780   16384 May 09 11:33 rm.exe
-rwx------ 10379780    8192 May 09 11:32 rmdir.exe
-rwx------ 10379780    8192 May 09 11:32 sleep.exe
drwx------ 10379780         May 10 08:50 spool/
drwx------ 10379780         May 10 08:50 tmp/
-rwx------ 10379780    9216 May 09 11:32 uname.exe
```

- Specify the `-q` option to show unprintable characters as a question mark (`?`).

```
C:\TEMP>%ADSH_OSCMD_DIR%\ls -q ..\dir1
.sub1     file2.txt sub4      wc2.c     wc4.c
.sub2     sub3      wc1.c     wc3.c     ????.txt
```

- Specify the `-R` option to list subdirectories recursively.

```
C:\TEMP>%ADSH_OSCMD_DIR%\ls -R ..\dir1
.sub1     file2.txt sub4      wc2.c     wc4.c
.sub2     sub3      wc1.c     wc3.c     ????.txt

..\dir1\.sub1:

..\dir1\.sub2:

..\dir1\sub3:

..\dir1\sub4:
```

- Specify the `-r` option to list entries sorted in reverse order.

```
C:\TEMP>%ADSH_OSCMD_DIR%\ls -r
spool        hostname.exe date.exe     Adshuxpl.dll
sleep.exe    mv.exe       grep.exe     cp.exe
uname.exe    rmdir.exe    mkdir.exe    cmp.exe
tmp          rm.exe       ls.exe       cat.exe
C:\TEMP>%ADSH_OSCMD_DIR%\ls -rl
total 1069363
-rwx------ 10379780    9216 May 09 11:32 uname.exe
drwx------ 10379780         May 10 08:50 tmp
drwx------ 10379780         May 10 08:50 spool
-rwx------ 10379780    8192 May 09 11:32 sleep.exe
-rwx------ 10379780    8192 May 09 11:32 rmdir.exe
-rwx------ 10379780   16384 May 09 11:33 rm.exe
-rwx------ 10379780   12288 May 09 11:33 mv.exe
-rwx------ 10379780    8192 May 09 11:33 mkdir.exe
-rwx------ 10379780   22528 May 09 16:27 ls.exe
-rwx------ 10379780    7680 May 09 11:33 hostname.exe
-rwx------ 10379780   43008 May 09 11:33 grep.exe
-rwx------ 10379780   10240 May 09 11:34 date.exe
-rwx------ 10379780   18432 May 09 11:34 cp.exe
-rwx------ 10379780   10240 May 09 11:34 cmp.exe
-rwx------ 10379780   10752 May 09 11:34 cat.exe
-rw------- 10379780  439808 May 09 11:31 Adshuxpl.dll
```

- Specify the `-S` option to sort by size, from largest to smallest.

```
C:\TEMP>%ADSH_OSCMD_DIR%\ls -S
Adshuxpl.dll rm.exe       cmp.exe      mkdir.exe
ls.exe       mv.exe       date.exe     rmdir.exe    spool
cat.exe      sleep.exe    tmp          grep.exe     cp.exe
uname.exe    hostname.exe
C:\TEMP>%ADSH_OSCMD_DIR%\ls -lS
total 1069363
-rw------- 10379780  439808 May 09 11:31 Adshuxpl.dll
-rwx------ 10379780   43008 May 09 11:33 grep.exe
-rwx------ 10379780   22528 May 09 16:27 ls.exe
-rwx------ 10379780   18432 May 09 11:34 cp.exe
-rwx------ 10379780   16384 May 09 11:33 rm.exe
```

```
-rwx------ 10379780   12288 May 09 11:33 mv.exe
-rwx------ 10379780   10752 May 09 11:34 cat.exe
-rwx------ 10379780   10240 May 09 11:34 cmp.exe
-rwx------ 10379780   10240 May 09 11:34 date.exe
-rwx------ 10379780    9216 May 09 11:32 uname.exe
-rwx------ 10379780    8192 May 09 11:33 mkdir.exe
-rwx------ 10379780    8192 May 09 11:32 rmdir.exe
-rwx------ 10379780    8192 May 09 11:32 sleep.exe
-rwx------ 10379780    7680 May 09 11:33 hostname.exe
drwx------ 10379780         May 10 08:50 spool
drwx------ 10379780         May 10 08:50 tmp
```

- Specify the `-s` option to display the number of blocks for each file. In Windows, `0` is displayed.

```
C:\TEMP>%ADSH_OSCMD_DIR%\ls -sl
total 1070036
0 -rw------- 10379780  439808 May 09 11:31 Adshuxpl.dll
0 -rw------- 10379780      90 May 10 10:46 abc.txt
0 -rw------- 10379780     152 May 10 15:25 abcd.txt
0 -rw------- 10379780     179 May 10 11:01 abcde.txt
0 -rw------- 10379780     146 May 10 15:22 abcdex.txt
0 -rwx------ 10379780   10752 May 09 11:34 cat.exe
0 -rwx------ 10379780   10240 May 09 11:34 cmp.exe
0 -rwx------ 10379780   18432 May 09 11:34 cp.exe
0 -rwx------ 10379780   10240 May 09 11:34 date.exe
0 -rw------- 10379780     106 May 10 13:58 file1.txt
0 -rwx------ 10379780   43008 May 09 11:33 grep.exe
0 -rwx------ 10379780    7680 May 09 11:33 hostname.exe
0 -rwx------ 10379780   22528 May 09 16:27 ls.exe
0 -rwx------ 10379780    8192 May 09 11:33 mkdir.exe
0 -rwx------ 10379780   12288 May 09 11:33 mv.exe
0 -rwx------ 10379780   16384 May 09 11:33 rm.exe
0 -rwx------ 10379780    8192 May 09 11:32 rmdir.exe
0 -rwx------ 10379780    8192 May 09 11:32 sleep.exe
0 drwx------ 10379780         May 10 08:50 spool
0 drwx------ 10379780         May 10 08:50 tmp
0 -rwx------ 10379780    9216 May 09 11:32 uname.exe
```

- Specify the `-T` option to display time information as the month, date, hour, minute, second, and year.

```
C:\TEMP>%ADSH_OSCMD_DIR%\ls -lT
total 1069363
-rw------- 10379780  439808 May 09 11:31:40 2011 Adshuxpl.dll
-rwx------ 10379780   10752 May 09 11:34:28 2011 cat.exe
-rwx------ 10379780   10240 May 09 11:34:20 2011 cmp.exe
-rwx------ 10379780   18432 May 09 11:34:35 2011 cp.exe
-rwx------ 10379780   10240 May 09 11:34:13 2011 date.exe
-rwx------ 10379780   43008 May 09 11:33:44 2011 grep.exe
-rwx------ 10379780    7680 May 09 11:33:34 2011 hostname.exe
-rwx------ 10379780   22528 May 09 16:27:40 2011 ls.exe
-rwx------ 10379780    8192 May 09 11:33:15 2011 mkdir.exe
-rwx------ 10379780   12288 May 09 11:33:53 2011 mv.exe
-rwx------ 10379780   16384 May 09 11:33:01 2011 rm.exe
-rwx------ 10379780    8192 May 09 11:32:54 2011 rmdir.exe
-rwx------ 10379780    8192 May 09 11:32:49 2011 sleep.exe
drwx------ 10379780         May 10 08:50:19 2011 spool
drwx------ 10379780         May 10 08:50:19 2011 tmp
-rwx------ 10379780    9216 May 09 11:32:44 2011 uname.exe
```

- Specify the `-t` option to sort files in the order of most recent modification date and time.

```
C:\TEMP>%ADSH_OSCMD_DIR%\ls -t
date.exe     hostname.exe rmdir.exe
spool        cp.exe       mkdir.exe    sleep.exe
tmp          cat.exe      mv.exe       uname.exe    Adshuxpl.dll
ls.exe       cmp.exe      grep.exe     rm.exe

C:\TEMP>%ADSH_OSCMD_DIR%\ls -lt
total 1069363
drwx------ 10379780         May 10 08:50 spool
drwx------ 10379780         May 10 08:50 tmp
-rwx------ 10379780   22528 May 09 16:27 ls.exe
-rwx------ 10379780   18432 May 09 11:34 cp.exe
-rwx------ 10379780   10752 May 09 11:34 cat.exe
-rwx------ 10379780   10240 May 09 11:34 cmp.exe
-rwx------ 10379780   10240 May 09 11:34 date.exe
-rwx------ 10379780   12288 May 09 11:33 mv.exe
-rwx------ 10379780   43008 May 09 11:33 grep.exe
-rwx------ 10379780    7680 May 09 11:33 hostname.exe
-rwx------ 10379780    8192 May 09 11:33 mkdir.exe
-rwx------ 10379780   16384 May 09 11:33 rm.exe
-rwx------ 10379780    8192 May 09 11:32 rmdir.exe
-rwx------ 10379780    8192 May 09 11:32 sleep.exe
-rwx------ 10379780    9216 May 09 11:32 uname.exe
-rw------- 10379780  439808 May 09 11:31 Adshuxpl.dll
```

- Specify the `-x` option to list entries in multiple columns sorted horizontally.

```
C:\TEMP>%ADSH_OSCMD_DIR%\ls -x
Adshuxpl.dll cat.exe      cmp.exe      cp.exe
date.exe       grep.exe     hostname.exe ls.exe       mkdir.exe
mv.exe         rm.exe       rmdir.exe    sleep.exe    spool
tmp          uname.exe
```

- Specify the `--full-time` option to display the date and time information in the complete long format.

```
C:\Program Files\HITACHI\JP1AS\JP1ASE\cmd>ls --full-time
total 2638901
-rwx------ SYSTEM  327168 2014-01-10 19:47:42.000000000 +0900 awk.exe
-rwx------ SYSTEM   10240 2014-01-10 19:45:32.000000000 +0900 basename.exe
-rwx------ SYSTEM   12800 2014-01-10 19:48:44.000000000 +0900 cat.exe
-rwx------ SYSTEM   11264 2014-01-10 19:48:44.000000000 +0900 cmp.exe
-rwx------ SYSTEM   19968 2014-01-10 19:48:40.000000000 +0900 cp.exe
-rwx------ SYSTEM   14848 2014-01-10 19:48:04.000000000 +0900 cut.exe
-rwx------ SYSTEM   10240 2014-01-10 19:48:36.000000000 +0900 date.exe
-rwx------ SYSTEM  237056 2014-01-10 19:48:14.000000000 +0900 diff.exe
-rwx------ SYSTEM  224256 2014-01-10 19:45:28.000000000 +0900 egrep.exe
```

- Display option error massages:

  This message might vary depending on the platform used to execute the command. The following is an example for Windows:

```
C:\>ls -z
ls: illegal option -- z
usage: ls [-1AaCcdFfghikLlmnpqRrSsTtux] [--format=word] [--full-time]
          [--indicator-style=word] [--sort=word] [--time=word] [file ...]
```

mkdir command (creates directories)

Format

```
mkdir [-p] [-m permissions] directory ...
```

Description

This command creates directories.

Arguments

-p

Specifies that missing intermediate directories are to be created as needed.

-m permissions

In UNIX, specifies permissions that are to be set for the created directories. The `umask` value is not applied.

You can specify the permissions either as an octal number or using symbols.

If you specify a number, an error occurs if you specify a non-octal number or an octal value greater than `07777` (`4095` decimal).

If you specify symbols, do so by setting, adding, and removing permissions, starting from the state in which nothing is specified (`0` in numeric representation).

A symbol consists of three parts. Specify one or more of the symbols, as explained below. If you specify more than one, separate them with the comma (`,`).

Order within the symbol	Permitted value
First	Specifies the items for which you want to set access permissions. You can specify more than one at the same time. The items below can be specified. If nothing is specified, `a` (all users) is assumed. • `u`: Owner • `g`: Group • `o`: Other • `a`: All users
Second	Specifies an operation on the mode. The following processing is performed on the items specified in the first part of a symbol. • `=`: Set (overwrite) access permissions. • `+`: Add access permissions. • `-`: Remove access permissions. The value to be set, added, or removed is specified in the third part of the symbol. You can specify the second and third parts of a symbol following the third part. The third part of a symbol can be omitted.
Third	Specifies the applicable access permissions. You can specify more than one at the same time. The following values can be specified: • `r`: Read. • `w`: Write. • `x`: Execute. • `s`: Set the user ID or group ID at run time. • `t`: Sticky bit • `u`: Owner access permissions currently set in the mode. • `g`: Group access permissions currently set in the mode. • `o`: Other access permissions currently set in the mode.

Order within the symbol	Permitted value
Third	When this part is omitted and = is specified in the second part of the symbol, the items for which access permissions are set are cleared. When this part is omitted and + or - is specified in the second part of the symbol, no processing occurs. Specifying `s` or `t` in this part will be ignored if only `o` is specified in the first part.

The following table shows examples of specifying symbols:

Value specified in -m	Numeric equivalent	Description
`u=x, g=w`	`120`	Set `u` to `x`, and set `g` to `w`.
`u=x, g=u`	`110`	Set `u` to `x`, and set the same values for `g` and `u`.
`u=x, =u`	`111`	Set `u` to `x`, and then set `a` (the default value) to the same values as `u`.
`u=x, u=w`	`200`	Set `u` to `x`, and then set (overwrite) `u` to `w`.
`u=x, u+w`	`300`	Set `u` to `x`, and then add `w` to `u`.
`ug=x`	`110`	Set `u` and `g` to `x`.
`u=rw`	`600`	Set `u` to `r` and `w`.
`u=r+x`	`500`	Set `u` to `r`, and then add `x`.
`u=r=w`	`200`	Set `u` to `r`, and then set (overwrite) it to `w`.
`=x, u=`	`011`	Set `a` (the default value) to `x`, and clear the settings for `u`.
`=`	`000`	Clear `a` (the default value).

In Windows, this specification is ignored.

directory

Specifies a name for a directory to be created. Multiple directory names can be specified.

Return codes

Return code	Meaning
`0`	Normal termination
`1` or greater	Error termination

Notes

- In Windows, the `-m` option is ignored. The mode cannot be specified.

Usage examples

- Create directory `Dir2` under `C:\USR\JP1`.

```
C:\TEMP>%ADSH_OSCMD_DIR%\mkdir C:\USR\JP1\Dir2
```

- Display an option error message.

  This message might vary depending on the platform on which the command is executed. The following shows an example in Windows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\mkdir -w
mkdir: illegal option -- w
usage: mkdir [-p] [-m mode] directory ...
```

mv command (moves files or directories)

Format

```
mv [-f] [-i] source destination
mv [-f] [-i] source ... destination-directory
```

Description

This command moves files or directories. It can also be used to change the name of a file or directory.

Arguments

-f

Specifies that path overwriting is to be performed without requesting confirmation. This option is ignored if it is specified before the `-i` option.

-i

Specifies that confirmation is to be requested before overwriting is performed. Overwriting will be performed only if the reply `y` or `Y` is read from the standard input. This option is ignored if it is specified before the `-f` option.

source

Specifies the path name to be moved. Multiple path names can be specified.

destination

Specifies the destination path name. You can also change the name of a file or directory by specifying a path name for both *source* and *destination*.

destination-directory

Specifies the destination directory name. You can move more than one file or directory by specifying multiple path names in *source*.

Return codes

Return code	Meaning
`0`	Normal termination
`1` or greater	Error termination

Notes

- If the `-i` and `-f` options are both specified, the one specified last takes effect.
- In Windows, when a file or directory is overwritten, access permissions other than the owner's are not displayed. For details about the permissions that are displayed, see *ls command (lists the contents of files or directories)*.
- In Windows, symbolic links are not supported.
- In Windows, group and mode are not preserved.

- In Windows, the destination file name will be created using the file name specified in the source. However, uppercase letters in the file name will be replaced with lowercase letters. For example, if the name of the file to be moved is `A.txt` and you execute `mv a.txt tmpdir`, the name of the file in `tmpdir` will be `a.txt`.
- In Windows, file input and output are performed in the binary mode. No conversion of end-of-line codes is performed.
- In UNIX, when the `mv` command is used to move a file or directory and all the following conditions are satisfied, the user who executes the `mv` command becomes the owner of the file or directory:
    - A general user executed the `mv` command.
    - The user executing the `mv` command is different from the owner of the source file.
    - The source and destination file systems are different.

  In addition, the following information will not be inherited:
    - The access permission information set in the `setuid` and `setgid` bits of the source file
    - The access permission information set in the `setuid`, `setgid`, and sticky bits of the source directory

Usage examples

- Specify the `-i` option to require confirmation before the destination file is overwritten.

```
C:\TEMP>%ADSH_OSCMD_DIR%\mv -i ..\dir1\file1.txt  ..\dir1\file2.txt
overwrite ..\dir1\file2.txt?
```

- Display an option error message.

  This message might vary depending on the platform on which the command is executed. The following shows an example in Windows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\mv -w
mv: illegal option -- w
usage: mv [-fi] source target
       mv [-fi] source ... directory
```

- Display an error message if a file does not exist.

```
C:\TEMP>%ADSH_OSCMD_DIR%\mv file3.txt file4.txt
mv: file3.txt: No such file or directory
```

paste command (concatenates multiple files in lines)

Format

```
paste[-s][-d list][path-name ...]
```

Description

This command concatenates multiple files in units of lines and then outputs the results to the standard output. You can also join all lines in a file into a single continuous line and then concatenate multiple files.

Arguments

-s

Joins all lines in the files into a single line (with separators).

If the `-s` option is omitted, the command joins the lines in all files that have the same line number (with separators).

-d list

Specifies a list of the separators that will be inserted between the lines to be concatenated. If the `-d` option is omitted, the tab character is assumed.

To specify a space or tab character, enclose the character in double quotation marks (`"`).

You can also specify special characters as separators.

Permitted special character	Meaning	Remarks
`\n`	End-of-line code	In Windows, an end-of-line code is denoted by `[CR]` + `[LF]`. In UNIX, an end-of-line code is denoted by `[LF]`.
`\t`	Tab character	--
`\\`	One backslash character	--
`\0`	Null character string	Character string with a length of zero (`""`). No separator is inserted between the lines to be concatenated.

Legend:

--: Not applicable.

Depending on the shell used to execute the `paste` command, special characters are treated as escape characters. Therefore, enclose special characters in double or single quotation marks (`"` or `'`). If a non-special character is immediately preceded by a backslash (`\`), the command ignores `\` and uses the character following `\` as the separator. If only `\` is specified, the command terminates with an error.

You can specify multiple separators. When multiple separators are specified, the command handles them as follows:

- Each time the command concatenates lines, it fetches a separator and inserts it between the lines. The separators are fetched in order from the beginning of the list.
- When the `-s` option is not specified, the command fetches separators in order from the beginning of the list again after it has output concatenated lines.
- When the `-s` option is specified, the command fetches separators in order from the beginning of the list again after it has joined all lines in a file and has output them.
- When the end of the list of separators specified in the `-d` option is reached, the command fetches separators from the list again in order from the beginning of the list.

path-name

Specifies the path name of a file to be concatenated and output. If no path name is specified or a hyphen (`-`) is specified as the path name, the command reads the path name from the standard input.

You can specify multiple path names and hyphens (`-`) or a mixture of path names and hyphens.

If multiple files are specified and an open error occurs on any of the files, the following occurs:

- When the `-s` option is omitted, the command outputs an error message and terminates with return code `1`. In this case, nothing is output to the standard output.
- When the `-s` option is specified, the command outputs an error message for the file resulting in the open error and resumes processing. When the command has processed all files, it terminates with return code `1`.

If only one path name is specified, the command runs as follows:

- When the `-s` option is omitted, the command only outputs the lines.
- When the `-s` option is specified, the command joins all lines in the file and then outputs them.

Input and output of lines

In input files, the command treats a record separated by an end-of-line code as one line.

- In Windows, `[CR]` + `[LF]` or by `[LF]` is treated as the end-of-line code.
- In UNIX, `[LF]` is treated as the end-of-line code.
  If each record in the input file is separated by `[CR]` + `[LF]`, the concatenated lines contain `[CR]`.

An end-of-line code is output at the end of the concatenated lines. The following is output as the end-of-line code:

- In Windows: `[CR]+[LF]`
- In UNIX: `[LF]`

Concatenating files in units of lines (when the -s option is omitted)

The command joins the lines that have the same line number in all the files and then outputs the results as a single line (with separators). Null lines are treated as null character strings when they are joined with other lines.

If the end-of-file is detected in any of the files while lines with the same line number are being read, any remaining lines for that file are joined as null characters with the lines from the other files.

The following example concatenates `file1`, `file2`, `file3`, and `file4` in units of lines.

Contents of `file1`:

```
a001
a002
```

Contents of `file2`:

```
b001
(null line)
b003
```

Contents of `file3`:

```
c001
c002
c003
c004
```

Contents of `file4`:

```
d001
```

Command that concatenates `file1`, `file2`, `file3`, and `file4` in units of lines:

```
$ paste file1 file2 file3 file4
```

The command concatenates the files as follows (`->` indicates a tab character used as a separator):

```
a001->b001->c001->d001   1.
a002->->    c002->       2.
->    b003->c003->       3.
->    ->    c004->       4.
```

1. The command joins the first lines of `file1`, `file2`, `file3`, and `file4` and then outputs the results. The command inserts a tab character between the lines.
2. The command joins the following values with tab characters and then outputs the results:
    - Contents of the second line of `file1`
    - Null character string because the second line of `file2` is the null line
    - Contents of the second line of `file3`
    - Null character string because the end of file is reached in `file4`

   A null character string is a character string with a length of 0. Therefore, the actual result that is output is the *contents of the second line of file1 + tab character + tab character + contents of the second line of file3 + tab character*.
3. The command joins the following values with tab characters and then outputs the results:
    - Null character string because the end-of-file is reached in `file1`
    - Contents of the third line of `file2`
    - Contents of the third line of `file3`
    - Null character string because the end of file is reached in `file4`

   The actual result that is output is a *tab character + contents of the third line of file2 + tab character + contents of the third line of file3 + tab character.*
4. The command joins the following values with tab characters and then outputs the results:
    - Null character string because the end-of-file is reached in `file1`
    - Null character string because the end-of-file is reached in `file2`
    - Contents of the fourth line of `file3`
    - Null character string because the end-of-file is reached in `file4`

   The actual result that is output is a *tab character + tab character + contents of the fourth line of file3 + tab character.*

If any of the specified files is empty, lines in that file are treated as null character strings and joined with other files' lines. However, if all the files specified in the argument are empty, no line is output.

Joining lines in files (when the -s option is specified)

The command joins all lines in one file with separators into a single line and then concatenates it with other files. If a file is empty, the command outputs only an end-of-line code. If all the files specified in the argument are empty, the command outputs an end-of-line code for each file.

The following example concatenates `file1`, `file2`, `file3`, and `file4`.

Contents of `file1`:

```
a001
a002
```

Contents of `file2`:

```
Null file
```

Contents of `file3`:

```
c001
c002
c003
c004
```

Contents of `file4`:

```
d001
```

Command that concatenates `file1`, `file2`, `file3`, and `file4`:

```
$ paste -s file1 file2 file3 file4
```

The command concatenates the files as follows (`->` indicates a tab character used as a separator):

```
a001->a002                 1.
                           2.
c001->c002->c003->c004     3.
d001                       4.
```

1. Joins all lines in `file1` with separators and then outputs the results.
2. Outputs only an end-of-line code because `file2` is an empty file.
3. Joins all lines in `file3` with separators and then outputs the results.
4. Outputs the contents of the line because `file4` contains only one line.

Joining lines read from the standard input

This subsection explains joining lines that are read from the standard input.

Concatenating files in units of lines (when the `-s` option is omitted)

The command reads only one line from the standard input and joins that line with a line from another file. If multiple hyphens (`-`) are specified, the command reads a line from the standard input for each `-` specified in order, and then joins the lines.

When a file is to be concatenated with the contents of the standard input, the command keeps reading lines from the standard input until the end-of-file is reached. Therefore, if `EOF` is read from the standard input, the command treats it as the null character string and joins that null character string with a line from the file.

The following example reads `file1` from the standard input and concatenates it with `file2` in units of lines.

Contents of `file1`:

```
a001
a002
a003
a004
a005
```

Contents of `file2`:

```
b001
b002
b003
b004
```

Command that concatenates `file1` and `file2` in units of lines:

```
$ cat file1 | paste - file2 -
```

The lines are joined in the following order:

1. Line input from `file1` (read from the standard input)
2. Line input from `file2`
3. Line input from `file1` (read from the standard input)

The command concatenates the files as follows (`->` indicates a tab character used as a separator):

```
a001->b001->a002     1.
a003->b002->a004     2.
a005->b003->         3.
->    b004->         4.
```

1. The command joins the following values with tab characters and then outputs the results:
   - Contents of the line read from the standard input (contents of the first line in `file1`)
   - Contents of the first line in `file2`
   - Contents of the line read from the standard input (contents of the second line in `file1`)
2. The command joins the following values with tab characters and then outputs the results:
   - Contents of the line read from the standard input (contents of the third line in `file1`)
   - Contents of the second line in `file2`
   - Contents of the line read from the standard input (contents of the fourth line in `file1`)
3. The command joins the following values with tab characters and then outputs the results:
   - Contents of the line read from the standard input (contents of the fifth line in `file1`)
   - Contents of the third line in `file2`
   - Null character string because `EOF` was read from the standard input (the end of file was reached in the file sent to the standard input)
4. The command joins the following values with tab characters and then outputs the results:
   - Null character string because `EOF` was read from the standard input (the end of file was reached in the file sent to the standard input)
   - Contents of the fourth line in `file2`
   - Null character string because `EOF` was read from the standard input (the end of file was reached in the file sent to the standard input)

Joining lines in files (when the `-s` option is specified)

The command repeats reading one line at a time from the standard input and joins the lines with separators until `EOF` is read. The command then concatenates the joined lines with another file. The following example reads `file1` from the standard input and concatenates it with `file2`.

Contents of `file1`:

```
a001
a002
a003
```

Contents of `file2`:

```
b001
b002
```

Command that concatenates `file1` and `file2` (`file1` is read from the standard input):

```
$ cat file1 | paste -s - file2
```

The command concatenates the files as follows (`->` indicates a tab character used as a separator):

```
a001->a002->a003    1.
b001->b002          2.
```

1. Joins all lines read from the standard input (all lines in `file1`) with separators, and then outputs them.
2. Joins all lines read from `file2` with separators, and then outputs them.

Return code

Return code	Meaning
0	Normal termination
1	Error termination • At least one of the files specified in the argument failed to open. If the `-s` option was omitted, the files are not concatenated. If the `-s` option was specified, an error message is output when a file open error occurs, and then the next file is processed.
2	Error termination • An invalid option was specified.
3	Error termination • An unresumable error occurred, such as a memory shortage.

Notes

- The `paste` command expects text files. Input from binary files and output of binary data are not guaranteed to work.
- When the `-s` option is omitted, the command opens all the files specified in the argument simultaneously. In UNIX, an error might occur when files are opened depending on OS settings, such as the maximum number of files that can be open at the same time in the entire OS or the maximum number of file descriptors permitted for a process (`ulimit`).

Usage examples

- Concatenate multiple files in units of lines. Use the tab character as the separator between the lines that are joined. In the output results, `->` indicates the tab character.

  Contents of input file `file01`:

```
a001
a002
a003
```

Contents of input file `file02`:

```
b001
b002
b003
```

Contents of input file `file3`:

```
c001
c002
c003
```

The following shows the specified command and the execution results:

```
$ paste file01 file02 file03
a001->b001->c001
a002->b002->c002
a003->b003->c003
```

- Concatenate multiple files in units of lines. Insert the separators = and `%` one at a time in this order between the lines that are joined.

  Contents of input file `file01`:

```
a001
a002
a003
```

Contents of input file `file02`:

```
b001
b002
b003
```

Contents of input file `file03`:

```
c001
c002
c003
```

Contents of input file `file04`:

```
d001
d002
d003
```

The following shows the specified command and the execution results:

```
$ paste -d "=%" file01 file02 file03 file04
a001=b001%c001=d001
a002=b002%c002=d002
a003=b003%c003=d003
```

- Concatenate multiple files in units of lines. Insert the separators =, `%`, and `@` one at a time in this order between the lines that are joined.

Contents of input file `file01`:

```
a001
a002
a003
```

Contents of input file `file02`:

```
b001
b002
b003
```

Contents of input file `file03`:

```
c001
c002
c003
```

The following shows the specified command and the execution results:

```
$ paste -d "=%@" file01 file02 file03
a001=b001%c001#
a002=b002%c002
a003=b003%c003
```

#

The concatenation count for the lines with the same line number is 2 as shown in the following:

- Concatenate one line of `file01` and one line of `file02`
- Concatenate one line of `file03` and the results of concatenating `file01` and `file02`

Note that the separator `@` specified in the `-d` option is not used.

- Read the list of file names displayed with the `ls` command from the standard input and then output four columns. Insert the comma (`,`) as the separator between file names.

```
$ ls
a001  a002  a003  a004  b001  b002  b003  b004  c001  c002
$ ls | paste -d "," - - - -
a001,a002,a003,a004
b001,b002,b003,b004
c001,c002,,
```

- Join all lines in each file into a single line and then concatenate all files. Insert the separators =, `%`, and `@` one at a time in this order between the lines that are joined.

Contents of input file `file01`:

```
a001
a002
a003
```

Contents of input file `file02`:

```
b001
b002
b003
b004
```

Contents of input file `file03`:

```
c001
c002
c003
c004
c005
```

Contents of input file `file04`:

```
d001
d002
```

The following shows the specified command and the execution results:

```
$ paste -s -d "=%@" file01 file02 file03 file04
a001=a002%a003
b001=b002%b003@b004
c001=c002%c003@c004=c005
d001=d002
```

- When multiple files are concatenated in units of lines, the following message is displayed if a nonexistent file is specified:

```
$ paste file01 file02 file03
paste: file02: No such file or directory
```

- Join all lines in each file and concatenate all files. Nonexistent files `file01` and `file03` are specified as input files.

Contents of input file `file02`:

```
b001
b002
b003
b004
```

Contents of input file `file04`:

```
d001
d002
```

The following shows the specified command and the execution results:

```
$ paste -s -d "=%@" file01 file02 file03 file04
paste: file01: No such file or directory#
b001=b002%b003@b004
paste: file03: No such file or directory#
d001=d002
```

#

Contents output to the standard error output.

rm command (removes files or directories)

Format

```
rm [-d] [-f] [-i] [-R] [-r] path-name ...
```

Description

This command removes files or directories.

Arguments

-d

Specifies that file or directory removal is to be performed. In the case of a directory, the entire directory is removed.

-f

Specifies that file removal is to be performed without requesting confirmation. Nonexistent files are ignored. This option is ignored if it is specified before the `-i` option.

-i

Specifies that confirmation is to be requested before removal is performed. Removal will be performed only if the reply `y` or `Y` is read from the standard input. This option is ignored if it is specified before the `-f` option.

-R|-r

Specifies that a directory tree is to be removed recursively.

path-name

Specifies a path name to be removed. Multiple path names can be specified.

Return codes

Return code	Meaning
`0`	Normal termination • The specified file or directory removal was successful. • If the `-f` option was specified, the files that existed among the specified files were removed successfully.
`1` or greater	Error termination

Notes

- In Windows, when you confirm removal without write permissions, non-access permissions other than the owner's are not displayed.

  For details about the permissions that are displayed, see *ls command (lists the contents of files or directories)*.
- If the options `-f` and `-i` are both specified, the one specified last takes effect.

Usage examples

- Specify the `-i` option to require confirmation before removing a file.

  ```
  C:\TEMP>%ADSH_OSCMD_DIR%\rm -i file2.txt
  remove file2.txt?
  ```
- Display an error message if the file does not exist.

```
C:\TEMP>%ADSH_OSCMD_DIR%\rm c.txt
rm: c.txt: No such file or directory
```

- Display an error message when you attempt to remove a directory without using the `-d` option.

```
C:\TEMP>%ADSH_OSCMD_DIR%\rm dir8
rm: dir8: is a directory
```

- Display an error message when you attempt to remove a directory that contains files.

```
C:\TEMP>%ADSH_OSCMD_DIR%\rm -d dir8
rm: dir8: Directory not empty
```

- Display an option error message.

  This message might vary depending on the platform on which the command is executed. The following shows an example in Windows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\rm -w
rm: illegal option -- w
usage: rm [-dfiRr] file ...
```

rmdir command (removes empty directories)

Format

```
rmdir directory-name ...
```

Description

This command removes empty directories.

Arguments

directory-name

Specifies a directory that is to be removed.

Return codes

Return code	Meaning
`0`	Normal termination • Directory removal was successful.
`1` or greater	Error termination

Notes

- This command does not accept options. If you specify an option as an argument, the option is interpreted as a directory name.

Usage examples

- Remove the `dir1` directory from `D:\temp\dir1`.

```
C:\TEMP>%ADSH_OSCMD_DIR%\rmdir  D:\temp\dir1
```

- Display an error message because you are trying to remove a directory that is not empty.

```
C:\TEMP>%ADSH_OSCMD_DIR%\rmdir dir8
rmdir: dir8: Directory not empty
```

- Display an error message when you do not specify a directory to be removed.

```
C:\TEMP>%ADSH_OSCMD_DIR%\rmdir
usage: rmdir directory ...
```

sed command (replaces character strings in text)

Format

```
sed [-a] [-E] [-n] [-r] [-u] command [input-file-path-name...]
sed [-a] [-E] [-n] [-r] [-u] [-e command]... [-f script-file-path-name]...
[input-file-path-name...]
```

Description

This command sends text from a file or the standard input to the standard output with specified character strings replaced.

Arguments

-a

Specifies that if a parsing error occurs in an editing command, no new file is to be created nor is any existing file to be overwritten (which could otherwise occur unintentionally). This option is used to maintain control over when the pattern space output file used by the `w` command or by the `s` command's `w` flag is created. The pattern space output file is created when the `w` command or the `s` command's `w` flag is applied. If the `-a` option is not specified, the pattern space output file is created during parsing of the `w` command or the `s` command's `w` flag.

-E|-r

Specifies that any patterns specified in the command are to be handled as extended regular expressions. The `-E` and `-r` options function identically.

-n

Specifies that output of the pattern space to the standard output is to be suppressed. When this option is specified, output of the pattern space to the standard output will not be performed unless the `p` or `P` command is executed.

-u

In Windows, specifies that output buffering of the execution results to the standard output is to be suppressed.

In UNIX, specifies that the execution results are to be buffered on a per-record basis when output to the standard output.

command|-e command

Specifies a command for editing the input file. More than one `-e` option can be specified. If you specify more than one `-e` option, the commands will be executed in the order they are specified. When the `-f` option is not used, *command* can be specified without the preceding `-e` option.

-f script-file-path-name

Specifies the path name of a script file. The script file contains edit commands for editing the records in the input file. More than one `-f` option can be specified. If you specify more than one `-f` option, or if you specify a combination of multiple `-f` and `-e` options, the execution order of the commands will be the order in which they were specified.

input-file-path-name

Specifies the path name of an input file to be edited. More than one input file path name can be specified. If no path name is specified, the standard input is read. If more than one file is specified, each file is opened when processing of the previous file reaches the end of the file, and reading of input records from the newly opened file begins.

Editing command description format

The following is the description format for an editing command for editing an input file:

```
[address[,address]] command[ arguments]
```

An address for identifying a record to be retrieved for editing can be specified in *address*. The address can be a record line number or a search pattern character string to be used for matching

The editing command to be applied to the retrieved records is specified in *command*.

Arguments to be passed to the editing command can be specified in *arguments*.

As each record is read from the input file, it is compared to the line number or search pattern character string specified in *address*. If there is a match, the editing specified in the command is performed. If *address* is omitted, all records will be retrieved as targets for editing. The results of executing the editing command are output to the standard output.

Addresses

Addresses are used to identify the records from the input file that are to be edited.

- Line number

  Specifies a line number, where `1` represents the first record of the input file. You can use the dollar sign (`$`) to indicate the last record. If multiple input files are specified, the records are numbered continuously across the multiple files. Note that if you specify `0` for the line number (or for the starting line number when specifying a range), the editing command will not be applied to any records.

- Search pattern character string

  Specifies a search pattern character string enclosed in forward slashes (`/`) for finding a matching character string in the input records. You can specify a regular expression for the search pattern character string. The following is an example for writing to a specified file the records that contain the character string `abc`:

  ```
  /abc/w file
  ```

  Instead of enclosing the search pattern character string in forward slashes (`/`), you can use any other single-byte character (except for `\` or an end-of-line character). To use a separator other than the forward slash, specify a `\` in front of the first separator.

  The following is an example of changing the separator enclosing the search pattern character string from `/` to `#`.

  ```
  \#abc#w file
  ```

- Address range specification

You can use *address*`,` *address* to specify a range of records to be edited. The execution range for the editing command will begin with the record that matches the first address and conclude with the record that matches the second address.

A range can be specified as follows:

- A range defined by two line numbers
- A range defined by two search pattern character strings
- A range defined by a combination of a line number and a search pattern character string

The following is an example of a range defined by two line numbers (this example writes to `outfile` the records in lines 5 through 20):

```
5,20w outfile
```

The execution range of the editing command begins with the record whose line number is specified in the first address and concludes with the record whose line number is specified in the second address.

Note that if the line number specified in the first address is greater than the line number specified in the second address (first address > second address), only the record corresponding to the line number specified in the first address will be subject to the editing command.

The following is an example of a range of two search pattern character strings:

```
/abc/, /xyz/w file
```

The execution range of the editing command begins with the record that matches the search pattern character string in the first address through and including the record that matches the search pattern character string in the second address.

If the search reaches the end of an input file without finding a record that contains the search pattern character string specified in the second address, the range concludes with the last line of the input file. However, if multiple input files are specified, the search for a record that matches the search pattern character string specified in the second address continues in the next input file.

If the first address is a search pattern character string and the second address is a line number and the line number of the record that matches the search pattern character string is greater than the line number in the second address (first address > second address), only the record matching the search pattern character string is subject to the editing command.

Pattern space and hold space

The `sed` command maintains two workspaces for text editing, called the *pattern space* and the *hold space*.

The pattern space stores records that are read from the input file.

The flow of processing in the pattern space is as follows:

1. Reads one record, delimited by the end-of-line code, from the input file.

   In Windows, the end-of-line code is `[CR]` + `[LF]` or `[LF]`. In UNIX, the end-of-line code is `[LF]`. In UNIX, if the input file uses `[CR]` + `[LF]` for the end-of-line code, the `[CR]` is stored in the pattern space.

2. Copies the contents of the input record into the pattern space.

3. Executes the editing command if the pattern space includes a line number specified in the address or the search pattern character string matches a character string in the pattern space.

   If the command to be executed is the `D` command and part of the input record remains in the pattern space after the `D` command has executed, steps 1 and 2 are skipped.

4. Outputs the contents of the pattern space to the standard output, unless the `-n` option has been specified.

5. Clears the contents of the pattern space.

The hold space is used as a temporary work area for operations such as saving the contents of the pattern space in the hold space and then returning the contents of the hold space to the pattern space.

Editing commands

The following editing commands can be used in `sed`:

[*address*[,*address*]]{ *command-list*}

Creates a group of multiple editing commands that are to be applied to the matched input records. The editing commands are delimited by an end-of-line character or a semicolon (`;`). If a right curly bracket (`}`) is specified on the same line as the last editing command, there must be a semicolon after the command name.

[*address*]a\*(end-of-line)*

text

Outputs the text set in *text* to the standard output before the next input record is read. To output multiple records, you must specify `\` immediately before each end-of-line character.

In the following example, two records are sent to the standard output before the next input record is read.

```
a\(end-of-line)
text1\(end-of-line)
text2
```

[*address*[,*address*]]b[*label*]

Branches to the `:label` command defined by the specified label *label*. If *label* is not specified, branching is to the end of the script.

[*address*[,*address*]]c\*(end-of-line)*

text

Clears the contents of the pattern space. When no address or a single address is specified, the text expressed in *text* is sent to the standard output. When two addresses are specified, the text expressed in *text* is sent to the standard output after processing the last record in the selected range. If multiple records are output, you must specify `\` immediately before each end-of-line character.

After the pattern space is cleared, the next input record is read and execution restarts from the beginning, without executing any commands that appear after the `c` command.

[*address*[,*address*]]d

Clears the contents of the pattern space. The contents of the pattern space are sent to the standard output before the pattern space is cleared. Then the next input record is read and execution restarts from the beginning, without executing any commands that appear after the `d` command.

[*address*[,*address*]]D

When the pattern space holds multiple records, deletes everything up to the first end-of-line character. The contents of the pattern space are not sent to the standard output. The next input record is then read and execution restarts from the beginning, without executing any commands that appear after the `D` command.

If the pattern space is empty as a result of executing the `D` command, the next input record is read and execution restarts from the beginning of the command.

[*address*[,address]]g

Copies the contents of the hold space into the pattern space. The previous contents of the pattern space are discarded.

[*address*[,*address*]]G

Appends the contents of the hold space to the pattern space, separated from the records already stored in the pattern space by the end-of-line character.

[*address*[,*address*]]h

Copies the contents of the pattern space into the hold space. The previous contents of the hold space are discarded.

[*address*[,*address*]]H

Appends the contents of the pattern space to the hold space, separated from the records already stored in the hold space by the end-of-line character.

[*address*]i\*(end-of-line)*

text

Sends the text expressed in *text* to the standard output before storing the current input record in the pattern space. To output multiple records, you must specify a backslash (\) immediately before each end-of-line character.

[*address*[,*address*]]l

Sends the contents of the pattern space to the standard output. For data other than single-byte characters (in the range 0x20 to 0x7e), the space, and multibyte characters, each byte is output as a three-digit octal number preceded by a backslash (\). The backslash itself is output as \\, and the control codes shown in the following table are output as escape characters.

Control code	Escape character that is output
Alert character (bell)	\a
Backspace character	\b
Formfeed character (page break)	\f
End-of-line character. Note that the end-of-line character at the end of a lone line (or of the last line in the case of multiple lines) is not output.	\n
Carriage return character	\r
Tab character	\t
Vertical tab character	\v

The value for the output width of a record is determined in the following priority order:

1. Value of the COLUMNS environment variable
2. For output to the console, width of the console screen
3. 60 single-byte characters

The dollar sign ($) is output at the end of each record. If a record exceeds the output width, it is split and a backslash (\) is set at the split location.

Note that the COLUMNS environment variable cannot be defined in a job definition script in JP1/Advanced Shell.

[*address*[,*address*]]n

Sends the current contents of the pattern space to the standard output and then reads the next input record from the input file and stores it in the pattern space. The current line number is incremented by 1. If the -n option was specified, the current contents of the pattern space are not sent to the standard output.

[*address*[,*address*]]N

Reads the next input record from the input file and appends it to the pattern space, separated from the records already stored in the pattern space by the end-of-line character. The current line number is incremented by 1.

[*address*[,*address*]]p

Sends the contents of the pattern space to the standard output.

[*address*[,*address*]]P

If the pattern space contains multiple records, outputs the contents only up to the first end-of-line character. If the pattern space contains only one record, this command is the same as the p command.

[*address*]q

Terminates processing of the script. No further execution of commands or reading of input records is performed after this command. Unless the -n option was specified, the contents of the pattern space are sent to the standard output at the time of termination. In addition, any records that had been added using the a or r command are output.

[*address*]r *path-name*

Before reading the next input record, reads the file specified in *path-name* and outputs its contents to the standard output. Any errors that occur during input of the file specified in *path-name* are ignored.

In Windows, end-of-line codes in the file are output as [CR] + [LF].

In UNIX, end-of-line codes in the file are output as is.

[*address*[,*address*]]s/*pattern*/*replacement*/*flags*

Substitutes the replacement character string (*replacement*) for the first character string that matches the pattern (*pattern*) in the pattern space. Instead of using a forward slash (/) to separate s, *pattern*, and *replacement*, you can use any single-byte character except for a backslash (\) or the end-of-line character. To include the separator character within *pattern* or *replacement*, it must be preceded by a backslash (\).

You can specify a regular expression for *pattern*.

The following characters can be used in the replacement character string for the indicated purposes:

- &: The ampersand (&) is replaced by the character string that matches *pattern*. To handle an & as a character to be replaced, it must be preceded by a backslash (\).
- \N (where *N* is a digit from 1 through 9): The \N is replaced by the character string that matches a tagged regular expression enclosed in parentheses (( )) in *pattern*. The numeric digit (*N*) specifies the sequential order of the tagged regular expression to be matched.
- \: To include an end-of-line code, specify backslash (\) immediately before the end-of-line code.

The flags that can be specified in *flags* include the values shown below. The flags are optional. More than one can be specified.

N

Only replace the *N*-th matched pattern in the pattern space.

g

Globally replace all the character strings that match the pattern in the pattern space, not just the first one.

p

If any substitution was made, output the contents of the pattern space to the standard output.

w *path-name*

If any substitution was made, output the contents of the pattern space to the file specified in *path-name*. If an existing file is specified in *path-name*, the following occurs:

- If the -a option was not specified

  Regardless of whether a substitution was made, the prior contents before execution of the sed command are discarded.

- If the -a option was specified

  If a substitution was made, the prior contents before execution of the sed command are discarded.

In Windows, the end-of-line code for the file is output as [CR] + [LF].

[*address*[,*address*]]t[*label*]

Branches to the : (colon) command defined by the specified label (*label*) if a substitution has been performed by the s command since the last input record was read or since the previously-executed t command was executed. If no label is specified, branching is to the end of the script.

[*address*[,*address*]] w *path-name*

Writes the contents of the pattern space to the file specified in *path-name*. If an existing file is specified in *path-name*, the following occurs:

- If the -a option was not specified

  Regardless of whether there is a match with the address (*address*), the prior contents before execution of the sed command are discarded.

- If the -a option was specified

  If there is a match with the address (*address*), the prior contents before execution of the sed command are discarded.

In Windows, the end-of-line code for the file is output as [CR] + [LF].

[*address*[,*address*]]x

Exchanges the contents of the pattern space and the contents of the hold space.

[*address*[,*address*]]y/*string1*/*string2*/

Searches the contents of the pattern space and replaces each character string specified in *string1* with the character string specified in *string2* (this is a character-by-character replacement, so each character position in *string1* is replaced with the character at the corresponding position in *string2*).

The number of characters in *string1* and *string2* must be the same.

To specify an end-of-line character in *string1* or *string2*, specify \n. Instead of using a / to separate y, *string1*, and *string2*, you can use any other single-byte character except for \ or the end-of-line character.

[*address*[,*address*]]!*command* or [*address*[,*address*]]!{*command-list*}

Applies the command, or list of grouped commands, to the records that are *not* selected by the address (*address*).

:*label*

Defines a label for the branch destinations specified in the b and t commands. The : (colon) command itself performs no processing.

[*address*]=

Outputs the current line number to the standard output as a single record.

(blank line)

Blank lines are ignored.

#

Indicates a comment. The hash mark (#) and everything following it is treated as a comment. Note that if the first column of the first record of the script file begins with #n, the -n option is assumed.

Escape characters

The escape characters listed below can be used within an address search pattern, the text portion of the a, c, and i commands, pattern and replacement character strings of the s command, and the search characters and replacement characters of the y command.

Escape character	Meaning
`\a`	Alert character (bell)
`\b`	Backspace character[#1]
`\f`	Formfeed character (page break)
`\n`	End-of-line character[#2]
`\r`	Carriage return character
`\t`	Tab character
`\v`	Vertical tab character
`\x`*hex*	Character represented by a one- or two-digit hexadecimal value (`0` to `9`, `a` to `f`, `A` to `F`)[#3]
`\c`	Any literal character (for example, `\"` for `"`)
`\\`	A single backslash character

#1

If you specify this in an address search pattern or in the pattern in the `s` command, it is treated as the `\b` regular expression operator. However, if you specify it in a character class enclosed in square brackets (`[ ]`), it is treated as the backspace character.

#2

In Windows, this is output as `[CR] + [LF]` when specified within the text portion of the `a`, `c`, and `i` commands.

#3

There are values that cannot be specified in a pattern depending on the character encoding at the time of execution. The values that can be specified for each character encoding are listed below in hexadecimal. Execution terminates with an error if you specify any other value.

- Shift JIS

`0x01-0x80, 0xA0-0xDF, 0xFD-0xFF`

- UTF-8

`0x01-0xBF, 0xFE-0xFF`

- EUC

`0x01-0x8D, 0x90-0xA0, 0xFF`

- C

`0x01-0xFF`

Return codes

Return code	Meaning
0	Normal termination
1 or greater	Error termination

Usage examples

- Specify the `d` command to delete the first through third records of the file. The input file is `file01.txt`.

  Contents of `file01.txt`:

```
hitachi group01 Tokyo
HITACHI group02 Yokohama
hitachi group03 Fukuoka
HITACHI group04 Hokkaido
HITACHI group05 Ooita
HITACHI group06 Hiroshima
```

  The results of executing the command are as follows:

```
C:\DIR>%ADSH_OSCMD_DIR%\sed "1,3d" file01.txt
HITACHI group04 Hokkaido
HITACHI group05 Ooita
HITACHI group06 Hiroshima
```

- Search for a pattern and then use the `i` command to add two records before each matched record and the `a` command to add one record before each matched record. In addition, use the `c` command to replace with another record each record that does not match the search pattern. The script file is `scpt01.sed`, and the input file is `file02.txt`.

  Contents of `scpt01.sed`:

```
/file/{
i\
<FILE-LINE>\
 [FILE-BEGIN]
a\
 [FILE-END]
}
/file/!{
c\
<DIR-LINE>
}
```

  Contents of `file02.txt`:

```
The file path used by trace is invalid.
Don't know current directory.
Input asc file is the same as output asc file.
Cannot change directory.
Merging two asc files is started.
```

  The results of executing the command are as follows:

```
C:\DIR>%ADSH_OSCMD_DIR%\sed -f scpt01.sed file02.txt
<FILE-LINE>
 [FILE-BEGIN]
The file path used by trace is invalid.
 [FILE-END]
<DIR-LINE>
<FILE-LINE>
 [FILE-BEGIN]
Input asc file is the same as output asc file.
 [FILE-END]
<DIR-LINE>
<FILE-LINE>
 [FILE-BEGIN]
Merging two asc files is started.
 [FILE-END]
```

- Replace the first character string that matches a pattern. The input file is `file03.txt`.

  Contents of `file03.txt`:

```
Hitachi Yokohama Office Hitachi Group
Hitachi Tokyo Office Hitachi Group
Hitachi Okinawa Office Hitachi Group
Hitachi Fukuoka Office Hitachi Group
```

  The results of executing the command are as follows:

```
C:\DIR>%ADSH_OSCMD_DIR%\sed "s/Hitachi/& Corporation/" file03.txt
Hitachi Corporation Yokohama Office Hitachi Group
Hitachi Corporation Tokyo Office Hitachi Group
Hitachi Corporation Okinawa Office Hitachi Group
Hitachi Corporation Fukuoka Office Hitachi Group
```

- Format the output by replacing character strings that match a pattern. The input file is `file04.txt`.

  Contents of `file04.txt`:

```
Hitachi Yokohama Office   Hitachi Group
Hitachi Tokyo Office   Hitachi Group
Hitachi Okinawa Office   Hitachi Group
Hitachi Hokkaido Office Hitachi Group
Hitachi Fukuoka Office   Hitachi Group
```

  The results of executing the command are as follows:

```
C:\DIR>%ADSH_OSCMD_DIR%\sed "s/\(Hitachi \)\(.*\) \(Office\)/\1\3 name:
\2/" file04.txt
Hitachi Office name: Yokohama   Hitachi Group
Hitachi Office name: Tokyo   Hitachi Group
Hitachi Office name: Okinawa   Hitachi Group
Hitachi Office name: Hokkaido Hitachi Group
Hitachi Office name: Fukuoka   Hitachi Group
```

- Replace the second character string that matches a pattern. The input file is `file05.txt`.

  Contents of `file05.txt`:

```
Hitachi Yokohama Office Hitachi Group Hitachi Corporation
Hitachi Tokyo Office Hitachi Group Hitachi Corporation
Hitachi Okinawa Office Hitachi Group Hitachi Corporation
Hitachi Fukuoka Office Hitachi Group Hitachi Corporation
```

  The results of executing the command are as follows:

```
C:\DIR>%ADSH_OSCMD_DIR%\sed "s/Hitachi/& Corporation/2" file05.txt
Hitachi Yokohama Office Hitachi Corporation Group Hitachi Corporation
Hitachi Tokyo Office Hitachi Corporation Group Hitachi Corporation
Hitachi Okinawa Office Hitachi Corporation Group Hitachi Corporation
Hitachi Fukuoka Office Hitachi Corporation Group Hitachi Corporation
```

- Replace all character strings that match a pattern in the records in a specified range. The input file is `file06.txt`.

  Contents of `file06.txt`:

```
Hitachi Yokohama Office Hitachi Group
Hitachi Tokyo Office Hitachi Group
Hitachi Okinawa Office Hitachi Group
Hitachi Fukuoka Office Hitachi Group
```

  The results of executing the command are as follows:

```
C:\DIR>%ADSH_OSCMD_DIR%\sed "/Tokyo/, /Okinawa/s/Hitachi/& Corporation/g"
file06.txt
Hitachi Yokohama Office Hitachi Group
Hitachi Corporation Tokyo Office Hitachi Corporation Group
Hitachi Corporation Okinawa Office Hitachi Corporation Group
Hitachi Fukuoka Office Hitachi Group
```

- Specify the `p` flag to the `s` command to output to the standard output the records in which a substitution occurred. The input file is `file07.txt`.

  Contents of `file07.txt`:

  ```
  Hitachi Yokohama Office Hitachi Group
  Hitachi Tokyo Office Hitachi Group
  Hitachi Okinawa Office Hitachi Group
  Hitachi Fukuoka Office Hitachi Group
  ```

  Examples of executing the command are given below.

  Example output with the `-n` option

  ```
  C:\DIR>%ADSH_OSCMD_DIR%\sed -n "/Tokyo/, /Okinawa/s/Hitachi/& Corporation/
  gp" file07.txt
  Hitachi Corporation Tokyo Office Hitachi Corporation Group
  Hitachi Corporation Okinawa Office Hitachi Corporation Group
  ```

  Example output without the `-n` option

  ```
  C:\DIR>%ADSH_OSCMD_DIR%\sed "/Tokyo/, /Okinawa/s/Hitachi/& Corporation/
  gp" file07.txt
  Hitachi Yokohama Office Hitachi Group
  Hitachi Corporation Tokyo Office Hitachi Corporation Group
  Hitachi Corporation Tokyo Office Hitachi Corporation Group
  Hitachi Corporation Okinawa Office Hitachi Corporation Group
  Hitachi Corporation Okinawa Office Hitachi Corporation Group
  Hitachi Fukuoka Office Hitachi Group
  ```

- Specify the `w` flag to the `s` command to output to a file the records in which a substitution occurred. The input file is `file08.txt`.

  Contents of `file08.txt`:

  ```
  Hitachi Yokohama Office Hitachi Group
  Hitachi Tokyo Office Hitachi Group
  Hitachi Okinawa Office Hitachi Group
  Hitachi Fukuoka Office Hitachi Group
  ```

  The results of executing the command are as follows:

  ```
  C:\DIR>%ADSH_OSCMD_DIR%\sed -n "/Tokyo/, /Okinawa/s/Hitachi/&
  Corporation/gw dir\\out.txt" file08.txt
  C:\DIR>%ADSH_OSCMD_DIR%\cat dir\out.txt
  Hitachi Corporation Tokyo Office Hitachi Corporation Group
  Hitachi Corporation Okinawa Office Hitachi Corporation Group
  ```

- Output to a file the records not found within a specified range. The input file is `file09.txt`.

  Contents of `file09.txt`:

  ```
  Hitachi Yokohama Office Hitachi Group
  Hitachi Tokyo Office Hitachi Group
  Hitachi Hokkaido Office Hitachi Group
  Hitachi Okinawa Office Hitachi Group
  Hitachi Fukuoka Office Hitachi Group
  ```

  The results of executing the command are as follows:

```
C:\DIR>sed -n "/Tokyo/, /Okinawa/!w dir\\out.txt" file09.txt
C:\DIR>cat dir\out.txt
Hitachi Yokohama Office Hitachi Group
Hitachi Fukuoka Office Hitachi Group
```

- Use the `y` command to substitute characters. The input file is `file10.txt`.

  Contents of `file10.txt`:

  ```
  a b c d e a b c
  e d c b a
  ```

  The results of executing the command are as follows:

  ```
  C:\DIR>%ADSH_OSCMD_DIR%\sed "y/abcde/12345/" file10.txt
  1 2 3 4 5 1 2 3
  5 4 3 2 1
  ```

- Output to the standard output the records that match a search pattern and their line numbers. Because the first record of the script file begins with `#n`, records that do not match the search pattern are not output. The script file is `scpt02.sed`, and the input file is `prog01.awk`.

  Contents of `scpt02.sed`:

  ```
  #n
  / print/{
  =
  p
  }
  ```

  Contents of `prog01.awk`:

  ```
  BEGIN{
      print "Extract record : group03 - group06" > "file06.txt"
  }
  /group03/, /group06/{#
      count++;
      print >> "file06.txt";

  }
  END{
      printf "total record : %03d\n",  count >> "file06.txt"
  }
  ```

  #

  The processing target begins with the record that matches `group03` and concludes with the record that matches `group06`.

  The results of executing the command are as follows:

  ```
  C:\DIR>%ADSH_OSCMD_DIR%\sed -f scpt02.sed prog01.awk
  2
      print "Extract record : group03 - group06" > "file06.txt"
  6
      print >> "file06.txt";
  10
      printf "total record : %03d\n",  count >> "file06.txt"
  ```

- Use the l command so that unprintable and escape characters will be visible in the output. The input file is file11.txt.

  Contents of file11.txt:

  ```
  Hitachi(tab)Yokohama\Office(tab)HitachiGroup
  Hitachi(tab)Tokyo\Office(tab)HitachiGroup
  Hitachi(tab)Fukuoka\Office(tab)Hitachi(0x12)#Group
  ```

  #

  1 byte of data.

  The results of executing the command are as follows:

  ```
  C:\DIR>%ADSH_OSCMD_DIR%\sed -n "l" file11.txt
  Hitachi\tYokohama\\Office\tHitachiGroup$
  Hitachi\tTokyo\\Office\tHitachiGroup$
  Hitachi\tFukuoka\\Office\tHitachi\022Group$
  ```

- Read the contents of the file specified by the r command at the location of the record that matches the search pattern, and then use the d command to delete the record that matches the search pattern. The script file is scpt03.sed, and the input files are prog02.awk and header.txt.

  Contents of scpt03.sed:

  ```
  /^<Header>/{
  r header.txt
  d
  }
  ```

  Contents of prog02.awk:

  ```
  ##################################################################
  <Header>
  ##################################################################
  BEGIN{
      str = "Hitachi#YokohamaOffice#HitachiGroup"
      num = split(str,  arry, "#")
      for (i = 1; i <= num; i++ ) {
          print arry[i]
      }
  }
  ```

  Contents of header.txt:

  ```
  # Sample program
  # Hitachi group list
  ```

  The results of executing the command are as follows:

  ```
  C:\DIR>%ADSH_OSCMD_DIR%\sed -f scpt03.sed prog02.awk
  ##################################################################
  # Sample program
  # Hitachi group list
  ##################################################################
  BEGIN{
      str = "Hitachi#YokohamaOffice#HitachiGroup"
      num = split(str, arry,"#")
      for (i = 1; i <= num; i++ ) {
          print arry[i]
  ```

```
    }
}
```

- Extract blocks of records from a file. The script file is `scpt04.sed`, and the input file is `file12.txt`.

  Contents of `scpt04.sed`:

```
/^Error01/{
:LOOP
      n#1
      /Error/{
          /^Error01/b LOOP#2
          /^Error01/!d
      }
      b LOOP#2
}
d
```

#1

This outputs the current contents of the pattern space to the standard output and then reads the next record.

#2

This branches to the `LOOP` label to execute the `n` command to read the next record.

Contents of `file12.txt`:

```
Error01001
The file path used by trace is invalid.
Error02001
Don't know current directory.
Error01002
Unable to get the date for the start of the job execution.
Spool job was not deleted.
Error01003
Cannot change directory.
Error02002
Asc file name size is exceeded limits
for batch coverage function.
Error01004
Failed to get the current time.
```

The results of executing the command are as follows:

```
C:\DIR>%ADSH_OSCMD_DIR%\sed -f scpt04.sed file12.txt
Error01001
The file path used by trace is invalid.
Error01002
Unable to get the date for the start of the job execution.
Spool job was not deleted.
Error01003
Cannot change directory.
Error01004
Failed to get the current time.
```

- Use the `q` command to terminate the script after reading a record that matches the pattern. The input file is `file13.txt`.

  Contents of `file13.txt`:

```
Error01001
The file path used by trace is invalid.
Error02001
Don't know current directory.
Error01002
Unable to get the date for the start of the job execution.
Spool job was not deleted.
Error01003
Cannot change directory.
```

The results of executing the command are as follows:

```
C:\DIR>%ADSH_OSCMD_DIR%\sed "/Error01002/q" file13.txt
Error01001
The file path used by trace is invalid.
Error02001
Don't know current directory.
Error01002
```

- Display an option error message.

  This message might vary depending on the platform on which the command is executed. The following shows an example in Windows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\sed -x
sed: illegal option -- x
usage: sed [-aEnru] command [file ...]
       sed [-aEnru] [-e command] ... [-f command_file] ... [file ...]
```

sleep command (stops for a specified period of time)

Format

```
sleep seconds
```

Description

This command suspends execution for a specified period of time.

Arguments

seconds

Specifies the amount of time in seconds that execution is to be suspended. If a non-numeric value is specified, the command's usage is displayed.

Return codes

Return code	Meaning
0	Normal termination
1 or greater	Error termination

Usage examples

- Suspend execution for five seconds.

```
C:\TEMP>%ADSH_OSCMD_DIR%\sleep  5
```

- Show what happens if a non-numeric value is specified for *seconds*.

```
C:\TEMP>%ADSH_OSCMD_DIR%\sleep poipoi
usage: sleep seconds
```

sort command (sorts text files)

Format

```
sort [-c|-m] [-b] [-f] [-n] [-r] [-u] [-z]
     [-k start-position[, end-position]] [-o output-path-name]
     [-T temporary-file-directory] [-t field-delimiter]
     [input-path-name ...]
```

Description

This command reads input from files or from the standard input and performs one of the following operations, then sends the results to the standard output:

- Sort
- Merge
- Check whether the input is already sorted

Arguments

Specifying the operation

If no operation is specified, the default is to sort. The `-r` option specifies whether items are to be sorted in ascending or descending order.

-c

Specifies that a single specified file is to be checked to determine if it is already sorted. This check functionality determines whether a specified file is already correctly sorted.

If the file is in sorted order, the command terminates with a return code of `0`. If the file is not in sorted order, the command outputs a message (`sort: found disorder:`*field-contents*) to the standard error output and terminates with a return code of `1`.

Specifying more than one file when this option is specified results in an error (`sort: too many input files for the -c option`). This option takes precedence when it is specified at the same time as any other option except for the `-u` option. Specifying this option more than once does not result in an error.

If this option is not specified, the default operation is to sort.

-m

Specifies that input files are to be merged (and assumes that they are already sorted). The `-m` option is ignored if it is specified at the same time as the `-c` option. Specifying this option more than once does not result in an error.

If this option is not specified, the default operation is to sort.

Input and output specifications

-o output-path-name

Specifies a destination for the output when the output is not to be sent to the standard output.

The output file is created if it does not already exist. In UNIX, the permissions for a newly created file are set according to the umask.

If the file already exists, the sort command first sends the output to a temporary file, then renames the temporary file to the output file, which overwrites the original file. The temporary output file is created in the same directory as the input files. In UNIX, the permissions for the file are reset according to the umask.

If this option is specified more than once, the last specification takes effect.

In UNIX, if you specify `/dev/stdout` (also written in lowercase `/dev/stdout` in Windows) for the output path name, the standard output is used.

If you specify a symbolic link for the output path name, the link is deleted and a new file is created.

-T temporary-file-directory

Specifies the directory to be used internally by the sort command for creating temporary files.

A temporary file is a work file that is used for sort and merge operations that cannot be performed entirely in memory.

If this option is specified more than once, the last specification takes effect.

If this option is omitted, the following directory is used:

Windows: *common-application-data-folder*`\HITACHI\JP1AS\misc`

UNIX: Directory specified in the `TMPDIR` environment variable (`/var/tmp` if the `TMPDIR` environment variable is not defined)

input-path-name

Specifies an input file. If this option omitted or specified as `-`, the standard input is read as the input. The standard input is also read when `/dev/stdin` (also written as lowercase `/dev/stdin` in Windows) is specified.

Sort key specifications

-b

Specifies that leading spaces are to be ignored in determining the start and end positions of a sort key specified with the `-k` option. The `-b` option is valid when a sort key is specified with the `-k` option. The `-b` option cannot be specified after the `-k` option.

-f

Specifies that lowercase letters are not to be distinguished from uppercase letters for purposes of sorting. Specifying this option more than once does not result in an error.

-n

Specifies numeric sorting, with the initial numeric character string in each line handled as a number.

The `-n` option takes precedence over the `-f` option. This option can be specified multiple times.

Numeric values are handled as follows.

- A numeric value is a character string composed of the ASCII characters 0 (`0x30`) through 9 (`0x39`).
- Leading whitespaces (`0x20` and `0x09`) and zeros (`0x30`) are ignored.
- A minus sign (`0x2d`) is allowed to precede a numeric value.
- No more than one decimal point can be specified.
- A numeric value can include a digits grouping character in the integer portion.
- The decimal point and the digits grouping character in the integer portion depend on the locale. Typically, the period (`.`) is used as the decimal point, and the comma (`,`) is used as the digits grouping character.

- Anything other than a numeric character string is treated as `0`.
- Do not specify for a sort key a numeric value that consists of more than 61 digits in the integer or more than 61 digits following the decimal point in a decimal value.

-r

Specifies sorting in descending order. If this option is not specified, the default is sorting in ascending order. This option can be specified multiple times.

Specifying the field separator

-t field-delimiter

Specifies the field delimiter. The field delimiter is not considered part of the field in determining the offset for the sort key. Consecutive field delimiters denote an empty field between them. You cannot specify the same character for the record delimiter.

If the `-t` option is omitted, fields are delimited by one or more consecutive whitespaces (consecutive spaces do not denote an empty field between them). Leading spaces are considered part of the field in determining the offset for the sort key.

If you specify more than one character for the field delimiter or if you specify a multibyte character, only the initial byte is used as the field delimiter (which cannot be the same byte value as the record delimiter).

If `-t` is specified but no field delimiter value is specified, the option or file name that follows immediately will be interpreted as the field delimiter during processing. To prevent this from happening, you must make sure to specify a delimiter. It is an error to specify this option multiple times (`sort: multiple field-delimiters`).

Specifying a sort key

-k start-position[, end-position]

Specifies the start and end positions of the sort key. If you specify more than one sort key, then all lines with the same value for the first sort key can be distinguished on the basis of the next sort key.

If *start-position* is greater than *end-position*, or if a specified field does not exist, the command assumes that no sort key is specified and all comparisons against this sort key are considered to be the same.

Specify *start-position* and *end-position* in the following format:

```
field-position[.indent][bfnr]
```

- *field-position*

  Specifies the position of the field in the record. It is an error to specify a non-numeric value (`sort: missing field number`) or a negative value (`sort: field numbers must be positive`).

  You cannot specify `0` for the start position.

  If you specify `0` for the end position, the sort key is considered to extend to the end of the record.

  The maximum value that can be specified for *field-position* is the maximum value for the `int` type (overflow will occur if you specify a greater value).

  If you specify `0` for the field position of an end position, you cannot specify the *indent* described below.

- *indent*

  Specifies an offset within the field. It is an error to specify a non-numeric or a negative value (`sort: missing offset`).

  The unit for the offset indentation is bytes. If the middle of a multibyte character is specified, evaluation occurs from that byte position.

  You cannot specify `0` for the indent of the start position.

  If you specify `0` for the indent of the end position, it is treated as though no indent were specified.

The maximum value that can be specified for *indent* is the maximum value for the `int` type (overflow will occur if you specify a greater value).

If you omit *indent* in *start-position*, the default is the first byte position of the field.

If you omit *indent* in *end-position*, the default is the last byte position of the field.

- Sort key options

  Specifies the `b`, `f`, `n`, or `r` option for sorting.

  The `b` option ignores leading spaces in determining the start or end position.

  The `f` option does not distinguish between lowercase and uppercase letters in sorting.

  The `n` option sorts numerically, treating the initial numeric character string in each line as a number.

  The `r` option sorts in descending order.

  The `b` option specified in *start-position* is valid only for *start-position*, and the `b` option specified in *end-position* is valid only for *end-position*. If no indent is specified in *end-position*, specification of the `b` option is disabled. For the options other than `b`, it does not matter whether they are specified for *start-position* or *end-position* (they function the same regardless of where they are specified.

Other specifications

-u

Specifies that when multiple records have the same sort key value, only one of them is to be output. If the `-u` option is specified at the same time as the `-c` option, a check is performed for whether there are records with the same sort key value. Specifying this option more than once does not result in an error.

-z

Specifies that the record delimiter is to be changed to `NULL` (`0x00`). It is an error to specify this option more than once (`sort: multiple record delimiters`).

In Windows, end-of-line codes are removed from the input data when the input is read and then are added back during output. For this reason, binary files must not be used as the input.

Sort function

Sorting works by reading one or more input files and running comparisons against one or more sort keys. The `-k` option is used to specify fields as sort keys. The `-t` option is used to specify a field delimiter for separating each record into fields.

If no sort key is specified, the entire record is considered to constitute the sort key. Sort keys are compared on a byte-by-byte basis.

If there are multiple sort keys, the first sort key specified is compared. If a match is found, the next sort key is compared, and comparing of sort keys continues until no match is found.

If there is a match on all the sort keys, the entire record is then compared byte-by-byte. Output is produced in ascending order with the `-r` option or in descending order without the `-r` option.

Sort key options

Two types of options apply to sort keys. When you specify one or more keys, for each *global* option that can be enabled, there is a corresponding *local* option that can be specified within the `-k` option. The `-fnrb` options are specified for the sort command globally. There are also corresponding local versions of the `fnbr` options that are specified within the `-k` option to the sort command. The global options cannot be specified after the `-k` option.

b

This option is enabled globally for both the start position and the end position specified in the -k option. However, it is disabled for the end position if no indent is specified for the end position or if an indent of 0 is specified for the end position.

The -b option is valid only when the -k option is specified.

f | n | r

When any of these options is specified locally, the local specification replaces the global specifications for the applicable field.

The following example illustrates global options:

```
-bfnr -k 1,1 -k 2,2
```

In this case, the -bnfr options are enabled for both the first and second fields. They are applied to the first and second fields as follows:

- -b option: Ignores leading blanks when determining the position of the sort key.
- -f option: Does not distinguish between lowercase and uppercase letters when sorting; this is disabled if the -n option is specified.
- -n option: Sorts numerically, handling the initial numeric character string in each line as a number.
- -r option: Sorts in descending order.

The following examples illustrate the range of sort keys when the global -b option is not specified.

-k 1

The sort key extends from the first field through the end of the record.

-k 1,1

The sort key is the entire first field.

-k 1,5

The sort key extends from the initial byte of the first field through the final byte of the fifth field.

-k 1.2,5.11

The sort key extends from the second byte of the first field through the eleventh byte of the fifth field.

-k 2,1

No sort key applies, because the fields are specified in reverse order, from the second to the first.

-k 2.1b, 5.1b

The sort key extends from the first byte (excluding leading whitespaces) of the second field through the first byte (excluding leading whitespaces) of the fifth field.

-k 2.1b, 5.0b

The sort key extends from the first byte (excluding leading whitespaces) of the second field through the final byte of the fifth field.

Merge function

The merge function aligns and integrates the input data by comparing the records of each pre-sorted input file. Even if the input files are not actually sorted, merging proceeds on the assumption that they are sorted. The example below illustrates merging of file1 and file2, whose contents are as follows:

`file1`

```
AAA
DDD
```

`file2`

```
BBB
AAA
```

The following command will merge `file1` and `file2`:

```
# sort -m file1 file2
```

The file will be as follows:

```
AAA   (1st line of file1)    <-- Result of comparing 1st line of file1 to
1st line of file2
BBB   (1st line of file2)    <-- Result of comparing 2nd line of file1 to
1st line of file2
AAA   (2nd line of file2)    <-- Result of comparing 2nd line of file1 to
2nd line of file2
DDD   (2nd line of file1)    <-- 2nd line of file1, because no 3rd line in
file2
```

Option to not distinguish between lowercase and uppercase (-f option)

In the examples below, the following input records are sorted:

`file1`

```
a:B
A:b
```

Sort with lowercase and uppercase letters distinguished:

```
$ sort -t : -k 2,2 file1
```

The sort key is set to the second field, which is delimited by `:`.

The following is the output of sorting with lowercase and uppercase letters distinguished:

```
a:B
A:b
```

Because `B` is smaller than `b`, `a:B` is output first.

Sort without distinguishing between lowercase and uppercase letters:

```
$ sort -f -t : -k 2,2 file1
```

In this case, the `-f` option is specified.

The following is the output of sorting without distinguishing between lowercase and uppercase letters:

```
A:b
a:B
```

In this case, the second fields are regarded to be the same value because they are compared without distinguishing between lowercase and uppercase, as specified by the `-f` option. Because the sort key values are the same, the records are compared byte-by-byte in their entirety. As a result, since `A` is smaller than `a`, `A:b` is output first.

Return codes

Return code	Meaning
`0`	Normal termination
`1`	Normal termination • The input data is not sorted (when the `-c` option is specified). • Duplicated key values exist (when the `-c` and `-u` options are specified).
`2`	Error termination

Notes

- If processing cannot be carried out in memory, it is performed using a temporary file. If the system runs out of disk space in the course of using the temporary file, the following error message is output:

  ```
  sort: fwrite: No space left on device
  ```

  If you receive this message, use the `-T` option to specify a disk with sufficient free space.
- If you interrupt execution of the sort command, the temporary file might remain in the directory containing the output file that was specified with the `-o` option. In such a case, it must be deleted manually. Similarly, in cases where the `-o` option is omitted, the temporary file might remain and will have to be deleted manually.
- References to whitespaces in the sort command include the tab character (`\t`) as well as the space character (`0x20`). Also, when the `-z` option is specified, `\n` (end-of-line) is also considered to be a whitespace.
- When the record delimiter is missing from the last record of the input file, the result of the sort or merge operation is output with the record delimiter appended.
- Processing can be carried out with the output end-of-line codes `[CR] + [LF]` or `[LF]`, but in the case of UNIX, the `[CR]` is treated as data. Regardless of the format of the end-of-line codes in the input file, the output results will follow the end-of-line code conventions of the platform.
- If a record cannot be accommodated, memory expands so that is can be stored. An error results if sufficient memory cannot be allocated.
- The size of the sort buffer is 16 megabytes. If this amount of space is not adequate, a temporary file is created. Therefore, this command is not recommended for sorting large amounts of data.
- If the `sort` command is cancelled during processing because sort or merge processing cannot be performed only in memory, a temporary file with the name shown below might remain. Delete such a temporary file manually.

  In Windows:

  `sort`*uuuu*`.tmp` (*uuuu*: any hexadecimal character string)

  In UNIX:

  `sort`*ppppp*`.`*XXXXXX* (*ppppp*: process ID consisting of five or more digits; *XXXXXX*: any character string consisting of six characters)
- If the `-o` option is specified and the `sort` command processing is cancelled, an intermediate file with the name shown below might remain. Delete such a temporary file manually.

  In Windows:

  first-three-characters-of-the-output-destination-path-nameuuuu`.tmp` (*uuuu*: any hexadecimal character string)

In UNIX:

output-destination-file-nameppppppXXXXXX (*ppppp*: process ID consisting of five or more digits; *XXXXXX*: any character string consisting of six characters)

- If multiple `sort` commands with the same output destination path name specified in the `-o` option are executed concurrently, they might terminate with an error. In such a case, the operation cannot be guaranteed.

Usage examples

The following shows the format of the files used in the examples below to illustrate the results of executing the `sort` command.

- file1

```
yyyy:101
tttt:8
ppppppp:14
```

- file2

```
cccccc:101
ggggg:31
rrrrrrrr:5
mmmmmmm:14
```

The files listed above are used as input files in the following examples.

- Combine and sort the two text files.

```
$ sort file1 file2
cccccc:101
ggggg:31
mmmmmmm:14
ppppppp:14
rrrrrrrr:5
tttt:8
yyyy:101
```

- Sort the two combined text files in descending order based on the numeric portion.

```
$ sort -t: -n -r -k 2 file1 file2
yyyy:101
cccccc:101
ggggg:31
ppppppp:14
mmmmmmm:14
tttt:8
rrrrrrrr:5
```

- Merge three files, using the first field as the sort key.

```
$ cat s1.txt
AAA s1
DDD s1

$ cat s2.txt
BBB s2
AAA s2
```

```
$ cat s3.txt
CCC s3
111 s3

$ sort -m -k 1,1 s1.txt s2.txt s3.txt
AAA s1
BBB s2
AAA s2

CCC s3
111 s3

DDD s1

$
```

- Sort data for which the keys are the same.

```
$ cat zr1.txt
aaa:999
$ cat zr2.txt
bbb:999

$ sort -k 2,2 -t : zr2.txt zr1.txt

aaa:999
bbb:999
$
```

- Sort the first field numerically and the second field as a character string.
    - Input command

```
sort -t : -k 1n, 1  -k 2,2
```

    - Input data

```
0010:aaa
10:AAA
-1:aaa
-1.00:ZZZ
1:zzz
```

    - Execution results

```
-1.00:ZZZ
-1:aaa
1:zzz
10:AAA
0010:aaa
```

- Sort from the beginning of the third field through the end of the line without distinguishing between lowercase and uppercase, and secondarily with the second field in descending order. In this example, because the second field is specified with a local option, it does not inherit the global options, so lowercase is distinguished from uppercase in the second field.

  Input command

```
sort -t : -f -k 3  -k 2,2r
```

Input data

```
aaa:aaa:cccc
aaa:AAA:cccc
aaa:aaa:AAAA
aaa:AAA:aaaa
aaa:aaa:BBBB
aaa:AAA:bbbb
```

Execution results

```
aaa:aaa:AAAA
aaa:AAA:aaaa
aaa:aaa:BBBB
aaa:AAA:bbbb
aaa:aaa:cccc
aaa:AAA:cccc
```

- Display an option error message.

  Windows example

```
C:\TEMP>%ADSH_OSCMD_DIR%\sort -w
sort: illegal option -- w
usage: sort [-cm][-bfnruz] [-k field1[, field2]] [-o output]
       [-T dir] [-t char] [file ...]
```

  Linux example

```
$ sort -w
sort: invalid option -- w
usage: sort [-cm][-bfnruz] [-k field1[, field2]] [-o output]
        [-T dir] [-t char] [file ...]
```

  AIX example

```
$ sort -w
sort: illegal option -- w
usage: sort [-cm][-bfnruz] [-k field1[, field2]] [-o output]
        [-T dir] [-t char] [file ...]
```

- Display the message that is output when you specify a directory for the input file.

```
$ ./sort dir01
sort: dir01: Is a directory
```

- Display the message that is output when you specify a nonexistent file as an input file.

```
$ ./sort xxxx
sort: xxxx: No such file or directory
```

- Display the message that is output when you specify a temporary file directory that does not exist.

  Windows example

```
C:\TEMP>%ADSH_OSCMD_DIR%\sort -mTxxx s0.txt s0.txt s0.txt s0.txt s0.txt
s0.t
xt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt
s0.txt
```

```
s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt
sort: xxx\sort: The directory name is invalid.
```

Linux example

```
$ ./sort -mT xxxx s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt
s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt
s0.txt s0.txt s0.txt s0.txt
sort: xxxx/sort.SDm1yr: No such file or directory
```

AIX example

```
$ ./sort -mT xxxx s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt
s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt s0.txt
s0.txt s0.txt s0.txt
sort: xxxx/sort.XXXXXX: No such file or directory
```

- Display the message that is output when you specify an invalid field position.

```
C:\TEMP>%ADSH_OSCMD_DIR%\sort -k xx
sort: missing field number
```

- Display the message that is output when you specify an invalid field position.

```
C:\TEMP>%ADSH_OSCMD_DIR%\sort -k 0 s0.txt
sort: field numbers must be positive
```

- Display the message that is output when you specify an invalid indent for the field position.

```
C:\TEMP>%ADSH_OSCMD_DIR%\sort -k 1.0 s0.txt
sort: illegal offset
```

split command (splits a file)

Format

```
split [-a suffix-length]
      [-b num-bytes [k|m]|-l num-lines] [input-path-name [prefix]]
```

Description

This command splits the contents of a file or of the standard input into segments and outputs the segments to separate files.

Arguments

-a suffix-length

Specifies the length of the suffix that is to be appended to the resulting file names after the input file is split.

Specify a value in the range `1` to `254`. An error results if you specify a value outside this range (`split:` *specified-value*`: too small` or `split:` *specified-value*`: too large`) or you specify a non-numeric value (`split:` *specified-value*`: invalid`). The default is `2`. If this option is specified multiple times, the last specification takes effect.

-b num-bytes [k|m]

Specifies in bytes the data size for each output file. If you specify both this option and the `-l` option, the command displays usage information and terminates.

- `k`: Specifies that the specified value is in kilobytes (`1k` = 1,024 bytes).
- `m`: Specifies that the specified value specified is in megabytes (`1m` = 1,048,576 bytes).

If this option is specified multiple times, the last specification takes effect.

-l num-lines

Specifies the number of lines for each output file. If you specify both this option and the `-b` option at the same time, the command displays usage information and terminates. If the `-b` and `-l` options are both omitted, the default value of `1000` (lines) is used.

input-path-name

Specifies the name of the input file. If this option is omitted, the standard input is read as the input.

prefix

Specifies a prefix for each file name after the split.

The file names of the output files after the split are constructed as follows:

prefix+suffix

If a prefix is specified, that character string is used. If no prefix is specified, `x`, `y`, and `z` are used in succession.

The suffix value is a character string consisting of lowercase alphabetic letters (`a` to `z`) of the length specified in *suffix-length*. Suffixes are created automatically, incrementing in alphabetical order.

For example, if the suffix length is set at two bytes, the suffix for the first file will be `aa`, and the successive files' suffixes will be `ab`, `ac`, ..., `az`, `ba`, `bb`, ...

Return codes

Return code	Meaning
`0`	Normal termination
`1` or greater	Error termination

Notes

- The input file will be overwritten if an output file has the same name. You can prevent this by specifying a prefix that will differentiate the output files from the input file, or else you can move the input file to a directory other than the current directory.
- If not enough file names can be generated to accommodate the split files, the command will terminate with an error (`split: too many files`). However, the files that were created are not deleted. In such a case, either increase the *suffix-length* value or increase the value of *num-bytes* or *num-lines*.
- If the length of the file name obtained after split processing exceeds the system's maximum value, the command outputs the following message and results in an error.

  In Windows:

  ```
  split : file-name : No such file or directory
  ```

  In UNIX:

  ```
  split : file-name : File name too long
  ```

- In Windows, input and output are performed in the binary mode for files and for the standard input and standard output. No conversion of end-of-line codes is performed.

Usage examples

- Split the file `test1.txt` into two-line segments.

```
$ ls
test1.txt
$ cat test1.txt
0001:test1.txt
0002:test1.txt
0003:test1.txt
0004:test1.txt
0005:test1.txt
0006:test1.txt
0007:test1.txt
0008:test1.txt
0009:test1.txt
0010:test1.txt
$ split -l2 test1.txt
$ ls
test1.txt  xaa  xab  xac  xad  xae
$ cat xaa
0001:test1.txt
0002:test1.txt
$ cat xab
0003:test1.txt
0004:test1.txt
$ cat xac
0005:test1.txt
0006:test1.txt
$ cat xad
0007:test1.txt
0008:test1.txt
$ cat xae
0009:test1.txt
0010:test1.txt
$
```

- Split the file `test1.txt` into 40-byte segments.

```
$ ls
test1.txt
$ cat test1.txt
0001:test1.txt
0002:test1.txt
0003:test1.txt
0004:test1.txt
0005:test1.txt
0006:test1.txt
0007:test1.txt
0008:test1.txt
0009:test1.txt
0010:test1.txt
$ split -a 5 -b 40 test1.txt new
$ ls
newaaaaa  newaaaab  newaaaac  newaaaad  test1.txt
```

```
$ cat newaaaaa
0001:test1.txt
0002:test1.txt
0003:test1$
```

- Display an option error message.

  This message might vary depending on the platform on which the command is executed. The following shows an example in Windows:

  ```
  C:\TEMP>%ADSH_OSCMD_DIR%\split -z
  split: illegal option -- z
  usage: split [-a suffix_length]
               [-b byte_count[k|m] | -l line_count] [file [name]]
  ```

stat command (outputs the statuses of files and directories to the standard output)

Format

```
(Windows only) stat [-c format] [-t] path-name ...
(UNIX only)   stat [-L] [-c format] [-t] path-name ...
```

Description

This command outputs file or directory statuses to the standard output. If a symbolic link file is specified for *path-name*, the command displays the status of the symbolic link file without using the link.

Arguments

-L

--dereference

(UNIX only)

Specifies that if a symbolic link file is specified for *path-name*, the command is to display the status of the file or directory at the link destination.

-c format

--format=format

Specifies the format in which the status of a file or directory is to be displayed . For *format*, you can specify a format specification code and any character string. For details about the display formats and format specification codes when this option is specified, see *Unique display format* in *Display formats*. If an unsupported format specification code is specified, the command outputs a warning message to the standard error output and a question mark (`?`) to the standard output, and then resumes the subsequent processing.

If this option is specified together with the `-t` option, this option takes effect.

-t

--terse

Specifies that the information is to be displayed in the concise format. For details about the concise display format, see *Concise display format* in *Display formats*.

path-name

Specifies the name of a file or a directory whose status is to be displayed.

If multiple path names are specified, file or directory statuses are displayed vertically. If the command is executed with multiple path names specified and the status display fails even for one of the files or directories, the command terminates with return code `1`.

Return codes

Return code	Meaning
`0`	Normal termination
`1` or greater	Error termination

Display formats

The three file information display formats are the normal display format, the concise display format, and the unique display format. The display format to be used depends on the options specification.

Normal display format

This is the display format used when no option is specified. The following file information is displayed with the indicated labels:

Output information	Label
Quoted file name For a symbolic link, the reference target of the symbolic link is also displayed. (UNIX only)	`File:`
Total size	`Size:`
Number of allocated blocks	`Blocks:`
Optimum block size for file system I/O operations	`IO Block:`
File type For details about the information that is displayed, see *Information displayed as file types* in *Display formats*.	-
Device number This information is displayed in the format *device-number-in-hexadecimal*-`h`/*device-number-in-decimal*-`d`. Nothing is displayed for non-device files.	`Device:`
Inode number	`Inode:`
Number of hard links	`Links:`
Device file type This information is displayed in the format *major-device-number*`,`*minor-device-number*. Nothing is displayed for non-device files.	`Device type:`
Permissions This information is displayed in the format *permissions-in-octal*/*permissions-character-string*.	`Access:`
Owner's user information This information is displayed in the format *owner's-user-ID*/*owner's-user-name*.	`Uid:`
Owner's group information This information is displayed in the format *owner's-group-ID*/*owner's-group-name*.	`Gid:`
File's most recent access date and time	`Access:`
File's most recent modification date and time	`Modify:`

Output information	Label
Most recent date and time file information was changed	`Change:`

Legend:

-: Displayed without a label.

Concise display format

This is the display format used when the `-t` option is specified. The following information is displayed sequentially separated by the space.

- File name
- Total size
- Number of allocated blocks
- `raw` mode in hexadecimal
- Owner's user ID
- Owner's group ID
- Device number in hexadecimal
- Inode number
- Number of hard links
- Major device number
- Minor device number
- File's most recent access date and time (number of seconds since the epoch)
- File's most recent modification date and time (number of seconds since the epoch)
- Most recent date and time file information was changed (number of seconds since the epoch)
- Optimum block size for file system I/O operations

Unique display format

This is the display format used when the `-c` option is specified. You can specify a unique display format by combining format specification codes and any character strings. Also, format specification code `%` can be followed by a flag character, field width, and precision.

- Format specification codes
  The following table lists and describes the supported format specification codes:

Format specification code	Meaning
`%a`	Permissions in octal In Windows, only the owner's permissions are displayed.
`%A`	Permissions character string In Windows, only the owner's permissions are displayed.
`%b`	Number of allocated blocks In Windows, `0` is always displayed.
`%B`	Size of one block (bytes) In Windows, `0` is always displayed.

Format specification code	Meaning
%d	Device number in decimal In Windows, a drive number is displayed, but the display varies in the following cases: • For paths whose full path has no colon (:) after the drive letter The command displays - as the device number and then resumes the subsequent processing. • If the device number acquisition processing results in an error The command outputs a warning message to the standard error output, displays a question mark (?) as the device number, and then resumes the subsequent processing.
%D	Device number in hexadecimal In Windows, a drive number is displayed, but the display varies in the following cases: • For paths whose full path has no colon (:) after the drive letter The command displays - as the device number and then resumes the subsequent processing. • If the device number acquisition processing results in an error The command outputs a warning message to the standard error output, displays a question mark (?) as the device number, and then resumes the subsequent processing.
%f	raw mode in hexadecimal In Windows, only the owner's permissions are displayed.
%F	File type For details about the information that is displayed, see *Information displayed as file types* in *Display formats*.
%g	Owner's group ID In Windows, 0 is always displayed.
%G	Owner's group name In Windows, an ellipsis (...) is always displayed. In UNIX, if the owner's group name cannot be acquired, the command displays the owner's group ID and then resumes the subsequent processing.
%h	Number of hard links In Windows, 0 is always displayed.
%i	Inode number In Windows, 0 is always displayed.
%n	File name
%N	File name enclosed in quotation marks For a symbolic link, the name of the referenced file is also displayed. In Windows, the name of the referenced file is not displayed even for a symbolic link. In UNIX, if acquisition of the name of the referenced file fails, the command outputs a warning message to the standard error output and then resumes the subsequent processing without displaying the name of the referenced file.
%o	Optimum block size for file system I/O operations In Windows, 0 is always displayed.
%s	Total size (bytes) In Windows, 0 is always displayed as the total size of directories. In UNIX, 0 is always displayed as the total size of device files.
%t	Major device number in hexadecimal In Windows, 0 is always displayed.
%T	Minor device number in hexadecimal In Windows, 0 is always displayed.
%u	Owner's user ID

Format specification code	Meaning
%u	In Windows, 0 is always displayed.
%U	Owner's user name In Windows, if the owner's user name cannot be acquired, the command displays an ellipsis (...), and then resumes the subsequent processing. In UNIX, if the owner's user name cannot be acquired, the command displays the owner's user ID, and then resumes the subsequent processing.
%x	File's most recent access date and time In Windows, the file's most recent modification date and time is displayed. If the command fails to display the file's most recent access date and time, the command outputs a warning message to the standard error output, displays a question mark (?) as the file's most recent access date and time, and then resumes the subsequent processing.
%X	File's most recent access date and time Number of seconds from the epoch (UTC January 1, 1970, 00:00:00) to the file's most recent access date and time. In Windows, the file's most recent modification date and time is displayed.
%y	File's most recent modification date and time# If the command fails to display the file's most recent modification date and time, the command outputs a warning message to the standard error output, displays a question mark (?) as the file's most recent modification date and time, and then resumes the subsequent processing.
%Y	Number of seconds from the epoch (UTC January 1, 1970, 00:00:00) to the file's most recent modification date and time
%z	Most recent date and time file information was changed# In Windows, the file's most recent modification date and time is displayed. If the command fails to display the most recent date and time file information was changed, the command outputs a warning message to the standard error output, displays a question mark (?) as the most recent date and time file information was changed, and then resumes the subsequent processing.
%Z	Number of seconds from the epoch (UTC January 1, 1970, 00:00:00) to the most recent date and time the file was changed. In Windows, the file's most recent modification date and time is displayed.
%%	Percent symbol (%)

#

The following format is used for display of *file's most recent access date and time*, *file's most recent modification date and time*, and *most recent date and time file information was changed*:

YYYY-*MM*-*DD hh*:*mm*:*ss*.*nnnnnnnnn* +/-*hhmm*

- *YYYY*: Calendar year
- *MM*: Month
- *DD*: Date
- *hh*: Hour
- *mm*: Minute
- *ss*: Second
- *nnnnnnnnn*: Date and time less than one second. 000000000 is always output.
- +/-*hhmm*: Time zone (time differential from UTC)

- Flag characters

You can specify (or omit) the following flag characters following the % format specification code:

Flag character	Description
#	Prefixes with 0 an octal number other than 0. Prefixes with 0x a hexadecimal number other than 0.

Flag character	Description
-	Left-aligns the output character strings in fields.
+	Always displays a symbol (+ or -) indicating a positive or negative numeric value. This specification is ignored for file information that is defined as unsigned integers.
space	Displays a space before a positive number for file information that is defined as signed integers. If this flag character is specified together with +, + takes effect.
0	Pads the leading part of fields with zeros, not spaces.

- Field width

  You can define a minimum field width by specifying a numeric value following the % format specification code or flag character. The permitted range for field width is from 0 to 2147483647. The field width can be omitted.

- Precision

  You can define a period (.) and one of the numeric values listed below following the % format specification code or flag character. The precision range is from 0 to 2147483647 in UNIX and from 0 to 512 in Windows. The precision can be omitted.

  - If the file information is a character string

    The maximum length to be displayed is defined.

  - If the file information is a numeric value

    The minimum number of digits is defined.

Information displayed as file types

The following table lists the file types that are displayed and their meanings:

File type	Meaning
regular file	Regular file
directory	Directory
symbolic link	Symbolic link (UNIX only)
fifo	FIFO (UNIX only)
socket	Socket (UNIX only)
block special file	Block special file (UNIX only)
character special file	Character special file (UNIX only)
unknown file	Unknown file (other than the above) (UNIX only)

Notes

- In Windows, a file other than a normal file or directory is handled as a regular file or directory.

  In UNIX, a file other than a regular file, directory, symbolic link, FIFO, socket, block special file, or character special file is handled as an unknown file.

- In Windows, the time zone set in the **Date and Time** control panel is used to display the date and time. The value of the TZ environment variable has no effect.

  Note that the value of the TZ environment variable and the time zone set in the **Date and Time** control panel are used to display the time zone. For this reason, you must ensure that the value of the TZ environment variable and the time zone set in the **Date and Time** control panel are the same. If they differ, the correct time zone will not be displayed for a file's most recent access date and time, a file's most recent modification date and time, and the most recent date and time file information was updated.

- In UNIX, the default block size is 512 bytes. You can change the block size with the `BLOCKSIZE` environment variable.

  The permitted value range for the `BLOCKSIZE` environment variable is from `512` to `1G` (1,024 × 1,024 × 1,024). If the specified value is outside this range, the command handles it as described below, outputs a warning message to the standard error output, and then performs the subsequent processing:

  - If a value smaller than 512 is specified in the `BLOCKSIZE` environment variable

    The block size is set to 512 bytes.

  - If a value greater than `1G` (1,024 × 1,024 × 1,024) is specified in the `BLOCKSIZE` environment variable

    The block size is set to `1G` (1,024 × 1,024 × 1,024).

  If you use the `BLOCKSIZE` environment variable to change the block size, specify a multiple of 512. If the specified value is not a multiple of 512, the remainder will be discarded. For example, if a size of 1,500 bytes is defined, the block size will be treated as being 1,024 bytes.

  You can specify following the numeric value a size character indicating a multiple, such as `G` (1,024 × 1,024 × 1,024), `M` (1,024 × 1,024), or `K` (1,024). If any value other than a numeric value and size character is specified, the command will assume 512 bytes as the block size, output a warning message to the standard error output, and then resume the subsequent processing.

- In Solaris, the total number of blocks including indirect blocks is displayed as the number of allocated blocks for files in a directory. If there are hard-linked files, the number of allocated blocks will not display correctly.

Examples

- Display a file's status in the normal display format.

  In Windows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\stat .\test.txt
  File: `.\test.txt'
  Size: 7               Blocks: 0          IO Block: 0      regular file
Device: 0003h/00003d    Inode: 0           Links: 1
Access: (0600/-rw-------)  Uid: (    0/   user1)   Gid: (    0/     ...)
Access: 2014-02-20 10:31:28.000000000 +0900
Modify: 2014-02-20 10:31:33.000000000 +0900
Change: 2014-02-20 10:31:28.000000000 +0900
```

  In UNIX:

```
$ stat ./test.txt
  File: `./test.txt'
  Size: 4               Blocks: 8          IO Block: 4096   regular file
Device: fd00h/64768d    Inode: 688407      Links: 2
Access: (0644/-rw-r--r--)  Uid: (  501/   user1)   Gid: (  502/  group1)
Access: 2014-02-11 18:35:52.000000000 +0900
Modify: 2014-02-11 18:35:52.000000000 +0900
Change: 2014-02-18 16:08:39.000000000 +0900
```

- Display a file's size only.

```
C:\TEMP>%ADSH_OSCMD_DIR%\stat -c %s .\test.txt
7
```

- Display the statuses of multiple files.

```
C:\TEMP>%ADSH_OSCMD_DIR%\stat .\test.txt .\test1.txt
  File: `.\test.txt'
  Size: 7              Blocks: 0         IO Block: 0      regular file
Device: 0003h/00003d   Inode: 0          Links: 1
Access: (0600/-rw-------)  Uid: (    0/   user1)   Gid: (    0/     ...)
Access: 2014-02-20 10:31:28.000000000 +0900
Modify: 2014-02-20 10:31:33.000000000 +0900
Change: 2014-02-20 10:31:28.000000000 +0900
  File: '.\test1.txt'
  Size: 7              Blocks: 0         IO Block: 0      regular file
Device: 0003h/00003d   Inode: 0          Links: 1
Access: (0600/-rw-------)  Uid: (    0/   user1)   Gid: (    0/     ...)
Access: 2014-02-20 14:34:01.000000000 +0900
Modify: 2014-02-20 14:34:01.000000000 +0900
Change: 2014-02-20 14:34:01.000000000 +0900
```

- Display a file's status in the unique display format.

```
C:\TEMP>%ADSH_OSCMD_DIR%\stat -c "Filename : %n" .\test.txt
Filename : .\test.txt
```

- Display option error messages:

  This message might vary depending on the platform used to execute the command. The following is an example for Windows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\stat -z
stat: illegal option -- z
usage: stat [-c format] [-t] file ...
```

tail command (displays the last part of files)

Format

```
tail [-r] [-b num-blocks|-c num-bytes|-n num-lines|-num-lines] [path-
name ...]
```

Description

This command outputs the last part of one or more files to the standard output. If no file is specified, the standard input is read as the input. The displayed portion of a file begins at a position that is expressed in units of bytes, lines, or blocks. All data found in the specified display range is displayed. No error occurs if there is no data within the specified range.

Arguments

A numeric value prefixed with the plus sign (+) indicates a position from the beginning of the input. For example, `-c +2` starts displaying from the second byte from the beginning of the input.

A numeric value prefixed with a minus sign (-) or without a sign indicates a position from the end. For example, `-n 2` indicates the second line from the end of the input. The default is `-n 10`, or 10 lines from the end of the input.

-r

Specifies that the display is to be line-by-line in reverse order.

When the `-r` option is specified together with the `-b` option, the display is line-by-line from the end of the file for the number of blocks specified in the `-b` option. If display starts from a position that is in the middle of a multibyte character, garbled characters might result.

When the `-r` option is specified together with the `-c` option, the display is line-by-line from the end of the file for the number of bytes specified in the `-c` option. If display starts from a position that is in the middle of a multibyte character, garbled characters might result.

When the `-r` option is specified together with the `-n` option or with *num-lines* specified, the display is line-by-line from the end of the file for the number of lines specified in *num-lines* or in the `-n` option.

When the `-r` option is specified by itself, all input lines are output line-by-line in reverse order from the end of the file. Specifying this option more than once does not result in an error.

-b num-blocks

Unless the `-r` option is specified, specifies the position from which to start displaying, in units of blocks (one block is 512 bytes).

A numeric value prefixed with a minus sign (`-`) or a numeric value without a sign indicates a position counted from the end of the input. A numeric value prefixed with the plus sign (+) indicates a position counted from the beginning of the input.

If *num-blocks* is omitted, the command displays usage information and terminates with an error message (`tail: option requires an argument -` *option*). If you specify a non-numeric value for *num-blocks*, an error message (`tail: illegal offset --` *specified-character-string*) is output.

If display starts from a position that is in the middle of a multibyte character, garbled characters might result. End-of-line codes are included in the byte count. For example, in Windows, a linefeed `[LF]` counts as one byte, and `[CR]` + `[LF]` counts as two bytes. If this option is specified more than once, usage information is displayed.

-c num-bytes

Unless the `-r` option is specified, specifies the position from which to start displaying, in units of bytes.

A numeric value prefixed with a minus sign (`-`) or a numeric value without a sign indicates a position counted from the end of the input. A numeric value prefixed with a plus sign (+) indicates a position counted from the beginning of the input.

If *num-bytes* is omitted, the command displays usage information and terminates with an error message (`tail: option requires an argument -` *option*). If you specify a non-numeric value for *num-bytes*, an error message (`tail: illegal offset --` *specified-character-string*) is output.

If display starts from a position that is in the middle of a multibyte character, garbled characters might result. End-of-line codes are included in the byte count. For example, in Windows, a linefeed `[LF]` counts as one byte, and `[CR]` + `[LF]` counts as two bytes. If this option is specified more than once, usage information is displayed.

-n num-lines|-num-lines

Unless the `-r` option is specified, specifies the position from which to start displaying, in units of lines.

A numeric value prefixed with a minus sign (`-`) or a numeric value without a sign indicates a position counted from the end of the input. A numeric value prefixed with a plus sign (+) indicates a position counted from the beginning of the input.

If *num-lines* is omitted, the command displays usage information and terminates with an error message (`tail: option requires an argument -` *option*). If you specify a non-numeric value for *num-lines*, an error message (`tail: illegal offset --` *specified-character-string*) is output. If this option is specified more than once, usage information is displayed.

path-name

Specifies an input file. If no input file is specified, the standard input is read. Multiple input files can be specified. If you specify more than one file, each file is identified at the beginning of the output from that file by a blank line (linefeed) and its file name in a header string in the following format:

==> *file-name* <==

When you execute the `tail` command with multiple files specified, all the files are processed. If any file fails to open, the command terminates with a return code of `1`.

Return codes

Return code	Meaning
`0`	Normal termination
`1` or greater	Error termination

Notes

- All data found in the specified display range is displayed. No error occurs if there is no data in the specified range.
- If no options are specified, `-n 10` is assumed.
- In Windows, input and output are performed in the binary mode for files and for the standard input and standard output. No conversion of end-of-line codes is performed.

Usage examples

- Display the last two lines of the files `test1.txt` and `test2.txt`.

```
$ cat test1.txt
0001:test1.txt
0002:test1.txt
0003:test1.txt
0004:test1.txt
0005:test1.txt
0006:test1.txt
0007:test1.txt
0008:test1.txt
0009:test1.txt
0010:test1.txt
$ cat test2.txt
0001:test2.txt
0002:test2.txt
0003:test2.txt
0004:test2.txt
0005:test2.txt
0006:test2.txt
0007:test2.txt
0008:test2.txt
0009:test2.txt
0010:test2.txt
$ tail -n2 test1.txt test2.txt
==> test1.txt <==
0009:test1.txt
0010:test1.txt

==> test2.txt <==
0009:test2.txt
0010:test2.txt
$
```

- Display the fifth and subsequent lines from the beginning of the file `test1.txt`.

```
$ cat test1.txt
0001:test1.txt
0002:test1.txt
0003:test1.txt
0004:test1.txt
0005:test1.txt
0006:test1.txt
0007:test1.txt
0008:test1.txt
0009:test1.txt
0010:test1.txt
$ tail -n+5 test1.txt
0005:test1.txt
0006:test1.txt
0007:test1.txt
0008:test1.txt
0009:test1.txt
0010:test1.txt
$
```

- The following is an example of specifying the `-r` option.

```
$ cat zztt1.txt
1:0001:zzzz:
2:0001:aaaa:
3:0001:JJJJ:
4:0001:cccc:
5:0001:cccc:
6:0001:cccc:
7:0001:cccc:
8:0001:cccc:
9:0001:cccc:
10:0001:cccc:
11:0001:cccc:
12:0001:cccc:
$ tail -r -n  2 zztt1.txt
12:0001:cccc:
11:0001:cccc:
$ tail -r  zztt1.txt   (display all lines rather than just 10)
12:0001:cccc:
11:0001:cccc:
10:0001:cccc:
9:0001:cccc:
8:0001:cccc:
7:0001:cccc:
6:0001:cccc:
5:0001:cccc:
4:0001:cccc:
3:0001:JJJJ:
2:0001:aaaa:
1:0001:zzzz:
$
```

- Below are two more examples that illustrate the `-r` option.

```
$ cat block.txt  ---> 101 lines of 100 bytes + end-of-line code (\n)
0000000000:1234567890123(omitted)7890123456789012345678901234567 8T
00001xxx00:1234567890123(omitted)78901234567890123456789012345678T
```

```
00002xxx00:1234567890123(omitted)78901234567890123456789012345678T
    (omitted)
00098xxx00:1234567890123(omitted)78901234567890123456789012345678T
00099xxx00:1234567890123(omitted)78901234567890123456789012345678T
00100xxx00:1234567890123(omitted)78901234567890123456789012345678T
$ tail -b 1 block.txt
45678T
00096xxx00:1234567890123(omitted)78901234567890123456789012345678T
00097xxx00:1234567890123(omitted)78901234567890123456789012345678T
00098xxx00:1234567890123(omitted)78901234567890123456789012345678T
00099xxx00:1234567890123(omitted)78901234567890123456789012345678T
00100xxx00:1234567890123(omitted)78901234567890123456789012345678T
$ tail -rb 1 block.txt
00100xxx00:1234567890123(omitted)78901234567890123456789012345678T
00099xxx00:1234567890123(omitted)78901234567890123456789012345678T
00098xxx00:1234567890123(omitted)78901234567890123456789012345678T
00097xxx00:1234567890123(omitted)78901234567890123456789012345678T
00096xxx00:1234567890123(omitted)78901234567890123456789012345678T
45678T
$ tail -c 110 block.txt
2345678T
00100xxx00:1234567890123(omitted)78901234567890123456789012345678T
$ tail -rc 110 block.txt
00100xxx00:1234567890123(omitted)78901234567890123456789012345678T
2345678T
$
```

- Display an option error message.

  This message might vary depending on the platform on which the command is executed. The following shows an example in Windows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\tail -z
tail: illegal option -- z
usage: tail [-r] [-b number | -c number | -n number | -number] [file ...]
```

touch command (changes a file's last access date and time or modification date and time)

Format

```
touch[-a][-c][-f][-m][-r path-name][-t date-and-time]path-name ...
touch[-a][-c][-f][-m]date-and-time path-name ...
```

Description

This command changes the most recent access date and time or the most recent modification date and time for specified files. In Windows, the command can change only the most recent modification date and time.

Arguments

Type of date and time to be changed

The `-a` and `-m` options specify the type of date and time to be changed. If both these options are omitted or both are specified, the command changes both the most recent access date and time and the most recent modification date and time. In Windows, the file's most recent access date and time is not changed.

-a

Changes the file's most recent access date and time.

When the `-m` option is omitted and the `-a` option is specified, the command changes only the file's most recent access date and time and does not change the file's most recent modification date and time.

In Windows, if the `-a` option is specified, the command does not change the file's most recent access date and time, but checks the format of the date and time specified in the argument and reads the file specified in the `-r` option.

-m

Changes the file's most recent modification date and time.

When the `-a` option is omitted and the `-m` option is specified, the command changes only the file's most recent modification date and time and does not change the file's most recent access date and time.

Specifying the time to be set

The `-r` or `-t` option or *date-and-time* in *MMDDhhmm*`[`*YY*`]` format specifies the date and time to be set. If none of these options is specified, the command sets the date and time this command executes.

The range of time values permitted in the `-t` option and in *MMDDhhmm*`[`*YY*`]` format is from `1970-01-01` at `00:00:00` to `2038-01-19` at `03:14:07` in UTC (Coordinated Universal Time). The specified date and time is interpreted according to the time zone in effect when the command is executed.

If the time zone is Japan Standard Time (UTC + 9), the permitted range of time values is `1970-01-01` at `09:00:00` to `2038-01-19` at `12:14:07`. Note that in AIX and Windows, the maximum time value permitted in Japan Standard Time (UTC + 9) is `2038-01-19` at `03:14:07`.

For the time zone, the command uses the value of the `TZ` environment variable. In Windows, if the `TZ` environment variable is not set, the command uses the time zone set in the **Date and Time** control panel. In Windows, the time zone set in the `TZ` environment variable must match the time zone set in **Date and Time**.

-r path-name

Specifies that the file's most recent access and modification dates and times to be set in the files are to be obtained from the file path specified here. An obtained date and time is set as a file's most recent access date and time when the `-a` option is specified and as a file's most recent modification date and time when the `-m` option is specified.

When a directory name is specified, the command obtains the most recent access and modification dates and times from the specified directory.

If this option is specified more than once, the last option specified takes effect. If the `-r` and `-t` options are both specified, the last option specified takes effect.

-t date-and-time

Specifies a date and time to be set as the most recent access or modification date and time in the files. The specified date and time is set as the most recent access date and time when the `-a` option is specified and as the most recent modification date and time when the `-m` option is specified.

If this option is specified more than once, the last option specified takes effect.

If the `-r` and `-t` options are both specified, the last option specified takes effect.

Specify the date and time in the following format:

```
[[CC]YY]MMDDhhmm[.SS]
```

CC

First two digits of the year.

YY

Last two digits of the year

If *CC* is omitted, the following value is set as *CC*:

If *YY* is from 69 to 99: `19` is set as *CC*.

If *YY* is from 00 to 68, `20` is set as *CC*.

If *CC* and *YY* are both omitted, the year in which this command executes is set.

MM

Month, as a numeric value from 01 to 12. To specify a single-digit number, add a leading zero.

DD

Date, as a numeric value from 01 to 31. To specify a single-digit number, add a leading zero.

hh

Hour, as a numeric value from 00 to 23. To specify a single-digit number, add a leading zero.

mm

Minute, as a numeric value from 00 to 59. To specify a single-digit number, add a leading zero.

SS

Second, as a numeric value from 00 to 61. To specify a single-digit number, add a leading zero. If this specification is omitted, `00` is set.

Note that if `60` or `61` is specified and the system does not support leap seconds, the date and time displayed by the `ls` command is advanced by one second for `60` and two seconds for `61`.

date-and-time

Specifies the date and time to be set as the file's most recent access or modification date and time. The specified date and time is set as the most recent access date and time when the `-a` option is specified and as the most recent modification date and time when the `-m` option is specified.

If the `-r` or `-t` option is specified, the specified value is treated as a file name.

If the specified date and time does not consist of eight or 10 digits, it is treated as a file name. If there is no such file, a file with the specified name is created.

Specify the date and time in the following format:

```
MMDDhhmm[YY]
```

MM

Month, as a numeric value from 01 to 12. To specify a single-digit number, add a leading zero.

DD

Date, as a numeric value from 01 to 31. To specify a single-digit number, add a leading zero.

hh

Hour, as a numeric value from 00 to 23. To specify a single-digit number, add a leading zero.

mm

Minute, as a numeric value from 00 to 59. To specify a single-digit number, add a leading zero.

YY

Last two digits of the year. If the specification is omitted, the year in which this command executes is set.

For the first two digits of the year, the following value is set:

If *YY* is from 69 to 99: `19`

If *YY* is from 00 to 68: `20`

Other options

path-name

Specifies the path name of a file whose most recent access or modification date and time is to be changed. You can specify multiple path names.

If the specified path does not exist, the command creates a new file with a size of zero bytes.

In Windows:

A directory's most recent modification date and time cannot be changed. If a directory name is specified, an error results. Read and write permissions are required to change the most recent modification date and time of an existing file.

In UNIX:

Permissions for a newly created file are set according to `umask`. If a directory name is specified, the directory's most recent access and modification dates and times are changed.

For a non-superuser to change the most recent access or modification date and time of an existing file, the following permissions are required:

- When the `-t` option or a *date-and-time* value in *MMDDhhmm*`[`*YY*`]` format is specified

  File owner permissions are required.

- When neither the `-t` option nor a *date-and-time* value in *MMDDhhmm*`[`*YY*`]` format is specified

  Write permission for the file is required.

-c

Specifies that no file is to be created when there is no file whose most recent modification date and time is to be changed. No error message is output (because this event is not handled as an error).

-f

This option is provided for compatibility with the `touch` OS command. This option is ignored, if specified.

Most recent access and modification dates and times that can be set in files

How the file's most recent access and modification dates and times are set depends on the `-r` or `-t` option or the *date-and-time* value in *MMDDhhmm*`[`*YY*`]` format that specifies a date and time and the `-a` or `-m` option that specifies the type of date and time to be changed, as described in the following.

When the -r option is specified

How the most recent access and modification dates and times obtained from the file specified in the `-r` option are set depends on whether the file specified in *path-name* exists, as described in the following table.

Table 8–11: Most recent access and modification dates and times that are set when the -r option is specified

Specification of the -a and -m options	Whether the file specified in path-name exists	Most recent access date and time that is set		Most recent modification date and time that is set
		Windows	UNIX	
Only `-a` is specified	Yes	--	T	--
	No	C	T	C

Specification of the -a and -m options	Whether the file specified in path-name exists	Most recent access date and time that is set		Most recent modification date and time that is set
		Windows	UNIX	
Only `-m` is specified	Yes	--		T
	No	C		T
`-a` and `-m` are both specified or neither `-a` nor `-m` is specified	Yes	--	T	T
	No	C	T	T

Legend:

T: The corresponding date and time in the file specified in the `-r` option is set.

C: This command's execution date and time is set.

--: The most recent access or modification date and time set in the file before this command executed is set.

When the -t option or a date-and-time value in MMDDhhmm[YY] format is specified

How the date and time specified in the `-t` option or the *date-and-time* value specified in *MMDDhhmm*`[`*YY*`]` format is set depends on whether the file specified in *path-name* exists, as described in the following table.

Table 8–12: Most recent access and modification dates and times that are set when a date-and-time value is specified in the argument

Specification of the -a and -m options	Whether the file specified in path-name exists	File's most recent access date and time that is set		File's most recent modification date and time that is set
		Windows	UNIX	
Only `-a` is specified	Yes	--	T	--
	No	C	T	C
Only `-m` is specified	Yes	--		T
	No	C		T
`-a` and `-m` are both specified or neither `-a` nor `-m` is specified	Yes	--	T	T
	No	C	T	T

Legend:

T: The date and time specified in the `-t` option or the *date-and-time* value specified in *MMDDhhmm*`[`*YY*`]` format is set.

C: This command's execution date and time is set.

--: The most recent access or modification date and time set in the file before this command executed is set.

When the -r option, the -t option, and a date-and-time value in MMDDhhmm[YY] format are all omitted

When the `-r` option, the `-t` option, and a *date-and-time* value in *MMDDhhmm*[*YY*] format are all omitted, how the date and time is set depends on whether the file specified in *path-name* exists, as described in the following table.

Table 8–13: File's most recent access and modification dates and times that are set when a date-and-time value is not obtained or specified

Specification of the -a and -m options	Whether the file specified in path-name exists	File's most recent access date and time that is set		File's most recent modification date and time that is set
		Windows	UNIX	
Only the `-a` option is specified	Yes	--	C	--
	No	C		C
Only the `-m` option is specified	Yes	--		C
	No	C		C

Specification of the -a and -m options	Whether the file specified in path-name exists	File's most recent access date and time that is set		File's most recent modification date and time that is set
		Windows	UNIX	
`-a` and `-m` are both specified or neither `-a` nor `-m` is specified	Yes	--	C	C
	No	C		C

Legend:

C: This command's execution date and time is set.

--: The most recent access or modification date and time set in the file before this command executed is set.

Return code

Return code	Meaning
`0`	Normal termination
`1`	Error termination • An invalid option was specified. • The date and time specified in the `-t` option or the *date-and-time* value specified in *MMDDhhmm*`[`*YY*`]` format is invalid. • A read error occurred in the file specified in the `-r` option.
`2`	Error termination • File creation processing failed. • The command failed to change the file's most recent access date and time and/or most recent modification date and time. • A directory was specified as the path name of a file whose most recent modification date and time was to be changed. (Windows only) If multiple files are specified in the argument, the command processes the next file.

Notes

- In UNIX, if symbolic links are specified for the path name in the `-r` option or for a path name whose most recent access or modification date and time is to be changed, the specified symbolic links are subject to command processing.

  In Windows, symbolic links are not supported.

- If the date and time in the file specified in the `-r` option is outside the range from `1970-01-01` at `00:00` to `2038-01-19` at `03:14` in UTC (Coordinated Universal Time), the most recent access or modification date and time that is set in the file by the command is not guaranteed to be correct.

- In Windows, the precision of the most recent modification date and time that is actually set in the file depends on specifications for the file system being used. For example, the most recent access and modification dates and times are set in files in a FAT file system as follows:

  - The range of times that can be specified is from `1980-01-01` at `00:00:00` to `2038-01-19` at `03:14:07`, regardless of the time zone in effect when the command executes.

  - The hour, minute, and second (*hhmm*`.`*ss*) values of a file's most recent access date and time are not set.

  - The number of seconds specified in the most recent modification date and time is rounded up to the next multiple of two seconds.

Usage examples

- Create a file:

```
$ touch file001
```

The most recent access and modification dates and times of the created file are the date and time this command executes. The file is created with a size of zero.

- Change the most recent access and modification dates and times of an existing file to the date and time specified in the `-t` option (`2012-05-12` at `03:49:05`):

  ```
  $ touch -t 1205120349.05 file001
  ```

  This command changes the most recent access and modification dates and times of the file to the date and time specified in the `-t` option.

- Change only the most recent access date and time of an existing file to the date and time specified in the `-t` option (`2013-11-01` at `15:08:00`):

  ```
  $ touch -a -t 201311011508 file001
  ```

  This command changes the most recent access date and time of the file to the date and time specified in the `-t` option. The file's most recent modification date and time remains the same as before the command executed.

- Change only the most recent modification date and time of an existing file to the *date-and-time* value specified in *MMDDhhmm*`[`*YY*`]` format (`2013-09-29` at `23:00:00`):

  ```
  $ touch -m 0929230013 file001
  ```

  This command changes the most recent modification date and time of the file to the *date-and-time* value specified in *MMDDhhmm*`[`*YY*`]` format. The file's most recent access date and time remains the same as before the command executed.

- Change the most recent access and modification dates and times of multiple files to the date and time specified in the `-t` option. Specify the `-c` option so that any file that does not already exist will not be created. In the following example, `file002` does not already exist:

  ```
  $ touch -c -t 201311011508 file001 file002 file003
  $ ls -lT *
  -rw-r--r--  1 usr1  grp1  5 Nov  1 15:08:00 2013 file001
  -rw-r--r--  1 usr1  rrp1  9 Nov  1 15:08:00 2013 file003
  ```

uname command (displays information about the OS or hardware)

Format

```
(Windows only) uname [-a] [-m] [-n] [-r] [-s] [-v] [-w]
(UNIX only) uname [-a] [-m] [-n] [-r] [-s] [-v]
```

Description

This command outputs information about the OS, the system host name, or hardware to the standard output.

Arguments

When this command is executed with no options specified, the processing is the same as when the `-s` option is specified.

-a

(Windows only)

When the `-a` option is specified and the `-w` option is omitted, the command displays the following information all on one line in the order shown in the following:

- OS name (always `Windows`)
- Node name
- Information about the OS
- Most recent service pack installed on the OS
- OS version
- Machine (hardware) type

When the `-a` option is specified and the `-w` option is also specified, the command displays the following information one item per line in the order shown in the following:

- OS name, installation folder for that OS, and partition information for the disk on which the OS is installed
- Node name
- OS release (always `unknown`)
- OS version
- Machine (hardware) type

(UNIX only)

The command displays the following information all on one line in the order shown in the following:

- OS name
- Node name
- OS release
- OS version
- Machine (hardware) type

-m

Specifies that the type of machine (hardware) is to be displayed.

-n

Specifies that the node name is to be displayed.

-r

Specifies that the OS release is to be displayed. In Windows, `unknown` is always displayed.

-s

Specifies that the OS name is to be displayed.

In Windows, the command displays the following information as the OS name:

- When the `-w` option is omitted, the command always displays `Windows`.
- When the `-w` option is also specified, the command displays the OS name, installation folder for that OS, and partition information for the disk on which the OS is installed.

-v

Specifies that the OS version is to be displayed.

-w (Windows only)

Specifies that the information is to be displayed in the format used in JP1/Advanced Shell version 10-01 or earlier. When the `-w` option is specified, the command displays the information as follows:

- The information about each option is displayed on a single line.
- The information that is displayed by the `-a` and `-s` options is variable. For details, see the description of each option.

If only this option is specified, the command's processing is the same as when the `-w` and `-s` options are both specified.

Return codes

Return code	Meaning
`0`	Normal termination
`1` or greater	Error termination

Notes

- In Windows, input and output are performed in the binary mode for files and for the standard input and standard output. No conversion of end-of-line codes is performed.
- In Windows, if an option other than those listed below is specified, a user with Administrators permissions must execute the command, because Administrators permissions are required to acquire the information. An error results if a user without Administrators permissions attempts to execute a `uname` command in which any option other than those listed below is specified.
    - `-r` option
    - `-s` option (when not specified together with the `-w` option)
- In Windows, in order for the `uname` command to use Windows OS functions for acquiring information about the OS and hardware, the `PATH` environment variable must contain the Windows system folder paths at the time the script executes. If you want to add another path to the `PATH` environment variable, be sure to append it to the `PATH` environment variable as in the following example.

  Example

  ```
  PATH="${PATH};C:\\home\\bin"
  ```

Usage examples

- Display the default with no options specified.

  Windows example:

  ```
  C:\TEMP>%ADSH_OSCMD_DIR%\uname
  Windows
  ```

  UNIX example (when the command is run in Linux):

  ```
  $ /opt/jp1as/cmd/uname
  Linux
  ```

- Specify the `-a` option to display the detailed information about the OS environment.

  Windows example (when the `-w` option is omitted):

```
Windows MyMachine Microsoft Windows Server 2008 R2 Enterprise Service
Pack 1 6.1.7601 x64-based PC
```

Windows example (when the `-w` option is also specified):

```
C:\TEMP>%ADSH_OSCMD_DIR%\uname -aw
Microsoft Windows Server 2008 R2 Enterprise|C:\Windows|\Device
\Harddisk0\Partition2
 MyMachine
 unknown
 6.1.7601
 x64-based PC
```

UNIX example (when the command is run in Linux):

```
$ /opt/jp1as/cmd/uname -a
Linux LINUX1 2.6.18-53.el5 #1 SMP Wed Oct 10 16:34:02 EDT 2007 i686
```

- Specify the `-m` option to display the name of the machine and hardware:

```
C:\TEMP>%ADSH_OSCMD_DIR%\uname -m
x64-based PC
```

- Specify the `-n` option to display the node name:

```
C:\TEMP>%ADSH_OSCMD_DIR%\uname -n
MyMachine
```

- Specify the `-r` option to display the OS release. The following shows a Windows example that always displays `unknown`:

```
C:\TEMP>%ADSH_OSCMD_DIR%\uname -r
unknown
```

- Specify the `-s` option to display the OS name.

  Windows example:

```
C:\TEMP>%ADSH_OSCMD_DIR%\uname -s
Windows
```

- Specify the `-v` option to display the OS version.

  Windows example:

```
C:\TEMP>%ADSH_OSCMD_DIR%\uname -v
```

- If multiple options are combined, the command displays the corresponding information according to the order defined for the `-a` option. A Windows example is shown below. This example specifies the `-v` and `-s` options in this order, but the information is displayed in the order of `-s` and `-v`:

```
6.1.7601
C:\TEMP>%ADSH_OSCMD_DIR%\uname -v -s
Windows 6.1.7601
```

- Display an option error message.

  This message might vary depending on the platform on which the command is executed.

  Windows example:

```
C:\TEMP>%ADSH_OSCMD_DIR%\uname -p
uname: illegal option -- p
usage: uname [-amnrsvw]
```

- In Windows, if a user without Administrators permissions attempts to execute the `uname` command in which is specified an option requiring Administrator permissions, an error message is displayed as shown below. This example executes the command with the `-m` option specified:

```
C:\TEMP>%ADSH_OSCMD_DIR%\uname -m
Failed to register mof file(s).
Only the administrator group members can use WMIC.EXE.
Reason:Win32 Error: Access is denied.

unknown
```

uniq command (removes duplicated lines from a sorted file)

Format

```
uniq [-c] [-d] [-u] [input-path-name [output-path-name]]
```

Description

This command outputs the results of consolidating duplicated lines in a file into single lines. Note that lines with identical content are considered to be duplicates only if they are consecutive.

Arguments

If no options are specified, the processing is the same as when the `-d` and `-u` options are both specified. That is, the command outputs duplicate lines as a single line, and it also outputs all non-duplicate lines.

-c

Specifies that each output line is to be preceded by a count of the number of times the line occurred, followed by a single space. A count is displayed as a four-digit number, but the number of digits will be increased if necessary to accommodate values that exceed four digits. A single space is displayed after each count.

-d

Specifies that only duplicate lines are to be output.

-u

Specifies that only lines that had no duplicates are to be displayed.

input-path-name

Specifies the input file. If *input-path-name* is not specified or is specified as `-`, the standard input is read.

output-path-name

Specifies the output file for the results. If *output-path-name* is not specified or is specified as `-`, the standard output is assumed.

Return codes

Return code	Meaning
`0`	Normal termination
`1` or greater	Error termination

Notes

- If you specify the same file for *input-path-name* and *output-path-name*, the file will be empty.
- The maximum number of bytes that can be compared in a single line is 8,192.
- Input from binary files and output of binary data are not guaranteed to work.

Usage examples

The following shows the format of the file used in the examples below to illustrate the results of executing the `uniq` command.

- `file1.txt`

```
aaaa
aaaaaaa                       ➜ duplicate
aaaaaaa                       ➜ duplicate
bbbbbbb
bbbbbbbbbbb                   ➜ duplicate
bbbbbbbbbbb                   ➜ duplicate
bbbbbbbbbbb                   ➜ duplicate
bbbbbbbbbbb                   ➜ duplicate
bcbcbcbcbcb
ddddddddddddddddddd
dddddddddddddddddddddd
dddddddddddddddddddddddd
ddddddddddddddddddddddddeee    ➜ duplicate
ddddddddddddddddddddddddeee    ➜ duplicate
```

The file listed above is used as the input file in the following examples.

- Display the default with no options specified.

```
C:\TEMP>%ADSH_OSCMD_DIR%\uniq file1.txt
aaaa
aaaaaaa
bbbbbbb
bbbbbbbbbbb
bcbcbcbcbcb
ddddddddddddddddddd
dddddddddddddddddddddd
dddddddddddddddddddddddd
ddddddddddddddddddddddddeee

C:\TEMP>
```

- Specify the `-c` option to precede each output line with a count of the number of times the line occurred.

```
C:\TEMP>%ADSH_OSCMD_DIR%\uniq -c file1.txt
   1 aaaa
   2 aaaaaaa
   1 bbbbbbb
   4 bbbbbbbbbbb
   1 bcbcbcbcbcb
   1 ddddddddddddddddddd
   1 dddddddddddddddddddddd
   1 dddddddddddddddddddddddd
   2 ddddddddddddddddddddddddeee

C:\TEMP>
```

- Specify the `-d` option to display only the duplicate lines (one instance of each set of lines that were duplicated).

```
C:\TEMP>%ADSH_OSCMD_DIR%\uniq -d file1.txt
aaaaaaa
bbbbbbbbbbb
ddddddddddddddddddddddddeee

C:\TEMP>
```

- Specify the `-u` option to display only the lines that had no duplicates.

```
C:\TEMP>%ADSH_OSCMD_DIR%\uniq -u file1.txt
aaaa
bbbbbbb
bcbcbcbcbcb
ddddddddddddddddddd
dddddddddddddddddddddd
dddddddddddddddddddddddd

C:\TEMP>
```

- Display an option error message.

  This message might vary depending on the platform on which the command is executed. The following shows an example in Windows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\uniq -w
uniq: illegal option -- w
usage: uniq [-cdu] [input_file [output_file]]
```

wc command (counts the number of bytes, lines, characters, and words in files)

Format

```
wc [-c] [-l] [-m] [-w] [path-name ...]
```

Description

This command counts the number of bytes, lines, characters, and words in files. Depending on the options that are specified, the counts of the numbers of lines, words, characters, and bytes are displayed in front of the name of each input file.

Arguments

-c

Specifies that the number of bytes in an input file is to be output to the standard output.

-l

Specifies that the number of lines in an input file is to be output to the standard output. The number of lines is determined by the number of end-of-line codes.

-m

Specifies that the number of characters in an input file is to be output to the standard output. A multibyte character is counted as a single character.

-w

Specifies that the number of words in an input file is to be output to the standard output. The number of words is determined by the number of character strings delimited by a space, tab, or end-of-line code.

path-name

Specifies the name of an input file. If *path-name* is not specified or is specified as `-`, the standard input is read.

Return codes

Return code	Meaning
`0`	Normal termination
`1` or greater	Error termination

Notes

- Any character in a character encoding that is different from the local character encoding is considered an invalid or incomplete character.
- Specifying no option is equivalent to specifying the `-c`, `-l`, and `-w` options.
- Regardless of the order in which the options are specified, output items are displayed in the order of *lines-count*, *words-count*, *multibyte-characters-count*, *bytes-count*, and *file-name*. Numeric values are displayed as seven-digit numbers separated by a single space. The number of digits is increased if necessary to accommodate values that exceed seven digits.
- An error results if an input file contains invalid or incomplete multibyte or wide characters, binary data, or a character encoding that is different from the local character encoding. In these cases, an error message (`wc: binaryfile: Invalid or incomplete multibyte or wide character`) is output.

Usage examples

- Display the default with no options specified.

```
C:\TEMP>%ADSH_OSCMD_DIR%\wc a.txt b.txt
       5       5      55 a.txt
       4       4      44 b.txt
       9       9      99 total
```

- Specify the `-c` option to display the number of bytes in the input file.

```
C:\TEMP>%ADSH_OSCMD_DIR%\wc -c a.txt
      55 a.txt
```

- Specify the `-l` option to display the number of lines in the input file.

```
C:\TEMP>%ADSH_OSCMD_DIR%\wc -l a.txt
       5 a.txt
```

- Specify the `-m` option to display the number of characters in the input file.

```
C:\TEMP>%ADSH_OSCMD_DIR%\wc -m a.txt
      50 a.txt
```

- Specify the `-w` option to display the number of words in the input file.

```
C:\TEMP>%ADSH_OSCMD_DIR%\wc -w a.txt
       5 a.txt
```

- Specify all the options to display the number of lines, words, characters, and bytes in the input file.

```
C:\TEMP>%ADSH_OSCMD_DIR%\wc -clmw a.txt
       5       5      50      55 a.txt
```

- Display an option error message.

  This message might vary depending on the platform on which the command is executed. The following shows an example in Windows:

```
C:\TEMP>%ADSH_OSCMD_DIR%\wc -z
wc: illegal option -- z
usage: wc [-clmw] [file ...]
```

- Display the error message that is output when there is an invalid or incomplete character in the file.

```
C:\TEMP>%ADSH_OSCMD_DIR%\wc binaryfile
wc: binaryfile: Invalid or incomplete multibyte or wide character
```

  The following are considered invalid or incomplete characters:

  - Invalid or incomplete multibyte or wide characters, or binary data
  - Characters in a character encoding that is different from the local character encoding

which command (obtains the paths of external commands)

Format

```
which[-a]command-name ...
```

Description

This command obtains the paths of external commands to be executed from the command search path set in the `PATH` environment variable. The command outputs the obtained command paths to the standard output.

Arguments

-a

Specifies that all executable command paths are to be obtained from the command search path set in the `PATH` environment variable.

When the `-a` option is omitted, the command output only the first command path obtained.

command-name

Specifies the name of an external command whose command path is to be obtained. You can specify multiple command names.

If the command path of a specified external command is not found, the `which` command outputs a message to that effect to the standard error output.

Command path search rules

The command searches for the command paths of external commands according to the rules described below.

In Windows:

Paths subject to external command search

The command searches the command search path set in the `PATH` environment variable for the external commands. If multiple command paths are set in the `PATH` environment variable, the command searches the command paths in order from the beginning. If the user executing the `which` command does not have permissions to read the external command storage directory, that directory is not subject to command path search.

External commands whose command paths are to be output

If the user executing the `which` command has permissions to read the external command storage directory, the `which` command outputs the corresponding command paths. The `which` command does not check whether the user has permissions to execute the external command.

The `which` command can output the paths of external commands if they are executable files with the extension `.com`, `.exe`, `.cmd`, or `.bat`.

If a specified external command contains no extension, the `which` command adds the extensions defined in the `PATHEXT` environment variable in the order defined and then searches the external commands. The supported extensions are `.com`, `.exe`, `.cmd`, and `.bat`. For details, see *5.1.11 Specifying external commands*.

In UNIX:

Paths subject to external command search

The command searches the command search path set in the `PATH` environment variable for the external commands. If multiple command paths are set in the `PATH` environment variable, the command searches the command paths in order from the beginning. If the user executing the `which` command does not have permissions to search the external command storage directory (including all directories in the path), that directory is not subject to command path search.

External commands whose command paths are to be output

If the user executing the `which` command has permissions to execute a specified external command, the `which` command determines that that external command is executable and outputs its command path. If the user does not have permissions to execute a specified external command, that external command's path is not output.

When the command names specified in the argument contain paths

In Windows:

If the user executing the `which` command has permissions to read the external command storage directory, the `which` command outputs the corresponding command paths. The `which` command does not check whether the user has permissions to execute the external command.

If the user executing the `which` command does not have permissions to read the external command storage directory, the `which` command outputs a message indicating that the external command's command path was not found.

The `which` command can output the paths of external commands if they are executable files with extension `.com`, `.exe`, `.cmd`, or `.bat`.

If the specified external commands contain no extension, the `which` command adds to the external command names the extensions defined in the `PATHEXT` environment variable in the order defined. The supported extensions are `.com`, `.exe`, `.cmd`, and `.bat`.

In UNIX:

If the user executing the `which` command has permissions to search the external command storage directory (including all directories in the path) and the external command execution permissions, the `which` command outputs the command names specified in the argument. If the user executing the `which` command does not have these permissions, the `which` command outputs a message indicating that the external command's command path was not found.

Return code

Return code	Meaning
`0`	Normal termination
`1`	Error termination The command path of the external command was not found; or, if multiple external commands were searched, at least one external command's command path was not found.
`2`	Error termination • An invalid option was specified. • The `PATH` environment variable is undefined. • In Windows, the `PATHEXT` environment variable is undefined.

Notes

- If the `PATH` environment variable is undefined, the command terminates with an error.
- In Windows, if the `PATHEXT` environment variable is undefined, the command terminates with an error.
- If the following names are specified as command names in the argument, the `which` command treats them as external commands:
    - Aliases defined by the `alias` command
    - Reserved words, standard shell commands, extended shell commands, and functions
- If a command name specified in the argument contains a path and the path name is subject to conversion by either of the environment setting parameters listed below, the `which` command outputs the path name obtained after conversion:
    - `PATH_CONV` parameter
    - `COMMAND_CONV_ARG` parameter
- In Windows, only external commands that satisfy the command path search rules are subject to output of command paths.

  In the case of commands that are specified in the `system` function of the `awk` command and commands that are executed from the command line specified in the `-exec` option of the `find` command, paths are searched as follows:
    - Commands specified in the `system` function of the `awk` command:

      The path search rules for command processor execution (such as the command prompt) are used as the path search rules for command execution.
    - Command line specified in the `-exec` option of the `find` command:

      The path search rules for the Windows API that executes the program are used as the path search rules for command line execution.

Note that the command path that is output when a command name specified in the above command is specified in the argument might differ from the path of the command that is executed.

Usage examples

- Obtain the command path of command `pgm01.exe`:

```
C:\TEMP>%ADSH_OSCMD_DIR%\which pgm01.exe
C:\Program Files\Hitachi\PP001\pgm01.exe
```

- Obtain the command path of command `pgm01`. This example omits the extension of the command name:

```
C:\TEMP>%ADSH_OSCMD_DIR%\which pgm01
C:\Program Files\Hitachi\PP001\pgm01.exe
```

- Specify the `-a` option to obtain all command paths of command `pgm01.exe`:

```
C:\TEMP>%ADSH_OSCMD_DIR%\which -a pgm01.exe
C:\Program Files\Hitachi\PP001\pgm01.exe
C:\Program Files\Hitachi\PP002\pgm01.exe
C:\Program Files\Hitachi\PP003\pgm01.exe
```

- Obtain the command path of command `pgm02`. In this example, the command search path does not contain `pgm02`:

```
C:\TEMP>%ADSH_OSCMD_DIR%\which pgm02
which: no pgm02 in (C:\WINDOWS\system32;C:\WINDOWS;C:\Program Files
\Hitachi\PP001
;C:\Program Files\Hitachi\PP002;C:\Program Files\Hitachi\PP003)
```

- Obtain the command paths of the commands `pgm01`,`pgm02`, `pgm03`, and `pgm04`. In this example, the command search path does not contain the commands `pgm02` and `pgm04`:

```
C:\TEMP>%ADSH_OSCMD_DIR%\which pgm01 pgm02 pgm03 pgm04
C:\Program Files\Hitachi\PP001\pgm01.exe
which: no pgm02 in (C:\WINDOWS\system32;C:\WINDOWS;C:\Program Files
\Hitachi\PP001
;C:\Program Files\Hitachi\PP002;C:\Program Files\Hitachi\PP003)
C:\Program Files\Hitachi\PP001\pgm03.exe
which: no pgm04 in (C:\WINDOWS\system32;C:\WINDOWS;C:\Program Files
\Hitachi\PP001
;C:\Program Files\Hitachi\PP002;C:\Program Files\Hitachi\PP003)
```

- Execute the `which` command with a command name containing a path specified. In this example, the specified program name exists:

```
C:\TEMP>%ADSH_OSCMD_DIR%\which "C:\Program Files\Hitachi\PP001\pgm01"
C:\Program Files\Hitachi\PP001\pgm01.exe
```

- Execute the `which` command with a command name containing a path specified. In this example, the specified program name does not exist:

```
C:\TEMP>%ADSH_OSCMD_DIR%\which "C:\Program Files\Hitachi\PP001\pgm02"
which: no pgm02 in (C:\Program Files\Hitachi\PP001)
```

8.5 UNIX-compatible commands (script format) (Windows only)

You can execute the UNIX-compatible commands listed below by using the sample script files provided by JP1/ Advanced Shell. These sample script files are for Widows only. In UNIX, use the commands provided by the OS.

Table 8–14: UNIX-compatible commands provided as sample script files

Command name	Name of sample script file	Overview of functionality
All commands	`script_0`	Disables the commands specified in job definition scripts.
`chmod`	`script_chmod1`	Changes the file read-only attribute setting (enable or disable).
	`script_chmod2`	Specifies file or folder permissions as numeric values.
	`script_chmod3`	Specifies file or folder permissions as symbols or numeric values.
`su`	`script_su1`	Executes programs with the permissions of the executing user.
`who`	`script_who1`	Outputs to logs information about the login user.

For details about the procedure for using the sample script files, see *2.6.6(2) Preparations for using the script-format UNIX-compatible commands (Windows only)*.

chmod command (disables the chmod commands specified in job definition scripts)

Format

```
chmod [option][mode][path-name]
```

You can create this command by using the `script_0` sample script file as the base. For details about how to create the command, see *2.6.6(2) Preparations for using the script-format UNIX-compatible commands (Windows only)*.

Description

This command disables all the `chmod` commands and their arguments that are specified in job definition scripts. This command always terminates normally with return code `0`.

In Windows, if access control is performed for each login user, it might not be necessary to change access permissions when job definition scripts are run. In such a case, you can use this command to disable all `chmod` commands specified in job definition scripts, thereby eliminating the need to modify job definition scripts that have been migrated from a UNIX system to a Windows system.

Arguments

option

Ignores the specification.

mode

Ignores the specification.

path-name

Ignores the specification.

Return code

Return code	Meaning
0	Normal termination

Notes

- In command substitution, the commands specified in arguments are still executed. If this affects the subsequent processing, check and, if necessary, revise the specification.

Usage examples

The following shows an example definition of a job definition script. This example assumes that the `chmod` command has been created by using the `script_0` sample script file as the base.

- Ignore the `chmod` commands specified in job definition scripts. The options of the `chmod` commands specified in this example are not executed:

```
chmod go-x test.txt
if [[ $? -ge 1 ]]; then     # Processing continues because the return
code of chmod is always 0.
  echo "chmod error." 1>&2
  exit 1
fi
```

chmod command (changes the file read-only attribute setting (enable or disable))

Format

```
chmod [-fhR] mode path-name
```

You can create this command by using the `script_chmod1` sample script file as the base. For details about how to create the command, see *2.6.6(2) Preparations for using the script-format UNIX-compatible commands (Windows only)*.

Description

This command changes the file read-only attribute setting (enable or disable).

Use this command when you want to suppress updating of files.

Arguments

-f

Ignores the specification.

-h

Ignores the specification.

-R

Ignores the specification.

mode

Specifies the mode as a symbol or a numeric value and enables or disables the read-only attribute. The table below explains how to specify this option. If any other mode is specified, the command outputs `chmod: invalid file mode:` *mode* to the standard error output, in which case the access permissions remain unchanged.

Specification	Symbol	Numeric value
Disabling the read-only attribute and permitting write operations (equivalent to when the execution results of the `adshscripttool -fmode -s w` command are `AAA` or `RRR`)	`+w` specified	Numeric value that turns on write permission mode bits `u`, `g`, and `o` (such as `777`, `666`, `333`, `222`, and `733`)
Enabling the read-only attribute and prohibiting write operations (equivalent to when the execution results of the `adshscripttool -fmode -s w` command are `DDD`)	`-w` specified	Numeric value that turns off write permission mode bits `u`, `g`, and `o` (such as `555`, `444`, `111`, `000`, and `511`)

path-name

Specifies the target file. You can specify multiple files. A folder cannot be specified.

Return code

Return code	Meaning
`0`	Normal termination
`1` or greater	Error termination

Notes

- If the executing user does not have permissions to change the file attribute, the `attrib` command outputs the message `Access denied -` *path-name* to the standard error output and then results in an error, in which case the permissions cannot be changed. Grant the executing user the permissions to change file attributes and then re-execute the command.
- If a folder is specified for *path-name*, the message `chmod: cannot access [`*path-name*`]: change for the directory is not supported` is output. In this case, the command does not change the folder read-only attribute and terminates with return code `1`.

Usage examples

The following shows example definitions of job definition scripts. These examples assume that the `chmod` command has been created by using the `script_chmod1` sample script file as the base.

- Use a symbol to prohibit write operations on files:

```
chmod -w test.txt
```

- Use a symbol to permit write operations on files:

```
chmod +w test.txt
```

- Use a numeric value to prohibit write operations on files:

```
chmod 444 test.txt
```

- Specify a mode whose specification is not permitted:

  ```
  chmod -r test.txt
  ```

  In this example, the following message is output to the standard error output:

  ```
  chmod: invalid file mode: -r
  ```

chmod command (specifies permissions as numeric values)

Format

```
chmod [-fhR] mode path-name
```

You can create this command by using the`script_chmod2` sample script file as the base. For details about how to create the command, see *2.6.6(2) Preparations for using the script-format UNIX-compatible commands (Windows only)*.

Description

This command deletes the existing access control list (ACL) and specifies a new ACL using numeric values for the mode specification.

Use this command to set access permissions as numeric values for the following purposes:

- Suppressing `write` and `read` by users other than `owner`
- Permitting `write` and `read` by all users
- Suppressing `write` by all users including `owner`

Arguments

-f

Ignores the specification.

-h

Ignores the specification.

-R

Ignores the specification.

mode

The table below lists the mode values and the corresponding access permissions in access control entries (ACEs) that are set. If any other mode is specified, the command outputs the message `chmod: invalid file mode:` *mode* to the standard error output, in which case the access permissions are not changed.

Mode value	Access permission that is set
777	Owner: `F`, Everyone: `F`
`766`	Owner: `F`, Everyone: `C`
`755`	Owner: `F`, Everyone: `R`

Mode value	Access permission that is set
`744`	Owner: `F`, Everyone: `R`
`733`	Owner: `F`, Everyone: `W`
`722`	Owner: `F`, Everyone: `W`
`700`	Owner: `F`
`666`	Owner: `C`, Everyone: `C`
`655`	Owner: `C`, Everyone: `R`
`644`	Owner: `C`, Everyone: `R`
`633`	Owner: `C`, Everyone: `W`
`622`	Owner: `C`, Everyone: `W`
`600`	Owner: `C`
`555`	Owner: `R`, Everyone: `R`
`544`	Owner: `R`, Everyone: `R`
`533`	Owner: `R`, Everyone: `W`
`522`	Owner: `R`, Everyone: `W`
`500`	Owner: `R`
`444`	Owner: `R`, Everyone: `R`
`433`	Owner: `R`, Everyone: `W`
`422`	Owner: `R`, Everyone: `W`
`400`	Owner: `R`
`333`	Owner: `W`, Everyone: `W`
`322`	Owner: `W`, Everyone: `W`
`300`	Owner: `W`
`222`	Owner: `W`, Everyone: `W`
`200`	Owner: `W`

Legend:

In the table, `F`, `C`, `R`, and `W` correspond to the following access permissions of the `cacls` command:

- `F`: Full control
- `C`: Change permission
- `R`: Read permission
- `W`: Write permission

The mode read and write permissions combined (mode bit 6) are defined as the change permission. All permissions combined (mode bit 7) are defined as full control.

A mode specification equivalent to execution permissions is ignored. Therefore, mode bit 5 is defined as being equivalent to mode bit 4, and mode bit 3 is defined as being equivalent to mode bit 2.

path-name

Specifies the target file or folder. You can specify multiple files or folders.

Return code

Return code	Meaning
0	Normal termination
1 or greater	Error termination

Notes

- This command deletes the existing ACL and sets only those ACEs listed in the explanation of mode. If there are ACEs for accounts that you want to keep, add them to the `cacls` command definition in the sample script.
- The Other users account is set to Everyone. Therefore, if Owner's access permissions are lower than Other users' access permissions, Owner can use Everyone's access permissions.
- If the executing user does not have permissions to access files and folders, the `cacls` command outputs the message `Access denied` to the standard error output and then results in an error, in which case the permissions cannot be changed. Grant the executing user the permissions to change access permissions and then re-execute the command.

Usage examples

The following shows example definitions of job definition scripts. These examples assume that the `chmod` command has been created by using the `script_chmod2` sample script file as the base.

- Set a specified file to be readable by all users:

  ```
  chmod 444 test.txt
  ```

- Specify a mode whose specification is not permitted:

  ```
  chmod 611 test.txt
  ```

  In this example, the following message is output to the standard error output:

  ```
  chmod: invalid file mode: 611
  ```

chmod command (specifies permissions as symbols or numeric values)

Format

```
chmod [-fhR] mode path-name
```

You can create this command by using the `script_chmod3` sample script file as the base. For details about how to create the command, see *2.6.6(2) Preparations for using the script-format UNIX-compatible commands (Windows only)*.

Description

This command deletes all ACEs except for those for Owner and Everyone and changes or sets access permissions according to the mode specified as symbols or numeric values.

Use this command to set access permissions as symbols or numeric values for the following purposes:

- Migrating to a Windows system job definition scripts in which `chmod` command modes are specified as symbols

- Adding or suppressing Owner's or all users' access permissions

Arguments

-f

Ignores the specification.

-h

Ignores the specification.

-R

Ignores the specification.

mode

The table below lists the mode values and the corresponding access permissions in access control entries (ACEs) that are set. If any other mode is specified, the command outputs the message `chmod: invalid file mode:` *mode* to the standard error output, in which case the access permissions are not changed.

Mode value (execution results of the adshscripttool -fmode command)	Access permission that is set
`u+r` (A00000000)	Owner: `R` is added.
`u+rw` (AA0000000)	Owner: `C` is added.
`u+rwx` (AAA000000)	Owner: `F` is added.
`u+w` (0A0000000)	Owner: `W` is added.
`u-rwx` (DDD000000)	Owner's ACE is deleted.
`u=r` (RDD000000)	Owner: Replaced with `R`.
`u=rw` (RRD000000)	Owner: Replaced with `C`.
`u=rwx` (RRR000000)	Owner: Replaced with `F`.
`u=w` (DRD000000)	Owner: Replaced with `W`.
`o+r` (000000A00)	Everyone: `R` is added.
`o+rw` (000000AA0)	Everyone: `C` is added.
`o+rwx` (000000AAA)	Everyone: `F` is added.
`o+w` (0000000A0)	Everyone: `W` is added.
`o-rwx` (000000DDD)	Everyone's ACE is deleted.
`o=r` (000000RDD)	Everyone: Replaced with `R`.
`o=rw` (000000RRD)	Everyone: Replaced with `C`.
`o=rwx` (000000RRR)	Everyone: Replaced with `F`.
`o=w` (000000DRD)	Everyone: Replaced with `W`.
`+r / ugo+r` (A00A00A00)	Owner: R, Everyone: R is added.
`+rw / ugo+rw` (AA0AA0AA0)	Owner: C, Everyone: C is added.
`+rwx / ugo+rwx` (AAAAAAAAA)	Owner: F, Everyone: F is added.
`+w / ugo+w` (0A00A00A0)	Owner: W, Everyone: W is added.

Mode value (execution results of the adshscripttool -fmode command)	Access permission that is set
`-rwx` / `ugo-rwx` (DDDDDDDDD)	Owner's and Everyone's ACEs are deleted.
`=r` / `ugo=r` (RDDRDDRDD)	Owner: R, Everyone: Replaced with `R`.
`=rw` / `ugo=rw` (RRDRRDRRD)	Owner: C, Everyone: Replaced with `C`.
`=rwx` / `ugo=rwx` (RRRRRRRRR)	Owner: F, Everyone: Replaced with `F`.
`=w` / `ugo=w` (DRDDRDDRD)	Owner: W, Everyone: Replaced with `W`.
`777` (RRRRRRRRR)	Owner: F, Everyone: Replaced with `F`.
`766` (RRRRRDRRD)	Owner: F, Everyone: Replaced with `C`.
`755` (RRRRDRRDR)	Owner: F, Everyone: Replaced with `R`.
`744` (RRRRDDRDD)	Owner: F, Everyone: Replaced with `R`.
`733` (RRRDRRDRR)	Owner: F, Everyone: Replaced with `W`.
`722` (RRRDRDDRD)	Owner: F, Everyone: Replaced with `W`.
`700` (RRRDDDDDD)	Owner: Replaced with `F`.
`666` (RRDRRDRRD)	Owner: C, Everyone: Replaced with `C`.
`655` (RRDRDRRDR)	Owner: C, Everyone: Replaced with `R`.
`644` (RRDRDDRDD)	Owner: C, Everyone: Replaced with `R`.
`633` (RRDDRRDRR)	Owner: C, Everyone: Replaced with `W`.
`622` (RRDDRDDRD)	Owner: C, Everyone: Replaced with `W`.
`600` (RRDDDDDDD)	Owner: Replaced with `C`.
`555` (RDRRDRRDR)	Owner: R, Everyone: Replaced with `R`.
`544` (RDRRDDRDD)	Owner: R, Everyone: Replaced with `R`.
`533` (RDRDRRDRR)	Owner: R, Everyone: Replaced with `W`.
`522` (RDRDRDDRD)	Owner: R, Everyone: Replaced with `W`.
`500` (RDRDDDDDD)	Owner: Replaced with `R`.
`444` (RDDRDDRDD)	Owner: R, Everyone: Replaced with `R`.
`433` (RDDDRRDRR)	Owner: R, Everyone: Replaced with `W`.
`422` (RDDDRDDRD)	Owner: R, Everyone: Replaced with `W`.
`400` (RDDDDDDDD)	Owner: Replaced with `R`.
`333` (DRRDRRDRR)	Owner: W, Everyone: Replaced with `W`.
`322` (DRRDRDDRD)	Owner: W, Everyone: Replaced with `W`.
`300` (DRRDDDDDD)	Owner: Replaced with `W`.
`222` (DRDDRDDRD)	Owner: W, Everyone: Replaced with `W`.
`200` (DRDDDDDDD)	Owner: Replaced with `W`.

Legend:

In the table, `F`, `C`, `R`, and `W` correspond to the following access permissions of the `cacls` command:

- `F`: Full control
- `C`: Change permission
- `R`: Read permission
- `W`: Write permission

The mode read and write permissions combined (mode bit 6) are defined as the change permission. All permissions combined (mode bit 7) are defined as full control.

A mode specification equivalent to execution permissions is ignored. Therefore, mode bit 5 is defined as being equivalent to mode bit 4, and mode bit 3 is defined as being equivalent to mode bit 2.

path-name

Specifies the target file or folder. You can specify multiple files or folders.

Return code

Return code	Meaning
`0`	Normal termination
`1` or greater	Error termination

Notes

- This command deletes all ACEs except for those for Owner and Everyone and sets only those ACEs listed in the explanation of mode. If there are ACEs for accounts that you want to keep, add them to the `cacls` command definition in the sample script.
- The Other users account is set to Everyone. Therefore, if Owner's access permissions are lower than Other users' access permissions, Owner can use Everyone's access permissions.
- If the executing user does not have permissions to access files and folders, the `cacls` command outputs the message `Access denied` to the standard error output and then results in an error, in which case the permissions cannot be changed. Grant the executing user the permissions to change access permissions and then re-execute the command.

Usage examples

The following shows example definitions of job definition scripts. These examples assume that the `chmod` command has been created by using the `script_chmod3` sample script file as the base.

- Add write permission to Other users:

  ```
  chmod o+w test.txt
  ```

- Specify a mode whose specification is not permitted:

  ```
  chmod g-w test.txt
  ```

  In this example, the following message is output to the standard error output:

  ```
  chmod: invalid file mode: g-w
  ```

su command (disables the su commands specified in job definition scripts)

Format

```
su [-] [user-name] [argument...]
```

You can create this command by using thescript_0 sample script file as the base. For details about how to create the command, see *2.6.6(2) Preparations for using the script-format UNIX-compatible commands (Windows only)*.

Description

This command disables all su commands and their arguments that are specified in job definition scripts. This command always terminates normally with return code 0.

If job definition scripts that use the su command in a UNIX system to start and stop subsystems are migrated to a Windows system that uses a different system to start and stop subsystems, the su command processing specified in the job definition scripts might no longer be needed. In such a case, you can use this command to disable all su command definitions in job definition scripts, thereby eliminating the need to modify job definition scripts that have been migrated from a UNIX system to a Windows system.

Arguments

-

Ignores the specification.

user-name

Ignores the specification.

argument

Ignores the specification.

Return code

Return code	Meaning
0	Normal termination

Notes

- In command substitution, the commands specified in arguments are still executed. If this affects the subsequent processing, check and, if necessary, revise the specification.

Usage examples

The following shows example definitions of job definition scripts. These examples assume that the su command has been created by using the script0 sample script file as the base.

- Ignore the su commands specified in job definition scripts. The options of the su commands specified in this example are not executed:

```
su - ${DBADMIN} -c 'export PDDIR=/home/db/db1; start -q'
if [[ $? -ge 1 ]]; then     # Processing continues because the return
code of su is always 0.
```

```
  echo "su error." 1>&2
  exit 1
fi
```

su command (executes programs with the permissions of the executing user)

Format

```
su [-] user-name {-c command-line|script-file-path-name}
   [run-time-parameters]
```

`script_su1` sample script file as the base. For details about how to create the command, see *2.6.6(2) Preparations for using the script-format UNIX-compatible commands (Windows only)*.

Description

This command executes the commands specified in the argument. The command ignores the specified user name and executes the specified commands with the permissions of the executing user.

If existing job definition scripts contain `su` commands, this command enables you to migrate them to a Windows system without having to rewrite the job definition scripts.

Arguments

-

Ignores the specification.

user-name

Ignores the specification.

-c command-line

Specifies the command line to be executed in the job.

You can specify for *command-line* any commands that can be specified in job definition scripts, such as shell operation commands and UNIX-compatible commands.

script-file-path-name

Specifies the path name of the script file that is to be executed.

run-time-parameters

Specifies the values to be set in *command-line* or *script-file-path-name* positional parameters. To include a space in a run-time parameter, you must enclose the corresponding character string in double quotation marks (").

Return code

Return code	Meaning
0	Normal termination
1 or greater	Error termination

Notes

- Before you execute this command, grant the necessary permissions to the executing user.

Usage examples

The following shows example definitions of job definition scripts. These examples assume that the su command has been created by using the script_su1 sample script file as the base.

- Grant the necessary permissions to the executing user and then execute this command with multiple commands specified in the -c argument:

```
# Execute the command by a user that has the required permissions
su - ${DBADMIN} -c 'export PDDIR=C:\\db\\db1; start -q'
```

- Grant the necessary permissions to the executing user and then execute this command with a script file name specified in the argument:

```
# Run the job definition script by a user that has the required
permissions
su - ${DBADMIN} '.\\DBSTART.ash'
```

who command (disables the who commands specified in job definition scripts)

Format

```
who [am i]
```

This command is created by using the sample script file script_0 as the base. For details about the creation procedure, see *2.6.6(2) Preparations for using the script-format UNIX-compatible commands (Windows only)*.

Description

This command disables all who commands and their arguments that are specified in job definition scripts. This command always terminates normally with return code 0.

You might want to disable the who command processing specified in job definition scripts in a situation where the who command was used in UNIX, but that information is no longer needed in Windows. By using this command, you eliminate the need to modify job definition scripts that have been migrated from UNIX to Windows because this command disables all who command definitions in job definition scripts.

Arguments

am i

Ignores the specification.

Return codes

Return code	Meaning
0	Normal termination

Notes

- In the event of command substitution, the commands specified in arguments will still be executed. If this affects the subsequent processing, check and revise the specifications as necessary.

who command (outputs login user information to logs)

Format

```
who [am i]
```

This command is created by using the sample script file script_who1 as the base. For details about the creation procedure, see *2.6.6(2) Preparations for using the script-format UNIX-compatible commands (Windows only)*.

Description

This command starts the quser.exe or qwinsta.exe command.

Specify this command to output to logs a list of the users who are logged in to the system when job definition scripts are run.

Arguments

am i

Ignores the specification.

Return codes

Return code	Meaning
0	Normal termination
Other than 0	Error termination (return code of the quser.exe or qwinsta.exe command)

Examples

The following shows an example definition for job definition scripts. This example assumes that the who command has been created by using the script_who1 sample script file as the base.

- Output to log file log.txt a list of the users who are logged in:

```
who >>log.txt
```

9 Job Definition Script Commands and Control Statements

This chapter explains the description formats and other details related to commands and control statements used in job definition script files, including the following:

- Standard shell commands
- Extended shell commands
- Extended script commands
- Script control statements
- Reserved script commands

9.1 Command and control statement description formats

The following types of commands and control statements can be used in job definition script files:

- Standard shell commands
- Extended shell commands
- Extended script commands
- Script control statements
- Reserved script commands

Note the following points about specifying job definition script files:

- If `NULL` (`0x00`, or `\0` in C language) is specified in the middle of a line, the location in the line where that `NULL` character occurs will be considered by the job controller to be the end of the line. Any character string (on the same line) following that `NULL` character will be ignored. You can prevent invalid execution results and run-time errors by avoiding use of `NULL`.

Example:

  - Input line (`0x00` is indicated by `\0`)

    ```
    echo "test\0null";echo "test after"
    ```

  - Output example

    ```
    echo "test
    ```

- To make job definition scripts easier to read, and to ensure that coverage information is displayed properly, we recommend that only one command be specified on a line. We recommend that you not specify multiple commands on a single line using the semicolon (`;`) separator between them.

Note the following points about collecting coverage information on the commands specified in job definition script files:

- Information about whether each command has executed can be displayed only if no more than four commands were specified on a single line.
- You will be able to determine whether all the commands in an entire job definition script have executed only if no more than 32 commands were specified on a single line.
- When the number of commands on one line exceeds 32, no coverage information will be acquired for the 33rd and subsequent commands. Even if all the commands in the job definition script have executed, the C0 execution ratio will not be shown as 100%.
- Rather than specifying an entire script control statement, such as an `if` statement, on a single line, we recommend that you begin a new line at each keyword.
- Specify the keywords listed below by themselves on a single line, rather than in combinations such as `fi;fi`. If these keywords are written on the same line, coverage information will not be displayed correctly.
  - `fi`, which marks the end of an `if` statement
  - `done`, which marks the end of a `do` block
  - `esac`, which marks the end of a `case` statement
- Only the following coverage information is output:
  - C0 information: Output for only the first four C0 target commands specified on a line

- C1 information: Output for only the first four execution paths on a line

- When multiple commands and execution paths are specified on a single line, some coverage information might not be displayed.

  Example 1:

  - Multiple commands and execution paths are specified on a single line

  ```
      echo 1; echo 2; echo 3; echo 4; echo 5
  ```

  - Each command and execution path is specified on a separate line

  ```
      echo 1
      echo 2
      echo 3
      echo 4
      echo 5
  ```

  Example 2:

  - Multiple commands and execution paths are specified on a single line

  ```
      if true ;then echo 1 ;elif true ;then echo 2 ;elif true ;then echo
  3 ;else echo 4 ;fi
  ```

  - Each command and execution path is specified on a separate line

  ```
      if true
      then
        echo 1
      elif true
      then
        echo 2
      elif true
      then
        echo 3
      else
        echo 4
      fi
  ```

- To execute a command for which you provide input to the standard input from the terminal keyboard, you must do the following to complete the input:
  - To read input terminated by `EOF`

    In Windows, press **Enter** followed by **Ctrl**+**Z**, and then press **Enter** again.

    In UNIX, type **Ctrl**+**D**.
  - To read one line of input

    Press **Enter**.

The following sections show the description formats of the commands and control statements used in job definition script files.

9.1.1 Standard shell command description format

The description format for standard shell commands is as follows:

```
Δ0command-name[Δ1option]...[Δ1option][Δ1operands]
```

- First specify options, and then specify operands. Operands include option names, option values, and other arguments that can be specified in commands. If an operand is specified before an option, all specified items are treated as operands.
- Specify an option in the format -*option-name*[Δ_1*value*]. When there is more than one option, they can be specified in any order.
- Options specified without a value can be grouped together in a block (for example, `-a -b -c` is equivalent to `-abc`). In this case, a value can be specified for the last option (for example, `xyz` in the specification `-abc xyz` is the value of option `-c`).
- An error results if you specify an invalid option or an option's value is outside the permissible range of values.
- Multibyte characters are not permitted in *option-name*.

9.1.2 Extended shell command description format

The description format for extended shell commands is as follows:

```
Δ0command-name[Δ1option][Δ1operand]
```

9.1.3 Extended script command description format

The description format for extended script commands is as follows:

```
Δ0command-nameΔ1attribute-value [...Δ1attribute-value] [Δ1-attribute-nameΔ
1attribute-value [...Δ1-attribute-nameΔ1attribute-value]]
```

- The command name of an extended script command always begins with `#-adsh_`.
- The command name is followed first by a list of attribute values and then by a list of pairs of an attribute name and an attribute value (-*attribute-name attribute-value*).
- The list of attribute values cannot be omitted, and they must be specified in a predetermined order. The list of -*attribute-name attribute-value* pairs can be specified in any order and their specification is optional.
- If you specify for an attribute value a character string beginning with a hyphen (-), it will be interpreted as a specification of an attribute name. To begin an attribute value with a hyphen (-), the hyphen must be escaped with a \, ", or '.
- A hash mark (#) at the beginning of an extended script command does not indicate a comment. You must specify two hash marks in succession (##) to set a comment in an extended script command.
- The double quotation ("), single quotation ('), and escape (\) characters can all be used.

  However, in an extended script command, any \ within a character string enclosed in double quotation marks will be interpreted as an escape character, regardless of what the following character is. To specify a \ within a character string enclosed in double quotation marks, you must specify it as \\.
- Uppercase and lowercase characters are distinguished in *command-name*, *attribute-name*, and *attribute-value* (in the case of reserved words).

- You can specify the name of an environment variable for *attribute-value*, and the environment variable's value will be set for the attribute value before the script starts. The name of the environment variable must be enclosed in curly brackets (`{ }`). The name of an environment variable must not exceed 255 bytes and must be specified in the **<environment variable name>** format shown below in *(2) Character set definitions*.
- Start the specification of an extended script command at the beginning of the line. Between the command name and the linefeed code, use the space as the delimiter between items. A parsing error will result if any non-space character is used as a delimiter.
- A parsing error will result if you specify more than one extended script command on a line (you cannot use a command delimiter to separate multiple commands on the same line).
- A parsing error will result if you specify an extended script command within a function.
- A pre-execution syntax error will result if you specify an extended script command within the block of a `for`, `while`, or `until` statement or within a function definition.
- You cannot specify an extended script command within an external script called with a dot (`.`). If you do, the extended script command will be treated as a comment.

When you specify an extended script command that spans multiple lines, the second and subsequent lines must be specified in the following format:

```
#-adshΔ1continuing-specification
```

- You can specify a continuation line only at the location of a delimiter following the command name or an attribute. A continuation line cannot be specified in the middle of the command name, an attribute name, or an attribute value.
- If a syntax error is detected in a continuation line, the line number displayed in the error message will be for the first line of the extended script command.

(1) Limitations

- An extended script command must not exceed 8,191 bytes per line, including continuation lines.
- To specify more than one attribute value, delimit them with the space or comma. A value cannot be omitted by specifying two commas in the manner of omitting positional parameters.

(2) Character set definitions

The following table defines the character sets that can be used in attribute values.

Table 9–1: Character sets that can be used in attribute values

<table>
<tr><th>Syntactic element</th><th>Permissible characters</th><th>Example target</th></tr>
<tr><td><b>&lt;symbolic name&gt;</b></td><td><b>{&lt;alphabetic character> | <numeric character> | <code>@</code> | # | _ }+</b></td><td>Job name</td></tr>
<tr><td><b>&lt;environment variable name&gt;</b></td><td><b>{&lt;alphabetic character> | _}{<alphabetic character> | _ | numeric character&gt;}*</b></td><td>File environment variable definition name</td></tr>
<tr><td><b>&lt;path name&gt;</b></td><td>A character string that conforms to the path naming conventions of the OS.
Because \ is treated as metacharacter (escape character), path names in Windows must be specified as in the examples below. For details about metacharacters, see <i>5.1.6 Metacharacters</i>.
Examples: <code>'C:\test'</code> or <code>C:\\test</code></td><td>Path name</td></tr>
<tr><td><b>&lt;any character string&gt;</b></td><td>A string of any characters.
Using only the following characters is recommended:</td><td>Value of an environment variable</td></tr>
</table>

Syntactic element	Permissible characters	Example target
<any character string>	**{<alphabetic character> | <numeric character>** | `@` | `#` | `_` **}+**	Value of an environment variable

9.1.4 Script control statement description format

The description format for script control statements is as follows:

```
Δ0control-statement [Δ1condition] [Δ1reserved-word [Δ1processing]]...
      [Δ1condition] [Δ1reserved-word [Δ1processing]]
 [Δ0control-statement-(end)-or-reserved-word]
```

condition, *reserved-word*, *processing*

Specify a condition, a reserved word, and processing, respectively.

9.1.5 Reserved script command description format

The description format for reserved script commands is as follows:

```
Δ0command-name[Δ1option]...[Δ1option][Δ1operands]
```

- First specify options, and then specify operands. Operands include option names, option values, and other arguments that can be specified in commands. If an operand is specified before an option, all specified items are treated as operands.
- Specify an option in the format `-`*option-name*`[` Δ_1*value*`]`. When there is more than one option, they can be specified in any order.
- Options specified without a value can be grouped together in a block (for example: `-a -b -c` is equivalent to `-abc`). In this case, a value can be specified for the last option (for example, `xyz` in the specification `-abc xyz` is the value of option `-c`).
- An error results if you specify an invalid option or an option's value is outside the permissible range of values.
- Multibyte characters are not permitted in *option-name*.

9.2 Lists of commands and control statements

This section provides overviews of the standard shell commands, extended shell commands, extended script commands, script control statements, and reserved script commands.

9.2.1 List of standard shell commands

The standard shell commands include special built-in commands and regular built-in commands.

Table 9–2: List of special built-in commands

Command name	Overview
`.`	Executes a shell script.
`:`	Expands arguments and returns `0` as the return code.
`break`	Exits from a loop.
`continue`	Interrupts loop processing and returns to the beginning of the next cycle of the loop.
`eval`	Concatenates arguments into a command that it executes.
`exec`	Executes a specified command and then exits.
`exit`	Exits the shell.
`export`	Exports shell variables.
`readonly`	Sets the read-only attribute for variables or displays all read-only variables.
`return`	Returns from a function or an external script.
`set`	Sets shell options, creates an array, or displays variable values.
`shift`	Shifts the run-time parameters.
`trap`	Specifies the action when signals and forced termination requests are received.
`typeset`	Declares explicitly the attributes and values of variables and functions.
`unset`	Unsets variable values and attributes.

Table 9–3: List of regular built-in commands

Command name	Overview
`alias`	Defines aliases.
`builtin`	Executes a built-in command.
`cd`	Changes the current directory.
`command`	Executes a built-in command or external command.
`echo`	Outputs what is specified in arguments to the standard output.
`false`	Returns `1` as the return code.
`getopts`	Parses option arguments.
`kill`	Sends a signal to processes.
`let`	Evaluates the values of arithmetic expressions.

Command name	Overview
print	Outputs what is specified in the arguments to the standard output.
pwd	Outputs the path of the current directory.
read	Reads from the standard input and stores the input in shell variables.
test	Determines the value of a conditional expression.
times	Displays the amount of CPU time used by the shell.
true	Returns 0 as the return code.
ulimit (UNIX only)	Sets or displays information about limits on system resources.
umask (UNIX only)	Sets or displays the file mode creation mask.
unalias	Removes alias definitions.
wait	Waits for child processes to complete.
whence	Displays how specified character strings would be interpreted if used as commands.

9.2.2 List of extended shell commands

The table below lists the extended shell commands. Both these commands are regular built-in commands.

Table 9–4: List of extended shell commands

Command name	Overview
adshecho	In the user-reply functionality, issues a specified event notification message as a JP1 event.
adshread	In the user-reply functionality, issue a specified reply-request message as a reply-waiting event.
adshscripttool (Windows only)	Collects and outputs information in order to make it easier to create job definition scripts.

9.2.3 List of extended script commands

The table below lists the extended script commands and shows for each command the maximum number of times it can be specified in a job.

Table 9–5: List of extended script commands

Command name	Overview	Maximum number of times specified#
#-adsh_file	Assigns and postprocesses a regular file.	4,095
#-adsh_file_temp	Assigns and postprocesses a temporary file.	4,095
#-adsh_job	Declares a name for a job.	1
#-adsh_job_stop	Defines termination conditions for a job.	1,023
#-adsh_path_var	Defines shell variables for handling path names.	1
#-adsh_rc_ignore	Defines commands to always terminate normally.	1,023

Command name	Overview	Maximum number of times specified#
#-adsh_script	Calls an external job definition script file from the job definition script that is running.	4,095
#-adsh_spoolfile	Assigns a program output data file.	In a job: 4,095 In a job step: 255 Outside a job step: 255
#-adsh_step	Defines a job step, using the following three commands: #-adsh_step_start, #-adsh_step_end, and #-adsh_step_error.	4,095

#

The maximum number of times a command can be specified in a job includes the sum of the number of times it is specified in the script defined in the job definition script file specified in an argument of the adshexec command plus the number of times it is specified in external scripts called from that script by the #-adsh_script command. External scripts include scripts in nested calls.

9.2.4 List of script control statements

The table below lists the script control statements.

Table 9–6: List of script control statements

Control statement	Overview
case	Performs one among several processing steps, depending on a match with a character string.
for	Repeats the same processing while incrementing a value.
if	Controls processing by executing branching based on the result of evaluating a condition.
until	Executes specified processing repeatedly until a specified condition becomes true.
while	Executes specified processing repeatedly as long as a specified condition is true.

9.2.5 List of reserved script commands

The table below lists the reserved script commands.

Table 9–7: List of reserved script commands

Command name	Overview
time	Displays the time used to execute a command.

9.3 Standard shell commands

The standard shell commands are divided into special built-in commands and regular built-in commands, as shown below. A built-in command is one that is included as part of the shell, and is executed by the shell itself.

- Special built-in commands
  If a special built-in command's syntax is invalid, it exits the shell that is executing the command.
- Regular built-in commands
  Even if a regular built-in command's syntax is invalid, it does not exit the shell that is executing the command.

. command (executes a shell script)

Format

```
. filename  [args]
```

Description

This command executes a shell script in the current shell. The shell script specified in *filename* is executed in the shell environment. The current directory of the shell script specified in *filename* will be the same as the current directory of the current shell being used by the . (dot) command.

Variables and functions that are set and defined in the specified shell script can be used in the current shell environment even after the specified shell script terminates. Also, variables and functions that were set or defined before the specified shell script was started can be used in the specified shell script. However, extended script commands cannot be used in the specified shell script. If an extended script command is encountered, it will be handled as a comment. For details about the extended script commands, see *9.5 Extended script commands*.

Arguments

filename

Specifies the file name of the shell script that is to be executed in the current shell.

args

Specifies positional parameters to be used in the specified shell script. How the value of a positional parameter is set depends on whether *args* was specified and whether the positional parameter is changed in the specified shell script file, as summarized in the following table:

args is specified	Positional parameter is changed in the specified shell script file	Positional parameter is not changed in the specified shell script file
Yes	• The value of the positional parameter in the file will be the value specified in the *args* argument. • The value of the positional parameter after the file has terminated will be its value just before execution of the . (dot) command.	Same as at the left.
No	• The value of the positional parameter in the file will be the value just before execution of the . (dot) command. • The value of the positional parameter after the file has terminated will be the new value acquired in the file.	• The value of the positional parameter in the file will be the value just before execution of the . (dot) command.

args is specified	Positional parameter is changed in the specified shell script file	Positional parameter is not changed in the specified shell script file
No	• The value of the positional parameter in the file will be the value just before execution of the . (dot) command. • The value of the positional parameter after the file has terminated will be the new value acquired in the file.	• The value of the positional parameter after the file has terminated will be its value just before execution of the . (dot) command.

Return codes

Return code	Meaning
0 to 255	Normal termination • The return code is set by the script that executed.
0	Normal termination • The command executed but no file was specified in the *filename* argument.
1	Error termination • A file other than a regular file was specified in the *filename* argument. • The file specified in the *filename* argument could not be read.

Notes

- The shell script specified in this command does not send output to the script image file in the spool directory. To output the execution history to the script image file, you must use the #-adsh_script command.
- When this command terminates normally, its execution results are not output to the job execution log. Note also that this command does not identify whether the job or job step terminated normally or with an error. Refer instead to the execution results of the external script that was called.
- If the command syntax of this special built-in command is invalid, it exits the shell that is executing the command.

Usage example

- Execute a shell script in the current shell.

```
. ./test.sh
```

: command (expands arguments)

Format

```
: [arguments]
```

Description

This command expands arguments. It always returns 0 as the return code.

For example, you can omit the else or elif clause from an if statement, but not the then clause. In such a case, specify the : command for the then clause as follows, to indicate that no action is to be taken when the condition is satisfied:

```
if [conditional-expression]; then
  :                  # Takes no action if the result of the conditional
expression is true.
else
  cmd1               # Executes command if the result of the conditional
expression is false.
fi
```

Arguments

arguments

Specifies arguments that are to be expanded as explained below.

Job definition script specification example

```
set -x
NUMBER=1
: $NUMBER
```

Result that is output to the standard error output

```
+ NUMBER=1
+ : 1                  # Outputs the expansion result of the variable
NUMBER.
```

Therefore, by specifying a variable substitution in the `arguments` argument, you can check whether a value is stored in the variable and substitute a value if there is no value.

Job definition script specification example

```
STRING01=ABC
: ${STRING01:=DEF}          # Because variable STRING01 stores ABC, no
action is taken.
: ${STRING02:=GHI}          # Because variable STRING02 is undefined,
CHI is substituted.
echo $STRING01 $STRING02    # "ABC GHI" is output to the standard
output.
```

If variable substitution in the format ${*variable*:?*word*} or ${*variable*?*word*} is specified in the `arguments` argument and no value is stored in *variable*, processing terminates with an error with `1` as the return code.

Note that this command accepts no options. If an option is specified, it is ignored and processing continues.

Return codes

Return code	Meaning
`0`	Normal termination
`1`	Error termination • A variable substitution in the format ${*variable*:?*word*} was specified in an argument, but *variable* was not defined or no value was set in *variable*. • A variable substitution in the format ${*variable*?*word*} was specified in an argument, but *variable* was not defined.

Notes

- If the value of *n* is 0 or smaller or exceeds 2,147,483,647, overflow will occur. The overflow digits are ignored and processing continues. We recommend that you specify a value in the range from 1 to 2,147,483,647.
- If the command syntax of this special built-in command is invalid, it exits the shell that is executing the command.

Usage example

- Only expand the arguments.

```
: $test
```

alias command (defines aliases)

Format

```
alias  [ -p|-x|+p|+x]  [name  [=value]...]
```

Description

This command defines aliases or outputs defined aliases to the standard output. If you do not specify at least one argument, it outputs the names and values of the aliases that are currently defined.

Arguments

-p

Outputs the aliases that have been defined, in the form `alias` *alias-name=value*.

-x|+x

Defines or outputs exported aliases.

+p

Outputs the aliases that have been defined, in the form `alias` *alias-name*.

name

Specifies a name for an alias to be defined or output. To output an alias, specify in *name* the name of an alias that has already been defined.

value

Specifies the command to set to the alias name specified in *name*. To define an alias, specify *value* in the form *alias-name=value*.

Return codes

Return code	Meaning
0	Normal termination
1	Error termination • An attempt was made to output an alias that has not been defined.

Notes

- Even if the command syntax of this regular built-in command is invalid, it does not exit the shell that is executing the command.

Usage example

- Output a defined alias (`functions`).

  Contents of the job definition script:

  ```
  alias functions='typeset -f'
  alias functions
  ```

  Contents of the `STDOUT` file of the execution job:

  ```
  ********   JOB SCOPE STDOUT    ********
  functions='typeset -f'
  ```

break command (exits from a loop)

Format

```
break  [n]
```

Description

This command exits from a loop that occurs in a `for` statement, `while` statement, or other looping control structure. If this command is executed while processing is not in a loop, it outputs a message and terminates normally.

Argument

n

Specifies the number of nested loops from which the command is to exit. Specify `1` or a greater integer.

When *n* is specified, the command exits out of *n* loops. When *n* is omitted, the command exits to the first enclosing loop.

If the command is executed with a value for *n* that exceeds the current number of nested looping structures, it continues exiting through the outermost loop, and then outputs a message and terminates normally.

Return codes

Return code	Meaning
`0`	Normal termination
`1`	Error termination • `0` was specified for *n*. • A non-numeric value was specified for *n*. • A negative value or an option character string (*-alphanumeric-character*) was specified for *n*.

Notes

- If the value of *n* is 0 or smaller or exceeds 2,147,483,647, overflow will occur. The overflow digits are ignored and processing continues. We recommend that you specify a value in the range from 1 to 2,147,483,647.
- If the command syntax of this special built-in command is invalid, it exits the shell that is executing the command.

Usage example

- Exit to the second enclosing loop.

```
break 2
```

builtin command (executes a built-in command)

Format

```
builtin  [command  [args ...]]
```

Description

This command executes a specified built-in command with arguments that are specified.

Arguments

command

Specifies the name of the built-in command that is to be executed. If this argument is omitted, the command terminates normally.

args

Specifies arguments for the built-in command.

Return codes

Return code	Meaning
0	Normal termination • The built-in command terminated normally.
1	Error termination • The specified command in the argument was not a built-in command, or the command terminated with an error.

Notes

- This command accepts no options. If an option is specified in the arguments, it will be interpreted as an argument of `command` and the job will terminate with an error.
- The execution results of this command are not output to the job execution log. Note also that this command does not identify whether the job or job step terminated normally or with an error. Refer instead to the execution results of the command that was called.
- Even if the command syntax of this regular built-in command is invalid, it does not exit the shell that is executing the command.

Usage example

- Use the `builtin` command to execute the `pwd` built-in command.

  Contents of the job definition script

  ```
  cd /tmp
  builtin pwd
  ```

  Contents of the `STDOUT` file of the execution job

  ```
  ********   JOB SCOPE STDOUT    ********
  /tmp
  ```

cd command (changes the current directory)

Format 1

```
cd [directory-path]
```

Format 2

```
cd old new
```

Description

This command changes the current directory. The destination can be specified in either of two formats.

The first format specifies the destination in *directory-path*. If the variable `CDPATH` is defined, this format specifies a directory relative to the location defined in `CDPATH`. If the variable `CDPATH` is not defined, this format specifies a directory relative to the current directory.

In the second format, the change is to the directory path in which the character string matching *old* is replaced with *new* in the path name of the current directory.

Arguments

directory-path

Specifies the new directory path.

If this argument is omitted, the change is to the user's home directory (`HOME` variable). If a hyphen (`-`) is specified for this argument, the change is to the previous working directory (`OLDPWD` variable).

old

Specifies a character string that is to be replaced in the path name of the current directory.

new

Specifies the character string that is to replace the specified character string in the path name of the current directory.

Return codes

Return code	Meaning
0	Normal termination

Return code	Meaning
1	Error termination

Notes

- Executing the cd command in Windows converts the directory delimiter from / to \.
- Even if the command syntax of this regular built-in command is invalid, it does not exit the shell that is executing the command.
- If HOME is not defined, an error results if you execute the cd command with no argument specified.
- A name in UNC format cannot be specified for the directory path name.
- In Format 2, if a character string that occurs multiple times in the current directory path name is specified for substitution, only the first instance of that character string will become subject to substitution. An example follows.

  Example: The current directory path name is /home/user/test/test

  ```
  cd test tmp.
  ```

  In this case, the cd command attempts to change the path to /home/user/tmp/test instead of /home/user/tmp/tmp.

Usage example

- Change from /var/log to /var/lib.

  Contents of the job definition script

  ```
  pwd
  cd log lib
  p2w2d1222
  ```

  Contents of the STDOUT file of the execution job

  ```
  ********   JOB SCOPE STDOUT    ********
  /var/log
  /var/lib
  /var/lib
  ```

command command (executes a command)

Format

UNIX

```
command  [-p] [command  [args ...]]
command  [-v|-V] [-p] [command  [command ...]]
```

Windows

```
command  [-w] [command  [args ...]]
command  [-v|-V] [command  [command ...]]
```

Description

This command executes a command or a built-in command.

The command specified in *command* is executed with the arguments specified in *args*.

When the `-v` option is specified, the same command path name as for the `whence` command is output to the standard output. When the `-V` option is specified, the same command interpretation as for the `whence -v` command is output to the standard output. If the `-v` and `-V` options are both specified, the `-V` option takes precedence.

For details about the output format, see *whence command (displays how character strings would be interpreted if used as commands)*.

Arguments

-p (UNIX only)

Specifies that the specified command is to be found on the standard path.

-w (Windows only)

Specifies that when an external command is executed in Windows, the following processing is not to be performed:

- Converting \ to \\ preceding double quotation marks (`"`) in arguments
- Prefixing \ to any double quotation mark (`"`) in arguments
- Enclosing arguments in double quotation marks (`"`)

However, even when this option is specified, the processing noted above is performed on the character string specified in *command*.

-v

Specifies that the command path for the character string specified in *command* is to be output as if it were to be treated as a command.

-V

Specifies that whether the character string specified in *command* is a command, reserved word, alias, standard shell command, extended shell command, or function is to be output.

command

Specifies the name of the command that is to be executed or a character string that is to be treated as a command. If this argument is omitted, the command terminates normally but without executing anything.

args

Specifies arguments to be passed to the specified command.

Return codes

Without the `-v` or `-V` option

Return code	Meaning
`0`	Normal termination
`127`	Error termination • The command could not be identified.
Other than the above	Error termination • The format of the command is invalid, or the command terminated with an error.

With the `-v` or `-V` option

Return code	Meaning
0	Normal termination
1	Error termination, or the value specified in *command* could not be found as a command.

Notes

- When this command terminates normally, its execution results are not output to the job execution log. Note also that this command does not identify whether the job or job step terminated normally or with an error. Refer instead to the standard output or to the execution results of the command that was called.
- Even if the command syntax of this regular built-in command is invalid, it does not exit the shell that is executing the command.

Usage example

- Use the `command` command to execute the `pwd` command.

  Contents of the job definition script

  ```
  command -p pwd
  ```

  Contents of the `STDOUT` file of the execution job

  ```
  ********   JOB SCOPE STDOUT    ********
  /tmp
  ```

continue command (interrupts loop processing and returns to the beginning of the loop)

Format

```
continue  [n]
```

Description

This command interrupts processing of a `for` statement, `while` statement, or other looping control structure and returns to the beginning of the loop. The argument specifies that the current iteration of the *n*th enclosing loop is to be interrupted. If this command is executed while processing is not in a loop, it outputs a message and terminates normally.

Argument

n

Specifies the number of nested loops for which the command is to skip processing. Specify `1` or a greater integer.

When *n* is specified, the command skips processing of *n* loops, and resumes processing from the beginning of the n^{th} enclosing loop. When *n* is omitted, the command skips processing in the current level loop, and resumes processing from the beginning of the enclosing loop.

If the command is executed with a value for *n* that exceeds the current number of nested looping structures, it resumes processing from the beginning of the outermost loop, and then outputs a message and terminates normally.

Return codes

Return code	Meaning
`0`	Normal termination
`1`	Error termination • `0` was specified for *n*. • A non-numeric value was specified for *n*. • A negative value or an option character string (-*alphanumeric-character*) was specified for *n*.

Notes

- If the value of *n* is 0 or smaller or exceeds 2,147,483,647, overflow will occur. The overflow digits are ignored and processing continues. We recommend that you specify a value in the range from 1 to 2,147,483,647.
- If the command syntax of this special built-in command is invalid, it exits the shell that is executing the command.

Usage example

- Interrupt the current iteration of the second enclosing loop.

```
continue 2
```

echo command (outputs what is specified in arguments to the standard output)

Format

```
echo  [-n] [-e|-E]  [args ...]
```

Description

This command outputs what is specified in arguments to the standard output.

During the output processing, escape characters that begin with a backslash (`\`) are replaced. The following table shows the escape characters that are replaced:

Escape character	Meaning
`\a`	Alert character (bell)
`\b`	Backspace character
`\c`	Suppress trailing linefeed (characters after `\c` are not output)
`\f`	Formfeed character (page break)
`\n`	Linefeed character
`\r`	Carriage return character
`\t`	Tab character
`\v`	Vertical tab character
`\0`*nnn*#1	ASCII character represented by one, two, or three octal digits (`0` to `7`)

Escape character	Meaning
`\x`*nn*[#2]	ASCII character represented by one or two hexadecimal digits (0 to 9, a to f, A to F)
`\\`	A single backslash character

#1

If a specified ASCII character consists of one or two digits and two or one leading zeros, respectively, are added to make it three digits, the ASCII character will still be treated as consisting of only one or two digits. For example, the following three specifications are all interpreted as being the same, in which case the alert character (bell) is output three times:

```
echo -e "\07"
echo -e "\007"
echo -e "\0007"
```

#2

Enabled only when `YES` is specified in the `ESCAPE_SEQ_ECHO_HEX` environment setting parameter. For details about the `ESCAPE_SEQ_ECHO_HEX` parameter, see *ESCAPE_SEQ_ECHO_HEX parameter (specifies whether ASCII code characters in hexadecimal notation are to be interpreted as escape characters)* in *7. Parameters Specified in the Environment Files*.

If a specified ASCII character consists of one digit and a leading zero is added to make it two digits, the ASCII character will still be treated as consisting of only one digit. For example, the following two specifications are interpreted as being the same, in which case the linefeed character is output twice:

```
echo -e "\xA"
echo -e "\x0A"
```

If you want to replace an escape character, enclose the `-e` option argument in single or double quotation marks (`'` or `"`), as in Example 2 below. The following examples show how escape characters are interpreted depending on whether quotation marks are used and whether the `-e` or `-E` option is specified.

1. In this example, `ta` is output to the standard output:

```
echo -e \ta
```

2. In this example, *tab-character*`a` is output to the standard output:

```
echo -e "\ta"
```

3. In this example, `ta` is output to the standard output:

```
echo -E \ta
```

4. In this example, `\ta` is output to the standard output:

```
echo -E "\ta"
```

Options specified for commands and interpretation of arguments

If the characters specified as the arguments to the `echo` command are all valid option characters, they are interpreted as options. For example, because the letters in `eEn` are all valid option characters, they are interpreted as options in the following example:

```
echo -eEn
```

However, if even one character is not a valid option character, the characters are interpreted as an *args* value. In the next example, because `a` is not a valid option character, `-eEna` is interpreted as an *args* value, which will be sent to the standard output.

```
echo -eEna
```

Finally, when an argument is enclosed in quotation marks, the entire enclosed character string is interpreted as a single argument. In the next example, because the space is not a valid option character, `-e a` is interpreted as an *args* value, which will be sent to the standard output.

```
echo "-e a"
```

Interpretation of escape characters (-e and -E options)

Escape characters are interpreted as follows, depending on the specification of the `-e` and `-E` options:

- When the `-e` option is specified, escape characters are interpreted.
- When the `-E` option is specified, escape characters are not interpreted.
- When the `-e` and `-E` options are both specified, the option specified last takes effect.
- When neither the `-e` nor the `-E` option is specified, the processing depends on the specification of the `ESCAPE_SEQ_ECHO_DEFAULT` environment setting parameter. For details about the `ESCAPE_SEQ_ECHO_DEFAULT` parameter, see *ESCAPE_SEQ_ECHO_DEFAULT parameter (defines the action of the echo command when the escape-character option is omitted)* in *7. Parameters Specified in the Environment Files*.

Arguments

-n

Specifies that trailing linefeeds are to be omitted from the output to the standard output.

-e

Specifies that escape characters are to be interpreted. The escape characters to be interpreted are determined by the specification of the `ESCAPE_SEQ_ECHO_HEX` environment setting parameter. If you want escape characters to be interpreted, enclose them in single or double quotation marks (`'` or `"`).

-E

Specifies that escape characters are not to be interpreted.

args

Specifies the arguments (what it is that is to be output).

Return codes

Return code	Meaning
0	Normal termination
1	Error termination

Notes

- Even if the command syntax of this regular built-in command is invalid, it does not exit the shell that is executing the command.
- If an escape character is specified using an ASCII code in hexadecimal notation, the result is the same as when the escape character is specified directly.

  In both of the following specifications, `a`*tab-character*`b` is output:

```
echo -e "a\tb"
echo -e "a\x09b"
```

In both of the following specifications, `a`*linefeed-character*`b` is output to `out.txt`, but the linefeed character will be `CR+LF` (Windows only):

```
echo -e "a\nb" > out.txt
echo -e "a\x0ab" > out.txt
```

- If a value outside the ASCII code range is specified when escape characters are expressed as ASCII character strings, the content to be output follows the character encoding specified for the terminal and unprintable characters might produce an incorrect output.
- If you specify a path name for an argument, execute the `echo` command in an environment in which the `-E` option is specified or `NO` is specified in the `ESCAPE_SEQ_ECHO_DEFAULT` parameter in order to prevent the backslash (`\`) from being replaced as an escape character.

  For example, the path name (`d:\a\b\c`) is not output correctly by any of the following commands:

  ```
  FILE="d:\\a\\b\\c"
  echo $FILE
  echo "$FILE"
  echo "d:\\a\\b\\c"
  echo 'd:\a\b\c'
  ```

  The path name (`d:\a\b\c`) is output correctly by all the following commands:

  ```
  FILE="d:\\a\\b\\c"
  echo -E $FILE
  echo -E "$FILE"
  echo -E "d:\\a\\b\\c"
  echo -E 'd:\a\b\c'

  FILE="d:\\\\a\\\\b\\\\c"
  echo $FILE
  echo 'd:\\a\\b\\c'
  echo d:\\\\a\\\\b\\\\c
  ```

Usage example

- Output the variable `LANG`.

  Contents of the job definition script

  ```
  echo $LANG
  ```

  Contents of the `STDOUT` file of the execution job

  ```
  ********   JOB SCOPE STDOUT    ********
  ja_JP.eucJP
  ```

eval command (concatenates arguments into a command and executes it)

Format

```
eval  [command  [args ...]]
```

Description

This command concatenates arguments into a command that it executes. The character strings provided as the arguments are interpreted as a single command that is executed.

Arguments

command

Specifies a name for the command that is to be executed. If this argument is omitted, the `eval` command terminates normally but without executing anything.

args

Specifies the arguments that are to be concatenated into a command and executed.

Return codes

Return code	Meaning
`0`	Normal termination
`127`	Error termination • The command could not be identified.
Other than the above	Error termination • The format of the command is invalid, or the command terminated with an error.

Notes

- The execution results of this command are not output to the job execution log. Note also that this command does not identify whether the job or job step terminated normally or with an error. Refer instead to the execution results of the command that was called.
- If the command syntax of this special built-in command is invalid, it exits the shell that is executing the command.

Usage example

- Change directories to `/home/adsh/script`.

```
eval cd /home/adsh/script
```

exec command (executes a command and exits)

Format

```
exec  [command  [args]...]
```

Description

This command executes a specified command and then exits.

If an external command is specified as the argument, it executes the command as a child process of the `adshexec` command. After waiting for the external command to complete, it performs postprocessing for the job, such as deleting temporary files.

If you specify only the input/output redirection symbol and a redirection target, it switches the input and output targets according to the input/output redirection symbol. For details about redirection, see *5.1.6(8) Input and output redirection*.

Arguments

command

Specifies the command name of the command that is to be executed. If nothing is specified in this argument, the `exec` command does nothing, and execution of the job definition script continues.

args

Specifies arguments for the command that is to be executed.

Return codes

Return code	Meaning
`0`	Normal termination
`127`	Error termination • The command could not be identified.
Other than the above	Error termination • The command terminated with an error.

Notes

- If the command syntax of this special built-in command is invalid, it exits the shell that is executing the command.

Usage examples

- Execute the user program `UAP01` and terminate the job:

```
exec UAP01
```

- Redirect the standard output destination to `file01`:

```
exec > file01
```

exit command (exits the shell)

Format

```
exit  [n]
```

Description

This command exits the shell. Regardless of the value of the return code, this command terminates normally or with an error on the basis of whether the command syntax is valid.

If no argument is specified, the command terminates normally with the return code of the command that executed last as its return code. When executed with an appropriate numeric value specified for the argument, the command terminates normally. When executed with an invalid value, such as non-numeric characters, specified as the argument, the command terminates with an error. When the command terminates with an error, it returns `1` as the return code.

When this command is executed within a job step error block, the results are as follows:

- If the argument is specified and the command terminates normally, the value specified in the argument is set as the job step's return code.
- If the argument is not specified and the command terminates normally, or if it terminates with an error with the argument specified, the return code of the job step will be the return code of the last command to execute within the job step normal block.

Arguments

n ~<unsigned integer>((0 to 255))

Specifies the return code to be set upon exiting the shell. If this argument is omitted, the command exits the shell with the return code of the last command that executed. If you specify `256` or a greater value for this argument, the command terminates normally with a return code that is the remainder of dividing the specified value by 256. If you specify a negative value for this argument, the command terminates normally with a return code that is the two's complement of the specified value.

Return codes

Return code	Meaning
`0` to `255`	Normal termination • Returns either *n* or the return code of the last command that executed.
`1`	Error termination • A non-numeric value was specified for *n*.

Notes

- You can specify for *n* a negative value or a value that is greater than `255`, but we recommend that you specify in JP1/Advanced Shell a value in the range of `0` to `255`.
- If the command syntax of this special built-in command is invalid, it exits the shell that is executing the command.
- When executing the `exit` command in a separate process, such as by using the `&` operator or a command substitution, also see the notes provided in *5.1.7 Execution in a separate process (UNIX only)*.

Usage example

- Exit the shell with a return code of `2`.

```
exit 2
```

export command (exports shell variables)

Format

```
export  [-p] [name[=value]...]
```

Description

This command exports specified shell variables. If the `-p` option and *name* argument are both specified, export of the variables specified in *name* takes precedence.

If the command is executed with no options specified, it outputs to the standard output the names of all the currently exported variables.

Arguments

-p

Specifies that all currently exported variables are to be output to the standard output in the format `export` *variable-name=value*.

name

Specifies the name of a variable that is to be exported.

When you export shell variables in Windows, the supported variable names vary as follows:

- When `DISABLE` is specified in the `VAR_ENV_NAME_LOWERCASE` parameter

  All letters contained in variable names must be in uppercase because shell variables whose names contain lowercase letters cannot be exported.

  If you attempt to export a shell variable whose name includes lowercase letters, the command outputs an error message and exits the batch job.

- When `ENABLE` is specified in the `VAR_ENV_NAME_LOWERCASE` parameter

  Shell variables whose names contains lowercase letters can be exported.

  Note that environment variables are not case sensitive. The last shell variable with the same spelling that was exported becomes the final environment variable value. Shell variables with the same spelling but in a different case are considered to be different values regardless of whether they are exported.

For *name*, you can specify multiple names of variables or arrays that are to be exported. If an array name is specified, the command exports all the elements that constitute the array. Even if you specify a single element of an array, all the elements of the array will be exported.

If you specify a variable that has not yet been created, the variable will be created and exported simultaneously. However, if you do not specify a value for the variable in such a case, the linefeed character will be set as the value and then the variable will be exported.

If the read-only attribute is set for a shell variable that is specified, and you attempt to specify a value for it, the command will terminate with an error.

value

Specifies a value that is to be assigned to the paired specified variable.

When =*value* is specified, the specified value is assigned and the variable with the specified value is exported simultaneously. When this argument is omitted, the specified variable is exported using the value that is already set for it.

Return codes

Return code	Meaning
`0`	Normal termination
`1`	Error termination

Notes

- If the command syntax of this special built-in command is invalid, it exits the shell that is executing the command.

Usage example

- Export the shell variable `HOME` after assigning `"/home/jp1as"` to it.

```
export HOME="/home/jp1as"
```

false command (returns 1 as the return code)

Format

```
false
```

Description

This command returns 1 as the return code.

Note that this command accepts no options. If an option is specified, it is ignored and processing continues.

Return code

Return code	Meaning
1	Normal termination • This command always returns 1.

Notes

- Even if the command syntax of this regular built-in command is invalid, it does not exit the shell that is executing the command.
- The execution result of this command is output in the KNAX6113-I message.

Usage example

- Set return code 1.

  Contents of the job definition script

  ```
  false
  echo $?
  ```

  Contents of the STDOUT file of the execution job

  ```
  ********   JOB SCOPE STDOUT    ********
  1
  ```

getopts command (parses option arguments)

Format

```
getopts optstr name  [args ...]
```

Description

This command parses specified option arguments.

Arguments

optstr

Specifies a string of valid option characters. When a character is followed by a colon (:), it indicates that the option has an associated value.

You specify a string of option characters that are valid as arguments for specification on the command line or in *args*. For example, specify ab for the options -a and -b. Multibyte characters cannot be used.

When a character specified in this argument is determined to be a valid option, the matching option is set in the variable whose name is specified in *name*. When a specified character is not found to be a valid option, ? is set in the variable whose name is specified in *name*.

If an option has a value, specify a colon (:) after the option character, in which case the value associated with the matching option will be set in the OPTARG shell variable. The getopts command parses the options starting at the argument index (which begins with 1) specified in the OPTIND shell variable. For example, if -a 10 is specified in *args*, the position of -a will be index 1.

name

Specifies a variable in which the option characters matched by the getopts command are to be set.

args

Specifies arguments that are to be parsed. When this option is omitted, getopts parses only the command-line options.

Return codes

Return code	Meaning
0	Normal termination
1	• Normal termination (detects option termination) • Error termination

Notes

- Even if the command syntax of this regular built-in command is invalid, it does not exit the shell that is executing the command.
- When executing the getopts command in a separate process, such as the & operator and a command substitution, also see the notes provided in *5.1.7 Execution in a separate process (UNIX only)*.

Usage example

- Parse -b as a valid option.

  Contents of the job definition script

```
getopts b: name -b 10
echo $name
```

  Contents of the STDOUT file of the execution job

```
********    JOB SCOPE STDOUT    ********
b
```

kill command (sends a signal)

Format

```
kill  [-s {signame|signum}] {pid|-pid}...
kill  [-signame|-signum] {pid|-pid}...
```

Description

This command sends a signal to one or more processes. The specified signal it is sent to one or more specified processes. If no signal to be sent is specified, `SIGTERM` is sent. When you specify by name the signal that is to be sent, specify its name without the leading characters `SIG` (for example, specify `INT` for `SIGINT`). For the specification for each signal, see the documentation for the OS being used.

Arguments

-s

Specifies that a signal to be sent is being specified by its signal number or signal name.

signame|-signame

Specifies the signal name of the signal that is to be sent.

signum|-signum

Specifies the signal number of the signal that is to be sent.

pid

Specifies the process ID of a process that is to receive the signal.

-pid

Specifies the process ID of a process when the signal is to be sent to all processes that belong to the specified process's process group. In Windows, you must specify a value greater than 0 to send the signal to multiple processes.

Return codes

Return code	Meaning
0	Normal termination
1	Error termination

Notes

- In Windows, an error results if you specify any signal other than `SIGKILL`.
- In Windows, when `SIGKILL` is specified, `TerminateProcess()` is used to forcibly terminate the process.
- Even if the command syntax of this regular built-in command is invalid, it does not exit the shell that is executing the command.
- To send the default signal (`SIGTERM`) in the *−signame* or *−signum* format to a process group (*−pid*), you must specify `--` as the signal. If you do not specify `--`, the process group (*−pid*) will be interpreted as *−signum*. An example follows:

  This example sends the default signal (`SIGTERM`) to process group `14588`:

  ```
  kill -- -14588
  ```

Usage examples

- In UNIX, send SIGINT to process ID 4725.

```
kill -INT 4725
```

- In Windows, forcibly terminate process ID 4725.

```
kill -KILL 4725
```

let command (evaluates the values of arithmetic expressions)

Format 1

```
let arithmetic-expression[,arithmetic-expression ... ]
```

Format 2

```
((arithmetic-expression))
```

Description

This command performs numeric calculations in order to evaluate specified arithmetic expressions.

In addition to using the let command, you can also calculate the result of an arithmetic expression using the syntax ((*arithmetic-expression*)), and the results will be evaluated in the same way as for the let command.

The let command accepts multiple arithmetic expressions delimited by the comma (,). If you specify more than one arithmetic expression, they are calculated in order from left to right. As a result, if you use a conditional expression to evaluate a comma-separated list of arithmetic expressions, a conditional is evaluated taking into account the result of the arithmetic expression that was executed last. Note that specifying any spaces before or after a delimiter comma results in termination of the command with an arithmetic error. You can group calculations in parentheses in order to change the priority of operations.

For details about arithmetic expressions, see *5.3 Arithmetic operations*. For details about conditionals, see *5.2 Conditionals*.

Return codes

Return code	Meaning
0	Normal termination • The value of the arithmetic expression is not 0.
1	Normal termination • The value of the arithmetic expression is 0. • The syntax (( )) executed but no arithmetic expression was specified.
	Error termination • The let command executed but no arithmetic expression was specified.
2	Error termination • An arithmetic error occurred (division by zero, invalid arithmetic expression).

Notes

- Even if the command syntax of this regular built-in command is invalid, it does not exit the shell that is executing the command.
- The arithmetic operators `*`, `&`, <, <<, >, and >> have special meanings as metacharacters. To use these characters in the `let` command, you must disable them as metacharacters.

  Example: Set the variable `RC` to 1 shifted 2 bits to the left.

  - Contents of the job definition script

  ```
    let "RC=1<<2"
    echo $RC
  ```

  Contents of the `STDOUT` file of the execution job

  ```
    ********    JOB SCOPE STDOUT    ********
    4
  ```

- The `let` command accepts no options. If you specify -*alphabetic-character* as an argument, it will be interpreted as a variable name, not an option.

  In the example shown below, `-a` is specified as the argument. It is interpreted as `-3`, with a return code of `0`.

  - Contents of the job definition script

  ```
    a=3
    let -a
    echo $?
  ```

  - Contents of the `STDOUT` file of the execution job

  ```
    ********    JOB SCOPE STDOUT    ********
    0
  ```

- When executing the `let` command in a separate process, such as the `&` operator and a command substitution, also see the notes provided in *5.1.7 Execution in a separate process (UNIX only)*.

Usage examples

- Add 3+4 and multiply the result by 2.

  Contents of the job definition script

  ```
  let "VAR=2*(3+4)"
  echo $VAR
  ```

  Contents of the `STDOUT` file of the execution job

  ```
  ********    JOB SCOPE STDOUT    ********
  14
  ```

- Set the variable `RC` to the result of 1+2.

  Contents of the job definition script

  ```
  ((RC=1+2))
  echo $RC
  ```

  Contents of the `STDOUT` file of the execution job

```
********   JOB SCOPE STDOUT    ********
3
```

print command (outputs to the standard output)

Format

```
print  [-n|-p|-r] [-u [num]] [--] [args]
```

Description

This command outputs what is specified in the arguments to the standard output. Trailing linefeeds are appended to the output.

Escape characters prefixed with \ are replaced in the output. The following table shows the escape characters that are replaced:

Escape character	Meaning
`\a`	Alert character (bell)
`\b`	Backspace character
`\c`	Suppress the trailing linefeed (characters after the `\c` are not output)
`\f`	Formfeed character (page break)
`\n`	Linefeed character
`\r`	Carriage return character
`\t`	Tab character
`\v`	Vertical tab character
`\0`*nnn*#	ASCII character represented by one, two, or three octal digits (`0` to `7`)
`\\`	A single backslash character

#
If a specified ASCII character consists of one or two digits and two or one leading zeros, respectively, are added to make it three digits, the ASCII character will still be treated as consisting of only one or two digits.

When the `-r` option is specified, escape characters are ignored.

Arguments

-n

Specifies that trailing linefeeds are to be omitted from the output to the standard output.

-p

Specifies that a pipe is to be used to send the output to the standard input of a background process, rather than to the standard output.

-r

Specifies that escape characters are to be ignored.

-u [num]

Specifies the file identifier to which the output is to be output. When no value is specified, `1` is assumed.

Specify either an output destination file identifier or `p`. Specifying `p` is equivalent to specifying the `-p` option.

--

Specifies the end of the option specifications. Any options specified after this option are interpreted as part of *args*.

args

Specifies the arguments (what is to be output).

Return codes

Return code	Meaning
`0`	Normal termination
`1`	Error termination

Notes

- Even if the command syntax of this regular built-in command is invalid, it does not exit the shell that is executing the command.
- If a value outside the ASCII code range is specified when escape characters are expressed as ASCII character strings, the content to be output follows the character encoding specified for the terminal and unprintable characters might produce an incorrect output.

Usage examples

- Output the character string `abc` with a trailing linefeed.

  Contents of the job definition script

  ```
  print "abc\n"
  ```

  Contents of the `STDOUT` file of the execution job

  ```
  ********   JOB SCOPE STDOUT    ********
  Abc
  ```

- Output the character string `abc` to the standard input of the `coproc.sh` background process.

  ```
  coproc.sh |&
  print -p abc
  ```

pwd command (outputs the path of the current directory)

Format

```
pwd  [-L|-P]
```

Description

This command outputs the path of the current directory to the standard output.

Arguments

Specifying no options is equivalent to specifying the `-L` option.

-L

Specifies that if the current directory path contains a symbolic link (a link using a file that contains the actual file path), the symbolic link is to be output. If the `-L` and `-P` options are both specified, the option specified last takes effect.

-P

Specifies that the path that does not contain a symbolic link is to be output. If the `-L` and `-P` options are both specified, the option specified last takes effect.

In Windows, the `-P` option is ignored.

Return codes

Return code	Meaning
0	Normal termination
1	Error termination

Notes

- In Windows, when you execute the `pwd` command, the directory delimiter `/` is displayed as `\`.
- Even if the command syntax of this regular built-in command is invalid, it does not exit the shell that is executing the command.

Usage example

- Output the path of the current directory.

  Contents of the job definition script

  ```
  cd /tmp
  pwd
  ```

  Contents of the `STDOUT` file of the execution job

  ```
  ********   JOB SCOPE STDOUT    ********
  /tmp
  ```

read command (reads from the standard input and stores the input in variables)

Format

```
read  [-p] [-r] [-u [num]] [varname ...]
```

Description

This command reads one line at a time from the standard input and stores the input in specified shell variables.

Arguments

-p

Specifies that the command is to read from the output of a background process through a pipe.

-r

Specifies that when there is a \ at the end of a line, the next line is not to be read as a continuation line.

-u [num]

Specifies the file identifier of the file from which the input is to be read. If no value is specified, the standard input is read.

Specify either a file identifier or `p`. Specifying `p` is equivalent to specifying the `-p` option.

varname

Specifies the name of a variable in which input that is read is to be stored.

When multiple variable names are specified, the input line is split into the fields delimited by the delimiter set in the `IFS` variable, and the fields are assigned to the variable names in sequential order (input line's first field is set in the first variable, second field is set in the second variable, and so on).

If the number of fields exceeds the number of variable names, the values of all the remaining fields are stored in the last variable that is specified.

If there are fewer fields than there are variables, a linefeed is set in each additional variable.

Return codes

Return code	Meaning
`0`	Normal termination
`1`	Normal termination • The end of file (EOF) was detected. Error termination • Other than the above.

Notes

- Even if the command syntax of this regular built-in command is invalid, it does not exit the shell that is executing the command.
- When executing the `read` command in a separate process, such as the `&` operator and a command substitution, also see the notes provided in *5.1.7 Execution in a separate process (UNIX only)*.
- In Windows, there are circumstances where keyboard input is accepted even before the `read` command starts. For example, this can happen while the system is executing a previous command, or when you are reading input from the console or command prompt while running the debugger within an editor and the debugger is stopped at a breakpoint. To prevent the input from being read incorrectly, do not make entries from the keyboard before the `read` command starts.

  If there is input from the keyboard before the `read` command starts, that input will be displayed when the `read` command starts. At that point, delete the keyboard input that is displayed and re-enter it for the `read` command.

Usage examples

- Read the file `string.txt` and send its contents to the standard output.

  Contents of the job definition script

```
while read LINE
do
 echo "$LINE"
done     < string.txt
```

Contents of the `STDOUT` file of the execution job

```
********   JOB SCOPE STDOUT    ********
HITACHI
JP1
Advanced Shell
```

- Read a character string output to the standard output by the `coproc.sh` background process and store it in the variable `NAME`.

```
coproc.sh |&
read -p NAME
```

readonly command (sets the read-only attribute for variables or displays all read-only variables)

Format

```
readonly  [-p] [name [=value]...]
```

Description

This command sets the read-only attribute for specified variables or displays all read-only variables. When the command is executed with no options, it outputs to the standard output the names of all the read-only variables.

To revert a variable's read-only attribute back to writable, specify the `+r` option to the `typeset` command.

Arguments

-p

Specifies that all read-only variables are to be output to the standard output in the format `readonly` *variable-name*=*value*. However, if you specify the `-p` option and *name* at the same time, the read-only attribute is set for the specified variable.

name

Specifies the name of a variable for which the read-only attribute is to be set.

More than one variable name or array name can be specified. However, if a function name is specified in *name*, the read-only attribute is not set for the function, instead the read-only attribute is set for a variable with the same name as the specified function.

If an array name is specified in *name*, the read-only attribute is set for all the elements that constitute the array. Even if you specify one element of an array, the read-only attribute is set for all the elements of the array.

If there is no variable with the specified name, such a variable is created, and the read-only attribute is set for it. In such case, if no value is specified for the attribute, the linefeed character is set as the new attribute's value and it has the read-only attribute.

If the read-only attribute is already set for the specified variable, the command terminates normally without doing anything.

value

Specifies a value to be set for the specified variable.

The specified value is set for the variable when *name* is followed by =*value*. If no value is specified, the read-only attribute is set for the specified variable without changing its value.

Return codes

Return code	Meaning
`0`	Normal termination
`1`	Error termination

Notes

- If the command syntax of this special built-in command is invalid, it exits the shell that is executing the command.

Usage example

- Set the read-only attribute for the `test` variable.

```
readonly test
```

return command (returns from a function or an external script)

Format

```
return  [n]
```

Description

This command returns from a function or an external script, then continues processing in the calling function. Note that this command exits the shell if it is executed at a location other than from within a function or an external script.

Regardless of the value of the return code, this command terminates normally or with an error on the basis of whether the command syntax is valid.

If no argument is specified, the command terminates normally with the return code from the command that executed last as the return code. If it executes with a valid numeric value specified as the argument, it terminates normally. If it executes with an invalid value specified as the argument, such as non-numeric characters, it terminates with an error. When this command terminates with an error, it returns `1` as the return code.

Arguments

n ~<unsigned integer>((0 to 255))

Specifies the return code. When this argument is omitted, the command returns with the return code from the command that executed last. If you specify `256` or a greater value for this argument, the command terminates normally with a return code that is the remainder from dividing the specified value by 256. If you specify a negative value, the command terminates normally with a return code that is the two's complement of the specified value.

Return codes

Return code	Meaning
`0` to `255`	Normal termination • Returns the value specified in *n* or the return code of the command that executed last.
`1`	Error termination • A non-numeric value was specified.

Notes

- You can specify for *n* a negative value or a value that is greater than `255`, but we recommend that you specify in JP1/Advanced Shell a value in the range of `0` to `255`.
- If the command syntax of this special built-in command is invalid, it exits the shell that is executing the command.
- When executing the `return` command in a separate process, such as the `&` operator and a command substitution, also see the notes provided in *5.1.7 Execution in a separate process (UNIX only)*.

Usage example

- Return with return code `2` from a function or external script and continue processing in the calling function.

```
return 2
```

set command (sets shell options, creates an array, or displays variable values)

Format

```
set  [-a|+a] [-f|+f] [-u|+u] [-v|+v] [-x|+x]
     [{-o|+o} [opt]]...
     [{-A|+A} name] [--] [val ...]
```

Description

The command sets specified shell options, creates an array, or displays variable values.

This command can be used both to set shell options and to create an array. If you do both at the same time, you must specify the shell options first, followed by array creation. If you specify the array creation first, everything after the `-A` option will be interpreted as array name and element value specifications.

Arguments

You can specify more than one option at the same time. If the same option is specified more than once, the last specification takes effect.

If no options are specified, all variables that have been assigned are output to the standard output in the format *variable-name=value*.

-a|+a

- `-a`: Enables the `allexport` option.

- `+a`: Disables the `allexport` option.

-f|+f

- `-f`: Enables the `noglob` option.
- `+f`: Disables the `noglob` option.

-u|+u

- `-u`: Enables the `nounset` option.
- `+u`: Disables the `nounset` option.

-v|+v

- `-v`: Enables the `verbose` option. When the `verbose` option is enabled, all lines that are read as input to the shell are output to the standard output. All input lines are output, regardless of category, such as control statement or command. The items that are output include even the following:
  - Comments
  - Nonexistent commands
  - Commands that are not executed because they do not satisfy the conditions in an `if`, `case`, or similar statement
  - Commands that are not executed because the loop in a `while` statement, `for` statement, or other looping structure is never entered
  - Job steps that are skipped by means of the `run` attribute
- `+v`: Disables the `verbose` option.

-x|+x

- `-x`: Enables the `xtrace` option.
- `+x`: Disables the `xtrace` option.

-o|+o

- `-o`: Enables the shell option specified in *opt*. In addition, displays a list of the shell options that are currently set.
- `+o`: Disables the shell option specified in *opt*. In addition, displays the shell options that are currently set, in a format that can be entered into the command line.

opt

Specifies the name of a shell option that is to be set. For details about the shell option names that can be specified, see *5.6 Shell options*.

-A|+A

Specifies that values are to be assigned to an array.

If you execute the command with the `-A` option specified and the array or variable specified in `name` already exists, the command deletes the contents of `name` and then creates the array or variable by assigning to it the values specified in *val*. The command creates as many array elements as there are arguments specified in `val`.

If you execute the command with the `-A` option specified and the array or variable specified in `name` already exists, the command assigns the values specified in `val` to the array or variable specified in `name` without deleting the contents of `name`. If the number of element values to be assigned is fewer than the number of elements existing in the array, the values of the array elements to which no value is assigned remain unchanged. If the array or variable specified in `name` does not exist, the command processing is the same as when the `-A` option is specified.

The number of array elements must be within the rage of 2 to 65,536. If only one value is specified, the command creates a variable rather than an array. Multiple arrays cannot be created simultaneously.

name

Specify the name to be assigned to the array that is being created. If you specify a read-only variable, the command terminates with an error.

--

Specifies the end of the option specifications. Any options specified after this option are interpreted as part of *val*.

val

Specifies a value to be assigned as an element of the array that is being created. If the command is executed with only one value specified, the specified value is assigned to a positional parameter. If multiple values are specified, they are assigned to the array from left to right in the order `$1`, `$2...`.

Return codes

Return code	Meaning
`0`	Normal termination
`1`	Error termination

Notes

- If the command syntax of this special built-in command is invalid, it exits the shell that is executing the command.
- If the `braceexpand` and `noglob` options are both specified, the `noglob` option takes effect and brace expansion is disabled.

Usage example

- Create the array `test` and assign `a01` to `test[0]`, `a02` to `test[1]`, and `a03` to `test[2]`.

```
set -A test a01 a02 a03
```

shift command (shifts the run-time parameters)

Format

```
shift  [n]
```

Description

This command shifts all the run-time parameters so that a specified subsequent parameter becomes set in the first position.

Argument

n

Specifies the number of places by which the run-time parameters are to be shifted. When this argument is specified, the run-time parameters are shifted by the specified number of places. When this argument is omitted, `1` is assumed (the second parameter becomes the first parameter, and so on). If you specify `0`, the run-time parameters are not shifted. If you specify a negative or non-numeric value, the command terminates with an error. If you specify a value greater than the number of run-time parameters, the command terminates with an error.

Return codes

Return code	Meaning
0	Normal termination
1	Error termination

Notes

- If you specify 0 for the argument, the run-time parameters are not shifted. If you intend to use the shift command to iterate through the run-time parameters in a for statement, while statement, or other looping control structure, be sure not to specify 0 for the argument.
- If the command syntax of this special built-in command is invalid, it exits the shell that is executing the command.

Usage example

- Shift the run-time parameters by 2 places.

```
shift 2
```

test command (determines the value of a conditional expression)

Format 1

```
test conditional-expression
```

Format 2

```
[ conditional-expression ]
```

Format 3

```
[[ conditional-expression ]]
```

Description

This command determines the value of a specified conditional expression. The command evaluates the value of a conditional expression specified using conditional operators. It returns 0 if the result is true or 1 if the result is false. If you execute the test command or [ ] with *conditional-expression* omitted, it returns 1.

For details about conditional expressions, see *5.2 Conditionals*.

Return codes

Return code	Meaning
0	Normal termination • The result of evaluating the conditional expression was true.
1	Normal termination • The result of evaluating the conditional expression was false.

Return code	Meaning
2	Error termination • The command terminated with an error.

Notes

- Operators such as angle brackets (< and >) have special meanings as metacharacters. To use these characters in the test command, you must disable them as metacharacters.
- When the square bracket format ([[ ]]) is used, neither a wildcard nor file name substitution is applied to a character string entered between the square brackets.

  An example is shown below. In this example, it is assumed that files test.ash and hhh are located in the current directory.

  ```
  [[ -f *est.ash ]]     ... (1)
  [ -f *est.ash ]       ... (2)
  test -f *est.ash      ... (3)
  [[ -f ?(hhh) ]]       ... (4)
  [ -f ?(hhh) ]         ... (5)
  test -f ?(hhh)        ... (6)
  ```

  In lines (1) and (4), the wildcard enclosed in the square brackets ([[ ]]) is not applied. Therefore, it is interpreted that no applicable file exists, and the return code will be 1.

  In lines (2), (3), (5), and (6), the wildcard is applied. Therefore, the determined result of the conditional expression becomes true, and the return code will be 0.
- Even if the command syntax of this regular built-in command is invalid, it does not exit the shell that is executing the command.

Usage example

- Determine whether the variables arg1 and arg2 have the same value.

  ```
  test $arg1 -eq $arg2
  ```

times command (displays the amount of CPU time used by the shell)

Format

```
times
```

Description

This command outputs to the standard output the amount of CPU time used by the shell and by processes launched from the shell. The following information is displayed:

- User CPU time (in seconds) used by the shell
- System CPU time (in seconds) used by the shell
- Total user CPU time (in seconds) used by processes launched from the shell
- Total system CPU time (in seconds) used by processes launched from the shell

The output format of the `times` command is shown in the following table:

Output format#	Meaning
`Shell:` *CPU-time* `user` *CPU-time* `system`	CPU time used by the shell is output in the order user CPU time, system CPU time.
`Kids:` *CPU-time* `user` *CPU-time* `system`	CPU time used by processes launched from the shell is output in the order user CPU time, system CPU time.

#

CPU-time is displayed to two decimal places.

Note that if an option is specified, it is ignored and processing of the command continues unaffected.

Return code

Return code	Meaning
`0`	Normal termination

Notes

- In Windows, the CPU time of child processes does not include the CPU time of grandchild processes.
- Even if the command syntax of this regular built-in command is invalid, it does not exit the shell that is executing the command.

Usage example

- Display the CPU time of the shell and the process launched from the shell (`ps` command).

  Contents of the job definition script

```
ps > /dev/null
times
```

  Contents of the `STDOUT` file of the execution job

```
********   JOB SCOPE STDOUT    ********
Shell:     0.00s user     0.01s system
Kids:      0.01s user     0.10s system
```

trap command (specifies the action when signals and forced termination requests are received)

The functionality of the `trap` command for the Windows edition differs from its functionality for the UNIX edition. For details about the functionality for the UNIX edition, see *(1) trap command (UNIX edition)*. For details about the functionality for the Windows edition, see *(2) trap command (Windows edition)*.

(1) trap command (UNIX edition)

Format

```
trap  [action] [signal ...]
```

Description

This command specifies the action to be taken when one or more specified signals are received. When the shell receives a specified signal, it executes the specified action.

When this command is executed with no arguments, it outputs to the standard output a list of the actions associated with all signals in the following format:

Category of signal	Output format
Signal whose name is defined	`trap --` *action signal-name-without-SIG-prefix*
Signal whose name is not defined	`trap --` *action* `UNKNOWN SIGNAL`

Actions can be set for signals for which multiple names are defined under a single signal number, as shown in the tables below.

Linux

Signal name	Alternate name	Whether an action can be set by the trap command	
`SIGSYS`	`SIGUNUSED`	`SIGSYS`	Can be set.
		`SIGUNUSED`	Can be set.

Note:

Because extended functions use `SIGSYS`, not `SIGUNUSED`, as the main signal name, the `trap` command also treats `SIGSYS` as the signal name.

AIX

Signal name	Alternate name 1	Alternate name 2	Whether an action can be set by the trap command	
`SIGABRT`	`SIGLOST`	`SIGIOT`	`SIGABRT`	Can be set.
			`SIGLOST`	Can be set.
			`SIGIOT`	Can be set.
`SIGIO`	`SIGPOLL`	None	`SIGIO`	Can be set.
			`SIGPOLL`	Can be set.

HP-UX or Solaris

Signal name	Alternate name	Whether an action can be set by the trap command	
`SIGABRT`	`SIGIOT`	`SIGABRT`	Can be set.
		`SIGIOT`	Can be set.
`SIGIO`	`SIGPOLL`	`SIGIO`	Can be set.
		`SIGPOLL`	Can be set.

When an action is specified for a signal for which multiple names are defined under a single signal number, one of the specified signal names is output.

The `trap` command does not display an action that is set for a signal when that action is registered in an extended function.

Arguments

action

Specifies the action to be taken when the specified signals are received. If this argument is omitted or a hyphen (-) is specified, the previously-specified traps for the specified signals are reverted to their default settings. If action is omitted and a signal number is specified for signal, the previously specified traps for the specified signals are also reverted to their default settings.

signal

Specifies a signal that is to be trapped. You can specify a signal number or a signal name. When you specify a signal name, you must specify it without the leading SIG (for example, specify INT for SIGINT). For the specifications for each signal, see the documentation for the OS being used.

The operation that is performed when SIGTERM is specified depends on the specification of the TRAP_ACTION_SIGTERM environment setting parameter. For details, see *TRAP_ACTION_SIGTERM parameter (defines the job controller's action when a forced termination request is received)* in *7. Parameters Specified in the Environment Files*.

You can specify multiple signals delimited by the space. You can also specify 0, EXIT, or ERR for this argument.

When you execute the trap command with 0 or EXIT specified for *signal*

Executes the command specified in *action* when the shell terminates.

When you execute the trap command with ERR specified for *signal*

Executes the action specified in *action* when the any of the following commands executed after the trap command terminates with a non-zero return code:

- Regular built-in command
- typeset command
- return command that results in an error in a function or external script because of an invalid format

In AIX, the SIGWAITING signal cannot be specified. If you execute this command in AIX with SIGWAITING specified, the command terminates with an error.

Return codes

Return code	Meaning
0	Normal termination
1	Error termination

Notes

- If you specify a value that is smaller than 0 for the *signal* argument, the trap command treats it as an invalid signal. When you specify a numeric value for the *signal* argument, specify a value that is within the permissible range of 0 to *signal*.
- If the adshread extended shell command is specified in the operation defined by the trap command and a forced termination request is received, the job waits for an entry and will not be terminated. If you specify TERM, or AUTO(and the job starts from JP1/AJS) in the TRAP_ACTION_SIGTERM parameter, do not specify the adshread extended shell command in the operation that is defined by the trap command.
- If the command syntax of this special built-in command is invalid, it exits the shell that is executing the command.
- If the *action* argument is omitted and only a signal number is specified in the *signal* argument, the trap command resets the action specified for *signal* to the default setting. However, if only a signal name is specified without SIG, the trap command terminates itself without resetting the action specified for *signal*.

Example: `trap 15`

➜ The command resets the action for signal number `15` to the default setting.

Example: `trap TERM`

➜ The command does not reset the action.

Usage examples

- Output the message `"trapped."` by the `echo` command when the `INT` signal is received.

```
trap 'echo trapped.' INT
```

- Display the actions that are associated with signals.

  Contents of the job definition script

```
trap -- 'echo Hangup.' HUP
trap -- 'echo trapped.' INT
trap
```

Contents of the `STDOUT` file of the execution job

```
********   JOB SCOPE STDOUT    ********
trap -- 'echo Hangup.' HUP
trap -- 'echo trapped.' INT
```

(2) trap command (Windows edition)

Format

```
trap [action][method]
```

Description

This command specifies the action to be taken when a forced termination request is received.

When TERM is specified in the TRAP_ACTION_SIGTERM parameter

The action to be taken upon receiving a forced termination request can be specified. When the forced termination request specified in the method is received, the job controller performs the operation specified for `action`.

If the forced termination request specified in `method` is neither `TERM` nor `15`, the command issues the `KNAX6718-I` message and terminates with return code `0` without performing the specified `action`.

If the command is executed with no argument specified, it outputs the action set for forced termination requests to the standard output in the following format:

Output format
`trap -- action` "*character-string-indicating-the-forced-termination-method*"

When DISABLE is specified in the TRAP_ACTION_SIGTERM parameter

The command issues the `KNAX6710-I` message and always terminates normally with return code `0`. The command does not perform processing for forced termination requests.

Arguments

action

Specifies the action to be taken when a forced termination request is received.

If a hyphen (`-`) is specified for *action*,the command resets the previously specified `action` setting for the specified method so that the method is not associated with any action setting. If `action` is omitted and 15 is specified in the `method` argument, the command also resets the previously specified `action` setting for the specified method so that the method is not associated with any `action` setting.

If you specify `""` for the action, the `trap` command terminates normally without changing the current setting for action.

method

Specifies the forced termination method that is to be subject to the trap.

You must specify either `TERM` or `15` for *method*.

`TERM` or `15`

Specifies that immediate process termination is to be performed by a function such as `TerminateProcess` (such as forced termination from JP1/AJS or the `taskkill` command).

Return codes

Return code	Meaning
`0`	Normal termination
`1`	Error termination

Notes

- When `kill -KILL` *process-ID* is executed to terminate `adshexec.exe` for a job, the *action* defined in the terminated job's `trap action TERM` is executed.
- If `TERM` is specified in the `TRAP_ACTION_SIGTERM` parameter, the `trap` command with an option specified results in an error. If a value other than `TERM` is specified in the `TRAP_ACTION_SIGTERM` parameter, the `trap` command with an option specified will not result in an error.
- If the adshread extended shell command is specified in the operation defined by the `trap` command and a forced termination request is received, the job waits for an entry and will not be terminated. If you specify TERM in the TRAP_ACTION_SIGTERM parameter, do not specify the adshread extended shell command in the operation that is defined by the `trap` command.
- If this special built-in command terminates with an error due to a syntax error, the shell running the command will also terminate.
- If the `action` argument is omitted and `15` is specified in the `method` argument, the `trap` command resets the action specified for immediate process termination by a function such as `TerminateProcess` so that the method is not associated with any `action` setting. However, if only `TERM` is specified, the `trap` command terminates itself without resetting the action specified for immediate process termination.

  Example: `trap 15`

  ➜ The command resets the action specified for immediate process termination by a function such as `TerminateProcess` so that the method is not associated with any `action` setting.

  Example: `trap TERM`

  ➜ The command does not reset the action.
- If multiple values are specified for *method* and at least one of those values is `TERM` or `15`, the command sets *action* for `TERM` or `15` without issuing the `KNAX6718-I` message. For any other *method*, the command does not set *action*.

  Example: If `trap date 28 15` is executed, the `trap` command sets the `date` command as the *action* for `15` without issuing the `KNAX6718-I` message.

Usage Examples

- "`trapped.`" is output by the `echo` command when a forced termination request performed by a function such as TerminateProcess is received.

  ```
  trap 'echo trapped.' TERM
  ```

- Display the action that is set for forced termination requests. `TERM` is displayed as the forced termination method even when `15` is set for *method* in the `trap` command that sets *action*.

  Contents of the job definition script:

  ```
  trap 'echo trapped.' 15
  trap
  ```

  Contents of the `STDOUT` file of the execution job:

  ```
  ********   JOB SCOPE STDOUT   ********
  trap -- 'echo trapped.' TERM
  ```

true command (returns 0 as the return code)

Format

```
true
```

Description

This command terminates successfully, returning `0` as the return code.

This command accepts no options. If an option is specified, it is ignored and processing continues.

Return code

Return code	Meaning
`0`	Normal termination • Always returns `0`.

Notes

- Even if the command syntax of this regular built-in command is invalid, it does not exit the shell that is executing the command.
- The execution result of this command is output in the `KNAX6113-I` message.

Usage example

- Set the return code to `0`.

  Contents of the job definition script

  ```
  true
  echo $?
  ```

  Contents of the `STDOUT` file of the execution job

```
********   JOB SCOPE STDOUT    ********
0
```

typeset command (declares explicitly the attributes and values of variables and functions)

Format

```
typeset  [{-L|+L}  [n]]  [{-R|+R}  [n]]  [{-Z|+Z}  [n]]
         [-l|+l|-u|+u] [{-i|+i} [n]] [-r|+r|-x|+x]
         [{-f|+f} [-t|+t] [-u]]
         [-p|+p]
         [--] [name  [=value]...]
```

Description

This command declares explicitly the attributes and values of variables and functions. You can use this command to define specified variables and functions (specified in *name*), as well as to declare explicitly attributes and values for them.

When this command is executed within a function, it defines local variables that are valid within that function only. Those values and attributes revert when the function terminates.

The options for this command are organized into the four categories of character string formatting options, attribute and type options, function options, and display options.

Arguments

You prefix each option specification with a minus sign (-) to enable the option (attribute) or a plus sign (+) to disable it.

[{-L|+L} [n]] [{-R|+R} [n]] [{-Z|+Z} [n]]

- `-L|+L`

  These are character string formatting options. The `-L` option left-justifies the contents of the variables. The `+L` option removes any left-justification that has been set.

  When a value is set in a variable and the length of the area is greater that the length of the specified value, this option pads the trailing remainder of the area with spaces. If the length of the specified value is greater than the length of the area, the trailing portion of the value is truncated to fit in the region

  If the `-Z` option is also specified, leading zeros are stripped. If the `-R` option is also specified, the one specified last takes effect.

  If the item whose name is specified in *name* is already defined with the `-R` option, its right-justification setting is disabled.

- `-R|+R`

  This is a character string formatting option. The `-R` option right-justifies the contents of the variables. The `+R` option removes any right-justification that has been set.

  When a value is set in a variable and the length of the area is greater than the length of the specified value, this option pads the beginning of the area with spaces. If the length of the specified value is greater than the length of the area, the beginning portion of the value is truncated to fit in the region. If `-L` and `-R` are both specified, the one specified last takes effect.

If the item whose name is specified in *name* is already defined with the `-L` option, its left-justification setting is disabled.

- `-Z|+Z`

  This is a character string formatting option. The `-Z` option pads the contents of the variables with zeros. The `+Z` option removes any zeros-padding that has been set.

  Unless the `-L` option is set, the contents will be right-justified. If the first character of the value specified in *value* is a numeric, the beginning of *value* is padded with zeros. If the first character of the value specified in *value* is a non-numeric, the beginning of *value* is padded with spaces.

- `n`

  Specifies the length of the area for *value*. If `0` is specified or *n* is omitted, the length of the area will be the length of the value specified in *value*. An error results if you specify a value for *n* that exceeds 16,385.

[-l|+l|-u|+u]

- `-l|+l`

  These are character string formatting options. The `-l` option converts the letters assigned to a variable specified in *name* from uppercase to lowercase. If the character string assigned to the variable contains a mixture of uppercase and lowercase, this option converts only the uppercase letters to lowercase letters. If the `-u` option is also specified, the option specified last takes effect.

  The `+l` option removes any uppercase-to-lowercase conversion that has been set.

- `-u|+u`

  This is a character string formatting option. The `-u` option converts the letters assigned to a variable specified in *name* from lowercase to uppercase. If the character string assigned to the variable contains a mixture of uppercase and lowercase, it converts only the lowercase letters to uppercase letters. If the `-l` option is also specified, the option specified last takes effect.

  The `+u` option removes any lowercase-to-uppercase-conversion attribute that has been set.

[{-i|+i} [n]] [-r|+r|-x|+x]

- `-i|+i`

  These are attribute and type options. The `-i` option declares that the type of a variable specified in *name* is integer. When this option is specified, the value assigned to *value* must be in decimal. If you use `-i` to specify a base other than 10, `'base#'` will be prefixed at the beginning of the contents of the variable specified in *name*. If you also specify the `-Z` option to set zero-padding, the area up to the beginning of `'base#'` will be padded with leading zeros.

  The `+i` option removes any integer type attribute for a variable specified in *name*.

- `n`

  Specifies the base in which *name* is to be displayed when it is output. If `0` is specified or *n* is omitted, and *name* is an undefined variable, the *name* item will be treated as a decimal number. If `0` is specified or *n* is omitted, and *name* is a defined variable, the *name* item will use the base that has already been defined for it. An error results if you specify a base of 1 or a base greater than 16.

- `-r|+r`

  These are attribute and type options. The `-r` option sets the read-only attribute for a variable specified in *name*. Once the read-only attribute is set for a variable, neither the variable's value nor any of its attributes can be changed.

  The `+r` option removes any read-only attribute that has been set.

- `-x|+x`

These are attribute and type options. The `-x` option exports a variable specified in *name*. The `+x` option cancels exporting of a variable specified in *name*.

When you export variables in Windows, the supported variable names are handled as follows:

When DISABLE is specified in the VAR_ENV_NAME_LOWERCASE parameter

All letters contained in variable names must be in uppercase because shell variables whose names contain lowercase letters cannot be exported.

If a variable containing a lowercase letter is specified for *name*, an error message is output and the batch job terminates.

When ENABLE is specified in the VAR_ENV_NAME_LOWERCASE parameter

Shell variables whose names contain lowercase letters can be exported.

Note that environment variables are not case sensitive. The last shell variable with the same spelling that was exported becomes the final environment variable value.

{-f|+f} [-t|+t] [-u]

- `-f|+f`

  These are function options. The `-f` option declares that the processing targets specified in *name* are to be treated as functions, not variables. When the command is executed with the `-f` option specified, functions specified in *name* are also output to the standard output. Functions are not displayed when the command is executed with the `+f` option.

  When the command is executed with only `-f` specified, and with no *name* specification, all currently defined functions are output to the standard output.

- `-t|+t`

  These are function options. When specified together with the `-f` option, the `-t` option enables the trace mode for a function specified in *name*.

  The `+t` option disables the trace mode for a function specified in *name*.

- `-u`

  This is a function option. When specified together with the `-f` option, the `-u` option enables the auto-load functionality for a function specified in *name*.

`-p|+p`

These are display options. The `-p` option outputs all defined variables to the standard output in the format `typeset` *variable-name=value*. If you specify the `-p` option and *name* at the same time, the declaration of attributes for the variable specified in *name* takes precedence.

The `+p` option outputs all defined variables to the standard output in the format `typeset` *variable-name*.

No options specified

This is a display option. When this command is executed with no options specified, all the defined variables are output to the standard output. Each variable name is preceded by the values of the attribute and type options declared in `typeset`. When no attribute and type options have been declared, the variable name is displayed left-justified.

Option only specified without a name specification

This is a display option. All the variables and functions with the specified property are output. If the option is prefixed with a hyphen (`-`), what is output to the standard output is *variable-name=value* or the contents of each function. If the option is prefixed with a plus sign (`+`), only the names of the variables or functions are output to the standard output.

--

Specifies the end of the option specifications. Any options specified after this option are interpreted as part of a variable name.

name

Specifies a variable name, array name, or function name whose attributes and values are to be declared. You can specify multiple variable names, array names, and function names.

When you specify an array name, the declaration applies to all the elements that constitute the array, even if you specify only one element of the array.

You can specify = after *name* to assign a value to *name* and declare its attributes at the same time.

If the read-only attribute is set for the variable specified in *name* and you attempt to set a value for it, the command will terminate with an error.

value

Specifies a value to be assign to the paired *name*. If *value* is omitted, the linefeed character is assigned, and then the attribute changes are made.

Return codes

Return code	Meaning
0	Normal termination
1	Error termination

Notes

- For the *n* argument, specify a value that is within the permissible range set for each of the arguments specified concurrently.
- If a variable that contains multibyte characters is truncated during left- or right-justification, the multibyte character data might be lost, leaving behind a character that is assigned to an incomplete byte sequence.
- You cannot use a single hyphen to specify multiple options and arguments at the same time. For example, to declare an octal integer that is right-justified within 16 digits, specify `-i8 -R16`. If you specify `-i8R16`, the command will terminate with an error.
- If the `-f` and `-x` options are both specified, the `-x` option is ignored and the processing is the same as when only the `-f` option is specified.

Usage examples

- Set the attributes for the variable `num` so that it is declared as an integer and displayed left-justified with a length of 10 digits.

```
typeset -L10 -i num
```

- Enable the trace mode for the function `func`.

```
typeset -ft func
```

ulimit command (sets limits on system resources) (UNIX only)

Format

```
ulimit  [-H] [-S] [-a] [-c] [-d] [-f] [-l] [-m]
        [-n] [-p] [-s] [-t] [limit]
```

Description

This command sets limits on system resources or displays information about limits on system resources that have been set. This command is used to set upper limits on system resources that are specified by option specifications, as well as to output to the standard output limits that have been set.

The following table shows the output formats for displaying resource limits.

Output format	Contents
`time(cpu-seconds)   upper-limit`	Maximum CPU time
`file(blocks)        upper-limit`	Maximum file size
`coredump(blocks)    upper-limit`	Maximum file size of core dump
`data(kbytes)        upper-limit`	Maximum data area size
`stack(kbytes)       upper-limit`	Maximum stack area size
`lockedmem(kbytes)   upper-limit`	Maximum size of physical memory that can be locked
`memory(kbytes)      upper-limit`	Maximum size of physical memory that can be used
`nofiles(descriptors) upper-limit`	Maximum number of file descriptors
`processes           upper-limit`	Maximum number of processes

Arguments

If you indicate multiple resources by specifying more than one resource option, the option specified last takes effect.

-H

Specifies that the limit being set (or displayed) is a hard limit. If the `-H` and `-S` options are both specified, the one specified last takes effect.

-S

Specifies that the limit being set (or displayed) is a soft limit. If the `-H` and `-S` options are both specified, the one specified last takes effect.

-a

Specifies that the upper limits for all resources are to be output.

-c

Specifies that a maximum size for a core dump file (in blocks) is to be set or displayed.

-d

Specifies that a maximum data area size (in kilobytes) is to be set or displayed.

-f

Specifies that a maximum file size (in blocks) for files written by a shell or by processes launched from a shell is to be set or displayed.

-l (Linux only)

Specifies that a maximum size (in kilobytes) for the physical memory that can be locked is to be set or displayed.

-m (AIX, HP-UX, and Linux only)

Specifies that a maximum size (in kilobytes) for the physical memory that can be used is to be set or displayed.

-n

Specifies that a maximum number of file descriptors that can be open is to be set or displayed.

-p (Linux only)

Specifies that a maximum number of processes that one user can start is to be set or displayed.

-s

Specifies that a maximum size (in kilobytes) of the stack area is to be set or displayed.

-t

Specifies that a maximum CPU time (in seconds) is to be set or displayed.

limit

Specifies the resource limit value that is to be set. If you specify `unlimited`, no upper limit is set for the resource. You can specify any numeric value as an upper limit, but for details about the upper limits that are valid in practice, see the documentation for the OS being used.

Return codes

Return code	Meaning
0	Normal termination
1	Error termination

Notes

- When either of the following conditions apply, the `ulimit` command outputs the `KNAX6710-I` message indicating that the command is not supported, and then terminates normally with a return code of `0`:
  - A specified option is not supported by the OS.
  - You are running in a Windows environment.
- Even if the command syntax of this regular built-in command is invalid, it does not exit the shell that is executing the command.
- If the maximum file size set with the `ulimit` command is too small to be able to generate the files required when you run a job definition script with the `adshexec` command, you might receive the `SIGXFSZ` signal.
- To increase the hard limit, you need the administrator's permission.
- The upper limits on resources that can be specified depend on the execution environment and the OS.
- For some resources whose limits are to be changed, if you specify a value that is not permitted, the execution environment or the OS might set a different value.

Usage example

- Output the upper limits for all resources.

  Contents of the job definition script

  ```
  ulimit -a
  ```

  Contents of the `STDOUT` file of the execution job

```
********   JOB SCOPE STDOUT    ********
time(cpu-seconds)    unlimited
file(blocks)         unlimited
coredump(blocks)     0
data(kbytes)         unlimited
stack(kbytes)        10240
lockedmem(kbytes)    32
memory(kbytes)       unlimited
nofiles(descriptors) 1024
processes            4096
```

umask command (sets the access permissions for creating a new file) (UNIX only)

Format

```
umask  [-S] [mask]
```

Description

This command sets the access permissions used when a new file is created. You specify in *mask* the file mode creation mask that is to be set. If the command is executed with *mask* omitted, it outputs the current `umask` value to the standard output.

Arguments

-S

Specifies that the value is to be set or output in symbolic format.

When the `-S` option is specified, the command sets or outputs the file mode in symbolic format. When the `-S` option is not specified, the access permissions that are set or output are expressed in octal. This specification also indicates when the specified access permissions are not granted at file creation time.

mask

Specifies a umask value to serve as the default file mode when a file is created. You can specify *mask* in numeric or symbolic format. If you use symbolic format, the specification follows the format `[`*who*`][`*op*`][`*perm*`]` `[,...]`. You can specify multiple sets of values delimited by the comma (you cannot use the space).

- `who`

  Specifies the targets for which the mask is set. Specify none, one, or multiple of the following letters:

  `u`: Permissions for the user (owner)

  `g`: Permissions for the group

  `o`: Permissions for others

  `a`: Permissions for all (`a=ugo`)

  (none): Permissions for all (`a=ugo`)

- `op`

  Specifies the mask setting operation. Specify one of the following symbols:

  `+`: Add *perm* to the current mask for each *who* entry

  `-`: Remove *perm* from the current mask for each *who* entry

=: Change the current mask for each *who* entry to *perm*

- perm

  Specifies the permissions to be granted at the time of file creation. Specify none, one, or multiple of the following letters:

  r: Read permission

  w: Write permission

  x: Execute permission

  u: Same permissions as for the user

  g: Same permissions as for the group

  o: Same permissions as for others

  X: If any of ugo has the execute permission, grant those execute permissions. If none of ugo have execute permissions, no permissions are set. If no permissions are set, nothing changes when *op* is + or -, but the mask for *who* is removed when *op* is =.

  s: No permissions. Nothing changes when *op* is + or -, but the mask for *who* is removed when *op* is =.

  (none): No permissions. Nothing changes when *op* is + or -, but the mask for *who* is removed when *op* is =.

Return codes

Return code	Meaning
0	Normal termination
1	Error termination

Notes

- When you use this command in a Windows environment, the KNAX6710-I message is output indicating that the command is not supported. The command then terminates normally with a return code of 0 (no file creation mask is set).
- Even if the command syntax of this regular built-in command is invalid, it does not exit the shell that is executing the command.
- For the permissions for temporary files created by the #-adsh_file_temp command, the specification of the umask value applies only to the file owner (creator) part and the group and other users' access permissions are always 0.

Usage example

- Set the file creation mask so that only the user has all access permissions:

```
umask 077
```

- Set the file creation mask so that only the user has write permission:

```
umask u=rwx,go=rx
```

unalias command (removes alias definitions)

Format

```
unalias  [-a]name  [name...]
```

Description

This command removes specified alias definitions. To do so, specify in *name* the name of the alias you want to remove. You can specify multiple names whose aliases are to be removed, using the space as the delimiter. If you specify in *name* an alias name that is not defined, or if you execute the command without any options or arguments, the command terminates with an error and returns `1` as the return code.

Arguments

-a

Specifies that all alias definitions are to be removed.

name

Specifies the name for an alias definition to be removed.

Return codes

Return code	Meaning
0	Normal termination
1	Error termination (such as that a name specified in *name* does not have a defined alias)

Notes

- Even if the command syntax of this regular built-in command is invalid, it does not exit the shell that is executing the command.

Usage example

- Remove the alias defined for `functions`.

  ```
  unalias functions
  ```

unset command (unsets variable values and attributes)

Format

```
unset  [-f] name [ name ...]
```

Description

This command unsets specified variables or functions. You can specify more than one variable or function name. When you execute the command with the `-f` option specified, each *name* is assumed to be a function name, and the specified function definitions are removed.

Arguments

-f

Specifies that this command applies to removal of function definitions.

name

Specifies the name of a variable or function that is to be removed. You can also specify the name of an array.

If an array name is specified, all the elements that constitute the array are unset. To unset only one element, specify *array-name*`[`*element-number*`]` in *name*.

If the read-only attribute is set for a variable whose name is specified, the command terminates with an error. When you execute this command with an undefined variable name or function name specified in *name*, it terminates with an error.

Return codes

Return code	Meaning
`0`	Normal termination
`1`	Error termination (such as that a name specified in *name* is not defined as a variable or function)

Notes

- If you use this command to unset shell variables such as `LINENO`, `OPTARG`, `OPTIND`, `RANDOM`, and `SECONDS`, the special meanings of the shell variables will be lost, even if you define them again later.
- If you use this command to unset array elements individually, they will still count as defined variables in the debugger, and you will still be able to use the `debug` command to display or set them.

  However, if you unset all the elements that constitute an array, they will be treated as undefined variables in the debugger, and you will no longer be able to use the `debug` command to display or set them. For details about using the `debug` command to display and set variable values, see *6. Debugging Job Definition Scripts*.
- If the command syntax of this special built-in command is invalid, it exits the shell that is executing the command.

Usage examples

- Remove a variable.

```
unset val
```

- Remove a function.

```
unset -f func
```

wait command (waits for child processes to complete)

Format

```
wait  [pid ...]
```

Description

This command waits for child processes to complete. Specify for *pid* one or more process IDs of child processes waiting for completion. If *pid* is not specified, the command waits for all child processes being executed to complete. If an invalid process ID that begins with a non-numeric value is specified for *pid*, processing terminates normally with return code `127`.

Note the following if a mixture of numeric values and non-numeric values is specified beginning with a numeric value, as shown in the example below: The character string up to the last position preceding the non-numeric value is interpreted as a process ID and the command waits for the child process to complete.

```
UAP &         # Shell variable ! stores the process that started the UAP.
wait $!ABC    # The wait command interprets the character string preceding
ABC and waits for the child process to complete.
```

When the *pid* argument is specified, the `wait` command terminates with the return code of the last process that waited for completion. An example follows:

```
UAP1 &                    # UAP1 terminates with a return code of 2.
PID1=$!                   #
UAP2 &                    # UAP2 terminates with a return code of 16.
PID2=$!                   #
UAP3 &                    # UAP3 terminates with a return code of 0.
PID3=$!                   #
wait $PID1 $PID2 $PID3    # The wait command terminates with a return code
of 0.
```

Argument

pid

Specifies the process ID of a child process that is waiting to complete.

Return codes

Return code	Meaning
`0`	Normal termination
`127`	Normal termination • Could not identify a specified child process. • A specified process ID was not for a currently executing child process.
Other than the above	Error termination

Notes

- Even if the command syntax of this regular built-in command is invalid, it does not exit the shell that is executing the command.

Usage example

- Wait for the child process with process ID 4848 to complete.

```
wait 4848
```

whence command (displays how character strings would be interpreted if used as commands)

Format

```
whence  [-p] [-v] name  [name...]
```

Description

This command displays how specified character strings would be interpreted if used as commands. When neither option is specified, the output is as follows.

- If a specified character string is a command, the path name of the command is output.
- If a specified character string is an alias, the value of the alias is output.
- If a specified character string is a reserved word, standard shell command, extended shell command, or function, the specified character string is output.
- If none of the above applies, nothing is output and the command terminates with a return code of `1`.

If the `-p` and `-v` options are both specified, the `whence` command assumes that a specified character string is a command and produces output accordingly. If the *name* argument is omitted, the command terminates with an error and returns `1` as the return code.

Arguments

-p

Specifies that the specified character strings are to be handled as commands whose paths are to be output.

-v

Specifies that whether each specified character string is a command, reserved-word, alias, standard shell command, extended shell command, or function is to be output.

The following table shows the contents of the output:

No.	Output contents	Meaning
1	*name* `is a reserved word`	*name* is a reserved word.
2	*name* `is a function`	*name* is a function.
3	*name* `is a traced function`	*name* is a function for which the trace mode is enabled. If the function is undefined and the trace mode is enabled for it, the contents shown in No. 4 are output.
4	*name* `is an undefined function`	*name* is an undefined function.
5	*name* `is an extended shell command`	*name* is an extended shell command.
6	*name* `is a shell builtin`	*name* is a built-in command.
7	*name* `is a special shell builtin`	*name* is a special built-in command.
8	*name* `is a shell builtin not supported`	*name* is a command that is not provided in JP1/Advanced Shell.
9	*name* `is` *path-name*	*name* is a command or executable file.

No.	Output contents	Meaning
10	*name* is an alias for '*alias-value*'	*name* is an alias.
11	*name* is an exported alias for '*alias-value*'	*name* is an exported alias.
12	*name* not found	*name* is not a command, reserved-word, alias, standard shell command, extended shell command, or function.

name

Specifies a character string to be interpreted as a command. If this argument is omitted, the command terminates with an error and returns 1 as the return code.

Return codes

Return code	Meaning
0	Normal termination
1	Error termination. Or, one of specified character strings could not be found as a command.

Notes

- Even if the command syntax of this regular built-in command is incorrect, it does not exit the shell that is executing the command.

Usage example

- Assume that pwd is a command and output its path.

  Contents of the job definition script

```
whence -p pwd
```

  Contents of the STDOUT file of the execution job

```
********   JOB SCOPE STDOUT    ********
/bin/pwd
```

9.4 Extended shell commands

The extended shell commands are built into JP1/Advanced Shell itself. A built-in command is one that is included as part of the shell, and is executed by the shell itself.

Even if the command syntax of an extended command is invalid, it does not exit the shell that is executing the command.

adshecho command (issues a specified event notification message as a JP1 event)

Format

```
adshecho [-d] event-notification-message
```

Description

This command issues a specified event notification message as a JP1 event. The issued JP1 event is displayed in JP1/IM - View. If you are running the debugger with the standard input and output redirected to the user-reply functionality, the specified event notification message is output to the standard output.

The JP1 event is issued after waiting for a fixed amount of time (specified in the `USERREPLY_JP1EVENT_INTERVAL` parameter) to elapse since the last JP1 event was issued. For details about the `USERREPLY_JP1EVENT_INTERVAL` parameter, see *USERREPLY_JP1EVENT_INTERVAL parameter (specifies the minimum interval at which JP1 events are to be issued)* in *7. Parameters Specified in the Environment Files*.

Arguments

-d

Specifies that during debugging, the output destination for the specified event notification message is to be set to the standard output. This option is ignored except during debugging.

When the specification is a character string that begins with a hyphen (`-`), it is treated as an option specification if it consists entirely of valid options. If such a character string does not consist entirely of valid hyphens, or if the character string does not begin with a hyphen (`-`), it is treated as an event notification message from that position.

event-notification-message ~<any character-string>((0 to 1,023 bytes))

Specifies the event notification message that is to be issued as a JP1 event.

The character encoding of the specified event notification message must be consistent with the character encoding of the JP1/Base running on the same host. If the character encodings are different, characters might become garbled.

The specified event notification message is issued as a JP1 event after its contents are converted by the `echo -E` *event-notification-message* command. If more than one event notification message is specified, an error results (`KNAX7403-E`).

Return codes

Return code	Meaning	Response	Whether to retry
0	Normal termination	None	--
1	A non-recoverable system error occurred: • Out of memory	Contact a system administrator.	N

Return code	Meaning	Response	Whether to retry
1	- Internal inconsistency detected	Contact a system administrator.	N
4	An error occurred in JP1 event processing.	Take action based on the information in the error message. See *11.4.4 Handling Error Information Displayed in the User-Reply Functionality*.	Y
5	An error occurred in JP1 event processing.	Take action based on the information in the error message. See *11.4.4 Handling Error Information Displayed in the User-Reply Functionality*.	N
6	The JP1 event failed to be transmitted to the specified host.	Check the following: • JP1/Base is installed on the host where JP1/IM - Manager is installed. • The JP1/Base event service is running on the host where JP1/IM - Manager is installed. • A JP1/Base connection has been established between the host where JP1/Advanced Shell is installed and the host where JP1/IM - Manager is installed.	Y
7	The JP1/Base library cannot be found.	Check if JP1/Base is installed on the host where JP1/Advanced Shell is installed. If JP1/Base is installed and this occurs anyway, re-install JP1/Base.	N
8	The connection to the JP1/Base event service on the local host failed.	Check that the JP1/Base event service is running on the host where JP1/Advanced Shell is installed.	Y
10	The specified format is invalid.	Check the format of the command.	N
128+ signal number (UNIX only)	The adshecho command received a signal and terminated.	Confirm that the job received a signal and terminated.	N
200 (Windows only)	The adshecho command was forcibly terminated.	Confirm that the job was forcibly terminated.	N

Legend:

Y: Retry

N: Do not retry

--: Not applicable

Notes

- Do not run this command in the background. If you run it in the background, flow control (specified by the USERREPLY_JP1EVENT_INTERVAL parameter) will not work.
- Do not execute this command in an environment without JP1/Base and JP1/IM, unless you are running the debugger with standard input and output redirected to the user-reply functionality.

  The following problems might occur during execution:

  - If JP1/Base is not installed on the host running JP1/Advanced Shell, the command will terminate with an error.
  - If the JP1/Base event service is not running on the host running JP1/Advanced Shell, the command will terminate with an error.
  - If JP1/Base or the JP1/Base event service is not running on the host specified in HOSTNAME_JP1IM_MANAGER, the command will terminate with an error.
- Even if JP1/IM - Manager is not running on the host specified in HOSTNAME_JP1IM_MANAGER, the command works and will successfully transmit a JP1 event once the event reaches the JP1/Base event service on the host specified in HOSTNAME_JP1IM_MANAGER.

- If the command terminates with an error from which it is possible to retry the command, the command might succeed when it is re-executed. Before re-executing the command, see the example job definition scripts in *3.7.5 How to handle adshecho and adshread commands that terminate with an error* for help in preparing the job definition script.
- Do not execute this command in conjunction with the pipe symbol.
- Do not execute this command in conjunction with the redirection symbols.

adshread command (issues a specified reply-request message as a reply-waiting event)

Format

```
adshread [-d] variable-name reply-request-message
```

Description

This command issues a specified reply-request message as a reply-waiting event. The reply-waiting event that is issued is displayed in JP1/IM - View, from where the operator can enter a reply. The reply that is entered is stored in the specified variable.

If you are running the debugger with the standard input and output redirected to the user-reply functionality, the specified reply-request message is output to the standard output and the reply is received from the standard input.

The JP1 event is issued after waiting for a fixed amount of time (specified in the `USERREPLY_JP1EVENT_INTERVAL` parameter) to elapse since the last JP1 event was issued. For details about the `USERREPLY_JP1EVENT_INTERVAL` parameter, see *USERREPLY_JP1EVENT_INTERVAL parameter (specifies the minimum interval at which JP1 events are to be issued)* in *7. Parameters Specified in the Environment Files*.

Arguments

-d

Specifies that during debugging, the I/O is to be redirected so that the output destination for the specified reply-request message is the standard output and the reply is received from the standard input. This option is ignored except during debugging.

A character string that begins with a hyphen (-) is treated as an option specification until a character string that begins with a character other than a hyphen (-) is encountered. If an invalid option is specified, an option error results.

variable-name ~<environment variable name>

Specifies a shell variable for storing the reply from the operator. Only one shell variable can be specified. If you specify more than one shell variable, everything that is specified following the first shell variable will be interpreted as part of the reply-request message.

The shell variable can accept an ASCII character string of 0-512 bytes.

When you run the debugger with the standard input and output redirected to the user-reply functionality, the portion in excess of 512 bytes in a character string consisting of more than 512 bytes that is specified in the reply from the standard input will be ignored. In addition, if a character string that contains a linefeed is specified, everything following the linefeed will be ignored.

To specify an array in this argument, you must specify each element. You can specify an array with 0 to 65,535 elements.

Examples:

Regular variable specification: `adshread ans "Continue (Y/N)?"`

Array specification: `adshread ans[1] "Continue (Y/N)?"`

The following table shows the execution result for this command, depending on the attributes of the specified shell variable:

Attribute of the specified shell variable	Execution result of the command
Read-only	Outputs `KNAX6008-E` and terminates.
Type is numeric.	Outputs `KNAX7404-E` and terminates.
Type is character string or an array of character strings.	Updates the value and terminates.
Non-existent variable	Creates a new variable of the character type, sets its value, and terminates.
Variable name is invalid.	Outputs `KNAX6003-E` and terminates.
Variable is outside the permitted range of array elements	Outputs `KNAX6007-E` and terminates.
Shell variable that cannot be used in JP1/Advanced Shell	Outputs `KNAX6002-E` and terminates the job forcibly.

reply-request-message ~<any character string>((0 to 1,023 bytes))

Specifies the reply-request message that is to be issued as a reply-waiting event.

The character encoding of the specified reply-request message must be consistent with the character encoding of the JP1/Base running on the same host. If the character encodings are different, characters might become garbled.

The specified reply-request message is issued as a JP1 event after its contents are converted by the `echo -E` *reply-request-message* command. If more than one reply request message is specified, an error results (`KNAX7403-E`).

Return codes

Return code	Meaning	Response	Whether to retry
`0`	Normal termination	None.	--
`1`	One of the following non-resumable errors occurred: ● Out of memory ● A variable that cannot be used in *variable-name* was detected ● Internal inconsistency detected	If you have specified a variable that cannot be used as *variable-name*, change the specification of the variable. If this does not resolve the problem, contact a system administrator.	N
`2`	Semaphore (Mutex): ● An error occurred in shared memory operations.	Take action based on the information in the error message. See *11.4.4 Handling Error Information Displayed in the User-Reply Functionality*.	N
`3`	There is no free space in shared memory.	Check and revise if necessary the setting for the `USERREPLY_WAIT_MAXCOUNT` parameter.	Y
`4`	An error occurred in JP1 event processing.	Take action based on the information in the error message. See *11.4.4 Handling Error Information Displayed in the User-Reply Functionality*.	Y
`5`	An error occurred in JP1 event processing.	Take action based on the information in the error message. See *11.4.4 Handling Error Information Displayed in the User-Reply Functionality*.	N

Return code	Meaning	Response	Whether to retry
6	The JP1 event failed to be transmitted to the specified host.	Check the following: • JP1/Base is installed on the host where JP1/IM - Manager is installed. • The JP1/Base event service is running on the host where JP1/IM - Manager is installed. • A JP1/Base connection has been established between the host where JP1/Advanced Shell is installed and the host where JP1/IM - Manager is installed.	Y
7	The JP1/Base library cannot be found.	Check if JP1/Base is installed on the host where JP1/Advanced Shell is installed. If JP1/Base is installed and this occurs anyway, re-install JP1/Base.	N
8	The connection to the JP1/Base event service on the local host failed.	Check that the JP1/Base event service is running on the host where JP1/Advanced Shell is installed.	Y
10	The specified format is invalid.	Check the format of the command.	N
128+ signal number (UNIX only)	The adshread command received a signal and terminated.	Confirm that the job received a signal and terminated.	N
200 (Windows only)	The adshread command was forcibly terminated.	Confirm that the job was forcibly terminated.	N

Legend:

Y: Retry

N: Do not retry

--: Not applicable

Notes

- Do not run this command in the background. If you run it in the background, flow control (specified by the USERREPLY_JP1EVENT_INTERVAL parameter) will not work. In addition, the reply from the operator will not be stored in the variable specified for the job that is executing the adshread command.
- Do not execute this command with a pipe specified.
- Do not specify processing that accepts a value and redirects it to this command.
- Do not execute this command in an environment without JP1/Base and JP1/IM, unless you are running the debugger with standard input and output redirected to the user-reply functionality.

  The following problems might occur during execution:

  - If JP1/Base is not installed on the host running JP1/Advanced Shell, the command will terminate with an error.
  - If the JP1/Base event service is not running on the host running JP1/Advanced Shell, the command will terminate with an error.
  - If JP1/Base or the JP1/Base event service is not running on the host specified in HOSTNAME_JP1IM_MANAGER, the command will terminate with an error.
  - Even if JP1/IM - Manager is not running on the host specified in HOSTNAME_JP1IM_MANAGER, the command works and will successfully transmit a JP1 event once the event reaches the JP1/Base event service on the host specified in HOSTNAME_JP1IM_MANAGER.
  - If the user-reply functionality's management daemon service is not running, the command will terminate with an error.

- If the command terminates with an error from which it is possible to retry the command, the command might succeed when it is re-executed. Before re-executing the command, see the example job definition scripts in *3.7.5 How to handle adshecho and adshread commands that terminate with an error* for help in preparing the job definition script.
- If the `adshread` command is waiting for a reply-request message and you terminate the job suddenly in a manner other than as described in *3.10.1 How to forcibly terminate jobs*, the reply-request message will be left in shared memory and the reply-waiting event might be retained in JP1/IM - View. In such a case, either use the `adshchmsg` command's `-d` option to cancel the reply-waiting status for the reply-request message, or restart the user-reply functionality's management daemon or service.
- Do not specify the `adshread` command to the action of the trap command if, in the TRAP_ACTION_SIGTERM parameter, you specify `TERM` or, in the UNIX edition, you specify `AUTO` (and the job starts from JP1/AJS).

Usage example

- Output the reply-request message and determine the processing based on the reply from the operator.

```
adshread ans "Continue (Y/N)?"

if  [ "$ans" = "Y" ] ; then
     echo "Continuing processing."
elif  [ "$ans" = "N" ] ; then
     echo "Terminating processing."
     exit 1
else
     echo "Invalid reply entered. Terminating processing."
     exit 1

fi
```

adshscripttool command (supports creation of job definition scripts) (Windows only)

Format

```
adshscripttool -fowner path-name
adshscripttool -fentry path-name
adshscripttool -fmode [-s {u|g|o|r|w|x}] mode
adshscripttool -exec {-m SIMPLE|MINIMUM}]
              {-r command-line|job-definition-script-file-path-name}
              [run-time-parameters]
```

Description

This command collects and outputs information in order to make it easy to create job definition scripts. The following table describes the arguments that can be specified and their purposes:

Argument	Execution content	Purpose
`-fowner`	Outputs the owner name of a file or folder.	Use this argument when you want to acquire the owner name of a file or folder in order to change the access privilege of the owner of the file or folder.

Argument	Execution content	Purpose
`-fentry`	Outputs a list of account names registered in the ACL of a file or folder.	Use this argument when you want to change the execution of the `cacls` or `attrib` command in the job definition script according to the account information in the acquired ACL.
`-fmode`	Parses the symbol or numeric value specified as the mode for the `chmod` command, and outputs the changes made to the permissions for owners, groups, and other users as 9-digit character strings so that they can be used easily in a job definition script. When this argument is specified together with the `-s` option, only the characters in the digits corresponding to the `ugorwx` specification are output out of the characters that would be output when the `-fmode` option is specified.	Use this argument when you want to change the execution of the `cacls` or `attrib` command in the job definition script according to the symbol or numeric value specified.
`-exec`	Executes the specified command line or job definition script as a child job.	Use this argument when you want to execute the specified command line or job definition script as a child job.

Arguments

-fowner

Specifies that the owner name of a file or folder is to be output to the standard output.

The owner name is output in the format *domain-or-computer-name*`\`*user-name* or *user-name* format.

Even if you use the `cacls` command to define Creator Owner, it might not be mapped to the file or folder owner. Before using the `cacls` command to specify an owner name, determine the owner name by executing the `adshscripttool` command with this option specified.

-fentry

Specifies that a list of account names registered in the ACL of a file or folder, separated by semicolons (`;`), is to be output to the standard output.

An account name is output in the format *domain-or-computer-name*`\`*user-name* or *user-name*.

path-name

Specifies the targeted file or folder.

-fmode

Specifies that the symbol or numeric value specified as the mode for the `chmod` command is to be parsed and that the changes made to the permissions for owners, groups, and other users are to be output as a 9-digit character string.

If a numeric value is specified, a value corresponding to mode bit ON is set to `R` and a value corresponding to mode bit OFF is set to `D`. If an error occurs, only `E` is output and processing terminates.

The following table explains the meaning of the character string that is output as the execution result of the `adshscripttool` command when the `-fmode` option is specified:

Digit number counted from the left side	Meaning
`1`	Owner's read permission
`2`	Owner's write permission
`3`	Owner's execute permission
`4`	Group's read permission
`5`	Group's write permission

Digit number counted from the left side	Meaning
6	Group's execute permission
7	Other users' read permission
8	Other users' write permission
9	Other users' execute permission

The following values are set in the individual digits:

Value	Meaning
A	Addition ((+) was set).
D	Deletion ((-) was set).
R	Substitution ((=) or numeric value was set).
0	Not specified.
E	The `adshscripttool -fmode` command terminated with an error.

-s[u|g|o|r|w|x]

Specifies one of the values listed below to indicate which of the digits specified when the `-fmode` option was specified are to be output. You can specify one of these values when you specify the `-fmode` option.

- `u`

  Corresponds to digits 1 to 3 of the output when the `-fmode` option was specified.

- `g`

  Corresponds to digits 4 to 6 of the output when the `-fmode` option was specified.

- `o`

  Corresponds to digits 7 to 9 of the output when the `-fmode` option was specified.

- `r`

  Corresponds to digits 1, 4, and 7 of the output when the `-fmode` option was specified.

- `w`

  Corresponds to digits 2, 5, and 8 of the output when the `-fmode` option was specified.

- `x`

  Corresponds to digits 3, 6, and 9 of the output when the `-fmode` option was specified.

mode

Specifies an 8-digit numeric value or symbol. You can specify *mode* when you specify the `-fmode` option.

- Specifying a numeric value

  Specify an octal number. If a non-octal number or a value that is greater than the octal value `07777` (`4095` in decimal) is specified, an error occurs.

- Specifying a symbol

  Specify setting, addition, or deletion when nothing is specified (`0` in the numeric value expression). The specification result of the symbol is output as the result.

  The table below shows what can be specified in a symbol. When specifying multiple items, separate them with a comma (`,`).

Order in the symbol	Value that can be specified
First	Specifies the item for which access permissions are being set. Multiple items can be specified concurrently. The items listed below can be specified. If the specification is omitted, *All users* is assumed. • u: Owner • g: Group • o: Other • a: All users
Second	Specifies the mode's operation. Executes one of the following processes for the item specified by the first symbol: • =: Sets access permission (overwriting). • +: Adds access permission. • -: Deletes access permission. You use the third symbol to specify the value to be set, added, or deleted. You can specify the second and third symbols following the third symbol. The third symbol can be omitted.
Third	Specifies the access permission to be set. Multiple values can be specified concurrently. The following values can be specified: • r: Read • w: Write • x: Execute If this symbol is omitted, the item for which access permission is to be set is erased. The erased value is set, added, or deleted according to the second symbol. Addition or deletion alone does not change the value. If s, t, u, g, or o is specified, it will be ignored.

-exec

Specifies that the command line specified in the -r option or the specified job definition script file is to be executed as a child job.

-m {SIMPLE|MINIMUM}

Specifies the mode to be used to output to the standard output or the standard error output. You can specify this option when the -exec option is specified.

For details about the simple output mode and the minimum output mode, see *3.3.4 Suppressing output of information and warning messages to job execution logs*.

- SIMPLE

  Runs the child job in the simple output mode.

- MINIMUM

  Runs the child job in the minimum output mode.

A sample script in the child job format is run in the minimum output mode. Therefore, if you use this command within a sample script in the child job format, we recommend that you specify MINIMUM in the -m option.

-r command line

Specifies what is to be executed by a job from the command line. You can specify this option when the -exec option is specified.

The information specified in the -r option is not created as a job definition script file in the spool job directory. Therefore, for the job definition script file name that is displayed, such as in messages, "-r CMDLINE" is displayed, not the absolute path of the job definition script file.

Command line specification

You can specify on the command line any commands that can be described in a job definition script, such as shell operation commands and UNIX-compatible commands. The following example specifies the pwd command, which is a standard shell command:

```
adshscripttool -exec -m MINIMUM -r pwd
```

You can specify any contents that can be described in a job definition script file, such as multiple commands separated by command separators. The following example specifies multiple commands on the command line:

```
adshscripttool -exec -m MINIMUM -r "export DATA=file01 ; pgm001"
```

If you specify a space on the command line, you must enclose the command line specification in single or double quotation marks (`'` or `"`). Because metacharacters, such as `$`, `*`, and the semicolon (`;`), are expanded, depending on the command-execution shell, you must enclose them in double quotation marks (`"`) or use an escape character (`\`).

Child job output information

If the `-r` option is specified, `"-r CMDLINE"` is displayed as the path name of the job definition script file in message texts and in the job definition script operation information that is output by JP1/Advanced Shell; the absolute path of the job definition script file is not displayed.

Positional parameter storage information

If positional parameter `$0` is specified for *command-line* in the `-r` option, `adshexec` is stored in `$0`.

Relationship to the `SPOOLJOB_CHILDJOB` parameter

Note the following when the command is executed with the `-r` option specified and `MERGE` is specified for the `SPOOLJOB_CHILDJOB` parameter: The values listed below, instead of the absolute path for the job definition script file, are output for the symbols that indicate the start and end of a child job's job execution logs to be output to the root job's job execution logs. For details about the output format when `MERGE` is specified for the `SPOOLJOB_CHILDJOB` parameter, see *3.4.1(3) Merging a child job's spool job into the root job's spool job*.

- Symbol indicating the start of a child job's `JOBLOG`
  `>>>>>> [JOBLOG] "-r CMDLINE"`
- Symbol indicating the end of a child job's `JOBLOG`
  `<<<<<< [JOBLOG] "-r CMDLINE"`
- Symbol indicating the start of a child job's standard error output (for normal execution)
  `>>>>>> [STDERR] "-r CMDLINE"`
- Symbol indicating the end of a child job's standard error output (for normal execution)
  `<<<<<< [STDERR] "-r CMDLINE"`
- Symbol indicating the start of a child job's standard error output and standard output (for debugging)
  `>>>>>> [STDERR,STDOUT] "-r CMDLINE"`
- Symbol indicating the end of a child job's standard error output and standard output (for debugging)
  `<<<<<< [STDERR,STDOUT] "-r CMDLINE"`

`"-r CMDLINE"` is also displayed as the job definition script file name in the script image file.

job-definition-script-file-path-name

~<path name>((1 to 247 bytes))

Specifies the path name of the job definition script file. You can specify a value when the `-exec` option is specified.

run-time parameters ~<character string>((1 to 1,022 bytes)))

Specifies the values to be stored in the positional parameters of the command line or job definition script file that is specified in the `-r` option. You can specify values when the `-exec` option is specified.

To include a space in a run-time parameter, enclose the character string in double quotation marks (`"`).

Return codes

Return code	Meaning
0	Normal termination
1	Error termination

Usage examples

- The following shows a specification example of the `-fowner` option and the result that is output to the standard output.

  Contents of job definition script:

  ```
  adshscripttool -fowner test.txt
  ```

  Result that is output to the standard output:

  ```
  MYPC\user1
  ```

- The following shows a specification example of the `-fentry` option and the result that is output to the standard output.

  Contents of job definition script:

  ```
  adshscripttool -fentry test.txt
  ```

  Result that is output to the standard output:

  ```
  BUILTIN\Administrators;NT AUTHORITY\SYSTEM;MYPC\user1;BUILTIN\Users
  ```

- The following shows an example in which `+w` is specified in the `-fmode` option and the result that is output to the standard output.

  Contents of job definition script:

  ```
  adshscripttool -fmode +w
  ```

  Result that is output to the standard output:

  ```
  0A00A00A0
  ```

- The following shows an example in which `ug-r` is specified in the `-fmode` option and the result that is output to the standard output.

  Contents of job definition script:

  ```
  adshscripttool -fmode ug-r
  ```

  Result that is output to the standard output:

  ```
  D00D00000
  ```

- The following shows an example in which `ug-w,u+w` is specified in the `-fmode` option and the result that is output to the standard output.

  Contents of job definition script:

  ```
  adshscripttool -fmode ug-w,u+w
  ```

  Result that is output to the standard output:

```
0A00D0000
```

- The following shows an example in which 655 is specified in the -fmode option and the result that is output to the standard output.

  Contents of job definition script:

  ```
  adshscripttool -fmode 655
  ```

  Result that is output to the standard output:

  ```
  RRDRDRRDR
  ```

- The following shows a specification example of the -fmode option and the result that is output to the standard output (in this example, +w is parsed with the -s w specification).

  Contents of job definition script:

  ```
  adshscripttool -fmode -s w +w
  ```

  Result that is output to the standard output (the second, fifth, and eighth digits of the result 0A00A00A0 are output):

  ```
  AAA
  ```

- The following shows a specification example of the -fmode option and the result that is output to the standard output (in this example, 655 is parsed with the -s r specification).

  Contents of job definition script:

  ```
  adshscripttool -fmode -s r 655
  ```

  Result that is output to the standard output (the first, fourth, and seventh digits of the result RRDRDRRDR are output):

  ```
  RRR
  ```

- The following shows a specification example of the -fmode option and the result that is output to the standard output (in this example, 655 is parsed with the -s uor specification).

  Contents of job definition script:

  ```
  adshscripttool -fmode -s uor 655
  ```

  Result that is output to the standard output (the first and seventh digits of the result RRDRDRRDR are output):

  ```
  RR
  ```

- The following shows an example in which a job definition script is specified to switch the execution of the cacls command on the basis of the symbol parsing result:

  ```
  username=`adshscripttool -fowner "$1"`       # Acquires the owner name.
  if [[ $? -ge 1 ]]                            # Error processing of
  adshscripttool -fowner
  then
    echo "adshscripttool -fowner error."
    return 1
  fi
  modebit=`adshscripttool -fmode $mode`        # Parses the mode. (mode=u+w)
  case $modebit in                             # 0A0000000 is stored in
  modebit.
    "AA0000000" ) cacls "$1" /E /G $username:C ;;
    "0A0000000" ) cacls "$1" /E /G $username:W ;;     # This cacls is
  ```

```
executed.
          "E" ) echo "adshscripttool -fmode error." # Error processing of
adshscripttool -fmode
                return 1 ;;
esac
```

- The following shows an example in which a job definition script is specified to delete all ACEs, except for those of the owner and `Everyone`:

```
IFS=\;
username=`adshscripttool -fowner "$1"`      # Acquires the owner name.
if [[ $? -ge 1 ]]                           # Error processing of
adshscripttool -fowner
then
  echo "adshscripttool -fowner error."
  return 1
fi
set -A entry `adshscripttool -fentry $1`    # Acquires an account name
list.
for i in "${entry[@]}"
do
  if ! [[ $i == "$username" || $i == "Everyone" ]]
  then
    cacls "$1" /E /R "$i"                   # Deletes all ACEs, except for
those of the owner and Everyone.
  fi
done
```

- The following shows an example in which a job definition script is specified to switch the execution of the `cacls` command according only to the definition content for the owner out of the numeric value parsing results:

```
username=`adshscripttool -fowner "$1"`      # Acquires the owner name.
if [[ $? -ge 1 ]]                           # Error processing of
adshscripttool -fowner
then
  echo "adshscripttool -fowner error."
  return 1
fi
modebit=`adshscripttool -fmode -s u $mode` # Parses the mode.(mode=644)
case $modebit in                            # RRD------ is stored in
modebit.
  "RRR" ) cacls "$1" /P $username:F ;;
  "RRD" ) cacls "$1" /P $username:C ;;           # This cacls is executed.
  "RDD" ) cacls "$1" /P $username:R ;;
  "DRD" ) cacls "$1" /P $username:W ;;
    "E" ) echo "adshscripttool -fmode error."    # Error processing of
adshscripttool -fmode
          return 1 ;;
esac
```

- The following shows an example of execution as a child job of the command line `export DBPATH=C:\\HOME\\DBUSER; start -q`:

```
adshscripttool -exec -m MINIMUM -r 'export DBPATH=C:\\HOME\\DBUSER; start
-q'
```

- The following shows an example in which the job definition script file `ppstart.ash` is executed as a child job:

```
adshscripttool -exec -m MINIMUM ppstart.ash
```

Notes

- When the `-fowner` and `-fentry` options are specified, the command must be executed by the owner of the targeted file or folder. If this is not the case, the command might output an error message and terminate with an error.
- You cannot specify the `-s` option before the `-fmode` option. If the `-s` option is specified before the `-fmode` option, a command parsing error occurs.
- If an internal conflict is detected, the shell being executed is terminated. When an error other than an internal conflict is detected, processing of the shell being executed continues.
- If an error occurs while the security information of the ACEs is being viewed with the `-fentry` option specified, only the account names of the ACEs that were viewed successfully are output and processing terminates with an error.
- If a child job is executed with the `-exec` option specified, the command name that will be output to the job execution log is a JP1/Advanced Shell command (`adshexecsub` command).

9.5 Extended script commands

The extended script commands begin with `#-adsh_` and are used in job definition script files.

You use extended script commands to create files and assign them to environment variables, to perform postprocessing of files after executing a job definition script or job step, and to declare the job name of a job definition script. In addition, you can define job steps, control the execution of jobs, and call external scripts with extended script commands.

The return code of an extended script command can be changed with the `ADSHCMD_RC_ERROR` and `ADSHCMD_RC_SUCCESS` environment setting parameters. However, the return code cannot be changed in the following circumstances:

- Normal termination or error termination of a job step by the `#-adsh_step_end` command
- Normal termination by the `#-adsh_script` command

For details about the environment setting parameters, see *7. Parameters Specified in the Environment Files*.

#-adsh_file command (specifies assignment and postprocessing of regular files)

Format

```
#-adsh_file file-environment-variable file-path
             [-chk {exist|no}]
             [-normal {del|keep}] [-abnormal {del|keep}]
```

Description

This command assigns a regular file, checks for the existence of a regular file, or specifies the postprocessing of a regular file. A maximum of 4,095 regular files can be assigned.

For details about assigning and postprocessing regular files, as well as the functional differences from the `adshfile` command, see *5.9.1 Allocating regular files and performing postprocessing*.

The regular files assigned by this command are managed separately from those assigned by the `adshfile` command. However, postprocessing is executed first for the regular files assigned by the `adshfile` command and then for the regular files assigned by the `#-adsh_file` command. If the same file is assigned by both commands, it will be postprocessed twice and an error might result.

Arguments

file-environment-variable

~<environment variable name>((1 to 31 bytes))

Specifies the name of a file environment variable that is being defined. This will serve as the key for identifying the regular file you are assigning to it.

In Windows, if `ENABLE` is specified in the `VAR_ENV_NAME_LOWERCASE` environment setting parameter, lowercase letters can be specified. If `DISABLE` is specified, lowercase letters cannot be specified.

file-path

Windows: ~<path name>((1 to 247 bytes))

UNIX: ~<path name>((1 to 1,023 bytes))

Specifies the path of the regular file that is to be assigned.

If a relative path is specified, it is converted into an absolute path before the command is executed. Make sure that the path length following conversion to an absolute path does not exceed the maximum permissible path length set by the OS. If the maximum set by the OS is exceeded, an execution error will occur.

-chk{exist|no}

Specifies whether a check is to be conducted for the existence of the specified regular file. If this specification is omitted, `no` is assumed.

- `exist`

  Check for the existence of the file.

- `no` or not specified

  Do not check for the existence of the file.

-normal{del|keep}

Specifies postprocessing that is to be performed when the applicable job or job step terminates normally. If this specification is omitted, `keep` is assumed.

- `del`

  After the applicable job or job step has completed, delete the assigned regular file.

- `keep`

  After the applicable job or job step has completed, do not delete the assigned regular file.

-abnormal{del|keep}

Specifies postprocessing that is to be performed when the applicable job or job step terminates with an error. If this specification is omitted, `keep` is assumed.

- `del`

  After the applicable job or job step has completed, delete the assigned regular file.

- `keep`

  After the applicable job or job step has completed, do not delete the assigned regular file.

Return codes

Return code	Meaning
0	Normal termination
1	Error termination

#-adsh_file_temp command (assigns and postprocesses a temporary file)

Format

```
#-adsh_file_temp file-environment-variable [ -id temporary-file-identifier]
                  [-chk {create|exist}]
                  [-normal {del|keep}]
```

Description

This command assigns a file to be used temporarily in a job definition script and specifies its postprocessing. A maximum of 4,095 temporary files can be assigned. For details about how to use the `#-adsh_file_temp` command to create temporary files, see *5.9.2 Allocating temporary files and performing postprocessing*.

Arguments

file-environment-variable

~<environment variable name>((1 to 31 bytes))

Specifies the name of a file environment variable that is being defined. This will serve as the key for identifying the temporary file you are assigning to it

In Windows, if `ENABLE` is specified in the `VAR_ENV_NAME_LOWERCASE` environment setting parameter, lowercase letters can be specified. If `DISABLE` is specified, lowercase letters cannot be specified.

-id temporary-file-identifier

~<symbolic name>((1 to 31 bytes))

Specifies an identifier for the temporary file, so that a temporary file created in a job step can be used in subsequent job steps. This argument can be omitted if you will not be using the assigned temporary file in any subsequent job step. This argument cannot be used to make assignments outside of the job's job steps.

The temporary file identifier must be unique among the temporary files that are created within the job. You cannot specify an identifier that is already in use for a temporary file created in another job step. However, you can specify the identifier of a file that was deleted during postprocessing in a previous job step.

-chk{create|exist}

Specifies whether to create a new temporary file or to assign an existing temporary file. If this specification is omitted, `create` is assumed.

- `create`

  Create a new temporary file. JP1/Advanced Shell will generate a file name and create a file whose size is zero bytes.

- `exist`

  Assign an existing temporary file that was created in a previous job step. You cannot use this specification to assign a file that was not created in a job step of the current job. When this option is specified, the `id` option and a temporary file identifier must also be specified.

-normal{del|keep}

Specifies the post-processing that is to be performed on the temporary file. If this specification is omitted, `del` is assumed.

- `del`

  After the applicable job or job step has completed, delete the assigned temporary file.

- `keep`

  After the applicable job step has completed, do not delete the assigned temporary file. The temporary file can be used again subsequently only in the current job's job steps. When this option is specified, the `id` option and a temporary file identifier must also be specified.

Return codes

Return code	Meaning
0	Normal termination

Return code	Meaning
1	Error termination

#-adsh_job command (declares a name for a job)

Format

```
#-adsh_job job-name
```

Description

This command declares a name for the job in the job definition script. The declaration of the job name can be specified on either the first or second line.

Argument

job-name

~<symbolic name>((1 to 31 bytes))

Specifies a name for the job, which will serve as one way to identify the job. The job name will be displayed in messages, such as in the job execution log, and will also be used as part of file names created by JP1/Advanced Shell.

Return codes

Return code	Meaning
0	Normal termination
1	Error termination

#-adsh_job_stop command (defines termination conditions for a job)

Format

```
#-adsh_job_stop return-code-definition[,return-code-definition ...]
```

Description

This command defines conditions to be used to determine whether to terminate the job at the end of a job step. A maximum of 1,023 job termination conditions can be specified.

Arguments

return-code-definition[,return-code-definition]...

Specifies definitions for job step return codes that are to be used to determine whether to terminate the job.

Multiple return code definitions can be specified, delimited by the comma (`,`), in which case the job will be terminated when any of the definitions is satisfied. You can specify a maximum of eight return code definitions.

return-code-definition

~<unsigned integer>((0 to 255))

- *return-code*

  Terminate the job when the specified return code is returned.

- *return-code-1*`:`*return-code-2*

  Terminate the job when the return code that is returned is in the range of the specified return codes, inclusive.

- *return-code*`:`

  Terminate the job when the return code that is returned is equal to or greater than the specified return code.

- `:`*return-code*

  Terminate the job when the return code that is returned is less than the specified return code.

- `:`

  Do not terminate the job on the basis of the return code that is returned. A syntax error results if you use this format while specifying multiple return code definitions.

Return codes

Return code	Meaning
`0`	Normal termination
`1`	Error termination

Notes

- A parsing error results if this command is used in a job step.
- Once the job has terminated, no subsequent job definition scripts will be executed, regardless of the value of the `run` attribute for the subsequent job steps.

#-adsh_path_var command (defines shell variables for handling path names)

Format

```
#-adsh_path_var shell-variable-name[, ... shell-variable-name]
```

Description

This command defines shell variables for handling path names. This command is enabled when the `PATH_CONV_ENABLE` parameter is defined in the environment file.

The `#-adsh_path_var` command can be specified at the following locations only:

- The line after `#!`*any-character-string* on the first line
- The line after the `#-adsh_job` command
- The first line (continuation lines are allowed).

When a shell variable specified by this command is described at the beginning of a path name, the path and directory separators in that path name as defined by the `PATH_CONV_ENABLE` parameter are converted to the path and directory separators of the target OS.

A text whose leading part matches the shell variable specified in either of the following formats is treated as a path name:

- `$`*shell-variable-name*
- `${`*shell-variable-name*`}`

Variable names must match completely. If *shell-variable-name* has a suffix consisting of a letter or the underscore (_), it will be evaluated as not to be a targeted shell variable and will not be converted.

After conversion, if the character string to be converted contains a path separator or directory separator defined by the `PATH_CONV_ENABLE` parameter, that separator is converted according to the OS under which the job definition script will be executed.

The conversion result depends on whether path conversion rule 1 or path conversion rule 2 was set by the `PATH_CONV_RULE` parameter. For details about the path conversion rules, see *PATH_CONV_RULE parameter (defines a rule for converting file paths) (Windows only)*. For conversion examples of job definition scripts, see *2.6.2 Converting path names*.

Arguments

shell-variable-name

~<environment variable name>((1 to 255 bytes))

Specifies the name of an environment variable that is to be used to define a shell variable for handling path names. A maximum of 255 shell variables can be specified. To use the defined shell variable in a job definition script, is must be specified as `$`*shell-variable-name* or `${`*shell-variable-name*`}`.

For details about pre-defined shell variables and shell variable names that cannot be used, see *5.5 Shell variables*.

Return codes

Return code	Meaning
0	Normal termination
1	Error termination

Notes

- The conversions specified by this command are performed on a line-by-line basis. For this reason, if there is a linefeed in a job definition script, the path name might not be converted correctly.

  In the following example, `$DIR1`*linefeed-code*`\\bar1\\...` is specified in the job definition script, to be converted based on path conversion rule 1 in Linux:

```
#-adsh_path_var DIR1,DIR2
echo foo > "$DIR1           ← Not converted because linefeed is entered
within "".
\\bar1\\"$DIR2\\bar2"bar3"  ← $DIR2\\bar2 is converted to $DIR2/bar2.
```

  In the following example, `$DIR1`*linefeed-code*`/bar1/...` is specified in the job definition script, to be converted based on path conversion rule 2 in Windows:

```
#-adsh_path_var DIR1,DIR2
echo foo > '$DIR1
/bar1/"$DIR2/bar2"bar3'      ← $DIR2/bar2 is converted to $DIR2\\bar2.
```

- Character strings with comments are also subject to conversion.
- If `DELETE` is specified for the `SPOOLJOB_CHILDJOB` parameter, no script images are output in the case of job definition scripts executed as child jobs. For this reason, if a job definition script executed as a child job is converted based on a conversion rule defined by this command, the conversion results will not be output.
- According to path conversion rule 2, you cannot nest single quotation marks (`'`) within a range enclosed in double quotation marks (`"`). Single quotation marks specified in such a case will be subject to path conversion.

#-adsh_rc_ignore command (defines commands to always terminate normally)

Format

```
#-adsh_rc_ignore command-name[,command-name ...]
```

Description

This command specifies commands that are always to terminate normally, regardless of the return code. The return code set by such a command will not be used to evaluate the success or failure of the job step. A maximum of 1,023 `#-adsh_rc_ignore` commands can be used to define commands that are to always terminate normally.

Note that if a specified command receives a signal and terminates, its termination will still be an error termination, regardless of this specification. For details about the definition method, see *5.8.4(2) Defining commands that always terminate normally*.

During execution of a job definition script, this command becomes applicable at the location where it is specified. If this command is specified outside of a job step, its definitions are applicable throughout the entire job definition script. If this command is specified within a job step, its definitions are applicable within that job step only, beginning at the point where it is specified through termination of the job step, during which period definitions specified outside the job step are disabled (however, before the point where it is specified within the job step, the definitions specified outside the job step are in effect).

Arguments

command-name [,command-name ...]

Specifies the commands that are to be defined to always terminate normally.

You can specify a maximum of 255 command names delimited by the comma (`,`).

- *command-name*

  Windows: ~<path name>((1 to 247 bytes))

  UNIX: ~<path name>((1 to 256 bytes))

  Specifies a command's base name.

Return codes

Return code	Meaning
0	Normal termination
1	Error termination

Notes

- A command must be specified by its base name. Do not use aliases or links within an applicable job step, because the names you use in the applicable job steps must match the base names you used to define the commands in the #-adsh_rc_ignore command.
- A maximum of 1,023 #-adsh_rc_ignore commands can be specified in a job definition script file.
- This command cannot be applied to an extended script command. The return code from an extended script command is always 0 for normal termination and 1 for error termination, and the job cannot continue when it terminates with an error.
- This command cannot be specified in a job step error block.
- When a batch job is interrupted and the KNAX6584-I message is output, the specification of this command will not have been applied to the command that executed last.
- If a command that is specified in the #-adsh_rc_ignore command will be executed in the format described in *5.1.7 Execution in a separate process (UNIX only)*, you must specify in the #-adsh_rc_ignore command the base name of the applicable command before character string substitutions have been applied.

Usage example

- Ignore the grep command's return code.

```
#-adsh_step_start STEP1
  #-adsh_rc_ignore grep
  UAP data|grep "TOTAL:"
#-adsh_step_end
```

#-adsh_script command (calls an external job definition script file from the job definition script that is running)

Format

```
#-adsh_script job-definition-script-file-name
```

Description

This command inserts the contents of an external job definition script file at the time the JP1/Advanced Shell was launched into the currently-running job definition script file. You can call a maximum of 4,095 external job definition script files from a job definition script that is running. An external script that is called is unpacked within the job definition script that contains the calling function. That job definition script is then parsed and executed as a single job definition script.

Arguments

job-definition-script-file-name

Windows: ~<path name>((1 to 247 bytes))

UNIX: ~<path name>((1 to 4,096 bytes))

Specifies the path of the job definition script file that to be deployed. If you specify a relative path, it is interpreted relative to the current directory at the time the job controller was launched.

Return codes

Return code	Meaning
Return code of the last command to terminate in the external script that was called	Normal termination
`1`	Error termination

Note:

You cannot use an environment setting parameter to change the return code when the `#-adsh_script` command terminates normally.

Notes

- This command differs from the standard shell `.` (dot) command in the following respects.
    - The `.` (dot) command executes the external script when the job definition script is executed. The `#-adsh_script` command executes the external script when the job definition script is parsed. You must not change the external script between the time the job definition script is parsed and the time it is executed.
    - The `.` (dot) command treats any extended script commands in the external script as comments. The `#-adsh_script` command is able to execute extended script commands contained in the external script.
    - If you execute the `#-adsh_script` command from an external script executed by another `#-adsh_script` command, do not call the same external script more than once.
    - The contents of an external script called by the `.` (dot) command are not output to the script image. The contents of an external script called by the `#-adsh_script` command are output to the script image.
    - If you specify an external script with a path that is relative to the `.` (dot) command, it resolves the path with reference to the value of the `PATH` environment variable. If you specify an external script with a path that is relative to the `#-adsh_script` command, it is interpreted as a path relative to the current directory when `adshexec` is started, without reference to the value of the `PATH` environment variable.
    - There is no limit to the number of times the `.` (dot) command can be used in a job. The `#-adsh_script` command can be used no more than 4,095 times in a job.
    - The `.` (dot) command allows you to specify arguments to the external script. The `#-adsh_script` command does not allow you to specify arguments to the external script.
- Do not specify a file name that begins with `.` (dot).
- Do not use a reserved device name (such as `CON`, `AUX`, and `NUL`) as a file name. (Windows only)
- Do not use an NTFS stream as a file name. (Windows only)
- Normal termination of this command cannot be used to identify the termination status (normal or error) of the job or job step. Refer instead to the execution results of the external script that was called.

#-adsh_spoolfile command (assigns a program output data file)

Format

```
#-adsh_spoolfile file-environment-variable
```

Description

This command assigns a program output data file. A maximum of 4,095 output files can be assigned. A maximum of 255 output files can be specified in a single job step or outside a job step. For details about the assignment method, see *5.9.3 Allocating program output data files and performing postprocessing*.

Arguments

file-environment-variable

~<environment variable name>((1 to 31 bytes))

Specifies the name of a file environment variable that is being defined. This will serve as the key for identifying the program output data file you are assigning to it.

In Windows, if ENABLE is specified in the VAR_ENV_NAME_LOWERCASE environment setting parameter, lowercase letters can be specified. If DISABLE is specified, lowercase letters cannot be specified.

Return codes

Return code	Meaning
0	Normal termination
1	Error termination

Notes

- Do not create directories under the spool job directory by using the path name assigned by the #-adsh_spoolfile command. If a directory is created under the spool job directory, an unexpected error might occur, such as an error resulting from spool job deletion by the adshhk command.
- The #-adsh_spoolfile command cannot be used when the spool job creation suppression functionality is being used. If the #-adsh_spoolfile command is used in such a case, the KNAX6385-E message is issued and the command is terminated.

#-adsh_step_start command, #-adsh_step_error command, #-adsh_step_end command (defines a job step)

Format

```
#-adsh_step_start
     [job-step-name]
     [-successRC return-code-definition[,return-code-definition ...]]
     [-stepVar shell-variable-name[,shell-variable-name ...]]
     [-run {normal|abnormal|always}]
     [-onError {cont|stop}]

... processing in the job step... (job step normal block)
```

```
  [#-adsh_step_error]

  [... processing at the time of a job step error... (job step error block)]

#-adsh_step_end
```

Description

This command groups a portion of the job definition script into a job step. A job step is a set of commands assembled into a group. A maximum of 4,095 job step definitions can be specified.

For details about how to use job steps, see *5.8.3 Defining job steps*.

For details about determining whether the commands executed within job steps terminate normally or result in an error, see *5.8.8 Return codes of jobs, job steps, and commands*.

For details about handling errors in job steps, see *5.8.10 Processing in the event of an error during job execution*.

Arguments

job-step-name

~<environment variable name>((1 to 31 bytes))

Specifies a name for the job step, which will serve as one way of identifying the job step. The specified job step name will be displayed in messages, such as in the job execution log, and will also be used as part of file names created by JP1/Advanced Shell.

Job step names can be duplicated within a job.

-successRC return-code-definition[,return-code-definition]...

Specifies definitions for the return codes from commands that execute in the job step normal block that will be considered to signify normal termination of the command. If you specify multiple return code definitions delimited by the comma (`,`), normal termination will be assumed if any of the definitions is satisfied.

If a command to be executed in the step normal block receives a signal and terminates, its termination will still be regarded as an error termination, regardless of this specification. A command executed within a step normal block might have returned a return code that does not match the return code of the command defined with the `successRC` attribute. Nevertheless, if the command matches the command name specified by the `#-adsh_rc_ignore` command, the specification by the `#-adsh_rc_ignore` command takes precedence regardless of the value specified for the `successRC` attribute.

return-code-definition

~<unsigned integer>((0 to 255))

You can specify a maximum of eight return code definitions.

- *return-code*

  Terminate normally when the return code that is returned matches the specified return code.

- *return-code-1*`:`*return-code-2*

  Terminate normally when the return code that is returned is in the range of the specified return codes, inclusive.

- *return-code*`:`

  Terminate normally when the return code that is returned is equal to or greater than the specified return code.

- `:`*return-code*

  Terminate normally when the return code that is returned is less than the specified return code.

-stepVar shell-variable-name[,shell-variable-name ...]

Specifies shell variables that are to be valid only within the job step. You can specify a maximum of 32 shell variable names delimited by the comma (`,`).

- *shell-variable-name*

  ~<environment variable name>((1 to 255 bytes))

  Specifies the name of a shell variable that is to be valid only within the job step. Names of function information arrays cannot be specified.

-run{normal|abnormal|always}

Specifies whether execution of the job step is to depend on the status of preceding job steps and commands in the job definition script. If this specification is omitted, `normal` is assumed.

- `normal`

  Execute the job step only if no earlier job step terminated with an error and no command in the previous portion of the job definition script terminated with an error.

- `abnormal`

  Execute the job step even if an earlier job step terminated with an error or a command in the previous portion of the job definition script terminated with an error.

- `always`

  Always execute the job step, regardless of the results of earlier job steps or the preceding portion of the job definition script.

-onError{cont|stop}

Specifies whether branching to the job step error block is to occur when a command in the job step normal block terminates with an error. When `stop` is specified, processing branches to the job step error block, and the subsequent portion of the job definition script in the job step normal block is not executed. When `cont` is specified, branching does not occur, and the subsequent portion of the job definition script in the job step normal block is executed. If this specification is omitted, `stop` is assumed.

- `cont`

  Execute the subsequent portion of the job definition script in the job step normal block.

- `stop`

  Execute the portion of the job definition script in the job step error block, without executing the subsequent portion of the job definition script in the job step normal block.

Return codes

For `#-adsh_step_start` and `#-adsh_step_error`

Return code	Meaning
`0`	Normal termination
`1`	Error termination

For `#-adsh_step_end`

Return code	Meaning
Return code of the last command that executed in the job step normal block	Job step terminated normally.
	Job step terminated with an error.
Argument to the `exit` command	Executed the `exit` command with an argument specified within the job step error block and terminated.

Return code	Meaning
1	#-adsh_step_end itself terminated with an error.

Note:

You cannot use an environment setting parameter to change the return code when a job step that contains the #-adsh_step_end command terminates normally or when a job step terminates with an error.

Notes

- If you specify a job step within the block of a control statement (if, for, while, until, or case), you must specify #-adsh_step_start through #-adsh_step_end within the same block; if you violate this rule, a syntax error will occur before execution.
- Do not define a job step within the block of a for, while, or until statement. If there is an external script expansion in such a block, the external script cannot include a job step. If it does, a pre-execution syntax error will result.
- You can define a job step within the block of an if or case statement. However, you cannot then specify abnormal or always in the run attribute.
- A job step cannot be defined inside another job step.
- If the command that executed last in a block terminates normally with a non-zero return code because of the specification of the #-adsh_rc_ignore command, the return code for the job step might be non-zero even though the job step terminates normally.

  Job definition script

```
  #-adsh_rc_ignore cmdA
  #-adsh_step_start S1 -onError cont
    cmdA    #rc=4 command
    cmdA    #rc=4 command
  #-adsh_step_end
```

  Execution log

```
  KNAX6117-I Execution of the command /home/hitachi/bin/cmdA (line=3)
finished. exit status=4 execution time=0.001s CPU time=0.000s
  KNAX6117-I Execution of the command /home/hitachi/bin/cmdA (line=4)
finished. exit status=4 execution time=0.001s CPU time=0.000s
  KNAX6597-I ADSH152257.S1 step succeeded. exit status=4 execution
time=0.005s CPU time=0.000s
```

- When a batch job is interrupted and the KNAX6584-I message is output, the successRC attribute will not have been applied to the command that executed last.
- When you define a function in a job step normal block or job step error block, you can use the defined function even if the job step is skipped using the run attribute.

Usage examples

- An error results if #-adsh_step_start is specified within the block of an if control statement and the corresponding #-adsh_step_end is specified outside the block.

```
if  [ [ $a = $b ]]; then
  #-adsh_step_start S1
fi
  #-adsh_step_end
```

- An error results if you define a job step in the block of a `while` control statement.

```
while  [ [ $a = $b ]] do
  #-adsh_step_start S1
  #-adsh_step_end
done
```

- Define a job step in the block of an `if` control statement.

```
if  [ [ $a = $b ]]; then
  #-adsh_step_start S1
  #-adsh_step_end
fi
```

9.6 Script control statements

Script control statements are used in job definition scripts.

A job definition script uses the results of conditional expressions in the control statements to determine the processing that is to be executed. You can specify any number of spaces or tab characters (including none) before the reserved words and processing instructions that constitute the control statements.

case statement (chooses from multiple processing paths)

Format

```
case expression in
        pattern-1)processing-a
                ;;
        pattern-2)processing-b
                ;;
        ...
        *)processing-x
                ;;
esac
```

Summary

This control statement determines the processing to be executed, based on finding a match with a specified character string (*expression*).

Description

The `in` keyword indicates the beginning of the types of processing defined in the `case` statement, and `esac` indicates the end of the `case` statement. When the specified expression matches a specified pattern, the processing described between the right parenthesis (`)`) and the double semicolons (`;;`) is executed. Multiple patterns can be specified, with the double semicolons (`;;`) serving as the delimiter between patterns. The pattern specified as an asterisk (`*`) is for the default processing when no match is detected with any of the other patterns. The matching of *expression* to the patterns is conducted in the order in which the patterns are specified. If *expression* matches multiple patterns, the processing prescribed for the first match is executed.

You can specify a left curly bracket (`{`) instead of `in`, and you can specify a right curly bracket (`}`) instead of `esac`. If you use `in`, you must also use `esac`; if you use `{`, you must also use `}`. If these specifications are not paired in this way, the syntax will be invalid and the control statement will terminate with an error.

A regular expression with wildcards can be specified in *pattern*.

Usage examples

- The `;;` indicating the end of a pattern can be specified on the same line as the processing.

```
case $cnt in
    0)
        echo "cnt is ZERO" ;;
    *)
```

```
        echo "cnt is not ZERO" ;;
esac
```

- If the last command in a pattern is an extended script command, add a linefeed to prevent `;;` from being interpreted as an argument of the extended script command.

```
case $cnt in
    0)
        #-adsh_step_start STEP01
        echo "cnt is ZERO"
        #-adsh_step_end ;;    <--Error. Specify linefeed before ;;.
    *)
        #-adsh_step_start STEP01
        echo "cnt is not ZERO"
        #-adsh_step_end
        ;;
esac
```

for statement (repeats the same processing)

Format 1

```
for variable [in wordlists]
do
        processing
done
```

Format 2

```
for variable [in wordlists];do
        processing
done
```

Summary

This control statement repeats the same processing while incrementing a value.

Description

This statement begins with `for`, followed by `do`, and ends with `done`. The number of times the loop is executed is determined by the number of elements in *wordlists*. The processing described between `do` and `done` is executed while *variable* is successively assigned to each element of *wordlists*, starting from the left. After all the elements of *wordlists* have been assigned, the `for` statement terminates.

The elements of *wordlists* are delimited by a whitespace and identified as *element-1 element-2 ...element-n*.

When a variable is specified in *wordlists*, even if the value of the specified variable changes between `do` and `done`, the value assigned to the variable in the `for` statement remains unchanged.

If `$@` is specified for *wordlists*, the arguments to the job definition script are used as the values of *wordlists*. Omitting `in` *wordlists* is the same as specifying `$@` for *wordlists* (`in $@`).

You can specify a left curly bracket ({) instead of do, and you can specify a right curly bracket (}) instead of done. If you use do, you must also use done; if you use {, you must also use }. If these specifications are not paired in this way, the syntax will be invalid and the control statement will terminate with an error.

If you specify a semicolon (;) immediately after *wordlists*, you can continue specifying the statement on the same line.

Usage example

- Iterate through three values, displaying each in turn.

```
for num in 1 2 3
do
    echo "num is $num"
done
```

if statement (branches conditionally)

Format 1

```
if condition-1
then
        processing-a
 [elif condition-2
then
        processing-b]
 [else
        processing-c]
fi
```

Format 2

```
if condition-1;then
        processing-a
 [elif condition-2; then
        processing-b]
 [else
        processing-c]
fi
```

Summary

This control statement controls branching based on whether the result of a specified condition is true (0) or false (non-zero).

Description

This statement begins with if and ends with fi. A condition can be any command or a group of commands combined by using operators such as &&, ||, ( ), and { }. If the return code from the command or the command group is 0, execution proceeds to the then clause; if the return code is non-zero, execution proceeds to the else or elif clause.

The elif and else clauses can be omitted, while then and fi must always be specified. Multiple elif clauses can be specified. If the statement is missing any of if, then, or fi, the syntax is invalid and the statement will terminate with an error.

If you specify a semicolon (;) immediately after *condition*, you can continue specifying the statement on the same line.

Usage example

- Display the result of comparing a value to 3.

```
if  [ [ $num -eq 3 ]]
then
    echo "num = 3"
elif  [ [ $num -lt 3 ]]
then
    echo "num < 3"
else
    echo "num > 3"
fi
```

until statement (loops until a condition is true)

Format 1

```
until condition
do
        processing
done
```

Format 2

```
until condition;do
        processing
done
```

Summary

This control statement executes specified processing repeatedly until a specified condition becomes true.

Description

This statement begins with until, followed by do, and ends with done. A condition can be any command or a group of commands combined using operators such as &&, ||, ( ), and { }. The processing described between do and done is performed repeatedly until the return code from executing the command or the command group specified as the condition becomes 0. To exit from the until statement, there must be a change created by the processing described between do and done such that the condition becomes satisfied. If the condition is already satisfied when the until statement begins, the statement terminates without executing the specified processing.

do and done cannot be omitted. Without a matching pair of do and done, the syntax is invalid and the statement will terminate with an error.

If you specify a semicolon (;) immediately after *condition*, you can continue specifying the statement on the same line.

Usage example

- Display numbers in a loop from 0 until the value 10 is reached.

```
num=0
until  [ [ $num -eq 10 ]]
do
    echo "num is $num"
    ((num+=1))
done
```

while statement (loops while a condition is true)

Format 1

```
while condition
do
        processing
done
```

Format 2

```
while condition;do
        processing
done
```

Summary

This control statement executes specified processing repeatedly as long as a specified condition is true.

Description

This statement begins with `while`, followed by `do`, and ends with `done`. A condition can be any command or a group of commands combined using operators such as `&&`, `||`, `( )`, and `{ }`. The processing described between `do` and `done` is performed repeatedly as long as the return code from executing the command or the command group specified as the condition is `0`. To exit from the `while` statement, there must be a change created by the processing described between `do` and `done` such that the condition ceases to be true.

`do` and `done` cannot be omitted. Without a matching pair of `do` and `done`, the syntax is invalid and the statement will terminate with an error.

If you specify a semicolon (`;`) immediately after *condition*, you can continue specifying the statement on the same line.

Usage example

- Repeat display while the value of *num* is in the range of `0` to `9`:

```
num=0
while  [ [ $num -ne 10 ]]
do
    echo "num is $num"
    ((num+=1))
done
```

9.7 Reserved script commands

The reserved script commands can be used as reserved words in job definition scripts.

time command (displays the time used to execute a command)

Format

```
time  [-p] [command]
```

Description

This command outputs to the standard error output the amount of time it took to execute a command.

Specifies the command whose execution time is to be output to the standard error output. If *command* is omitted, the shell's execution time is output. The output formats are as follows.

- When *command* is specified

  ```
  command-name command-execution-time  command-name user-CPU-time   command-
  name system-CPU-time
  ```

  In Windows, the CPU times of grandchild processes are not included in the command's user CPU time and system CPU time.

- When *command* is omitted

  ```
  user-CPU-time-of-shell#  system-CPU-time-of-shell#
  ```

  #: Includes CPU time of processes launched from the shell.

  In Windows, the CPU times of grandchild processes are not included in the shell's user CPU time and system CPU time.

Arguments

-p

Specifies that the amounts of execution time, user CPU time, and system CPU time are to be output, each on a separate line.

command

Specifies the name of the command whose execution time and CPU time are to be output.

Return codes

Return code	Meaning
Return code of the specified command: `0` when no command is specified.	Normal termination

Notes

- The results of the `time` command cannot be redirected to a file other than the standard error output.

- The execution results of this command are not output to the job execution log. Note also that this command does not identify whether the job or job step terminated normally or with an error. Refer instead to the execution time that was output to the standard error output and to the execution results of the command that was called.

Usage example

- Output a command's execution time and CPU time.

  Contents of the job definition script

  ```
  time date
  ```

  Contents of the `STDOUT` file of the execution job

  ```
  ********   JOB SCOPE STDOUT    ********
  Thu Jul  7 11:06:38 JST 2011
  ```

  Contents of the `STDERR` file of the execution job

  ```
  ********   JOB SCOPE STDERR    ********
      0.01s real     0.00s user     0.00s system
  ```

10 Troubleshooting

This chapter describes troubleshooting, including how to respond when problems occur, the types of log information, the troubleshooting information that needs to be collected, and how to collect it.

10.1 Response procedure

If you encounter a problem while running a job definition script in JP1/Advanced Shell, such as the script terminating with an error, attempt to determine what caused the problem.

If a message was output, check its contents. For details about the corrective actions to be taken for each message, see *11. Messages*. Then, take the following actions according to the cause indicated in the output message:

- Problem with the job definition script
  If a message is output indicating a problem with the job definition script, do the following:
  - Investigate and handle problems
    Based on your investigation of the problem, modify the job definition script in the development environment, and then verify the results in the debugger.
  - Repeat the operation
    Attempt to perform the operation again in the execution environment.
- Problem that requires contacting a system administrator
  If a message is output indicating that you need to contact a system administrator, do the following:
  - Collect information
    Collect information that will be needed to investigate the cause of the problem, as described in *10.2 Information needed when a problem occurs*.
  - Investigate the problem
    Investigate the cause of the problem based on the collected information, and then narrow the scope of the problem by isolating the conditions under which it occurs.
- Problem entering a response to a request for a user reply
  - Investigate the problem and take corrective action
    See *10.1.1 Corrective action when using the user-reply functionality* to investigate the problem and take the necessary corrective action.

10.1.1 Corrective action when using the user-reply functionality

Replies to reply-request messages are sent from JP1/IM - View. However, in the following cases, you cannot enter a reply from JP1/IM - View.

Table 10–1: When you cannot enter a reply to a reply-waiting event

No.	Circumstance	Notification sent to the user
1	An error occurs in JP1/Advanced Shell when you enter a reply from JP1/IM, and the reply's success or failure is unknown in JP1/IM.	Invalid data (or a similar message) is sent.
2	A backlog of reply-waiting events was cleared because the backlog exceeded 2,000 events.	JP1/IM - View displays the `KAVB0551-E` error message.
3	JP1/IM has become unusable due to a communication failure or other problem	None (JP1/IM - View is not available).

Because reply-request messages are managed in shared memory on the machine where JP1/Advanced Shell is installed, a JP1/Advanced Shell administrator is able to use the following commands to check the status of reply-request messages and to reply to reply-request messages.

- Displaying a list of reply-request messages in reply-waiting status

  The administrator uses the `adshlsmsg` command to display a list of reply-request messages from operators that are in reply-waiting status. For details, see *adshlsmsg command (displays a list of reply-request messages when a failure occurs)* in *8.3 Shell operation commands*.

- Replying to (manually) or canceling a reply-request message in reply-waiting status

  The administrator uses the `adshchmsg` command to enter a reply to or cancel a reply-request message from an operator that is in reply-waiting status. For details, see *adshchmsg command (replies manually to a reply-request message when a failure occurs)* in *8.3 Shell operation commands*.

10.1.2 When the root job terminates before its child jobs terminate

If a child job that was executed from another child job is terminated abruptly from an intermediate job by a means such as `SIGKILL` in UNIX or `TerminateProcess` in Windows, the root job might terminate without waiting for all its child jobs to terminate. For this reason, do not execute an abrupt termination of this type. For details, see *3.2.3(4) Notes about child jobs that are executed from another child job*.

If this occurs, check the execution results of the related root job and its child jobs. For the child jobs other than the abruptly terminated job, the spool job directory might remain even after an attempt to delete it. Even if it was deleted, the contents of `JOBLOG` will have been output to the standard error output, so the log is not lost.

10.2 Information needed when a problem occurs

The following table shows the information to be collected when a problem occurs.

Table 10–2: Information to be collected when a problem occurs

Category	Contents	What to collect
Logs	Logs output by JP1/Advanced Shell	System execution log, trace log
Error information	Error information collected by the system	Dump file (Windows only) Core file (UNIX only)
Spool information	Information for managing the spool	The specified environment file and job ID file (either `.jobid` or `adsh.jobid`)
Environment information	System status	Basic system information, process information, memory usage information, file information, network usage, JP1 event information, error logs
User-reply functionality's management daemon information (UNIX only)	Information related to start and end of the user-reply functionality's management daemon	Start log and pid file of the user-reply functionality's management daemon

You can use JP1/Advanced Shell's `adshcollect` command to batch-collect the needed information. For details about the `adshcollect` command, see *10.3 How to collect information*.

The following sections provide details about the information that is needed in each of the above categories, with the exception of environment information, which is omitted because it is product-specific.

10.2.1 Logs

The following table shows the logs to be collected.

Table 10–3: Logs to be collected

Type	Contents	Output destination
System execution log	Entire execution log of JP1/Advanced Shell	This is output as specified by the `LOG_DIR` parameter# in the environment file.
Trace log	JP1/Advanced Shell internal trace log	This is output as specified by the `TRACE_DIR` parameter# in the environment file. Trace logs for custom jobs, the editor, and common commands are output in accordance with system specifications.

#

For the parameter default values, see *Table 10-8 Rules for keyword specifications in the environment file* in *adshcollect command (collects information)*.

10.2.2 Error information

The following table shows the error information to be collected. Error information is collected only when `DUMP` or `CORE` is specified in the definition file for collecting maintenance information.

Table 10–4: Error information to be collected

Type	Contents	Output destination
Dump file (Windows only)	Error information collected in the Watson log and similar sources	This file is output when you run a debug tool such as Dr. Watson. In the case of Dr. Watson, the default directory for output of the error information is: • *common-application-data-folder*`\Microsoft\Dr Watson\drwtsn32.log`
Core file (UNIX only)	Error information collected by the system	When a process terminates with an error, this file is output by the system to the directory specified in the system settings. If the directory is not specified in the system settings, the file is output to the directory that was current when `adshexec` started.

10.2.3 Spool information

The following table shows the spool information to be collected.

Table 10–5: Spool information to be collected

Type	Contents	Output destination
Environment file	JP1/Advanced Shell definition information	Path of the environment file created in the `ADSH_ENV` environment variable Environment file specified with the `-e` option of the `adshcollect` command (the job environment file defined for **Job environment file** in the Define Script Execution dialog box of a custom job or the job environment file defined in the Job environment file of the Runtime Environment Settings dialog box of the editor)
Files under the spool directory	Batch job information output to the spool	Job ID file (either `.jobid` or `adsh.jobid`)

10.2.4 User-reply functionality's management daemon information (UNIX only)

The following table shows the user-reply functionality's management daemon information to be collected.

Table 10–6: User-reply functionality management daemon information to be collected

Type	Contents	Output destination
Start log	User-reply functionality's management daemon start log	The user-reply functionality's management daemon start log is output to `/opt/jp1as/system`. For details, see *2.6.8 Defining job execution results and log output information*.

pid file	User-reply functionality's management daemon pid file	The user-reply functionality's management daemon pid file is output to `/opt/jp1as/system`.
		The pid file corresponds to the following file name:
		• User-reply functionality's management daemon on a physical host
		`adshmd.pid`
		• User-reply functionality's management daemon on a logical host
		`adshmd_`*logical-host-name*`.pid`

10.3 How to collect information

If JP1/Advanced Shell terminates with an error or becomes unresponsive, the system administrator will require data from core dumps (core files or dump files), logs, and other sources for purposes of investigating the cause of the failure. You can use the `adshcollect` command to batch-collect this required information.

This section describes how to use the `adshcollect` command, how to set up a definition file for collecting maintenance information, and how to set up an environment file. Note that the maintenance information (data) collected by the `adshcollect` command might be different in Windows and UNIX.

adshcollect command (collects information)

Format

```
adshcollect maintenance-information-output-directory [-f definition-file-
name] [-e environment-file-name] [-h logical-host-name]
```

Description

The `adshcollect` command enables batch-collection of the information required to investigate a failure.

To execute the `adshcollect` command, you start it from the command prompt in the case of Windows or from the shell in the case of UNIX.

In order to collect error information in the event of a failure, this command must be executed with the permissions of the executing user. However, to collect user-reply functionality information, it must be executed with an administrator role.

1. Collect the environment file that was being used when the failure occurred. If the environment file has been modified since the error occurred, reconstruct the environment file to match the operational environment at the time the failure occurred. If no environment file was being used when the failure occurred, there is no need to collect one.

   Note:

   In the Windows edition, the `adshcollect` command might result in an error if an ampersand (`&`) is specified in an environment file.

   If an ampersand (`&`) is specified in the job environment file, make a copy of the job environment file and delete the ampersand (`&`) from the copied file. In step 3, specify the copied job definition file in the `-e` option.

   If an ampersand (`&`) is specified in the system environment file, make a backup by copying the system environment file to another directory and then delete the ampersand (`&`).

2. Create a definition file.

   For collecting a core file or dump file, create a definition file at any location. There is no need to create a definition file except when a core file or dump file is required.

3. Execute the `adshcollect` command.

   Specify the arguments described below and execute the `adshcollect` command. For notes on executing the `adshcollect` command, see *Notes*, below.

   Maintenance information output directory

   Note the following points about the directory specified for the maintenance information:

   - The output directory for the maintenance information must be writable, and it must have sufficient space.

- It must also be a directory that is not being used in JP1/Advanced Shell.

Environment file name

Specify the path of the environment file collected in step 1 in the `-e` option or the `ADSH_ENV` environment variable. This specification is required only if an environment file was collected in step 1.

Definition file name

Specify in the `-f` option the path of the definition file created in step 2. This specification is required only if a definition file was created in step 2.

Logical host name

If the environment in which the failure occurred is a logical host, specify the logical host name in the `-h` option. This specification is required only if the environment in which the failure occurred is a logical host.

Arguments

maintenance-information-output-directory

Windows only

The files containing maintenance information are output to a destination directory. The directory name is in the following format:

`ADSH`*yyyymmddhhmmss*

- *yyyymmdd*: Date when the `adshcollect` command was started
- *hhmmss*: Time, in 24-hour local time, when the `adshcollect` command was started

Because Windows does not provide as a standard feature the equivalent of the UNIX `tar` command for handling maintenance information, you must use a user compression tool to compress the files in a standard format (such as ZIP or LZH).

UNIX only

Specifies a destination directory for the `tar` archive files of collected information. Any required temporary files will also be created in this directory. The name of the archive file is in the following format:

`ADSH`*yyyymmddhhmmss*`.tar`

- *yyyymmdd*: Date when the `adshcollect` command was started
- *hhmmss*: Time, in 24-hour local time, when the `adshcollect` command was started

The disk space required for the compressed file containing maintenance information is as follows:

Size of system execution logs and trace logs + size of files specified in DUMP or CORE#

#

`DUMP` files in a Windows environment and `CORE` files in a UNIX environment.

-f definition-file-name

Specifies the name of the definition file that defines the maintenance information to be collected. You can specify the definition file in terms of an absolute path or a path relative to the current directory. For the contents to be set up, see *Definition file and environment file settings*, below.

Specification of a definition file name is optional. If *definition-file-name* is omitted, the `DUMP` or `CORE` maintenance information will not be collected.

-e environment-file-name

Specifies an environment file when you want to specify a different file path from the one specified in the `ADSH_ENV` environment variable. You can specify an absolute path or a path relative to the current directory.

- When this option is not specified

  The file path specified in the `ADSH_ENV` environment variable is used for the environment file.

- When this option and the ADSH_ENV environment variable are both not specified

  Information is collected on the basis of the applicable settings in the system environment file.
- When this option and the ADSH_ENV environment variable are both not specified and there is no system environment file

  The default values for SPOOL_DIR, LOG_DIR, and TRACE_DIR are used.

-h logical-host-name

Specifies the name of the logical host where the error information is to be collected. The environment file is parsed based on the specified logical host name.

If the -h option is specified but no logical host name is specified, the logical host name is obtained from the JP1_HOSTNAME environment variable. If the JP1_HOSTNAME environment variable is not defined, the command outputs usage information and terminates with an error. For details about the JP1_HOSTNAME environment variable, see the *Job Management Partner 1/Base User's Guide*.

Definition file and environment file settings

Define the information to be collected in a definition file and define the output destination for the collected information in an environment file.

- Defining the definition file

  The definition file contains #-adsh_conf Δ $_1$, followed by keywords and their values delimited by the space. Specify all file names in terms of their absolute paths.

  The table below shows the rules for keyword specifications in the definition file. Although all keywords are optional, nothing other than keywords, including comments, is permitted in the definition file. Note that no wildcard characters can be specified in any keyword value.

Table 10–7: Rules for keyword specifications in the definition file

Keyword	Specification contents	Specify more than once
DUMP (Windows only)	Specifies a dump file you want to collect, such as a Watson log. For details about Watson logs, see the Windows documentation. If there are any spaces in the path, enclose the path in double quotation marks.	Y (maximum of 16)
CORE (UNIX only)	Specifies a directory where core files can be found that need to be collected as error information. Files under the specified directory that have core as part of their name will be batch-collected.	Y

Legend:

Y: Can be specified.

- Defining the environment file

  The table below shows the rules for keyword specifications in the environment file. All keywords are optional. If no keyword is specified, the information described under *Default path name* in the table will be collected. Note that no wildcard characters can be specified in any keyword value.

Table 10–8: Rules for keyword specifications in the environment file

Keyword (environment setting parameter)	Specification contents	Default path name	Specify more than once
SPOOL_DIR	Path name of the spool root directory#	• In the execution environment (Windows only) *shared-documents-folder*\Hitachi\JP1AS\JP1ASE\spool	N

Keyword (environment setting parameter)	Specification contents	Default path name	Specify more than once
`SPOOL_DIR`	Path name of the spool root directory#	- In the development environment (Windows only) *shared-documents-folder*`\Hitachi\JP1AS\JP1ASD\spool` - In the execution environment (UNIX only) `/var/opt/jp1as/spool`	N
`LOG_DIR`	Path name of the system execution log output directory#	- In the execution environment (Windows only) *shared-documents-folder*`\Hitachi\JP1AS\JP1ASE\log` - In the development environment (Windows only) *shared-documents-folder*`\Hitachi\JP1AS\JP1ASD\log` - In the execution environment (UNIX only) `/opt/jp1as/log`	N
`TRACE_DIR`	Path name of the trace log output directory#	- In the execution environment (Windows only) *common-application-data-folder*`\Hitachi\JP1AS\JP1ASE\trace` - In the development environment (Windows only) *commmon-application-data-folder*`\Hitachi\JP1AS\JP1ASD\trace` - In the execution environment (UNIX only) `/opt/jp1as/trace`	N

Legend:

N: Cannot be specified.

#

In Windows, if there are spaces in the path, enclose the path in double quotation marks.

Example definition file and environment file specifications

- In Windows

  The following is an example of specifying the definition file:

```
#-adsh_conf DUMP "C:\Program Files\Hitachi\JP1AS\JP1ASE\dump"
```

  The following is an example of specifying the environment file:

```
#-adsh_conf SPOOL_DIR "C:\Documents and Settings\All Users\Documents
\Hitachi\JP1AS\JP1ASE\spool"
#-adsh_conf LOG_DIR "C:\Documents and Settings\All Users\Documents\Hitachi
\JP1AS\JP1ASE\log"
#-adsh_conf TRACE_DIR "C:\Documents and Settings\All Users\Application
Data\Hitachi\JP1AS\JP1ASE\trace"
```

- In UNIX

  The following is an example of specifying the definition file:

  ```
  #-adsh_conf  CORE  /home/user1/program1
  ```

  The following is an example of specifying the environment file:

  ```
  #-adsh_conf  SPOOL_DIR  /var/opt/jp1as/spool
  #-adsh_conf  LOG_DIR  /opt/jp1as/log
  #-adsh_conf  TRACE_DIR  /opt/jp1as/trace
  ```

List of files collected by the adshcollect command

The files collected by the `adshcollect` command and their maximum sizes are different in Windows and UNIX, as shown in the following tables.

Table 10–9: Files collected by the adshcollect command and their maximum sizes (Windows only)

File type	File name	Maximum size	Collected
Spool management file	[`SPOOL_DIR` in the environment file#]`\adsh.jobid`	About 1 KB	Y
System execution log (JP1/Advanced Shell)	[`LOG_DIR` in the environment file#]`\AdshLog.log` [`LOG_DIR` in the environment file#]`\AdshLog_`*n*`.log` (where *n* is the log file count)	[`LOG_FILE_SIZE` in the environment file] × (*n* + 1) (MB)	Y
	[`LOG_DIR` in the environment file#]`\AdshLog.conf`	About 1 KB	Y
Execution log for JP1/Advanced Shell internal processing	*commmon-application-data-folder*`\Hitachi\JP1AS\JP1ASE\uxpl\spool\uxpllog[`*n*`].txt` (where *n* is the log file count, maximum 2)	5 MB	Y
	commmon-application-data-folder`\Hitachi\JP1AS\JP1ASD\uxpl\spool\uxpllog[`*n*`].txt` (where *n* is the log file count, maximum 2)	5 MB	Y
	commmon-application-data-folder`\Hitachi\JP1AS\misc\uxpl\spool\uxpllog[`*n*`].txt` (where *n* is the log file count, maximum 2)	5 MB	Y
Trace log (JP1/Advanced Shell)	[`TRACE_DIR` in the environment file#]`\AdshTrace_[`*n*`].log` (where *n* is the log file count: fixed at 4) Can be changed in the environment file.	[`TRACE_FILE_SIZE` in the environment file] × *n* (MB)	Y
Trace log (custom job)	*commmon-application-data-folder*`\Hitachi\JP1AS\JP1ASV\trace\AdshTrace_1.log`	1 MB	Y
Trace log (editor)	*commmon-application-data-folder*`\Hitachi\JP1AS\JP1ASD\adshedit\trace\AdshTrace_1.log`	1 MB	Y
Trace log (JP1/Advanced Shell, JP1/Advanced Shell - Developer common commands)	*commmon-application-data-folder*`\Hitachi\JP1AS\misc\trace\AdshTrace_[`*n*`].log` (where *n* is the log file count)	8 MB	Y

File type	File name	Maximum size	Collected
Trace log (editor-specific features)	*commmon-application-data-folder*`\Hitachi\JP1AS\JP1ASD\adshedit\trace\adshedit.txt`	Depends on user environment settings.	Y
Dump file	Dump file in `DUMP` in the definition file	Depends on user environment settings.	O
Environment file	File in the `ADSH_ENV` environment variable, or file specified in the `-e` option	About 1 KB	O
System environment file	*commmon-application-data-folder*`\Hitachi\JP1AS\`*product-name*`\conf\adshrc.ase`	About 1 KB	O
Host name set in the machine	*system-root-folder*`\system32\drivers\etc\hosts`	Depends on user environment settings.	Y
Service ports set in the machine	*system-root-folder*`\system32\drivers\etc\services`	Depends on user environment settings.	Y
Environment information file	`ADSHTMP`*yyyymmddhhmmss*`.txt` ● *yyyymmdd*: Date when the `adshcollect` command was started ● *hhmmss*: Time when the `adshcollect` command was started	Depends on user environment settings.	Y

Legend:

Y: Always collected by the `adshcollect` command.

O: Collected when the applicable `adshcollect` command option is specified.

#

Can be changed in the environment file. For the default path names, see *Table 10-8 Rules for keyword specifications in the environment file*.

Table 10–10: Files collected by the adshcollect command and their maximum sizes (UNIX only)

File type	File name	Maximum size	Collected
Spool	[`SPOOL_DIR` in the environment file#]`/.jobid`	About 1 KB	Y
System execution log	[`LOG_DIR` in the environment file#]`/AdshLog.log` [`LOG_DIR` in the environment file#]`/AdshLog_[`*n*`].log` (where *n* is the log file count)	[`LOG_FILE_SIZE` in the environment file] × (*n* + 1) (MB)	Y
	[`LOG_DIR` in the environment file#]`/AdshLog.conf`	About 1KB	Y
Trace log	[`TRACE_DIR` in the environment file#]`/AdshTrace_[`*n*`].log` (where *n* is the log file count)	[`TRACE_FILE_SIZE` in the environment file] × *n* (MB)	Y
Core file	Core file in the `CORE` keyword that is collected in the definition file	Depends on user environment settings	O
User-reply functionality management daemon information	Start log and pid file under `/opt/jp1as/system`	About 1 KB × *number of executing user-reply functionality management daemons*	Y
Environment file	File in the `ADSH_ENV` environment variable or file specified in the `-e` option	About 1 KB	O

File type	File name	Maximum size	Collected
System environment file	/opt/jp1as/conf/adshrc.ase	About 1 KB	Y
Installed Hitachi products	/etc/.hitachi/pplistd/pplistd	Depends on user environment settings.	Y
Environment variables	• AIX or Linux /etc/environment • HP-UX /etc/profile • Solaris /etc/skel/.profile	Depends on user environment settings.	Y
Environment information file	ADSHTMP*yyyymmddhhmmss*.txt ● *yyyymmdd*: Date when the adshcollect command was started ● *hhmmss*: Time when the adshcollect command was started	Depends on user environment settings.	Y
Tar logs	ADSHTARyyyymmddhhmmss.txt ● *yyyymmdd*: Date when the adshcollect command was started ● *hhmmss*: Time when the adshcollect command was started	About 1 KB	Y

Legend:

Y: Always collected by the adshcollect command.

O: Collected when the applicable adshcollect command option is specified.

#

Can be changed in the environment file. For the default path names, see *Table 10-8 Rules for keyword specifications in the environment file*.

Notes

- The maintenance information output directory must have adequate free space for the output files and temporary files to be created there.
- The maintenance information output directory must be writable so that the output files and temporary files can be created there.
- If the adshcollect command is forcibly terminated during execution, temporary files might still remain in the maintenance information output directory. In such a case, you will have to delete the temporary files manually.
- (UNIX only) If you are using the user-reply functionality, execute the adshcollect command as a user with root privileges. If you execute it as a user without root privileges, you will not be able to collect the user-reply functionality information.
- (Windows only) If you are using the user-reply functionality, execute the adshcollect command as a user with Administrators permissions. If you execute it as a user without Administrators permissions, you will not be able to collect the user-reply functionality information.
- Do not specify any of the following special characters in the path of an output directory for maintenance information, environment file path, definition file path, path specified in SPOOL_DIR, LOG_DIR, or TRACE_DIR, DUMP path, CORE path, or the current directory path for executing the adshcollect command:

  ```
  & ( ) [ ] { } ^ = ; ! ' + , ` ~ # %
  ```
- If you specify an option for any of the adshcollect command arguments *maintenance-information-output-directory*, *environment-file-name*, *definition-file-name*, or *logical-host-name*, the option will be interpreted as the directory or file name or logical host name.

11 Messages

This chapter lists the messages output by JP1/Advanced Shell and provides detailed information about errors that might occur.

11.1 Message format

This section explains the format of the messages that are issued by JP1/Advanced Shell.

11.1.1 Message output format

JP1/Advanced Shell issues messages in the following format:

```
KNAXnnnn-t message-text
```

- KNAX
  Message prefix that identifies a message as having been issued by JP1/Advanced Shell.
- *nnnn*
  Message number.
- *t*
  Indicator of the type of message. The following table lists and explains the types.

Table 11–1: Types of messages

Type identifier	Type	Meaning
E	Error	• A failure that disables a library, command, or server function has occurred. • Operation has been disabled because of an invalid definition or command argument.
W	Warning	Processing continues once this message has been output.
I	Information	Information for the user.

(1) Format of messages output to job execution logs

When messages are output to job execution logs, the time and job ID are added to the messages as follows:

```
time job-ID KNAXnnnn-t message-text
```

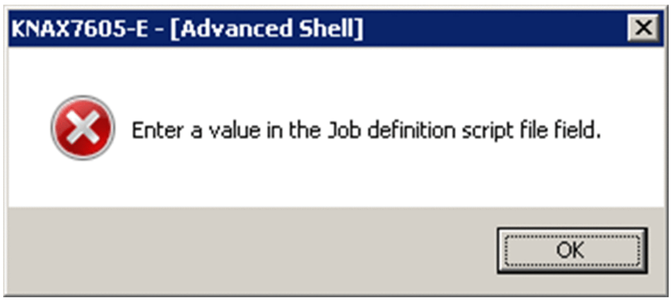
- *time*
  Time the message was output, in the format *hh*:*mm*:*ss*.
- *job-ID*
  Six-digit identifier of the job that issued the message. If the job ID consists of fewer than six digits, it is padded with leading zeros to make it six digits in length.

(2) Format of messages output in a message dialog box or error window

Some messages are output in a message dialog box or an error window, as shown in the following example.

- Message dialog box

Figure 11–1: Message dialog box



KNAX7605-E - [Advanced Shell]

Enter a value in the Job definition script file field.

A message dialog box contains an icon that indicates the type of message. The following table explains the icons that are displayed in message dialog boxes.

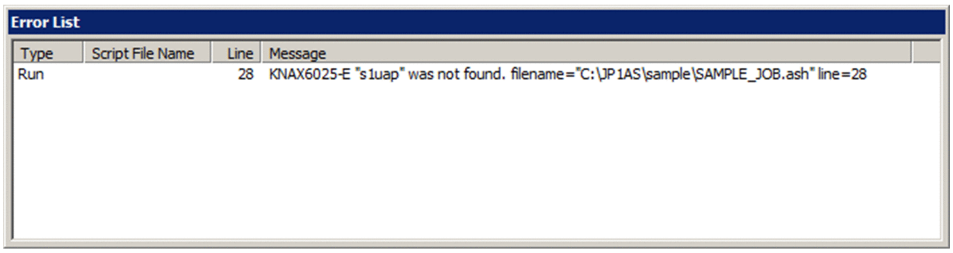
Table 11–2: Icons displayed in message dialog boxes

Type	Icon	Meaning	User's action
Information	i	An event that needs to be reported to the user occurred during processing.	Click **OK**.
Query	?	An event requiring an action by the user has occurred. The message asks the user to select one of two choices.	Select **Yes** or **No**.
Warning	⚠	A warning event that needs to be reported to the user occurred during processing. The message asks the use to select one of two choices.	Select **OK** or **Cancel**.
Error	✕	An error occurred during processing.	Click **OK**.

- Error window

Messages might be displayed in the error window shown in the following while you are using JP1/Advanced Shell Editor.

Figure 11–2: Error window



Error List

Type	Script File Name	Line	Message
Run		28	KNAX6025-E "s1uap" was not found. filename="C:\JP1AS\sample\SAMPLE_JOB.ash" line=28

11.1.2 Format of message explanations

The format of the message explanations in this chapter is shown in the following.

The *italics* font indicates a placeholder (variable) for the value that will actually be set in the message text. The notation *(Windows only)* or *(UNIX only)* to the right of a message ID indicates that the message is displayed only in a Windows or UNIX environment, respectively.

For details about the meaning of *error-details* described in a message text and the action to be taken, see *11.4 Details of errors*.

The messages are listed in order of message ID. The following shows an example of the message explanations:

message-ID [(Windows only)|(UNIX only)]

message-text

Explanation of the message

(S)

Indicates the system processing.

(O)

Indicates the action to be taken by the developer or operator when the message is output.

11.1.3 Assignment of message numbers

Messages are grouped by ranges of message numbers into broad subject categories. The following table lists the subject categories of the ranges of message numbers (these are the numbers that follow JP1/Advanced Shell's message prefix `KNAX`).

Table 11–3: Subject categories assigned to ranges of message numbers

Message numbers	Subject category of messages
`0001` through `0299`	Basic job processing
`0300` through `0399`	Command arguments
`0400` through `0699`	Environment files
`0700` through `0899`	Job execution logs
`1600` through `1899`	Area allocation
`1900` through `2199`	Job step execution
`2200` through `2499`	Message processing
`3000` through `3999`	Daemons
`4414` through `4429`	Spool job manipulation
`5300` through `5399`	Adapter commands
`5400` through `5499`	User-reply commands
`6000` through `6699`	Batch job execution control
`6700` through `6999`	Cross-platform operability
`7000` through `7399`	Development environment
`7400` through `7599`	User-reply functionality
`7600` through `7799`	JP1 program linkage functions
`7800` through `7999`	Common functions
`9000` through `9999`	License-related

11.2 Message output destinations

The table below lists the output destinations of the messages issued by JP1/Advanced Shell. This table shows the output destinations of messages in the expansion output mode. For details about the output destinations of messages in the simple output mode and the minimum output mode, see the explanations that follow the table.

Table 11–4: Output destinations of messages issued by JP1/Advanced Shell (expansion output mode)

Message IDs	Output destinations of messages				
	stdout	stderr	JOBLOG	System execution log	GUI
KNAX0001-E	N	Y	Y	Y	Y
KNAX0004-I	N	N	N	Y	N
KNAX0030-E, KNAX0031-E	N	Y	N	N	Y
KNAX0091-I, KNAX0092-I	N	N	Y	Y	N
KNAX0098-I	N	Y	Y	Y	N
KNAX0101-E	N	Y	Y	Y	Y
KNAX0220-E	N	Y	N	Y	N
KNAX0235-E to KNAX0239-E	N	Y	N	Y	Y
KNAX0240-I	N	Y	N	Y#1	N
KNAX0299-E	N	Y	Y	Y	Y
KNAX0300-I	N	Y	N	N	N
KNAX0301-E to KNAX0307-E	N	Y	N	N	Y
KNAX0308-E to KNAX0309-I	N	Y	N	N	N
KNAX0310-E to KNAX0336-E	N	Y	N	N	Y
KNAX0401-E to KNAX0702-E	N	Y	N	N	Y
KNAX0703-E	N	Y	N	Y	Y
KNAX0704-E to KNAX0708-E	N	Y	N	N	Y
KNAX0719-I	N	Y	N	N	N
KNAX0720-E to KNAX0723-E	N	Y	N	N	Y
KNAX0724-I	N	Y	N	N	N
KNAX0725-E	N	Y	N	N	Y
KNAX0726-I	N	Y	N	N	N
KNAX0727-E, KNAX0728-E	N	Y	N	N	Y
KNAX0800-E	N	Y	N	N	Y
KNAX0801-E	N	Y	N	N	N
KNAX0802-E	N	Y	N	N	Y
KNAX0803-E	N	Y	Y	Y	Y

Message IDs	Output destinations of messages				
	stdout	stderr	JOBLOG	System execution log	GUI
KNAX0804-E	N	Y	N	N	Y
KNAX0805-E	N	Y	Y	Y	Y
KNAX1600-I to KNAX1605-I	N	N	Y	N	N
KNAX1632-E	N	N	Y	N	Y
KNAX1871-E to KNAX1880-E	N	Y	N	N	N
KNAX1890-I	N	N	Y	N	N
KNAX1891-E to KNAX1892-E	N	N	Y	N	G
KNAX1893-W	N	N	Y	N	N
KNAX1910-E, KNAX1911-E	N	N	Y	Y	Y
KNAX2201-E to KNAX2205-E	N	Y	N	Y	Y
KNAX2206-E	N	Y	N	N	Y
KNAX2207-E	N	Y	N	Y	Y
KNAX2208-E to KNAX2213-E	N	Y	N	N	Y
KNAX2214-E, KNAX2400-E	N	Y	N	Y	Y
KNAX2499-E	N	Y	N	Y	N
KNAX3000-I	Y	Y#2, #3	N	Y	N
KNAX3001-I	N	Y#2, #3	N	Y	N
KNAX3002-E	Y	Y#2, #3	N	Y	N
KNAX3003-E	N	Y#2, #3	N	Y	N
KNAX3006-I	N	Y#2	N	Y	N
KNAX3008-W, KNAX3009-E	Y	Y#2	N	Y	N
KNAX3020-E to KNAX3029-E	N	Y#2	N	Y	N
KNAX3261-I	N	N	N	Y	N
KNAX3400-I to KNAX3542-W	N	Y#2	N	Y	N
KNAX3700-I to KNAX3799-I	Y	N	N	N	N
KNAX3998-E, KNAX3999-E	N	Y	N	N	N
KNAX4414-E to KNAX4429-E	N	Y#4	N	N	N
KNAX5300-I to KNAX5372-E	N	Y#5	N	Y	N
KNAX5380-I, KNAX5381-I	N	N	N	Y	N
KNAX5396-I to KNAX5399-E	N	Y	N	N	N
KNAX5407-E to KNAX5499-E	N	Y	N	N	N
KNAX6000-E to KNAX6071-E	N	N	Y	N	G

Message IDs	Output destinations of messages				
	stdout	stderr	JOBLOG	System execution log	GUI
KNAX6072-E	N	N	Y	N	N
KNAX6075-E to KNAX6099-E	N	N	Y	N	G
KNAX6100-E	N	Y	N	N	G
KNAX6110-I to KNAX6127-I	N	N	Y	Y	N
KNAX6130-E	N	Y	Y	Y	N
KNAX6200-I	N	Y	Y	Y	Y
KNAX6201-E	N	Y	N	N	N
KNAX6202-E to KNAX6208-E	N	Y	N	N	Y
KNAX6209-W	N	Y	N	N	N
KNAX6210-E to KNAX6215-E	N	Y	N	Y	N
KNAX6219-E	N	Y	N	Y	Y
KNAX6220-I to KNAX6222-I	N	Y	N	Y	N
KNAX6223-E to KNAX6241-E	N	Y	N	Y	Y
KNAX6242-I to KNAX6243-I	N	Y	N	Y	N
KNAX6244-E	N	Y	N	Y	Y
KNAX6290-E to KNAX6298-E	N	N	N	N	Y
KNAX6301-E to KNAX6303-E	N	Y	N	N	Y
KNAX6304-E	N	N	Y	N	Y
KNAX6305-E	N	Y	N	N	Y
KNAX6306-E	N	N	Y	N	Y
KNAX6307-W	N	N	Y	N	N
KNAX6308-E, KNAX6309-E	N	N	Y	N	Y
KNAX6310-E to KNAX6319-E	N	N	Y	N	G
KNAX6320-E	N	N	Y	N	Y
KNAX6321-E	N	N	Y	N	G
KNAX6323-E	N	Y	N	N	Y
KNAX6324-E to KNAX6330-E	N	N	Y	N	G
KNAX6332-E	N	N	Y	N	Y
KNAX6333-E	N	N	Y	N	G
KNAX6380-I	N	Y	N	N	N
KNAX6381-E	N	Y	N	N	Y
KNAX6382-I to KNAX6385-E	N	Y	N	N	N
KNAX6399-E, KNAX6400-E	N	N	Y	N	Y

Message IDs	Output destinations of messages				
	stdout	stderr	JOBLOG	System execution log	GUI
KNAX6401-E	N	N	Y	N	G
KNAX6403-E	N	N	Y	N	G
KNAX6404-E	N	N	Y	N	Y
KNAX6405-E to KNAX6407-E	N	N	Y	N	G
KNAX6408-E	N	N	Y	N	Y
KNAX6409-I, KNAX6410-I	N	N	Y	N	N
KNAX6411-E to KNAX6413-E	N	N	Y	N	Y
KNAX6414-E	N	N	Y	N	G
KNAX6507-I to KNAX6511-I	N	N	Y	Y	N
KNAX6512-I	N	Y	N	N	N
KNAX6521-E, KNAX6522-E	N	N	Y	Y	G
KNAX6530-E, KNAX6531-E	N	N	Y	N	G
KNAX6540-I	N	N	Y	Y	N
KNAX6541-E, KNAX6542-E	N	N	Y	Y	Y
KNAX6551-E to KNAX6586-E	N	N	Y	Y	N
KNAX6587-E	N	Y	N	Y	N
KNAX6588-E	N	Y	N	N	N
KNAX6589-W	N	N	N	Y	N
KNAX6590-E	N	N	Y	Y	Y
KNAX6591-E to KNAX6592-E	N	N	Y	Y	N
KNAX6593-E	N	Y	Y	Y	Y
KNAX6594-E	N	N	Y	Y	N
KNAX6596-E	N	Y	Y	Y	Y
KNAX6597-I	N	Y	Y	Y	N
KNAX6598-E, KNAX6599-E	N	N	Y	Y	Y
KNAX6600-E to KNAX6646-E	N	Y	N	Y	N
KNAX6701-W	N	N	Y	Y	N
KNAX6710-I	N	N	Y	N	N
KNAX6711-E, KNAX6712-E	N	N	Y	N	G
KNAX6713-E	N	N	Y	Y	Y
KNAX6714-E, KNAX6715-E	N	N	Y	N	G
KNAX6718-I	N	N	Y	N	N
KNAX6750-E to KNAX6753-E	N	Y	N	Y	N

Message IDs	Output destinations of messages				
	stdout	stderr	JOBLOG	System execution log	GUI
KNAX6800-I[#6], KNAX6801-I[#6]	N	N	N	N	N
KNAX6803-I to KNAX6806-I	N	N	Y	N	N
KNAX6810-E to KNAX6812-E	N	N	Y	Y	Y
KNAX6813-E	N	N	Y	N	G
KNAX6814-E, KNAX6815-E	N	N	Y	Y	Y
KNAX6830-I to KNAX6832-I	N	N	Y	Y	N
KNAX6997-E	N	N	Y	Y	Y
KNAX6998-E	N	N	Y	Y	G
KNAX6999-E	N	N	Y	Y	Y
KNAX7000-E to KNAX7004-E	N	Y	N	Y	Y
KNAX7006-W to KNAX7009-I	N	Y	N	Y	N
KNAX7010-E	N	Y	N	Y	Y
KNAX7011-I, KNAX7012-W	N	Y	N	Y	N
KNAX7013-E, KNAX7014-E	N	Y	N	Y	Y
KNAX7015-W	N	Y	N	Y	N
KNAX7016-E, KNAX7017-E	N	Y	N	Y	Y
KNAX7018-I	N	Y	N	Y	N
KNAX7019-E to KNAX7022-E	N	Y	N	Y	Y
KNAX7023-I	N	Y	N	Y	N
KNAX7024-E	N	Y	N	Y	Y
KNAX7025-I	N	Y	N	Y	N
KNAX7026-E to KNAX7029-E	N	Y	N	Y	Y
KNAX7032-I to KNAX7034-I	N	Y	N	Y	N
KNAX7035-E	N	Y	N	Y	Y
KNAX7036-I, KNAX7037-I	N	Y	N	Y	N
KNAX7038-I	N	Y	Y	Y	N
KNAX7039-E, KNAX7040-E	N	Y	N	Y	Y
KNAX7043-I	N	Y	N	Y	N
KNAX7044-E to KNAX7046-E	N	Y	N	Y	Y
KNAX7047-I, KNAX7048-I	N	Y	N	Y	N
KNAX7049-E to KNAX7052-E	N	Y	N	Y	Y
KNAX7053-I	N	Y	N	Y	N
KNAX7054-E, KNAX7055-E	N	Y	N	Y	Y

Message IDs	Output destinations of messages				
	stdout	stderr	JOBLOG	System execution log	GUI
KNAX7056-I, KNAX7057-I	N	Y	Y	Y	N
KNAX7058-I	N	Y	N	N	N
KNAX7062-E	N	Y	N	Y	Y
KNAX7063-I, KNAX7064-I	N	Y	Y	Y	N
KNAX7065-I to KNAX7067-I	N	Y	N	Y	N
KNAX7068-I	N	Y	Y	Y	N
KNAX7070-E	N	Y	N	Y	Y
KNAX7071-E, KNAX7072-E	N	Y	N	Y	N
KNAX7073-I	N	Y	Y	Y	N
KNAX7090-W	N	N	N	N	Y
KNAX7099-E, KNAX7101-E	N	Y	Y	Y	Y
KNAX7102-I, KNAX7103-I	N	N	N	Y	N
KNAX7104-E to KNAX7106-E	N	Y	Y	Y	Y
KNAX7107-I	N	N	N	Y	N
KNAX7108-E	N	Y	Y	Y	Y
KNAX7109-I	N	N	N	Y	N
KNAX7110-E	N	Y	Y	Y	Y
KNAX7111-I	N	N	N	Y	N
KNAX7112-E to KNAX7116-E	N	Y	Y	Y	Y
KNAX7117-I	N	N	N	Y	N
KNAX7118-E	N	N	N	N	Y
KNAX7119-E	N	Y	Y	Y	Y
KNAX7120-W	N	N	N	Y	N
KNAX7121-E to KNAX7125-E	N	Y	Y	Y	Y
KNAX7126-I, KNAX7127-E	N	Y	Y	Y	N
KNAX7128-E	N	Y	N	Y	N
KNAX7400-E to KNAX7402-E	N	N	Y	Y	Y
KNAX7403-E to KNAX7405-E	N	N	Y	Y	G
KNAX7408-E	N	N	Y	Y	Y
KNAX7420-E	N	Y	N	N	N
KNAX7450-I#7	N	N	N	N	N
KNAX7451-I	N	N	Y	Y	N
KNAX7460-E to KNAX7465-W	N	N	Y	Y#8	N

Message IDs	Output destinations of messages				
	stdout	stderr	JOBLOG	System execution log	GUI
KNAX7470-I	N	N	Y	Y	N
KNAX7500-I to KNAX7509-I#7	N	N	N	N	N
KNAX7550-I, KNAX7551-E#9	Y	N	N	N	N
KNAX7552-E to KNAX7556-E#9	N	N	N	N	N
KNAX7560-I, KNAX7561-E#9	Y	N	N	N	N
KNAX7600-E to KNAX7773-E	N	N	N	N	Y
KNAX7800-I to KNAX7880-E	Y	N	N	N	N
KNAX7892-I to KNAX7897-E	N	Y	N	N	N
KNAX7900-I	N	N	N	N	Y#10
KNAX7901-I	N	Y	Y#11	Y#11	N
KNAX7902-I	N	N	Y	Y	N
KNAX7999-I	N	Y	N	N	N
KNAX9000-E to KNAX9002-E	N	Y	N	N	N

Legend:

The following table describes the meaning of the columns under *Output destinations of messages* in the table above:

Column heading	Output destination of messages	Notes
stdout	Standard output	In the simple output mode and the minimum output mode, output of messages for jobs during normal execution and for child jobs during debugging are handled as follows: • Messages of message types I and W are not output. • Messages of message type E are output to the applicable output destinations and to JOBLOG. Messages of message type E for root jobs during debugging are also output to JOBLOG.
stderr	Standard error output	
JOBLOG	Job execution log	• In the expansion output mode During debugging, the same messages are also output to the standard error output during job execution. In the case of child jobs, these messages are not output to JOBLOG, but are output to the standard error output when job execution is completed. • In the simple output mode or the minimum output mode Messages of message type E are also output to the standard error output.
System execution log	System execution log	Depending on the job's status during message output, messages might not be output to this output destination.
GUI	Message dialog box or error window	None

Note that the following messages are also output in the simple output mode and the minimum output mode:

- KNAX0240-I (message indicating that the value of ADSH_JOBRC_FATAL has been applied)
- KNAX0300-I (usage)
- KNAX0309-I (displays version information)
- Messages that are output when a signal is received (however, in the minimum output mode during normal execution, part of the message that is output when a signal is received is suppressed)
- Messages other than those that are output to job execution logs during debugging (excluding KNAX0473-W)

G: GUI message that is output with a line number to a JP1/Advanced Shell Editor error window. If the line number is omitted from the message, the message is not output.

Y: Output.

N: Not output.

#1

This message is output only during CUI debugging.

#2

Output to the start log.

#3

The following is output to syslog:

- Facility: LOG_USER
- Level: LOG_NOTICE

#4

While the log file specified in the argument of the adshhk command is open, the message is output to the specified log file, not to the standard error output.

#5

The adapter command's output is displayed in the JP1/IM - View window.

#6

Output to the script image file.

#7

JP1 events are issued.

#8

Output while the adshecho or adshread command is executing.

#9

Output to the event log.

#10

Displayed in the web browser that is started when Help is selected in the GUI.

#11

Not output to this destination during CUI or GUI debug execution.

11.2.1 Notes about the row numbers that are output in messages

The following notes apply to the row numbers that are output in messages `KNAX6000-E` through `KNAX6100-E`, `KNAX6710-I` through `KNAX6712-E`, and `KNAX6998-E`.

- If a command error occurs in a command substitution spanning multiple lines, the last line number in the command substitution is displayed in the message as the erroneous line number.

  Example:

  If an error occurs in `unset` command in the following code, the erroneous line is shown as line number 3:

  ```
  1: `unset
  2:  echo pwd
  3:  `
  ```

- If an error occurs during syntax analysis of an external script, the name of the job definition script that called the external script is output in the error message as the job definition script file name. The line whose number is displayed is the line in the job definition script where the external script was called.
- If a syntax or command error occurs while the `trap` command's `action` is running, the line number of the `trap` command is displayed in the message as the erroneous line number.

  Example 1:

  This example spans multiple lines. The erroneous line is shown as line number 1.

  ```
  1: trap 'pwd
  2: unset
  3: date' INT
  ```

  Example 2:

  This example calls a function. The erroneous line is shown as line number 4.

  ```
  1: func1() {
  2:  unset
  3: }
  4: trap func1 INT
  ```

11.3 List of messages

This section explains the messages that are issued by JP1/Advanced Shell and how to handle them.

KNAX0001-E

Memory is insufficient. details=*maintenance-information*

A memory shortage occurred.

maintenance-information, which is displayed as eight hexadecimal characters, indicates the system's internal status. This message is output to the error notification destinations. The message is output to only some of the destinations, depending on the timing of the error occurrence.

(S)

Terminates processing.

(O)

Contact the system administrator. The system administrator must check and, if necessary, revise the memory estimation.

KNAX0004-I

Job ID=*Advanced Shell-job-ID*, JP1NBQSQueueName=*environment-variable-value*, scheduler job ID=*scheduler-job-number*

This message displays the JP1/AJS job information and the JP1/Advanced Shell job ID for the batch job that has been started.

Advanced Shell-job-ID

Job ID assigned to the batch job by JP1/Advanced Shell

environment-variable-value

Value of the `JP1NBQSQueueName` environment variable in the batch job

scheduler-job-number

JP1 job number assigned by JP1/Advanced Shell in the batch job

(S)

Resumes processing.

KNAX0030-E (Windows only)

An error occurred while starting adshexec. function="*function-name*",error code=*error-code*,reason="*error-details*"

An error occurred during job controller start processing. The message displays *function-name*, *error-code*, and *error-details*.

(S)

Terminates processing.

(O)

Contact the system administrator. The system administrator must eliminate the cause of the error based on the displayed *function-name*, *error-code*, and *error-details*, and then re-execute the batch job.

KNAX0031-E (Windows only)

An error occurred while completing adshexec process. function="*function-name*",error code=*error-code*,reason="*error-details*"

An error occurred during job controller termination processing. The message displays *function-name*, *error-code*, and *error-details*.

(S)

Terminates processing.

(O)

Contact the system administrator. The system administrator must eliminate the cause of the error based on the displayed *function-name*, *error-code*, and *error-details*, and then re-execute the batch job.

KNAX0091-I

job-name The job started.

The batch job indicated by *job-name* has started.

(S)

Resumes processing.

KNAX0092-I

job-name.job-step-name step started.

The job step indicated by *job-step-name* that is defined in the batch job indicated by *job-name* has started.

(S)

Resumes processing.

KNAX0098-I

job-name The job ended. exit status=*exit-status-code* execution time=*execution-time-in-seconds* CPU time=*CPU-time-in-seconds*

The batch job indicated by *job-name* has ended.

exit-status-code

Return code indicating the batch job execution results.

For details about the return code, see the description of the `adshexec` command's return codes.

If an error occurs during `adshexec` command postprocessing after this message was output, the return code displayed in this message might not be the `adshexec` command's return code. The `adshexec` command's final return code is output to `KNAX7999-I`.

execution-time-in-seconds

Total amount of time (in seconds) required for execution, from the beginning to the end of the batch job. This is a reference value obtained by using the OS's API.

CPU-time-in-seconds

Total amount of CPU time (in seconds) used, from the beginning to the end of the batch job. This is a reference value obtained by using the OS's API.

(S)

Resumes processing.

KNAX0101-E

job-name An error occurred during execution of the job.

An error occurring while the batch job indicated by *job-name* was running.

(S)

Resumes processing.

(O)

See the other messages output together with this message, eliminate the cause of the error, and then re-execute the batch job.

KNAX0220-E

The environment variable "*environment-variable-name*" is not specified.

The environment variable indicated by *environment-variable-name* was not specified.

If *environment-variable-name* is `JP1_HOSTNAME` and the logical host name for a logical host operation was not specified explicitly, the job might have been started from a system other than JP1/AJS.

(S)

Terminates processing.

(O)

Contact the system administrator, and then re-evaluate the value set for the environment variable.

KNAX0235-E

The value specified for the environment variable "*environment-variable-name*" is invalid.

The value for the environment variable indicated by *environment-variable-name* is invalid.

environment-variable-name

Name of an environment variable

(S)

Terminates processing.

(O)

Check and, if necessary, revise the value of the environment variable. If the problem cannot be resolved, contact the system administrator.

KNAX0236-E

The value specified for the environment variable "*environment-variable-name*" is too long.

The value set for the environment variable indicated by *environment-variable-name* is too long.

(S)

Terminates processing.

(O)

Re-evaluate the value set for the environment variable. If the problem cannot be resolved, contact the system administrator.

KNAX0237-E

> The value specified for the environment variable "*environment-variable-name*" is an empty string.

The value specified for the environment variable *environment-variable-name* is an empty string.

(S)

Terminates processing.

(O)

Contact the system administrator, and then re-evaluate the value set for the environment variable.

KNAX0238-E

> The value specified for the environment variable "*environment-variable-name*" contains an invalid character.

The value set for the environment variable indicated by *environment-variable-name* contains an invalid character.

(S)

Terminates processing.

(O)

Re-evaluate the value set for the environment variable. If the problem cannot be resolved, contact the system administrator.

KNAX0239-E

> The value specified for the environment variable "*environment-variable-name*" is out of range.

The value set for the environment variable indicated by *environment-variable-name* is outside the permitted range.

(S)

Terminates processing.

(O)

Contact the system administrator, and then re-evaluate the value set for the environment variable.

KNAX0240-I

> The setting specified for the environment variable *environment-variable-name* was applied. value=*environment-variable-value*

The setting of the environment variable indicated by *environment-variable-name* was applied.

environment-variable-value

Terminates processing.

(S)

Resumes processing.

KNAX0299-E

An internal error occurred. details=*maintenance-information*

An internal conflict occurred during memory allocation.

This message is output to the error notification destinations. The message is output to only some of the destinations, depending on the timing of the error occurrence.

(S)

Terminates processing.

(O)

Contact the system administrator.

KNAX0300-I

Usage: *command-name command-argument*

The command indicated by *command-name* and *command-argument* is invalid.

(S)

Terminates processing.

(O)

Specify the correct command and then execute it.

KNAX0301-E

No value is specified for the option "*option-name*".

The value specified for *option-name* is invalid.

(S)

Cancels processing.

(O)

Specify the correct option.

KNAX0302-E

The option "*option-name*" is not a valid option.

The specified *option-name* is invalid.

(S)

Cancels processing.

(O)

Specify the correct option name.

KNAX0303-E

No script file name is specified.

No job definition script file name was specified.

(S)

Cancels processing.

(O)

Specify a job definition script file name, and then re-enter the command.

KNAX0305-E

The specified argument "*argument*" is invalid.

The command argument indicated by *argument* is invalid.

(S)

Cancels processing.

(O)

Specify the argument correctly, and then re-enter the command.

KNAX0306-E

The value specified for the option "*option-name*" is invalid.

The value of the option indicated by *option-name* is invalid.

(S)

Cancels processing.

(O)

Specify the correct option, and then re-enter the command.

KNAX0307-E

A required option is not specified.

A required option is missing.

(S)

Terminates processing.

(O)

Specify the required option, and then re-enter the command.

KNAX0308-E

The options "*option-name-1*" and "*option-name-2*" cannot be specified at the same time.

The options indicated by *option-name-1* and *option-name-2* are mutually exclusive.

(S)

Cancels processing.

(O)

Specify the correct option, and then re-enter the command.

KNAX0309-I

The version of *program-name* is *version*.

The version of the command indicated by *program-name* is displayed in *version*.

(S)

Terminates processing.

KNAX0310-E

Too many operands are specified.

Too many operands are specified.

(S)

Cancels processing.

(O)

Specify the operand correctly.

KNAX0311-E

One or more options or parameters required for *command-name* are not specified.

One or more options or parameters required for processing of the command indicated by *command-name* are missing.

(S)

Terminates processing.

(O)

Specify the required options or parameters, and then re-enter the command.

KNAX0336-E

The length of the value specified for the option "*option-name*" is invalid.

The size of the specified option name is not valid. Possible causes are as follows:

- The specified option name is too long.
- The size of the specified option name is 0.

(S)

Cancels processing.

(O)

Specify the correct option, and then re-enter the command.

KNAX0401-E

Failed to open the environment file. reason="*error-details*"

An open error occurred on the environment file for the reason indicated by *error-details*.

(S)

Terminates processing.

(O)

Based on the displayed error details, determine the cause of the error (such as permissions), and then correct the error so that the environment file can be imported. If the problem cannot be resolved, contact the system administrator.

KNAX0402-E

Failed to read the environment file. reason="*error-details*"

A read error occurred on the environment file for the reason indicated by *error-details*.

(S)

Terminates processing.

(O)

Based on the displayed error details, determine the cause of the error (such as permissions), and then correct the error so that the environment file can be imported. If the problem cannot be resolved, contact the system administrator.

KNAX0403-E

The environment file name is too long.

The file name of the environment file is too long.

(S)

Terminates processing.

(O)

Check the specified environment file name for any error.

KNAX0406-E

An error occurred during collection of the host name. reason="*error-details*"

A host name acquisition error occurred for the reason indicated by *error-details*. In UNIX, this message might be displayed when the length of the host name exceeds 255 characters.

(S)

Terminates processing.

(O)

Contact the system administrator to check the host name in the network.

KNAX0407-E

The file "*file-name"* is not a regular file.

The file indicated by *file-name* is not a regular file.

(S)

Terminates processing.

(O)

Check the file.

KNAX0410-E

An error occurred when parsing the environment file "*file-name*". For details, see the message output before this one.

A parsing error occurred in the environment file indicated by *file-name*. For details about the error, see the message output before this message.

(S)

Terminates processing.

(O)

Correct the error in the environment file.

KNAX0411-E

Line size exceeds limits. line=*line-number*

The line indicated by *line-number* in the environment file is too long.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the environment file.

KNAX0431-E

A parameter name is invalid. line=*line-number*

An invalid parameter name was found on the line indicated by *line-number* in the environment file.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the environment file.

KNAX0432-E

The value specified for the parameter "*parameter-name*" is invalid. line=*line-number*

An invalid value was found in the parameter indicated by *line-number* in the environment file.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the environment file.

KNAX0433-E

No value is specified for the parameter "*parameter-name*". line=*line-number*

No value was specified in the parameter indicated by *line-number* in the environment file.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the environment file.

KNAX0434-E

The parameter "*parameter-name*" is specified multiple times. line=*line-number*

The parameter indicated by *line-number* is duplicated in the environment file.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the environment file.

KNAX0435-E

The number of "*parameter-name*" parameters exceeds the limit. line=*line-number*

The number of times the indicated parameter on the line indicated by *line-number* was specified in the environment file exceeds the maximum value.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the environment file.

KNAX0436-E

The value specified for the parameter "*parameter-name*" is too long. line=*line-number*

The value of the indicated parameter on the line indicated by *line-number* in the environment file is too long.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the environment file.

KNAX0437-E

The value specified for the parameter "*parameter-name*" is out of range. line=*line-number*

The value of the indicated parameter on the line indicated by *line-number* in the environment file is outside the permitted range.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the environment file.

KNAX0438-E

The value specified for the parameter "*parameter-name*" contains an invalid character. line=*line-number*

The value of the indicated parameter on the line indicated by *line-number* in the environment file contains an invalid character.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the environment file.

KNAX0439-E

The path specified for the parameter "*parameter-name*" is not an absolute path. line=*line-number*

The file path indicated by *parameter-name* on the indicated *line-number* in the environment file is not an absolute path.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the environment file.

KNAX0441-E

The directory specified for the parameter "*parameter-name*" does not exist. line=*line-number*

The directory specified in the indicated parameter on the line indicated by *line-number* in the environment file does not exist.

In UNIX, this message might be issued due to receipt of a signal.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the environment file, or check the operating environment.

KNAX0442-E

The value specified for the parameter "*parameter-name*" is not a directory. line=*line-number*

The value specified in the indicated parameter on the line indicated by *line-number* in the environment file is not a directory.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the environment file.

KNAX0444-E

Too many operands are specified for the parameter "*parameter-name*". line=*line-number*

Too many operands were specified in the indicated parameter on the line indicated by *line-number* in the environment file.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the environment file.

KNAX0445-E

The default directory specified for the parameter "*parameter-name*" does not exist. directory="*default-directory-name*"

There is no default directory.

In UNIX, this message might be issued due to receipt of a signal.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the operating environment.

KNAX0446-E

The default directory specified for the parameter "*parameter-name*" is not a directory. directory="*default-directory-name*"

The default directory name is not a directory.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the operating environment.

KNAX0449-E

The required directory "*directory-name*" does not exist.

The required directory indicated by *directory-name* is missing.

In UNIX, this message might be issued due to receipt of a signal.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the environment file or the operating environment.

KNAX0450-E

The required directory "*directory-name*" is not a directory.

The required directory indicated by *directory-name* is not a directory.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the environment file or the operating environment.

KNAX0451-E (Windows only)

An error occurred during a request for the default directory. parameter name="*parameter-name*"

An error occurred while obtaining the default directory name.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the operating environment.

KNAX0456-E (Windows only)

A value is specified multiple times for the parameter "*parameter-name*". line=*line-number*

The value specified as *parameter-name* is duplicated.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the environment file.

KNAX0458-E

The combination of parameters is invalid. line=*line-number*

The combination of parameters is invalid. Possible causes are as follows:

- The combination of `phost_start` and `phost_end` parameters is invalid.
- The combination of `lhost_start` and `lhost_end` parameters is invalid.
- The parameter specification order is invalid.
- A start parameter is specified, but no end parameter is specified.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the environment file.

KNAX0459-E

The operands are specified in the wrong order, or the same operand is specified multiple times. parameter="*parameter-name*" line=*line-number*

The order in which the operands are specified is not correct or the same operand is specified more than once.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the environment file.

KNAX0471-E

The value specified for the parameter *parameter-name* in "*file-name*" is different from that specified in the system environment file.

The value specified for the *parameter-name* parameter in the *file-name* environment file differs from the value specified in the system environment file.

If the default value is specified in the system environment file, specifying a different value in the job environment file results in an error. The system execution log and trace settings specified in the system environment file cannot be changed.

If the value is not specified explicitly in the job environment file, no error results.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the environment file.

KNAX0472-E

An unexpected error occurred. (function="*function-name*", error details="*error-details*", maintenance information=*maintenance-information*)

An unexpected error occurred during environment file analysis processing.

function-name

Internal function name

error-details

Character string indicating the nature of the error

maintenance-information

Maintenance code

(S)

Terminates processing.

(O)

Correct the error, and then re-execute the command. If the error cannot be corrected, contact the system administrator.

KNAX0473-W

The parameter "*parameter-name*" is redundantly specified in *environment-file-type*.

A redundant parameter is specified in the file indicated by *environment-file-type*.

For details about the problems that might result, see the explanation for the particular parameter.

We recommend that you do not specify this parameter in the indicated type of environment file.

environment-file-type

Job environment file

(S)

Executes the processing.

(O)

If the problem needs to be eliminated, delete the corresponding parameter.

KNAX0474-E

The value "*parameter-value*" for the parameter "*parameter-name*" cannot be specified for the current execution method of a batch job. filename="*file-name*"

The specified value is not permitted for the current job start method that is set in the environment file indicated by *file-name*.

parameter-value

Parameter value that is not permitted

parameter-name

Name of the parameter resulting in the error

file-name

Environment file containing the parameter that resulted in the error

(S)

Terminates processing.

(O)

Check and, if necessary, revise the parameter specified in the environment file.

KNAX0700-E

A spool job directory could not be created. reason=*error-details*

A directory for the batch job cannot be created under the spool root directory.

(S)

Terminates processing.

(O)

Eliminate the cause indicated by *error-details*, and then re-execute the command. If the problem cannot be resolved, contact the system administrator. The system administrator must check and, if necessary, revise the spool root directory specified in the environment file or the spool directory itself.

KNAX0701-E

Failed to open the file "*file-name*". reason=*error-details*

An open error occurred on the file indicated by *file-name* in the spool job directory. Alternatively, an open error occurred in the output console.

In the case of the output console, `CONOUT$` is displayed for *file-name*.

(S)

Terminates processing.

(O)

Eliminate the cause indicated by *error-details*, and then re-execute the command. If the problem cannot be resolved, contact the system administrator. The system administrator must check and, if necessary, revise the spool job directory or files in the spool job directory.

KNAX0702-E

An I/O error occurred during an attempt to write to JOBLOG file. reason=*error-details*

A write error occurred in the job execution log file in the spool job directory.

(S)

Terminates processing.

(O)

Eliminate the cause indicated by *error-details*, and then re-execute the command. If the problem cannot be resolved, contact the system administrator. The system administrator must check and, if necessary, revise the spool job directory or files in the spool job directory.

KNAX0703-E

The file "*file-name*" does not exist.

The file indicated by *file-name* does not exist in the spool job directory.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the file name.

KNAX0704-E

Failed to get the date.

Acquisition of the date failed.

(S)

Terminates processing.

(O)

Contact the system administrator.

KNAX0706-E

Failed to create the file path "*file-path*". reason=*error-details*

Creation of *file-path* has failed.

(S)

Terminates processing.

(O)

Eliminate the cause indicated by *error-details*, and then re-execute the command. If the problem cannot be resolved, contact the system administrator. The system administrator must check and, if necessary, revise the spool job directory or files in the spool job directory.

KNAX0708-E

The number of JOBLOG files exceeded the limit.

The number of `JOBLOG` files allocated exceeded the limit. Possible causes are as follows:

- A large number of spool jobs remain in the spool directory, leaving few available job IDs.
- A large number of child jobs have been started in a job.

(S)

Terminates processing.

(O)

Delete unneeded spool jobs, and then re-execute the job. Or, check the job definition script to make sure that an unnecessarily large number of child jobs have not been started.

KNAX0719-I

STEP. step number=*step-number* step name=*step-name* output destination=*output-destination*

Following this message, *information-about-output-destination* for the job step is displayed.

(S)

Resumes processing.

KNAX0720-E

Failed to open the job ID file. reason=*error-details*

An open error occurred in the job ID file.

(S)

Terminates processing.

(O)

Eliminate the cause indicated by *error-details*, and then re-execute the command. If the problem cannot be resolved, contact the system administrator. The system administrator must check and, if necessary, revise the spool directory specified in the environment file, the spool directory itself, or files in the spool directory.

KNAX0721-E

An I/O error related to the job ID file occurred. reason=*error-details*

An input/output error occurred on the job ID file.

(S)

Terminates processing.

(O)

Eliminate the cause indicated by *error-details*, and then re-execute the command. If the problem cannot be resolved, contact the system administrator. The system administrator must check and, if necessary, revise the spool directory specified in the environment file, the spool directory itself, or files in the spool directory.

KNAX0722-E

Failed to allocate a job ID.

Assignment of a job ID failed. This message might be displayed because no more spool job directories can be created.

(S)

Terminates processing.

(O)

Contact the system administrator. The system administrator must eliminate the cause of the error based on the messages output before and after this message, and then re-execute the command. Alternatively, delete an unneeded spool job directory, and then re-execute the command.

KNAX0723-E

Failed to lock the job ID file. reason=*error-details*

An attempt to lock the job ID file failed.

If an NFS directory is specified in the `SPOOL_DIR` parameter, this message might be displayed because an error resulted. Do not specify an NFS directory in the `SPOOL_DIR` parameter.

(S)

Terminates processing.

(O)

Eliminate the cause indicated by *error-details*, and then re-execute the command. If the problem cannot be resolved, contact the system administrator. The system administrator must check and, if necessary, revise the spool directory specified in the environment file, the spool directory itself, or files in the spool directory.

KNAX0724-I

The job ID was assigned. job ID=*job-ID*

The indicated *job-ID* was assigned.

This message is not output for a child job that was started with `MERGE` specified as the operand in the `SPOOLJOB_CHILDJOB` parameter of the root job's environment file.

(S)

Resumes processing.

KNAX0725-E

An API error occurred. (API="*API-name*", reason="*cause*", maintenance information="*maintenance-information*")

An error occurred in the API.

(S)

Terminates processing.

(O)

Eliminate the cause indicated by *error-details*, and then re-execute the command. If the problem cannot be resolved, contact the system administrator.

KNAX0726-I

The child job ID was assigned. job ID=*job-ID*

A job ID has been assigned for a child job.

(S)

Resumes processing.

KNAX0727-E

Failed to lock the file that manages the start order of child jobs. reason=*error-details*

An attempt to lock the file that manages the start order of child jobs failed.

(S)

Terminates processing.

(O)

Eliminate the cause indicated by *error-details*, and then re-execute the command. If the problem cannot be resolved, contact the system administrator. The system administrator must check and, if necessary, revise the spool job directory or files in the spool job directory.

KNAX0728-E

The format of the file that manages the start order of child jobs is invalid.

The format of the file that manages the start order of child jobs is not valid.

(S)

Terminates processing.

(O)

Check whether the file that manages the start order of child jobs has been updated illegally. If the problem cannot be resolved, contact the system administrator.

The system administrator must check whether there is a problem in the spool job directory or with the files in the spool job directory.

KNAX0800-E

Failed to create the {.sysout|sysout.ini} file. reason=*error-details*

An attempt to create a spool job management file failed.

(S)

Terminates processing.

(O)

Eliminate the cause indicated by *error-details*, and then re-execute the command. If the problem cannot be resolved, contact the system administrator. The system administrator must check and, if necessary, revise the spool job directory or files in the spool job directory.

KNAX0801-E

Failed to lock the .sysout file. reason=*error-details*

An attempt to lock the spool job management file failed.

If an NFS directory is specified in the `SPOOL_DIR` parameter, this message might be displayed because an error resulted. Do not specify an NFS directory in the `SPOOL_DIR` parameter.

(S)

Terminates processing.

(O)

Eliminate the cause indicated by *error-details*, and then re-execute the command. If the problem cannot be resolved, contact the system administrator. The system administrator must check and, if necessary, revise the spool job directory or files in the spool job directory.

KNAX0802-E

Failed to open the {.sysout|sysout.ini} file. reason=*error-details*

An attempt to open the spool job management file failed.

(S)

Terminates processing.

(O)

Eliminate the cause indicated by *error-details*, and then re-execute the command. If the problem cannot be resolved, contact the system administrator. The system administrator must check and, if necessary, revise the spool job directory or files in the spool job directory.

KNAX0803-E

An I/O error related to the {.sysout|sysout.ini} file occurred. reason=*error-details*

An input/output error occurred in the spool job management file.

This message is output to the error notification destinations. The message might be output to only some of the destinations, depending on the timing of the error occurrence.

(S)

Terminates processing.

(O)

Eliminate the cause indicated by *error-details*, and then re-execute the command. If the problem cannot be resolved, contact the system administrator. The system administrator must check and, if necessary, revise the spool job directory or files in the spool job directory.

KNAX0804-E

Failed to get the current time to use for STARTTIME in the {.sysout|sysout.ini} file.

Acquisition of the current time used for `STARTTIME` in the spool job management file failed.

(S)

Resumes processing.

(O)

Contact the system administrator.

KNAX0805-E

Failed to get the current time to use for ENDTIME in the {.sysout|sysout.ini} file.

Acquisition of the current time used for `ENDTIME` in the spool job management file failed.

(S)

Resumes processing.

(O)

Contact the system administrator.

KNAX1600-I

job-name Allocation of file(s) for a job started.

The file for the batch job indicated by *job-name* has been allocated.

(S)

Resumes processing.

KNAX1601-I

job-name.job-step-name Allocation of file(s) for a step started.

The file for the job step indicated by *job-name* and *job-step-name* has been allocated.

(S)

Resumes processing.

KNAX1604-I

The file *file-path* was deleted.

The file indicated by *file-path* was deleted based on the value specified for postprocessing.

(S)

Resumes processing.

KNAX1605-I

Deletion of the file *file-path* failed. reason="*error-details*"

An attempt was made to delete *file-path* based on the value specified for postprocessing, but the error indicated by *error-details* occurred.

error-details

Details of the error. This is an error information string representation for `errno`.

(S)

Resumes processing.

KNAX1632-E

The value specified for the environment variable *environment-variable-name* is too long.

An attempt was made to set the environment variable whose name is indicated by *environment-variable-name*, but the specified value exceeded the maximum permissible length in JP1/Advanced Shell.

(S)

Terminates processing.

(O)

Correct the environment variable value specified in the environment file, and then re-execute the batch job.

KNAX1871-E

command-name: Failed to normalize the file name. (function="*function-name*", reason="*cause*", maintenance information="*maintenance-information*")

An error occurred while the command specified by *command-name* was normalizing the specified file path by converting it to an absolute path.

If *function-name* is `_fullpath`, this message might be issued when the path name specified by the command is too long (in this case, `Invalid argument` might be indicated for *cause*).

cause

The following table explains the causes that can be displayed and their meanings:

Cause	Meaning
`The path contains one or more invalid multibyte characters`	(UNIX only) The path name contains an invalid multibyte character. Correct the file path specified in the command.

Cause	Meaning
`The path contains too many components`	(UNIX only) The path name contains more than 4,096 components. Correct the file path specified in the command.
`File name too long`	(UNIX only) After being converted to an absolute path, the size of the file path name exceeded the maximum permissible length. Correct the file path specification.
`error cause determined from errno`	• A memory shortage occurred. • An error occurred while an API was being executed by the OS (error in `getcwd` or `_fullpath`). • (Windows only) If *function-name* is `_fullpath` and *cause* is `Invalid argument`, the size of the file path name after conversion to an absolute path might have exceeded the maximum permissible length. In this case, correct the file path specified in the command.

maintenance-information

Internal information

(S)

Terminates processing.

(O)

Eliminate the cause of the error based on *function-name* and *cause*, and then re-execute the command. If the cause of the error cannot be eliminated, contact the system administrator.

KNAX1872-E

command-name: The file path is invalid. (file path="*file-path*", reason="*cause*", maintenance information="*maintenance-information*")

An error occurred while the file path specified by *file-path* was being checked during the processing of the command indicated by *command-name*. Make sure that the file path specified in the command is usable.

cause

Cause of the error reported by the system

maintenance-information

Internal information

(S)

Terminates processing.

(O)

Eliminate the cause of the error, and then re-execute the command.

If the cause of the error cannot be eliminated, contact the system administrator.

KNAX1873-E

command-name: The specified file path is a directory. (file path="*file-path*", maintenance information="*maintenance-information*")

The *file path* indicated by file-path specified by the command indicated by *command-name* is a directory. Specify a normal file for the file path.

maintenance-information

Internal information

(S)

Terminates processing.

(O)

Eliminate the cause of the error, and then re-execute the command.

KNAX1875-E

command-name: An error occurred. (reason="*cause*", maintenance information="*maintenance-information*")

An unexpected error occurred while the command indicated by *command-name* was being processed.

cause

The following table explains the causes that can be displayed and their meanings:

Cause	Meaning
`The size of the buffer is insufficient for storing the path`	(UNIX only) A shortage occurred in the buffer for storing path names.
`An error occurred in the _time64 function`	(Windows only) An error occurred in the process for determining the time.
`An error occurred in the clock_gettime function`	(UNIX only) An error occurred in the process for determining the time.
`An error occurred in the nanosleep function`	(UNIX only) An error occurred in the suspension process.
`Invalid argument`	An invalid argument was specified.

maintenance-information

Internal information

(S)

Terminates processing.

(O)

Contact the system administrator.

KNAX1877-E

command-name: An environment variable is invalid. (environment variable="*environment-variable-name*", maintenance information="*maintenance-information*")

The environment variable indicated by *environment-variable-name* does not exist. Alternatively, the size of the character string in the specified value is 0 bytes or less.

maintenance-information

Internal information

(S)

Terminates processing.

(O)

Check the job definition script to determine whether the applicable environment variable has been changed. Or, check whether the command was started in a program other than JP1/Advanced Shell.

If the cause cannot be determined, contact the system administrator.

KNAX1878-E

command-name: An I/O error occurred. (file path="*file-path*", reason="*cause*", maintenance information="*maintenance-information*")

An I/O error occurred in the allocation management file.

file-path

File path name of the allocation management file

cause

The following table explains the causes that can be displayed and their meanings:

Cause	Meaning
`The file is not a regular file`	The file is not a regular file.
`The file was replaced`	The file was replaced while it was being opened.
api-name `error :` *error-details*	The error indicated by *error-details* occurred during the processing of the API indicated by *api-name*. *error-details* is the error information that is set by the API.

maintenance-information

Internal information

(S)

Terminates processing.

(O)

Eliminate the cause of the error, and then re-execute the command.

If the cause of the error cannot be determined, contact the system administrator.

KNAX1879-E

command-name: The number of files exceeded the limit. (maintenance information="*maintenance-information*")

The maximum permissible number of files that can be registered, which is 64, was exceeded.

maintenance-information

Internal information

(S)

Terminates processing.

(O)

Modify the job definition script so that the number of files to be registered does not exceed the maximum.

KNAX1880-E

command-name: The command cannot be executed with the current setting of the environment setting parameter. parameter="*environment-setting-parameter*"

The indicated command cannot be used with the current setting of the indicated environment setting parameter.

command-name

Command name

environment-setting-parameter

Environment setting parameter and value that caused this error

(S)

Terminates processing.

(O)

Do not execute this command. If you want to execute this command, check and revise the specified environment setting parameter as necessary.

KNAX1890-I

The file was deallocated as "*processing-value*". path=*file-path*

The file indicated by *file-path* was released in accordance with the processing indicated by *processing-value*.

processing-value

One of the following values is displayed as the specified postprocessing for files during normal times or abnormal times:

- `del`: Delete.
- `keep`: Do not delete.

(S)

Resumes processing.

KNAX1891-E

An I/O error occurred during deallocation of the file. (file path="*file-path*", reason="*cause*", maintenance information="*maintenance-information*")

An I/O error occurred in the allocation management file during postprocessing of the file.

file-path

File path name of the allocation management file

cause

The following table explains the causes that can be displayed and their meanings:

Cause	Meaning
`The file is not a regular file`	The file is not a regular file.
`The file was replaced`	The file was replaced while it was being opened.
api-name `error :` *error-details*	The error indicated by *error-details* occurred during the processing of the API indicated by *api-name*. *error-details* is the error information that is set by the API.

maintenance-information

Internal information

(S)

Terminates the job.

(O)

Contact the system administrator.

KNAX1892-E

An error occurred during deallocation of the file. (reason="*cause*", maintenance information="*maintenance-information*")

An unexpected error occurred.

cause

The following table explains the causes that can be displayed and their meanings:

Cause	Meaning
`An error occurred in the _time64 function`	(Windows only) An error occurred in the process for determining the time.
`An error occurred in the clock_gettime function`	(UNIX only) An error occurred in the process for determining the time.
`An error occurred in the nanosleep function`	An error occurred in the suspension process.

maintenance-information

Internal information

(S)

Terminates the job.

(O)

Contact the system administrator.

KNAX1893-W

An invalid entry in the file was skipped. (file path="*file-path*", maintenance information="*maintenance-information*")

The postprocessing of the `adshfile` command was skipped because an invalid entry existed in the allocation management file. The postprocessing of the file specified by the `adshfile` command of the corresponding job might not have been executed.

file-path

File path name of the allocation management file

maintenance-information

Internal information

(S)

Resumes processing.

(O)

If files specified to be deleted by the `adshfile` command remain, delete them manually if necessary.

KNAX1910-E

The CPU time is invalid.

The calculation result of `C-Time` is invalid.

(S)

Resumes the batch job using `0` as the time value for output to job execution logs.

(O)

Contact the system administrator.

KNAX1911-E

The execution time is invalid.

The calculation result of `E-Time` is invalid.

(S)

Resumes the batch job using `0` as the time value for output to job execution logs.

(O)

Contact the system administrator.

KNAX2201-E

The message is too long. message number=*message-number*

Some of the text could not be output to the message indicated by *message-number* (number following `KNAX`) because the text was too long.

(S)

Resumes processing.

(O)

Check the batch job processing for any problem. If necessary, correct the job definition script.

KNAX2202-E

Output to JOBLOG failed. message ID=*message-ID*

Output to the job execution log of the message indicated by *message-ID* failed.

(S)

Resumes processing.

(O)

Contact the system administrator. Check and, if necessary, correct the access permissions and disk status of the spool job directory.

KNAX2204-E

Output to stdout failed. message ID=*message-ID*

Output to the standard output of the message indicated by *message-ID* failed.

(S)

Resumes processing.

(O)

Contact the system administrator. Check and, if necessary, correct the access permissions and disk status of the standard output.

KNAX2205-E

Output to stderr failed. message ID=*message-ID*

Output to the standard error output of the message indicated by *message-ID* failed.

(S)

Resumes processing.

(O)

Contact the system administrator. Check and, if necessary, correct the access permissions and disk status of the standard error output.

KNAX2206-E

The system execution log output failed. message ID=*message-ID*

Output to the system execution log of the message indicated by *message-ID* failed.

(S)

Resumes processing.

(O)

Contact the system administrator. Check and, if necessary, correct the specification of system execution logs in the environment file or the access permissions and disk status of the system execution log directory.

KNAX2207-E

Trace output failed. message ID=*message-ID*

Output to the trace log of the message indicated by *message-ID* failed.

(S)

Resumes processing.

(O)

Contact the system administrator. Check and, if necessary, correct the specification of traces in the environment file or the access permissions and disk status of the trace directory.

KNAX2208-E

System execution log initialization failed. code=*maintenance-information*, reason=*error-cause*

Initialization of the system execution log failed.

If an NFS directory is specified in the `LOG_DIR` parameter, this message might be displayed because an error resulted. Do not specify an NFS directory in the `LOG_DIR` parameter.

(S)

Terminates processing.

(O)

Contact the system administrator. Check and, if necessary, correct the specification of system execution logs in the environment file or the access permissions and disk status of the system execution log directory.

KNAX2209-E

Trace initialization failed. code=*maintenance-information*

Initialization of the trace log failed.

If an NFS directory is specified in the `TRACE_DIR` parameter, this message might be displayed because an error resulted. Do not specify an NFS directory in the `TRACE_DIR` parameter.

(S)

Terminates processing.

(O)

Contact the system administrator. Check and, if necessary, correct the specification of traces in the environment file or the access permissions and disk status of the trace directory.

KNAX2211-E

Trace settings for the file size or the number of files used are invalid.

The trace settings are invalid.

(S)

Terminates processing.

(O)

Contact the system administrator. Check and, if necessary, revise the specification of traces in the environment file.

KNAX2213-E

The location specified for trace output is invalid.

The specified trace output destination is invalid.

(S)

Terminates processing.

(O)

Contact the system administrator. Check and, if necessary, correct the specification of traces in the environment file or the access permissions and disk status of the trace directory.

KNAX2214-E

Failed to get the current time.

Acquisition of the current time failed.

(S)

Terminates processing.

(O)

Contact the system administrator.

KNAX2400-E

Output failed because initialization has not been completed. output location=*output-location*, message ID=*message-ID*, destination=*maintenance-information-1*, set destination=*maintenance-information-2*

The message indicated by *message-ID* cannot be output because initialization has not been completed.

(S)

Resumes processing.

(O)

Contact the system administrator.

KNAX2499-E

Message number "*message-number*" is not defined.

The message indicated by *message-number* does not exist.

(S)

Terminates processing.

(O)

Contact the system administrator.

KNAX3000-I

adshmd started.

The user-reply functionality's management daemon has started.

(S)

Resumes processing.

KNAX3001-I

adshmd stopped.

The user-reply functionality's management daemon has stopped.

(S)

Terminates processing.

KNAX3002-E

An attempt to start adshmd failed.[*detailed-message*]

An attempt to start the user-reply functionality's management daemon failed.

This message is output to the applicable error notification destinations. The message might be output to only some of the destinations, depending on the timing of the error occurrence. In the case of output to `syslog`, the message text for a message related to the user-reply functionality's management daemon (message number from `3000` through `3999`) might be added to this message.

(S)

Terminates processing.

(O)

Eliminate the cause of the error, and then re-execute the user-reply functionality's management daemon.

KNAX3003-E

An error was detected with adshmd.[*detailed-message*]

The user-reply functionality's management daemon terminated with an error or an error occurred during its termination processing. In the case of output to `syslog`, the message text for a message related to the user-reply functionality's management daemon (message number from `3000` through `3999`) might be added to this message.

(S)

Terminates processing.

(O)

Eliminate the cause of the error, and then re-execute the user-reply functionality's management daemon.

KNAX3006-I

adshmd will now start. (PID=*process-ID*, UID=*user-ID*, GID=*group-ID*, user name=*user-name*)

The user-reply functionality's management daemon has started.

(S)

Terminates processing.

KNAX3008-W

The old PID was *process-ID*.

This message displays the process ID (*PID*) used the last time the user-reply functionality's management daemon terminated with an error.

(S)

Resumes processing.

KNAX3009-E

adshmd could not start because another instance is already running.

The user-reply functionality's management daemon is already running.

This message is output to the applicable error notification destinations. The message might be output to only some of the destinations, depending on the timing of the error occurrence.

(S)

Terminates processing.

(O)

If necessary, terminate the running user-reply functionality's management daemon, and then restart it.

KNAX3020-E

A file error occurred. (function=*function-name*, target=*target-name*, details=*error-details*)

A file manipulation error occurred in the user-reply functionality's management daemon.

function-name, *target-name*, and *error-details* provide error information.

This message is output to the applicable error notification destinations. The message might be output to only some of the destinations, depending on the timing of the error occurrence.

(S)

Terminates processing.

(O)

Check the error information, eliminate the cause of the error, and then re-execute the user-reply functionality's management daemon.

KNAX3023-E

A signal error occurred. (function=*function-name*, target=*target-name*, details=*error-details*)

A signal error occurred in the user-reply functionality's management daemon.

function-name, *target-name*, and *error-details* provide error information.

This message is output to the applicable error notification destinations. The message might be output to only some of the destinations, depending on the timing of the error occurrence.

(S)

Terminates processing.

(O)

Check the error information, eliminate the cause of the error, and then re-execute the user-reply functionality's management daemon.

KNAX3024-E

A signal error occurred. (function=*function-name*, target=*target-name*)

A signal error occurred in the user-reply functionality's management daemon.

function-name and *target-name* indicate the error information.

(S)

Terminates processing.

(O)

Check the error information, eliminate the cause of the error, and then re-execute the user-reply functionality's management daemon.

KNAX3025-E

A signal error occurred. (function=*function-name*, details=*error-details*)

A signal error occurred in the user-reply functionality's management daemon.

function-name and *error-details* provide error information.

(S)

Terminates processing.

(O)

Check the error information, eliminate the cause of the error, and then re-execute the user-reply functionality's management daemon.

KNAX3026-E

A signal error occurred. (function=*function-name*)

A signal error occurred in the user-reply functionality's management daemon.

function-name indicates the error information.

(S)

Terminates processing.

(O)

Check the error information, eliminate the cause of the error, and then re-execute the user-reply functionality's management daemon.

KNAX3027-E

A process error occurred. (function=*function-name*, target=*target-name*, details=*error-details*)

A processing error occurred in the user-reply functionality's management daemon.

function-name, *target-name*, and *error-details* provide error information.

(S)

Terminates processing.

(O)

Check the error information, eliminate the cause of the error, and then re-execute the user-reply functionality's management daemon.

function-name indicates the system's function name. Determine the cause of the error based on the system function's error message.

KNAX3029-E

An error occurred in a system function. (function=*function-name*, details=*error-details*)

A system function error occurred in the user-reply functionality's management daemon.

function-name and *error-details* provide error information.

This message is output to the applicable error notification destinations. The message might be output to only some of the destinations, depending on the timing of the error occurrence.

(S)

Terminates processing.

(O)

Check the error information, eliminate the cause of the error, and then re-execute the user-reply functionality's management daemon.

KNAX3261-I

adshmd received the signal *signal-name*.

The user-reply functionality's management daemon received the termination request signal indicated by *signal-name*.

(S)

Resumes processing.

KNAX3400-I

The parameters specified in the environment file were correct.

Checking of the environment file was completed successfully.

(S)

Resumes processing.

KNAX3402-E

One or more parameters specified in the environment file were incorrect.

An error occurred while checking the environment file.

For details about the error, see the message that was output before this message.

(S)

Terminates processing.

(O)

Eliminate the cause of the error.

KNAX3508-I

A response request was canceled. (job ID=*job-ID*, line=*line-number*, host name=*host-name*)

The user-reply functionality's management daemon canceled the reply-request message that was initiated by the `adshread` command.

job-ID

Job ID assigned to the batch job by JP1/Advanced Shell

line-number

Line number in the job definition script where the `adshread` command was issued *host-name*

Name of the host on which the user-reply functionality's management daemon is running

(S)

Resumes processing.

KNAX3522-E

A shared memory error occurred. (function=*function-name*, details=*error-details*)

A shared memory error occurred in the user-reply functionality's management daemon.

function-name and *error-details* provide error information.

(S)

Terminates processing.

(O)

Check the error information, eliminate the cause of the error, and then re-execute the user-reply functionality's management daemon.

function-name indicates the system's function name. Determine the cause of the error based on the system function's error message.

KNAX3542-W

The shared memory object or named semaphore already exists, and will be initialized for reuse. ({*Named semaphore* | *Shared memory object*})

The existing shared memory or semaphore will be initialized and used because the `-f` option was specified for the user-reply functionality's management daemon.

target-name indicates a shared memory object or a named semaphore.

(S)

Initializes the existing shared memory or semaphore, and then resumes processing.

KNAX3700-I

The adshmd will now start.

The user-reply functionality's management daemon will now start.

(S)

Resumes processing.

KNAX3701-I

The adshmd will now stop.

The user-reply functionality's management daemon will now stop.

(S)

Resumes processing.

KNAX3703-E

The adshmd is not running.

The user-reply functionality's management daemon is not running.

(S)

Terminates processing.

(O)

Check if the user-reply functionality's management daemon is running.

KNAX3709-E

The adshmd could not start because another one is already running.

The user-reply functionality's management daemon is already running.

(S)

Terminates processing.

(O)

If necessary, terminate the running user-reply functionality management daemon, and then restart it.

KNAX3710-I

The adshmd is running.

The user-reply functionality's management daemon is running.

(S)

Terminates processing.

KNAX3711-I

The adshmd is not running.

The user-reply functionality's management daemon is not running.

(S)

Terminates processing.

KNAX3799-I

Usage *command-name* [-h *LogicalHostName*] {start [reuse]|stop|status|conftest [*EnvironFile*]|help}

This message displays the usage of the `adshmdctl` command.

(S)

Terminates processing.

KNAX3998-E

An error occurred during adshmd signal handler processing.

The signal handler terminated with an error in the user-reply functionality's management daemon.

(S)

Terminates processing.

(O)

Contact the system administrator.

KNAX3999-E

The adshmd ended abnormally because of an unexpected exception.

The user-reply functionality's management daemon terminated with an error.

(S)

Terminates processing.

(O)

Contact the system administrator.

KNAX4414-E

The size of the specified spool directory path is invalid. filename="*file-name*" line=*line-number*

The length of the path name for a specified spool directory is invalid.

file-name

Name of the target list file

line-number

Line number in the target list file where the error occurred

(S)

Performs the specified processing in the next line.

(O)

Check that the spool directory name specified in the target list file is correct, and then re-execute the command.

KNAX4415-E

The data format is incorrect. filename="*file-name*" line=*line-number*

The format of the target list file is invalid. Or, a number of days is not specified in the command's argument or in the target list file.

file-name

Name of the target list file

line-number

Line number in the target list file where the error occurred

(S)

Performs the specified processing in the next line.

(O)

Check and, if necessary, revise the specification in the target list file, or specify a number of days in the command's argument.

KNAX4416-E

A line exceeds the maximum line size. filename="*file-name*" line=*line-number*

A line in the target list file exceeds the permitted maximum length.

file-name

Name of the target list file

line-number

Line number in the target list file where the error occurred

(S)

Cancels processing.

(O)

Check and, if necessary, revise the specification in the target list file.

KNAX4417-E

A file was unexpectedly modified. filename="*file-name*"

The file entities differ before and after the file was opened.

file-name

File name

(S)

Cancels processing the corresponding file.

(O)

Check the file for any error.

KNAX4418-E

The file "*file-name*" is not a regular file.

The indicated file is not a regular file.

file-name

File name

(S)

Cancels processing the corresponding file.

(O)

Check the file for any error.

KNAX4419-E

An I/O error occurred. path="*path-name*" error="*error-details*"

An input or output error occurred in the file.

path-name:

Path name

One of the reasons why this message is output is that there is a file under the spool directory that cannot be recognized by JP1/Advanced Shell, such as a user-created file or directory.

(S)

Cancels processing the corresponding path.

(O)

Check the corresponding path for any error.

KNAX4420-E

A fatal error occurred. error information="*error-details*, *internal-information*"

An unexpected error occurred (an open error in a log file or trace file, or an error during time processing).

(S)

Cancels spool job processing if the error occurred during spool job processing; otherwise, terminates the command.

(O)

Check the operating environment for any problem.

KNAX4422-E

A required operand is not specified. Correct syntax: {*target-list-file* | *report-file* | *log-file*}

A mandatory operand is missing in the command.

target-list-file

Target list file

report-file

Report file

log-file

Log file

(S)

Terminates the command.

(O)

Specify the mandatory operand, and then re-execute the command.

KNAX4423-E

The number of days is specified incorrectly.

The format of the number of days specification in the command is invalid.

(S)

Terminates the command.

(O)

Specify the number of days in the correct format, and then re-execute the command.

KNAX4424-E

The spool job was not deleted because the start date for job execution could not be obtained. path="*spool-job-directory-name*"

The spool job was not deleted because the date the batch job is to start cannot be determined. The file used to manage the spool job might be corrupted.

spool-job-directory-name

Name of the erroneous spool job directory

(S)

Resumes processing without deleting the corresponding spool job.

(O)

If necessary, delete the corresponding spool job manually.

KNAX4425-E

The spool directory is being used by another program. path="*spool-directory-name*"

The spool directory indicated by *spool-job-directory-name* cannot be processed because it is being used by another program.

(S)

Resumes processing without processing the corresponding spool job.

(O)

Wait a while, check that the corresponding spool directory is not being used by another program, and then re-execute the command.

KNAX4427-W

An invalid spool job directory was skipped.

An invalid spool directory was found and that directory was skipped.

The name of the spool job directory for a completed job is not in one of the following formats:

- *job-ID*−*job-name*
- *job-ID*−

This message is displayed when the spool directory contains a directory or a file with an invalid name, such as one of the following:

- The name consists of fewer than six bytes.
- Byte 7 in the name is not a hyphen (−).
- The job name part of the name consists of 32 bytes or more.

These requirements do not apply to a file managed by JP1/Advanced Shell (`jobid` in UNIX and `adsh.jobid` in Windows). They are also not applicable to a name consisting of a job ID only (name length is six bytes) because such a job is currently executing.

(S)

Continues processing.

(O)

Delete any invalid directory or file manually.

KNAX4428-I

A spool job was removed. path="*path-name*"

The spool job was deleted.

path-name

Name of the spool job directory

(S)

Continues processing.

KNAX4429-E

An error occurred.

An error occurred during command execution.

(S)

Continues processing.

(O)

Determine the nature of the error by checking the message output to the log file, report file, or standard error output. If necessary, correct the error, and then re-execute the command.

KNAX5300-I

Usage: *command-name* [-jbspglogicalhost *LogicalHostName*]

An argument in the adapter command is invalid.

(S)

Terminates processing.

KNAX5301-E

No value is specified for the option "*option-name*".

A specified option value in the adapter command is invalid.

option-name

Option name in the adapter command

(S)

Terminates processing.

KNAX5305-E

> The specified argument "*argument*" is invalid.

An invalid *argument* was specified in the adapter command.

argument

Name of the adapter command's option.

(S)

Terminates processing.

KNAX5308-E

> An API error occurred. (maintenance information=*maintenance-information*, details=*error-details*)

An API error occurred in the adapter command.

maintenance-information and *error-details* provide error information.

(S)

Terminates processing.

(O)

If *maintenance-information* is `sem_open`, one of the following environment errors might have occurred:

- The user-reply functionality's management daemon or service is not running.

  Start the user-reply functionality's management daemon or service.

  If the reason the user-reply functionality's management daemon or service is not running is known, resolve the problem, and then restart the user-reply functionality's management daemon or service.

  If the cause is not known, contact the system administrator.

- The user-reply functionality's management service has not been registered (in Windows).

  Use the service registration procedure to register the user-reply functionality's management service, and then start the service.

  If the service cannot be registered or has been registered but will not start and the reason is known, resolve the problem, and then register or start the service.

  If the reason preventing the service from being registered or running is not known, contact the system administrator.

If *maintenance-information* is not `sem_open`, contact the system administrator.

KNAX5309-E

> An internal error occurred.

An internal error occurred.

(S)

Terminates processing.

(O)

Contact the system administrator.

KNAX5323-E

A signal error occurred. (function=*function-name*, target=*target-name*, details=*error-details*)

A signal error occurred in the adapter command.

function-name, *target-name*, and *error-details* provide error information.

(S)

Terminates processing.

(O)

Contact the system administrator.

KNAX5340-E

You do not have permission to execute the command *command-name*.

The user does not have execution permissions for the adapter command.

This command must be executed by a user with Administrators permissions.

(S)

Terminates processing.

(O)

The adapter command is a program that is started from JP1/Base's plug-in service. If this message is displayed when the adapter command was started as a plug-in service, check and, if necessary, revise the JP1/Base settings.

KNAX5350-E

The request header is invalid.

The request header that was passed to the adapter command is invalid.

(S)

Terminates processing.

(O)

Contact the system administrator.

KNAX5360-E

The request data is invalid.

The request data that was passed to the adapter command is invalid.

(S)

Terminates processing.

(O)

Contact the system administrator.

KNAX5361-E

Failed to get the identifier.

Acquisition of the identifier in the request data that was passed to the adapter command failed.

(S)

Terminates processing.

(O)

Contact the system administrator.

KNAX5362-E

The response contains one or more non-ASCII characters.

The data entered as a reply from the Enter Replies window in JP1/IM - View contains non-ASCII characters.

(S)

Terminates processing.

(O)

Specify an ASCII character string as the reply, and then re-enter the reply.

KNAX5371-E

The userreply function is busy.

JP1/Advanced Shell processing is busy.

(S)

Terminates processing.

(O)

Wait a while, and then reply to the reply-request message again.

KNAX5372-E

The message is not found.

There is no reply-request message. Possible reasons are as follows:

- The user-reply functionality's management daemon or service is not running.
- The logical host settings in JP1/Base are not valid.

When a response to a message was entered by proxy with the `adshchmsg` command, this message might be displayed depending on the timing. If this is the case, no action is needed.

(S)

Terminates processing.

(O)

Contact the system administrator.

KNAX5380-I

The following data was received: *received-data*

The user-reply functionality received a message from JP1.

(S)

Resumes processing.

KNAX5381-I

The following information was sent: *sent-data*

The user-reply functionality sent a message to JP1.

(S)

Resumes processing.

KNAX5396-I

adshuserreply.adapter completed because signal is detected.

The adapter command received a termination signal and terminated.

(S)

Terminates processing.

KNAX5397-I

Signal handler processing completed.

The adapter command received a signal.

(S)

Terminates processing.

(O)

The core has been output. Contact the system administrator.

KNAX5398-E

An error occurred during adshuserreply.adapter signal handler processing.

The adapter command received a signal, but an error occurred in the signal handler processing.

(S)

Terminates processing.

(O)

Contact the system administrator.

KNAX5399-E

The adshuserreply.adapter ended abnormally because of an unexpected exception.

The adapter command terminated with an error.

(S)

Terminates processing.

(O)

Contact the system administrator.

KNAX5407-E

Non-ASCII character is found in a response.

Non-ASCII characters are specified in the `-r` option (response) of the `adshchmsg` command.

(S)

Terminates processing.

(O)

Specify an ASCII character string in the `-r` option of the `adshchmsg` command, and then re-enter the response.

KNAX5409-E

No valid response request was found.

An attempt was made to cancel the reply-request message or to enter a reply to the reply-request message whose number was specified in the `-n` option in the `adshchmsg` command, but the specified reply-request message was not found.

(S)

Terminates processing.

(O)

Check the following:

- Whether the correct reply-request message number was specified in the `-n` option.
- Whether the reply-request message number specified in the `-n` option is displayed by executing the `adshlsmsg` command.

If the correct number was specified but it is not displayed by executing the `adshlsmsg` command, a reply might have already been entered.

KNAX5410-E

An API error occurred. (maintenance information=*maintenance-information*, details=*error-details*)

An API error occurred in the `adshchmsg` or `adshlsmsg` command.

maintenance-information and *error-details* provide error information.

(S)

Terminates processing.

(O)

If *maintenance-information* is `sem_open`, one of the following environment errors might have occurred:

- The user-reply functionality's management daemon or service is not running.
  Start the user-reply functionality's management daemon or service.

If the reason the user-reply functionality's management daemon or service is not running is known, resolve the problem, and then restart the user-reply functionality's management daemon or service.

If the cause is not known, contact the system administrator.

- The user-reply functionality's management service has not been registered (in Windows).

  Use the service registration procedure to register the user-reply functionality's management service, and then start the service.

  If the service cannot be registered or has been registered but will not start and the reason is known, resolve the problem, and then register or start the service.

  If the reason preventing the service from being registered or running is not known, contact the system administrator.

If *maintenance-information* is not `sem_open`, contact the system administrator.

KNAX5423-E

A signal error occurred. (function=*function-name*, target=*target-name*, details=*error-details*)

A signal error occurred in the `adshchmsg` or `adshlsmsg` command.

function-name, *target-name*, and *error-details* provide error information.

(S)

Terminates processing.

(O)

Check the error information, eliminate the cause, and then re-execute the command.

KNAX5424-E

A signal error occurred. (function=*function-name*, target=*target-name*)

A signal error occurred in the `adshchmsg` or `adshlsmsg` command.

function-name and *target-name* provide error information.

(S)

Terminates processing.

(O)

Check the error information, eliminate the cause, and then re-execute the command.

KNAX5425-E

A signal error occurred. (function=*function-name*, details=*error-details*)

A signal error occurred in the `adshchmsg` or `adshlsmsg` command.

function-name and *error-details* provide error information.

(S)

Terminates processing.

(O)

Check the error information, eliminate the cause, and then re-execute the command.

KNAX5426-E

A signal error occurred. (function=*function-name*)

A signal error occurred in the `adshchmsg` or `adshlsmsg` command.

function-name provides error information.

(S)
Terminates processing.

(O)
Check the error information, eliminate the cause, and then re-execute the command.

KNAX5429-E

An internal error occurred. (*maintenance-information*)

An internal error occurred in the `adshchmsg` or `adshlsmsg` command.

(S)
Terminates processing.

(O)
Contact the system administrator.

KNAX5440-E

You do not have permission to execute the command *command-name*.

The user does not have execution permissions for the command indicated by *command-name*.

This command must be executed by a user with Administrators permissions.

(S)
Terminates processing.

(O)
Have a user with Administrators permissions execute the command.

KNAX5498-E

An error occurred during *command-name* signal handler processing.

An error occurred in the `adshchmsg` or `adshlsmsg` command during signal handler processing.

(S)
Terminates processing.

(O)
Contact the system administrator.

KNAX5499-E

The *command-name* ended abnormally because of an unexpected exception.

The `adshchmsg` or `adshlsmsg` command terminated with an error.

(S)

Terminates processing.

(O)

Contact the system administrator.

KNAX6000-E

The built-in command "*command-name*" is not supported. [filename="*file-name*" line=*line-number*]

The specified built-in command is not supported in JP1/Advanced Shell.

command-name

Name of the built-in command that is not supported in JP1/Advanced Shell

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Terminates processing.

(O)

Check the erroneous line and correct the job definition script.

KNAX6001-E

The shell option "*shell-option-name*" is not supported. [filename="*file-name*" line=*line-number*]

The specified shell option is not supported in JP1/Advanced Shell.

shell-option-name

Name of the shell option that is not supported in JP1/Advanced Shell

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Terminates processing.

(O)

Check the erroneous line and correct the job definition script.

KNAX6002-E

The shell variable "*shell-variable-name*" cannot be specified. [filename="*file-name*" line=*line-number*]

The specified shell variable name is not supported in JP1/Advanced Shell.

shell-variable-name

Name of the shell variable that is not supported in JP1/Advanced Shell

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Terminates processing.

(O)

Correct the erroneous shell variable name.

KNAX6003-E

The variable "*variable-name*" is not an identifier. [filename="*file-name*" line=*line-number*]

The specified variable name contains invalid characters.

variable-name

Variable name determined to be invalid

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues processing if it was an extended shell command, regular built-in command, or the `typeset` command that resulted in the error; otherwise, terminates processing.

(O)

Correct the variable name resulting in the error.

KNAX6004-E

The specified value "*invalid-value*" is invalid. [filename="*file-name*" line=*line-number*]

The cause might be one of the following:

- An attempt was made to assign characters to an integer-type variable.
- An argument requires a numeric value, but characters were specified.
- The specified numeric value is invalid.

invalid-value

Value determined to be invalid or a non-numeric value

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues processing if it was an extended shell command, regular built-in command, or the `typeset` command that resulted in the error; otherwise, terminates processing.

(O)

If an assignment expression resulted in the error, check the value or the attributes of the variable to be assigned, and then correct the job definition script as necessary. If a command resulted in the error, check the specified argument values, and then correct the job definition script.

KNAX6005-E

Too many arguments are specified. [filename="*file-name*" line=*line-number*]

Too many arguments were specified in the command.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues processing if it was a regular built-in command that resulted in the error; otherwise, terminates processing.

(O)

Check the command argument that resulted in the error, and then correct the job definition script.

KNAX6006-E

A substitution is specified incorrectly. [filename="*file-name*" line=*line-number*]

The specified substitution is invalid. Or, a character string that is not part of the current directory path name is specified as an argument in the `cd` command.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues processing if it was the `cd` command that resulted in the error; otherwise, terminates processing.

(O)

Check the variable or command substitution or argument specification that resulted in the error, and then correct the job definition script.

KNAX6007-E

The subscript of the array "*array-name*" is out of range.[filename="*file-name*" line=*line-number*]

The number of array elements is out of range.

array-name

Specified array name

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues processing if it was an extended shell command that resulted in the error; otherwise, terminates processing.

(O)

Correct the job definition script so that the array element number that is specified is in the range from `0` through `65535`.

When `TYPE_A` or `TYPE_B` is specified in the `VAR_SHELL_FUNCINFO` environment setting parameter, the function nesting level exceeds the maximum number of array elements. Either specify `NONE` in the `VAR_SHELL_FUNCINFO` environment setting parameter or correct the job definition script.

KNAX6008-E

The variable "*variable-name*" is read-only. [filename="*file-name"* line=*line-number*]

An attempt was made to assign a value to a read-only variable.

variable-name

Specified variable name

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Sets the command's return code and continues processing when any of the following commands has resulted in the error:

- Extended shell command
- Regular built-in command
- `typeset` command

  If your specification was attempting to assign a value to a variable that is defined as an array without specifying an element number while the attribute of that variable has been set to read-only by the `typeset` command, the assignment processing that resulted in this error is not performed. However, in the case of an assignment expression, the system sets return code `0` and continues processing.

Otherwise, terminates processing.

(O)

Check the attribute or name of the variable that resulted in the error, and then correct the job definition script.

KNAX6009-E

The option "*option*" is invalid. [filename="*file-name*" line=*line-number*]

An invalid option was specified in the command.

option

Option specified in the command

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues processing if it was an extended shell command, regular built-in command, or the `typeset` command that resulted in the error; otherwise, terminates processing.

(O)

Check the option specified in the command, and then correct the job definition script.

KNAX6010-E

The option "*shell-option*" is invalid. [filename="*file-name*" line=*line-number*]

An invalid shell option was specified in the `set` command.

option

Specified shell option

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Terminates processing.

(O)

Check the option specified in the `set` command, and then correct the job definition script.

KNAX6011-E

The signal number or signal name "*signal-information*" is invalid. [filename="*file-name*" line=*line-number*]

An invalid signal number or signal name was specified.

signal-number-or-name

Specified signal number or signal name

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues processing if it was a regular built-in command that resulted in the error; otherwise, terminates processing.

(O)

Check the signal number or signal name specified in the command, and then correct the job definition script.

KNAX6012-E

The mask "*mask*" is invalid. [filename="*file-name*" line=*line-number*]

An invalid mask was specified.

mask
Specified mask

file-name
Name of the job definition script file

line-number
Line number in the job definition script file where the error occurred

(S)
Continues processing.

(O)
Check the mask specified in the command, and then correct the job definition script.

KNAX6013-E

The value of an upper limit could not be changed. error details=*error-details* [filename="*file-name*" line=*line-number*]

The maximum value could not be changed because the error indicated by *error-details* occurred.

error-details
Details of the error. This is an error information string representation for `errno`.

file-name
Name of the job definition script file

line-number
Line number in the job definition script file where the error occurred

(S)
Continues processing.

(O)
Eliminate the cause of the error based on the error details. If necessary, correct the job definition script.

KNAX6014-E

No value is set for the specified variable "*variable-name*". [filename="*file-name*" line=*line-number*]

A variable for which no value has been set was specified while the `nounset` shell option was enabled.

variable-name
Specified variable name

file-name
Name of the job definition script file

line-number
Line number in the job definition script file where the error occurred

(S)

Terminates processing.

(O)

Evaluate whether the `nounset` shell option is required. If it is required, correct the job definition script so that a value is assigned when the variable is used.

KNAX6015-E

No argument is specified. [filename="*file-name*" line=*line-number*]

A built-in command requires an argument, but the command was executed with no argument specified.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues processing if it was a regular built-in command that resulted in the error; otherwise, terminates processing.

(O)

Check the command specification resulting in the error, and then correct the job definition script.

KNAX6016-E

The option "*option*" requires an argument. [filename="*file-name*" line=*line-number*]

The command was executed with no option value specified.

option

Specified option

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues processing if it was a regular built-in command that resulted in the error; otherwise, terminates processing.

(O)

Check the command specification resulting in the error, and then correct the job definition script.

KNAX6017-E

A syntax error occurred in a statement item ("*item-name*"). [file name="*file-name*" line=*line-number*]

The specified control statement is not valid.

item-name

Word determined to constitute a syntax error

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues processing if it was a regular built-in command that resulted in the error; otherwise, terminates processing.

(O)

Correct the job definition script.

KNAX6018-E

A required part ("*item-name*") of a statement is not present. [filename="*file-name*" line=*line-number*]

A correspondence between words is invalid in a control statement.

item-name

Word determined to constitute a syntax error

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6019-E

EOF was unexpectedly reached during syntax analysis. [filename="*file-name*" line=*line-number*]

A specified control statement is invalid.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6020-E

The specified directory path "*directory-path*" is invalid. [filename="*file-name*" line=*line-number*]

An invalid directory path was specified.

directory-path

Specified directory path

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues the processing without changing the directory.

(O)

Correct the job definition script.

KNAX6021-E

Too many instances of "<<" are specified. [filename="*file-name*" line=*line-number*]

A redirect specification is invalid in a here document.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6022-E

Too many redirections are specified. [filename="*file-name*" line=*line-number*]

Too many redirections are specified.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6023-W

The {break|continue} command cannot be executed in this context. [filename="*file-name*" line=*line-number*]

The `break` or `continue` command was executed outside a loop.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues processing.

(O)

Correct the job definition script.

KNAX6024-E

The name specified for the function "*function-name*" is invalid. [filename="*file-name*" line=*line-number*]

An invalid function name was specified in a function definition.

function-name

Specified function name

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6025-E

"*entity-name*" was not found. [filename="*file-name*" line=*line-number*]

A specified file name, command name, or function name cannot be identified.

file-name, command-name, or function-name

Unidentifiable name that was specified for a file name, command name, or function name

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Terminates the processing if it was a `.` (dot) command that resulted in the error; otherwise, continues processing.

(O)

Check for an error in the specified file name, command name, or function name, and then correct the job definition script.

KNAX6026-E

The command "*command-name*" cannot be executed. reason=*error-details* [filename="*file-name*" line=*line-number*]

The specified command could not be executed because the error indicated by *error-details* occurred.

command-name

Specified command name

error-details

Details of the error. This is an error information string representation for `errno`.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues the processing without executing the specified command.

(O)

Eliminate the cause of the error based on the error details. If necessary, correct the job definition script.

KNAX6027-W

The value specified in the {break|continue} command is greater than the number of nested loops (*nested-loop-count*). [filename="*file-name*" line=*line-number*]

The value specified in the argument of the `break` or `continue` command is greater than the number of nesting loops.

nested-loop-count

Number of nesting loops when control is to exit the looped processing (`break` command) or when the looped processing is to be canceled and control returned to the beginning (`continue` command)

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Executes the `break` or `continue` command as many times as there are nesting loops, and then continues processing.

(O)

Check the argument specified in the `break` or `continue` command, and then correct the job definition script.

KNAX6028-E

The command "*command-name*" is not a built-in command. [filename="*file-name*" line=*line-number*]

The command specified in the `builtin` command is not a built-in command.

command-name

Specified command name

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Terminates processing.

(O)

Check the argument specified in the `builtin` command, and then correct the job definition script.

KNAX6029-E

A coprocess is already being executed. [filename="*file-name*" line=*line-number*]

A background process is already running.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6030-E

The directory cannot be changed. directory=*directory-path* details=*error-details* [filename="*file-name*" line=*line-number*]

The directory cannot be changed.

directory-path-name

Specified directory path name

error-details

Details of the error. This is an error information string representation for `errno`.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues the processing without changing the directory.

(O)

Eliminate the cause of the error based on the error details. If necessary, correct the job definition script.

KNAX6031-E

The shell variable HOME is not set. [filename="*file-name*" line=*line-number*]

The directory cannot be changed because the `HOME` shell variable has not been specified.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues the processing without changing the directory.

(O)

Specify the home directory in the `HOME` shell variable, and then re-execute the job definition script.

KNAX6032-E

The shell variable OLDPWD is not set. [filename="*file-name*" line=*line-number*]

The directory cannot be changed because the `OLDPWD` shell variable has not been specified.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues the processing without changing the directory.

(O)

Correct the job definition script.

KNAX6033-E

The current directory could not be identified. [filename="*file-name*" line=*line-number*]

The directory cannot be changed because the current directory cannot be identified.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues the processing without changing the directory.

(O)

Re-execute the job definition script.

KNAX6034-E

No coprocess exists. [filename="*file-name*" line=*line-number*]

The job definition script was executed without a background process.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues processing.

(O)

Correct the job definition script.

KNAX6035-E

The specified file descriptor "*file-descriptor*" is invalid. reason=*error-details* [filename="*file-name*" line=*line-number*]

An invalid file descriptor was specified.

file-descriptor

Specified file descriptor

error-details

Details of the error. This is an error information string representation for `errno`.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues processing.

(O)

Eliminate the cause of the error based on the error details. If necessary, correct the job definition script.

KNAX6036-E

The value specified for the process ID ("*process-ID*") is invalid. [filename="*file-name*" line=*line-number*]

A specified process ID is invalid.

process-ID

Specified process ID

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues the processing in the case of a regular built-in command; terminates the processing in the case of a special built-in command.

(O)

Correct the job definition script.

KNAX6037-E

The getopts command was executed without a required option. [filename="*file-name*" line=*line-number*]

The `getopts` command was executed with no option specified.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues processing.

(O)

Correct the job definition script.

KNAX6038-E

The getopts command was executed without the required name argument. [filename="*file-name*" line=*line-number*]

The `getopts` command was executed with `name` omitted.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues processing.

(O)

Correct the job definition script.

KNAX6039-E

The shift command was executed with more arguments specified than were specified in the command line. [filename="*file-name*" line=*line-number*]

The `shift` command was executed while the number of specified arguments was greater than the number of arguments in the command line.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Terminates processing.

(O)

Check the number of arguments specified in the `shift` command or the number of arguments in the command line, and correct the job definition script if necessary.

KNAX6040-E

A "]" character is missing. [filename="*file-name*" line=*line-number*]

A right square bracket (`]`) is missing.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues processing if it was a regular built-in command that resulted in the error; otherwise, terminates processing.

(O)

Correct the job definition script.

KNAX6041-E

The specified test command or conditional expression is invalid. [filename="*file-name*" line=*line-number*]

There is an error in the specified `test` command or in a conditional expression.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Resumes the processing in the case of the `test` command; terminates the processing in the case of a conditional expression.

(O)

Correct the job definition script.

KNAX6042-E

The expression contains a read-only variable. [filename="*file-name*" line=*line-number*]

A read-only variable is specified in an arithmetic expression.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues the processing without processing the arithmetic expression.

(O)

Check the attribute of the variable specified in the arithmetic expression, and then correct the job definition script.

KNAX6043-W

The command to execute in another process is not specified. [filename="*file-name*" line=*line-number*]

A command to be executed in another process has not been specified.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues processing.

(O)

Correct the job definition script.

KNAX6044-E

The function definition file "*function-name*" was not found in the FPATH directory. [filename="*file-name*" line=*line-number*]

No directory is specified in the `FPATH` shell variable. Or, no function definition file for the directory specified in the `FPATH` shell variable was found.

function-name

Function name

file-name

Script file name

line-number

Line number in the script file where the error occurred

(S)

Continues processing.

(O)

Check if the directory containing the function definition file is specified in the `FPATH` shell variable.

If the directory is specified, check if the name of the function whose execution was attempted is correct and whether the directory specified in the `FPATH` shell variable contains the function definition file for the function that is to be executed.

KNAX6045-E

The function definition file "*function-name*" cannot be opened. [filename="*file-name*" line=*line-number*]

A function definition file cannot be opened.

function-name

Function name

file-name

Script file name

line-number

Line number in the script file where the error occurred

(S)

Continues processing.

(O)

Check the directory specified in the `FPATH` shell variable and the permissions for the function definition file of the function that is to be executed.

KNAX6046-E

The function "*function-name*" is not defined in the function definition file "*function-definition-file*". [filename="*file-name*" line=*line-number*]

The indicated function is not defined in the function definition file.

function-name

Function name

function-definition-file-name

Name of the function definition file in which the function that is to be executed is defined

file-name

Script file name

line-number

Line number in the script file where the error occurred

(S)

Continues processing.

(O)

Check and, if necessary, correct the name of the function to be executed or the function name defined in the function definition file.

KNAX6047-E

The value specified for the upper limit ("*upper-limit-value*") is invalid. [filename="*file-name*" line=*line-number*]

A specified maximum value is invalid.

maximum-value

Specified option

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues the processing without changing the maximum value.

(O)

Correct the job definition script.

KNAX6048-E

Failed to set or modify the upper limit of a system resource. [filename="*file-name*" line=*line-number*]

An error occurred before the hard limit for resources was changed by the `ulimit` command. This message is issued in one of the following cases:

- The user has the permission needed to change the hard limit, but specified a value not permitted by the system.
- The user does not have the permission needed to change the hard limit, and specified a value that exceeded the set hard limit.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues the processing without changing the maximum value. Note, however, that depending on the resource to be changed, the execution environment or OS might set a different value if a value not permitted by the system is specified.

(O)

Take the corrective action required for the cause of the error as described below, and then re-execute the job definition script:

- When the user has the permission needed to change the hard limit, but specified a value not permitted by the system

  Change the argument of the `ulimit` command to a value permitted by the system.

- When the user does not have the permission needed to change the hard limit, and specified a value that exceeded the set hard limit

  Grant to the executing user the administrator permission and change the argument of the `ulimit` command to a value permitted by the system. Note that the administrator permission is not required to reduce the hard limit.

KNAX6049-E

You cannot modify the resource specified in the ulimit command. [filename="*file-name*" line=*line-number*]

The resource specified in the `ulimit` command cannot be changed.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues the processing without changing the maximum value.

(O)

Correct the job definition script.

KNAX6050-E

The variable "*variable-name*" is null or has not been defined. [filename="*file-name*" line=*line-number*]

The variable specified for variable substitution has not been created or has no value set in it.

variable-name
Specified variable name

file-name
Name of the job definition script file

line-number
Line number in the job definition script file where the error occurred

(S)
Terminates processing.

(O)
Check the variable name specified for variable substitution, and, if necessary, correct the job definition script.

KNAX6051-E

The specified redirection character "*redirection-character*" is invalid. [filename="*file-name*" line=*line-number*]

redirect-character specified in command substitution is invalid.

redirection-character
Specified redirect character

file-name
Name of the job definition script file

line-number
Line number in the job definition script file where the error occurred

(S)
Terminates processing.

(O)
Correct the job definition script.

KNAX6052-E

The input file "*input-file-name*" cannot be opened. [filename="*file-name*" line=*line-number*]

The input file specified in command substitution cannot be opened.

input-file-name
Specified input file name

file-name
Name of the job definition script file

line-number
Line number in the job definition script file where the error occurred

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6053-E

Pipe creation failed. [filename="*file-name*" line=*line-number*]

Pipe creation failed.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Terminates processing.

(O)

Check the indicated line in the job definition script whose file name is displayed in the message for any specification error. Also, check if too many files are open in the job definition script. If there is a specification error, correct it and then re-execute the job definition script.

If the problem is still unresolved after re-execution, contact the system administrator.

KNAX6054-E

Process creation failed. [filename="*file-name*" line=*line-number*]

Process creation failed.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Terminates processing.

(O)

Check the indicated line in the job definition script whose file name is displayed in the message for any specification error. If there is a specification error, correct it and then re-execute the job definition script.

If the problem is still unresolved after re-execution, possible causes are as follows:

- The path of the executable file cannot be found.
- The executable file is not a regular file.
- The user does not have the permission to search the components of the executable file.
- The path name of the executable file is too long.
- There are too many arguments in the executable file, or a specified argument is invalid.
- The specified file is not executable.

- Too many symbolic links were found while the path names of the executable files were being converted.
- The total number of processes that can be executed exceeds the system's maximum.
- There is inadequate swapping area or physical memory for creating a new process.
- There are too many files to be opened.

Correct the applicable cause listed above, and then re-execute the job definition script.

If the problem is still unresolved after re-execution, contact the system administrator.

KNAX6055-E

Failed to send the signal. PID=*process-ID* signal number=*signal-number* reason=*error-details* [filename="*file-name*" line=*line-number*]

Transmission of a signal with the specified *process-ID* failed because the error indicated by *error-details* occurred.

process-ID

Specified process ID

signal-number

Signal number whose transmission failed

error-details

Details of the error. This is an error information string representation for `errno`.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues processing.

(O)

Eliminate the cause of the error based on the error details. If necessary, correct the job definition script.

KNAX6056-W

The specified process ID "*process-ID*" is invalid and will be ignored. [filename="*file-name*" line=*line-number*]

A specified process ID was ignored because it was not valid.

process-ID

Specified process ID

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues processing.

(O)

Correct the job definition script.

KNAX6057-E

Memory could not be allocated because of insufficient memory. [filename="*file-name*" line=*line-number*]

A memory shortage occurred.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Terminates processing.

(O)

Contact the system administrator. The system administrator must check and, if necessary, revise the memory estimation.

KNAX6058-E

The maximum recursion count for the shell variable ("*shell-variable-name*") exceeds the limit. [filename="*file-name*" line=*line-number*]

Processing was stopped because the permitted maximum recursive conversion count was exceeded for the following shell variable:

- Permitted recursive conversion count for a variable specified in `offset` when shell variables are referenced: 1,024
- Permitted recursive conversion count for a variable specified in `length` when shell variables are referenced: 1,025

shell-variable-name

Name of the variable resulting in the circular or recursive reference

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Terminates processing.

(O)

Because a circular or recursive reference specification has been made for the specified variable, check and revise the value of the variable as necessary.

KNAX6059-E

Too many files are open in the script. [filename="*file-name*" line=*line-number*]

Too many files are open in the job definition script.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Terminates processing.

(O)

Check the indicated line in the job definition script whose file name is displayed in the message for any specification error. Also check if too many files are open in the job definition script. If there is a specification error, correct it and then re-execute the job definition script.

If the problem is still unresolved after re-execution, contact the system administrator.

KNAX6061-E

Creation of the here document failed. [filename="*file-name*" line=*line-number*]

Creation of a here document failed.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues processing.

(O)

Correct the job definition script.

KNAX6062-E

The temporary file could not be {created|opened|deleted}. temporary filename=*temporary-file-name* reason=*error-details* [filename="*file-name*" line=*line-number*]

A temporary file cannot be created, opened, or deleted because the error indicated by *error-details* occurred.

temporary-file-name

Name of temporary file to be created

error-details

Details of the error. This is an error information string representation for `errno`.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Terminates processing. However, if it was the here document processing that resulted in the error, the system continues processing.

(O)

Eliminate the cause of the error based on the error details. If necessary, correct the job definition script.

KNAX6063-E

> Failed to write to the file *write-file-name*. reason=*error-details* [filename="*file-name*" line=*line-number*]

A write error occurred in the indicated file because the error indicated by *error-details* occurred.

target-file-name

Name of the file into which data was to be written

error-details

Details of the error. This is an error information string representation for `errno`.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Resumes processing.

(O)

Eliminate the cause of the error based on the error details. If necessary, correct the job definition script.

KNAX6064-E

> Failed to create the temporary variable "*variable-or-expression-name*" in the expression. [filename="*file-name*" line=*line-number*]

Creation of a variable to be used temporarily in an arithmetic expression failed.

variable-name

Name of the variable contained in the arithmetic expression that resulted in the error

arithmetic-expression

Arithmetic expression resulting in the error

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues the processing without performing the arithmetic operation that resulted in the error.

(O)

Correct the job definition script.

KNAX6065-E

> The expression contains an invalid variable or expression "*variable-or-expression-name*". [filename="*file-name*" line=*line-number*]

An invalid variable was used in an arithmetic expression.

variable-name

Name of the variable contained in the arithmetic expression that resulted in the error

arithmetic-expression

Arithmetic expression resulting in the error

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues the processing without performing the arithmetic operation resulting in an error.

(O)

Correct the job definition script.

KNAX6066-E

The expression syntax is invalid. [filename="*file-name*" line=*line-number*]

The format of an arithmetic expression is invalid.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues the processing without performing the arithmetic operation that resulted in the error.

(O)

Correct the job definition script.

KNAX6067-E

A divisor in the expression is zero. [filename="*file-name*" line=*line-number*]

A divide-by-zero error occurred.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues the processing without performing the arithmetic operation that resulted in the error.

(O)

Correct the job definition script.

KNAX6068-E

A negative value is specified for an exponent. [filename="*file-name*" line=*line-number*]

A negative value was specified as the exponent for the arithmetic operator (`**`).

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues the processing without performing the arithmetic operation that resulted in the error.

(O)

Correct the job definition script.

KNAX6070-E

No more jobs need to be waited on for completion.[filename="*file-name*" line=*line-number*]

There is no job to be executed.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues processing.

(O)

Correct the job definition script.

KNAX6071-E

The current directory could not be identified. reason=*error-details* [filename="*file-name*" line=*line-number*]

The current directory cannot be identified because the error indicated by *error-details* occurred.

error-details

Details of the error. This is an error information string representation for `errno`.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues the processing without changing the directory.

(O)

Eliminate the cause of the error based on the error details. If necessary, correct the job definition script.

KNAX6072-E

A required option is not specified.[filename="*file-name*" line=*line-number*]

A required option is missing.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues processing if a regular built-in command resulted in the error; otherwise, terminates processing.

(O)

Check the command option that resulted in the error and correct the job definition script.

KNAX6075-E

Redirection by using *redirection-character* failed. reason=*error-details* [filename="*file-name*" line=*line-number*]

The file ID cannot be copied by using `dup` because the error indicated by *error-details* occurred.

redirection-character

Specified redirect character

error-details

Details of the error. This is an error information string representation for `errno`.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues processing.

(O)

Eliminate the cause of the error based on the error details. If necessary, correct the job definition script.

KNAX6076-E

Arguments for the getopts command were altered while the command was running.[filename="*file-name*" line=*line-number*]

The contents of an argument were changed during execution of the `getopts` command.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues processing.

(O)

Check the indicated line in the job definition script where the error occurred for any specification error. If there is a specification error, correct it and then re-execute the job definition script.

KNAX6077-E

An invalid option is specified. [filename="*file-name*" line=*line-number*]

A specified option is invalid.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues processing if it was a regular built-in command that resulted in the error; otherwise, terminates processing.

(O)

Correct the job definition script.

KNAX6078-E

Closing quotation marks are missing. [filename="*file-name*" line=*line-number*]

The correspondence of quotation marks is invalid.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6079-E

The label "*label*" specified in the here document was not found. [filename="*file-name*" line=*line-number*]

The label specified in the here document was not found.

label

Label specified in the here document

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6080-E

The file descriptor for the file *target-file-name* could not be duplicated. reason=*error-details* [filename="*file-name*" line=*line-number*]

A file ID cannot be copied because the error indicated by *error-details* occurred.

target-file-name

Name of the file resulting in the file ID duplication error

error-details

Details of the error. This is an error information string representation for `errno`.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues processing if it was a regular built-in command that resulted in the error; otherwise, terminates processing.

(O)

Eliminate the cause of the error based on the error details. If necessary, correct the job definition script.

KNAX6081-E

The file *target-file-name* could not be opened. reason=*error-details* [filename="*file-name*" line=*line-number*]

An open error occurred in a file because the error indicated by *error-details* occurred.

target-file-name

Name of the file resulting in the file open error

error-details

Details of the error. This is an error information string representation for `errno`.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues processing if it was a regular built-in command that resulted in the error; otherwise, terminates processing.

(O)

Eliminate the cause of the error based on the error details. If necessary, correct the job definition script.

KNAX6082-E

The file *target-file-name* could not be created. reason=*error-details* [filename="*file-name*" line=*line-number*]

File creation failed because the error indicated by *error-details* occurred.

target-file-name

Name of the file that resulted in the creation error

error-details

Details of the error. This is an error information string representation for `errno`.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues processing if it was a regular built-in command that resulted in the error; otherwise, terminates processing.

(O)

Eliminate the cause of the error based on the error details. If necessary, correct the job definition script.

KNAX6085-E

The trap cannot be set to the specified signal "*signal-number-or-name*". [filename="*file-name*" line=*line-number*]

A trap cannot be set for a specified signal.

signal-number-or-name

Specified signal number or signal name

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Terminates processing.

(O)

Check and correct the signal number or signal name specified in the `trap` command.

KNAX6097-E

The attribute of the specified extended shell variable "*variable-name*" cannot be changed. [filename="*file-name*" line=*line-number*]

An attempt was made to change the attribute of an extended shell variable that cannot be changed. Possible causes are as follows:

- An attempt was made to release an extended shell variable's read-only attribute.
- An attempt was made to change an extended shell variable to a local variable.

variable-name

Name of the extended shell variable on which an attempt was made to change the attribute

file-name

Script file name

line-number

Line number in the script file where the error occurred

(S)

Terminates processing.

(O)

Take one of the following actions:

- Check the name of the variable resulting in the error or the affected attribute, and then correct the job definition script.
- Change the `VAR_SHELL_FUNCINFO` environment setting parameter value to `NONE` and then re-execute the command.

If the problem cannot be resolved when the job definition script is re-executed, contact the system administrator.

KNAX6098-E

An error occurred. reason=*source-row-number*, *error-analysis-data* [filename="*file-name*" line=*line-number*]

An error occurred.

source-row-number

Line number in the source where the error occurred

error-analysis-data

Error analysis information

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Continues processing if it was a regular built-in command that resulted in the error; otherwise, terminates processing.

(O)

Check the indicated line in the job definition script whose file name is displayed in the message for any specification error. If there is a specification error, correct it and then re-execute the job definition script. If the problem is still unresolved after re-execution, contact the system administrator.

KNAX6099-E

An internal error occurred. reason=*source-row-number*, *error-analysis-data* [filename="*file-name*" line=*line-number*]

An internal error occurred.

source-row-number

Line number in the source where the error occurred

error-analysis-data

Error analysis information

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Terminates processing.

(O)

Contact the system administrator.

KNAX6100-E

Execution of the command "*command-name*" failed. exit status=*exit-status-code* [file name="*file-name*" line=*line-number*]

Execution of a command failed. When this message is displayed, the return code displayed in the `KNAX7999-I` message is ignored and *return-code* displayed in this message is set as the job's return code.

command-name

Name of the command that failed to execute

exit-status-code

Job's return code

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Terminates processing.

(O)

Eliminate the cause of the error based on the `KNAX6098-E` message output immediately before this message, and then re-execute the job definition script. If the problem is still unresolved after re-execution, contact the system administrator.

KNAX6110-I

Execution of the command *command-name* (line=*line-number*) finished successfully. exit status=*exit-status-code* execution time=*execution-time-in-seconds* CPU time=*CPU-time-in-seconds*

The command terminated normally.

command-name

Name of the command that executed. If the command name exceeds the maximum path length permitted by the OS, it is truncated at the maximum length.

line-number

Number of lines in the script on which the command is specified

exit-status-code

Command's return code

execution-time-in-seconds

Command execution time (a reference value obtained by using the OS's API)

CPU-time-in-seconds

Command's CPU time (a reference value obtained by using the OS's API)

(S)

Resumes processing.

KNAX6111-I

Execution of the command *command-name* (line=*line-number*) finished. exit status=*exit-status-code* execution time=*execution-time-in-seconds* CPU time=*CPU-time-in-seconds*

The command was terminated. This message is displayed when a command whose result (success or failure) cannot be determined from the return code was terminated.

command-name

Name of the command that executed. If the command name exceeds the maximum path length permitted by the OS, it is truncated at the maximum length.

line-number

Number of lines in the script on which the command is specified

exit-status-code

Command's return code

execution-time-in-seconds

Command execution time (a reference value obtained by using the OS's API)

CPU-time-in-seconds

Command's CPU time (a reference value obtained by using the OS's API)

(S)

Resumes processing.

KNAX6112-I

Execution of the command *command-name* (line=*line-number*) finished successfully. exit status=*exit-status-code* execution time=*execution-time-in-seconds* CPU time=*CPU-time-in-seconds*

The command terminated normally.

command-name

Name of the command that executed. If the command name exceeds the maximum path length permitted by the OS, it is truncated at the maximum length.

line-number

Number of lines in the script on which the command is specified

exit-status-code

Command's return code

execution-time-in-seconds

Command execution time (a reference value obtained by using the OS's API)

CPU-time-in-seconds

Command's CPU time (a reference value obtained by using the OS's API)

(S)

Resumes processing.

KNAX6113-I

Execution of the command *command-name* (line=*line-number*) finished. exit status=*exit-status-code* execution time=*execution-time-in-seconds* CPU time=*CPU-time-in-seconds*

The command was terminated. This message is displayed when a command whose result (success or failure) cannot be determined from the return code was terminated.

command-name

Name of the command that executed. If the command name exceeds the maximum path length permitted by the OS, it is truncated at the maximum length.

line-number

Number of lines in the script on which the command is specified

exit-status-code

Command's return code

execution-time-in-seconds

Command execution time (a reference value obtained by using the OS's API)

CPU-time-in-seconds

Command's CPU time (a reference value obtained by using the OS's API)

(S)

Resumes processing.

KNAX6114-I

Execution of the command *command-name* (line=*line-number*) finished successfully. exit status=*exit-status-code* execution time=*execution-time-in-seconds* CPU time=*CPU-time-in-seconds*

The command terminated normally.

command-name

Name of the command that executed. If the command name exceeds the maximum path length permitted by the OS, it is truncated at the maximum length.

line-number

Number of lines in the script on which the command is specified

exit-status-code

Command's return code

execution-time-in-seconds

Command execution time (a reference value obtained by using the OS's API)

CPU-time-in-seconds

Command's CPU time (a reference value obtained by using the OS's API)

(S)

Resumes processing.

KNAX6115-I

Execution of the command *command-name* (line=*line-number*) finished. exit status=*exit-status-code* execution time=*execution-time-in-seconds* CPU time=*CPU-time-in-seconds*

The command was terminated. This message is displayed when a command whose result (success or failure) cannot be determined from the return code was terminated.

command-name

Name of the command that executed. If the command name exceeds the maximum path length permitted by the OS, it is truncated at the maximum length.

line-number

Number of lines in the script on which the command is specified

exit-status-code

Command's return code

execution-time-in-seconds

Command execution time (a reference value obtained by using the OS's API)

CPU-time-in-seconds

Command's CPU time (a reference value obtained by using the OS's API)

(S)

Resumes processing.

KNAX6116-I

Execution of the command *command-name* (line=*line-number*) finished successfully. exit status=*exit-status-code* execution time=*execution-time-in-seconds* CPU time=*CPU-time-in-seconds*

The command terminated normally.

command-name

Name of the command that executed. If the command name exceeds the maximum path length permitted by the OS, it is truncated at the maximum length.

line-number

Number of lines in the script on which the command is specified

exit-status-code

Command's return code

execution-time-in-seconds

Command execution time (a reference value obtained by using the OS's API)

CPU-time-in-seconds

Command's CPU time (a reference value obtained by using the OS's API)

(S)

Resumes processing.

KNAX6117-I

Execution of the command *command-name* (line=*line-number*) finished. exit status=*exit-status-code* execution time=*execution-time-in-seconds* CPU time=*CPU-time-in-seconds*

The command was terminated. This message is displayed when a command whose result (success or failure) cannot be determined from the return code was terminated.

command-name

Name of the command that executed. If the command name exceeds the maximum path length permitted by the OS, it is truncated at the maximum length.

line-number

Number of lines in the script on which the command is specified

exit-status-code

Command's return code

execution-time-in-seconds

Command execution time (a reference value obtained by using the OS's API)

CPU-time-in-seconds

Command's CPU time (a reference value obtained by using the OS's API)

(S)

Resumes processing.

KNAX6120-I

Execution of the command *command-name* for the function *function-name* finished successfully. exit status=*exit-status-code* execution time=*execution-time-in-seconds* CPU time=*CPU-time-in-seconds*

The command executed by the function indicated by *function-name* terminated normally.

command-name

Name of the command that executed. If the command name exceeds the maximum path length permitted by the OS, it is truncated at the maximum length.

function-name

One of the following is displayed as the name of the function that executed the command:

- `command substitution`: Command substitution function
- `trap action`: Action of the `trap` command

exit-status-code

Command's return code

execution-time-in-seconds

Command execution time (a reference value obtained by using the OS's API)

CPU-time-in-seconds

Command's CPU time (a reference value obtained by using the OS's API)

(S)

Resumes processing.

KNAX6121-I

Execution of the command *command-name* for the function *function-name* finished. exit status=*exit-status-code* execution time=*execution-time-in-seconds* CPU time=*CPU-time-in-seconds*

The command executed by the function indicated by *function-name* was terminated. This message is displayed when a command whose result (success or failure) cannot be determined from the return code was terminated.

command-name

Name of the command that executed. If the command name exceeds the maximum path length permitted by the OS, it is truncated at the maximum length.

function-name

One of the following is displayed as the name of the function that executed the command:

- `command substitution`: Command substitution function
- `trap action`: Action of the `trap` command

exit-status-code

Command's return code

execution-time-in-seconds

Command execution time (a reference value obtained by using the OS's API)

CPU-time-in-seconds

Command's CPU time (a reference value obtained by using the OS's API)

(S)

Resumes processing.

KNAX6122-I

Execution of the command *command-name* for the function *function-name* finished successfully. exit status=*exit-status-code* execution time=*execution-time-in-seconds* CPU time=*CPU-time-in-seconds*

The command executed by the function indicated by *function-name* terminated normally.

command-name

Name of the command that executed. If the command name exceeds the maximum path length permitted by the OS, it is truncated at the maximum length.

function-name

One of the following is displayed as the name of the function that executed the command:

- `command substitution`: Command substitution function
- `trap action`: Action of the `trap` command

exit-status-code

Command's return code

execution-time-in-seconds

Command execution time (a reference value obtained by using the OS's API)

CPU-time-in-seconds

Command's CPU time (a reference value obtained by using the OS's API)

(S)

Resumes processing.

KNAX6123-I

Execution of the command *command-name* for the function *function-name* finished. exit status=*exit-status-code* execution time=*execution-time-in-seconds* CPU time=*CPU-time-in-seconds*

The command executed by the function indicated by *function-name* was terminated. This message is displayed when a command whose result (success or failure) cannot be determined from the return code was terminated.

command-name

Name of the command that executed. If the command name exceeds the maximum path length permitted by the OS, it is truncated at the maximum length.

function-name

One of the following is displayed as the name of the function that executed the command:

- `command substitution`: Command substitution function
- `trap action`: Action of the `trap` command

exit-status-code

Command's return code

execution-time-in-seconds

Command execution time (a reference value obtained by using the OS's API)

CPU-time-in-seconds

Command's CPU time (a reference value obtained by using the OS's API)

(S)

Resumes processing.

KNAX6124-I

Execution of the command *command-name* for the function *function-name* finished successfully. exit status=*exit-status-code* execution time=*execution-time-in-seconds* CPU time=*CPU-time-in-seconds*

The command executed by the function indicated by *function-name* terminated normally.

command-name

Name of the command that executed. If the command name exceeds the maximum path length permitted by the OS, it is truncated at the maximum length.

function-name

One of the following is displayed as the name of the function that executed the command:

- `command substitution`: Command substitution function
- `trap action`: Action of the `trap` command

exit-status-code

Command's return code

execution-time-in-seconds

Command execution time (a reference value obtained by using the OS's API)

CPU-time-in-seconds

Command's CPU time (a reference value obtained by using the OS's API)

(S)

Resumes processing.

KNAX6125-I

> Execution of the command *command-name* for the function *function-name* finished. exit status=*exit-status-code* execution time=*execution-time-in-seconds* CPU time=*CPU-time-in-seconds*

The command executed by the function indicated by *function-name* was terminated. This message is displayed when a command whose result (success or failure) cannot be determined from the return code was terminated.

command-name

Name of the command that executed. If the command name exceeds the maximum path length permitted by the OS, it is truncated at the maximum length.

function-name

One of the following is displayed as the name of the function that executed the command:

- `command substitution`: Command substitution function
- `trap action`: Action of the `trap` command

exit-status-code

Command's return code

execution-time-in-seconds

Command execution time (a reference value obtained by using the OS's API)

CPU-time-in-seconds

Command's CPU time (a reference value obtained by using the OS's API)

(S)

Resumes processing.

KNAX6126-I

> Execution of the command *command-name* for the function *function-name* finished successfully. exit status=*exit-status-code* execution time=*execution-time-in-seconds* CPU time=*CPU-time-in-seconds*

The command executed by the function indicated by *function-name* terminated normally.

command-name

Name of the command that executed. If the command name exceeds the maximum path length permitted by the OS, it is truncated at the maximum length.

function-name

One of the following is displayed as the name of the function that executed the command:

- `command substitution`: Command substitution function
- `trap action`: Action of the `trap` command

exit-status-code

Command's return code

execution-time-in-seconds

Command execution time (a reference value obtained by using the OS's API)

CPU-time-in-seconds

Command's CPU time (a reference value obtained by using the OS's API)

(S)

Resumes processing.

KNAX6127-I

Execution of the command *command-name* for the function *function-name* finished. exit status=*exit-status-code* execution time=*execution-time-in-seconds* CPU time=*CPU-time-in-seconds*

The command executed by the function indicated by *function-name* was terminated. This message is displayed when a command whose result (success or failure) cannot be determined from the return code was terminated.

command-name

Name of the command that executed. If the command name exceeds the maximum path length permitted by the OS, it is truncated at the maximum length.

function-name

One of the following is displayed as the name of the function that executed the command:

- `command substitution`: Command substitution function
- `trap action`: Action of the `trap` command

exit-status-code

Command's return code

execution-time-in-seconds

Command execution time (a reference value obtained by using the OS's API)

CPU-time-in-seconds

Command's CPU time (a reference value obtained by using the OS's API)

(S)

Resumes processing.

KNAX6130-E

An I/O error related to an event file occurred. filename="*event-file-path*"

An I/O error occurred when job definition script operation information was output to the event file.

event-file-path

Path name of the event file resulting in the I/O error

(S)

Resumes processing.

(O)

Determine the cause of the I/O error by referencing the messages output before and after this message, and then correct the error.

KNAX6200-I

Usage: *command-name command-argument*

This message displays the syntax of the command arguments for the command indicated by *command-name*.

(S)

Terminates processing.

KNAX6201-E

No value is specified for the option "*option-name*".

No value is specified for the option indicated by *option-name*.

(S)

Terminates processing.

(O)

Specify a value for the displayed option.

KNAX6202-E

The specified option "*option-name*" is invalid.

An unknown option name, indicated by *option-name*, was specified.

(S)

Terminates processing.

(O)

Specify the correct option.

KNAX6203-E

The specified option value "*option-value*" is invalid.

An option value that was specified is invalid. *option-value* indicates the specified option value.

(S)

Terminates processing.

(O)

Specify the correct value for the option.

KNAX6204-E

The option "*option-name*" is specified concurrently with an option that it cannot be specified with.

The option indicated by *option-name* cannot be specified together with some other mutually exclusive option.

(S)

Terminates processing.

(O)

Specify the correct combination of options.

KNAX6206-E

No asc file name is specified as a command argument.

No `asc` file is specified in the command's argument.

(S)

Terminates processing.

(O)

Specify an `asc` file in the command's argument.

KNAX6207-E

Too many options are specified.

The command has too many arguments.

(S)

Terminates processing.

(O)

Specify the correct arguments in the command.

KNAX6208-E

One or more operands are missing.

There are not enough arguments in the command.

(S)

Terminates processing.

(O)

Specify the correct arguments in the command.

KNAX6209-W

The specified line number range falls outside of the actual range for script line numbers.

A specified range of line numbers is not within the range of line numbers constituting the job definition script.

(S)

Interprets the specified range of line numbers as described below and resumes processing:

- If the start line number is outside the range of line numbers in the job definition script, the system ignores the invalid range specification.
- If the start line number is within the range of line numbers in the job definition script but the end line number is outside the range of line numbers in the job definition script, the system assumes the largest line number in the job definition script as the end line number.

If there are no valid ranges of lines as a result of ignoring the invalid range specification, the system outputs only the header information at the top. Neither the job definition script nor the coverage information is output.

(O)

Specify the correct line numbers.

KNAX6210-E

Failed to open the asc file "*file-name*". reason="*error-details*"

An open error occurred on the `asc` file indicated by *file-name*.

file-name might be the name of an `asc` file specified in the command, or it might be the name of an `asc` file created temporarily by the command.

(S)

Terminates processing.

(O)

Determine the cause of the `asc` file open error, correct the error, and then re-execute the command.

In Windows, `No such file or directory` is displayed for *error-details* if a directory containing the directory separator (\) is specified at the end of the path name for the `asc` file in the `adshexec` command. In this case, specify a file name, not the directory.

KNAX6211-E

Failed to lock the asc file "*file-name*". reason="*error-details*"

The `asc` file indicated by *file-name* cannot be locked.

error-details displays the nature of the file lock error.

file-name might be the name of an `asc` file specified in the command, or it might be the name of an `asc` file created temporarily by the command.

(S)

Terminates processing.

(O)

Determine the cause of the `asc` file lock error, correct the error, and then re-execute the command. Most often, an `asc` file lock error occurs because another program is using the `asc` file.

KNAX6212-E

Failed to read the asc file "*file-name*". reason="*error-details*"

A read error occurred in the `asc` file indicated by *file-name*.

file-name might be the name of an `asc` file specified in the command, or it might be the name of an `asc` file created temporarily by the command.

(S)

Terminates processing.

(O)

Determine the cause of the `asc` file read error, correct the error, and then re-execute the command.

KNAX6213-E

The format of the asc file "*file-name*" is invalid. details=*maintenance-information*

A format error was detected in the `asc` file indicated by *file-name*.

file-name might be the name of an `asc` file specified in the command, or it might be the name of an `asc` file created temporarily by the command.

(S)

Terminates processing.

(O)

Delete the `asc` file indicated by *file-name* and collect new coverage information. Check if the `asc` file was updated illegally. If this is not the problem but the same event recurs, save the indicated `asc` file and contact the product provider.

KNAX6214-E

> Failed to update the asc file "*file-name*". reason="*error-details*"

An updating error occurred in the `asc` file indicated by *file-name*.

file-name might be the name of an `asc` file specified in the command, or it might be the name of an `asc` file created temporarily by the command.

(S)

Terminates processing.

(O)

Determine the cause of the `asc` file read error, correct the error, and then re-execute the command.

KNAX6215-E

> Failed to unlock the asc file "*file-name*". reason="*error-details*"

An error occurred when an attempt was made to unlock the `asc` file indicated by *file-name*.

file-name might be the name of an `asc` file specified in the command, or it might be the name of an `asc` file created temporarily by the command.

(S)

Terminates processing.

(O)

Determine the cause of the `asc` file unlock error, correct the error, and then re-execute the command.

KNAX6219-E

> The script "*job-definition-script-name*" and the asc file "*asc-file-name*" contain different script data.

The coverage information cannot be accumulated in the `asc` file indicated by *asc-file-name* because the following job definition scripts are not the same:

- Job definition script indicated by *job-definition-script-name*
- Job definition script used to collect the coverage information in the `asc` file indicated by *asc-file-name*

(S)

Terminates processing.

(O)

Take one of the following actions:

1. When accumulating the coverage information in the `asc` file indicated by *asc-file-name*

   Use the job definition script file used to collect the coverage information in the `asc` file indicated by *asc-file-name.*

2. When not accumulating the coverage information in the `asc` file indicated by *asc-file-name*

   If the coverage information in the `asc` file indicated by *asc-file-name* is not needed, specify the `-f` option in the `adshexec` command. The existing coverage information will be discarded and new coverage information will be stored (during the initial accumulation).

   If you need the coverage information in the `asc` file indicated by *asc-file-name*, specify the output `asc` file with the `-o` option in the `adshexec` command. Check that the specified `asc` file does not already exist at the output destination.

KNAX6220-I

> The two asc files will now be merged. output file="*file-name*"

Coverage information merge processing has started.

file-name indicates the name of the `asc` file that will store the merge results.

(S)

Resumes processing.

KNAX6221-I

> Base file="*file-name-1*", file to be merged="*file-name-2*"

file-name-1 indicates the base `asc` file.

file-name-2 indicates the merge `asc` file.

(S)

Resumes processing.

KNAX6222-I

> The two asc files have been merged. output file="*asc-file-name*"

Coverage information merge processing has finished.

asc-file-name indicates the name of the `asc` file containing the merged results.

(S)

Resumes processing.

KNAX6223-E

> The two asc files "*file-name-1*" and "*file-name-2*" contain different script data.

The job definition scripts used to collect coverage information differ between `asc` files *file-name-1* and *file-name-2*.

file-name-1 indicates the base `asc` file.

file-name-2 indicates the merge `asc` file.

(S)

Terminates processing.

(O)

Specify in the `adshcvmerg` command as the base and merge `asc` files `asc` files obtained by using the same job definition script.

KNAX6225-E

An internal error occurred. details=*maintenance-information*

An internal conflict was detected during processing.

(S)

Terminates processing.

(O)

Contact the product provider and provide the displayed *maintenance-information*.

KNAX6226-E

The number of nested control statements exceeded the limit.

There are too many nesting levels in control statements.

(S)

Terminates processing.

(O)

If you need to collect coverage information, edit the job definition script to avoid too many nesting levels in control statements.

KNAX6227-E

The version "*version-number*" in the asc file is not supported by this command.

An `asc` file's version number is not supported by the command.

version indicates the version number.

(S)

Terminates processing.

(O)

Specify an `asc` file whose version is supported by the command.

KNAX6228-E

Failed to get time information. reason="*error-details*"

An error occurred while obtaining the date and time.

(S)

Terminates processing.

(O)

Determine the cause of the date and time acquisition error, and then take appropriate action. The `time` function is used to obtain the date and time.

KNAX6229-E

> Failed to get information about the file "*file-name*". reason="*error-details*"

An error occurred while obtaining information about the file indicated by *file-name*.

file-name might be the name of an `asc` file specified in the command, or it might be the name of an `asc` file created temporarily by the command.

(S)

Terminates processing.

(O)

Check the cause of the file information acquisition error, and then correct the error. This error often occurs when the user does not have the permission to access the file.

KNAX6231-E

> The asc file to be merged is the same file as the base asc file.

The merge `asc` file is the same as the base `asc` file.

(S)

Terminates processing.

(O)

Specify different merge and base `asc` files whose coverage information is to be merged.

KNAX6232-E

> Failed to get the user name.

The name of the user executing the command cannot be obtained.

(S)

Terminates processing.

(O)

Determine why the name of the user executing the command cannot be obtained, and then correct the error. In UNIX, the name of the user executing the `/etc/passwd` command might not be registered.

KNAX6233-E

> The file "*file-name*" already exists.

The file indicated by *file-name* already exists.

file-name might be the name of an `asc` file specified in the command, or it might be the name of an `asc` file created temporarily by the command.

(S)

Terminates processing.

(O)

Rename or delete the file, inasmuch as the processing cannot be performed because the file indicated by *file-name* already exists.

KNAX6236-E

The file "*file-name*" is not a regular file.

The file indicated by *file-name* is not a regular file.

file-name might be the name of an `asc` file specified in the command, or it might be the name of an `asc` file created temporarily by the command.

(S)

Terminates processing.

(O)

Check the type of the file indicated by *file-name*.

KNAX6237-E

The path of the *file-type* is the same as that of the script file.

The path name of the job definition script file is the same as the path name of an `asc` file (including an `asc` file created by the command).

file-type

`asc` file whose path name is the same as the path name of the job definition script file:

- `"asc file"`: The path name of an omitted or specified `asc` file is the same as the path name of the job definition script file.
- `"temporary asc file"`: The path name of a temporary `asc` file is the same as the path name of the job definition script file.
- `"backup asc file"`: The path name of a backup `asc` file is the same as the path name of the job definition script file.

(S)

Terminates processing.

(O)

Specify the path name of the `asc` file explicitly so that it differs from the path name of the job definition script file. If the path name of the `asc` file was already specified explicitly, change the explicitly specified path name so that it differs from the path name of the job definition script file.

KNAX6238-E

Failed to rename the asc file "*file-name*". reason="*error-details*"

An error occurred while renaming the `asc` file indicated by *file-name*.

file-name might be the name of an `asc` file specified in the command, or it might be the name of an `asc` file created temporarily by the command.

(S)

Terminates processing.

(O)

Determine the cause of the `asc` file renaming error, and then correct the error. The write-protect settings or access permissions for the file indicated by *file-name* might have been changed during command execution.

KNAX6239-E

Failed to remove the asc file "*file-name*". reason="*error-details*"

An error occurred while deleting the `asc` file indicated by *file-name*.

file-name might be the name of an `asc` file specified in the command, or it might be the name of an `asc` file created temporarily by the command.

(S)

Terminates processing.

(O)

Determine the cause of the `asc` file deletion error, and then correct the error. The file indicated by *file-name* might be write-protected or it might have been created during command execution file with no access permission.

KNAX6240-E

Failed to set the access position in the asc file "*file-name*". reason="*error-details*"

An error occurred while positioning the `asc` file indicated by *file-name*.

file-name might be the name of an `asc` file specified in the command, or it might be the name of an `asc` file created temporarily by the command.

(S)

Terminates processing.

(O)

Determine the cause of the `asc` file positioning error, and then correct the error.

KNAX6241-E

The input asc file is the same as the output asc file.

The same files were specified in the `adshcvmerg` command for the output file and an input `asc` file (base or merge `asc` file).

(S)

Terminates processing.

(O)

Specify for the output `asc` file a different file than that of either input `asc` file.

KNAX6242-I

The asc file "*file-name*" was updated.

The coverage information in an `asc` file was updated by the `adshexec` command. *file-name* indicates the path name of the updated `asc` file.

(S)

Resumes processing.

KNAX6243-I

The asc file "*file-name*" was restored from a backed-up asc file.

An `asc` file was restored from its backup `asc` file by the `adshexec` command. *file-name* indicates the path name of the restored `asc` file.

(S)

Resumes processing.

KNAX6244-E

Failed to truncate the asc file "*file-name*". reason="*error-details*"

An error occurred while initializing the `asc` file indicated by *file-name*.

In Windows, the error occurred in the `_chsize` function.

In UNIX, the error occurred in the `ftruncate` function.

(S)

Terminates processing.

(O)

Determine the cause of the `asc` file initialization error, and then correct the error.

KNAX6290-E

The settings of the coverage view program are invalid. details code=*maintenance-information*

The settings for the program used to start the display program are invalid.

(S)

Terminates processing.

(O)

Contact the product provider and provide the *maintenance-information*.

KNAX6291-E

Startup of the coverage view program failed. error details=*error-details*

Startup of the coverage information display program failed. *error-details* provide the reason for the startup error.

(S)

Terminates processing.

(O)

Determine the cause of the startup error, and then correct the error.

KNAX6292-E

The file "*file-name*" is not a regular file.

The file indicated by *file-name* is not a regular file.

(S)

Terminates processing.

(O)

Check the type of file indicated by *file-name*.

KNAX6293-E

Failed to get information about the file "*file-name*". reason="*error-details*"

An error occurred while obtaining information about the file indicated by *file-name*.

(S)

Terminates processing.

(O)

Determine the cause of the file information acquisition error, and then correct the error. The cause most often is that the user does not have the permission to access the file.

KNAX6294-E

Failed to open the file "*file-name*" used by the coverage view program. reason="*error-detail*"

An open error occurred in a file used by the coverage information display program.

(S)

Terminates processing.

(O)

Determine the cause of the open error in the file used by the coverage information display program, correct the error, and then display the coverage information.

KNAX6295-E

Failed to execute the coverage view program. (reason=*error-details*)

Execution of the coverage display program failed for the reason indicated by *error-details*.

(S)

Terminates processing.

(O)

Eliminate the cause of the error, and then re-execute the command. If the problem cannot be resolved, contact the system administrator.

KNAX6296-E

Failed to update the file "*file-name*" used by the coverage view program. reason="*error-detail*"

An error occurred while updating the file indicated by *file-name* that is used by the coverage information display program.

(S)

Terminates processing.

(O)

Determine the cause of the write error in the file that is used by the coverage information display program, correct the error, and then re-execute the command.

KNAX6297-E

Failed to get the temporary directory. reason="*error-details*"

An error occurred while obtaining information about the temporary directory.

(S)

Terminates processing.

(O)

Determine the cause of the error in obtaining information about the temporary directory, and then correct the error.

KNAX6298-E

An error occurred during an attempt to view coverage. response code=*error-details*

An error occurred while displaying coverage information. For details about the information displayed as the return code, see the description of the `adshcvshow` command's return value.

(S)

Terminates processing.

(O)

Determine the cause of the error in displaying coverage information, and then correct the error. The message issued when the error occurred might be stored in the coverage information accumulation file.

If no job definition script has been executed, response code `6` is set. Display coverage information after you have executed a job definition script.

KNAX6301-E

The coverage option is specified in batch coverage mode.

A batch job was executed with coverage information collected in the batch mode, but the `adshexec` command was executed with the `-t` option specified.

(S)

Cancels execution.

(O)

Re-execute the batch job with the `-t` option omitted.

KNAX6302-E

The length of the asc file name exceeds the limit for the batch coverage function.

Coverage information was to be collected in the batch mode, but the name of the `asc` file to be used exceeded the maximum length.

If `asc` files are created in conformity with the `asc` file naming rules specified in the environment file, the created file name exceeds the maximum length.

(S)

Cancels execution.

(O)

Check and, if necessary, revise the specification in the environment file and the name of the job definition script to be executed.

KNAX6303-E

Failed to duplicate file descriptors. filename="*file-name*" error="*error-details*"

Duplication of the file ID of *file-name* failed.

file-name

File name of the job definition script

error-details

Details of the error

(S)

Cancels execution.

(O)

Eliminate the cause of the error, and then re-execute the batch job.

KNAX6304-E

Initialization failed.

The JP1/Advanced Shell initialization processing failed.

(S)

Terminates processing.

(O)

Check if the job definition script file is correct.

KNAX6305-E

The existence of the script file could not be verified. filename="*file-name*" error="*error-details*"

The existence of the job definition script file cannot be confirmed.

file-name

File name of the job definition script

error-details

Details of the error

(S)

Terminates processing.

(O)

Check if the job definition script file is correct.

KNAX6306-E

Failed to read the script file. filename="*file-name*" function=*function-name* error="*error-details*"

A read error occurred in the job definition script file.

file-name

File name of the job definition script

function-name

Name of the function resulting in the error

error-details

Details of the error

(S)

Terminates processing.

(O)

Check if the job definition script file is correct.

KNAX6307-W

A line was truncated because the limit on line length was exceeded. filename="*file-name*" line=*line-number*

A line exceeds the maximum length of a line of a job definition script that can be displayed. The line is displayed with the excess part discarded.

file-name

File name of the job definition script

line-number

Line number in the job definition script

(S)

Displays the line with the excess part discarded, and then resumes processing.

(O)

Correct the job definition script file.

KNAX6308-E

The script file is empty. filename="*file-name*"

The job definition script file is empty.

file-name

File name of the job definition script

(S)

Terminates processing.

(O)

Correct the job definition script file.

KNAX6309-E

Failed to output SCRIPT file to the spool job directory. reason="*error-details*"

An output error occurred in the job definition script file in the spool.

error-details

Details of the error

(S)

Terminates processing.

(O)

Correct the job definition script file.

KNAX6310-E

The item "*item-name*" is incorrect. filename="*file-name*" line=*line-number*

The item indicated by *item-name* in an extended script command is invalid.

item-name

Name of an option or positional operand in the extended script command

file-name

File name of the job definition script

line-number

Line number in the job definition script

If the error is on a continuation line, the message might display the number of the first line of the extended script command, not the line number of the continuation line.

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6311-E

The item "*item-name*" is not specified. filename="*file-name*" line=*line-number*

The item indicated by *item-name* is missing in an extended script command.

item-name

Name of an option or positional operand in the extended script command

file-name

File name of the job definition script

line-number

Line number in the job definition script

If the error is on a continuation line, the message might display the number of the first line of the extended script command, not the line number of the continuation line.

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6312-E

The value specified for the option "*option-name*" is invalid. filename="*file-name*" line=*line-number*

An invalid value was specified for *option-name* in an extended script command.

option-name

Option name in the extended script command

file-name

File name of the job definition script

line-number

Line number in the job definition script

If the error is on a continuation line, the message might display the number of the first line of the extended script command, not the line number of the continuation line.

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6313-E

The option "*option-name*" is specified multiple times. filename="*file-name*" line=*line-number*

The indicated option name is specified multiple times in an extended script command.

option-name

Option name in the extended script command

file-name

File name of the job definition script

line-number

Line number in the job definition script

If the error is on a continuation line, the message might display the number of the first line of the extended script command, not the line number of the continuation line.

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6314-E

The expression "*character-string*" is incomplete. filename="*file-name*" line=*line-number*

The specification indicated by *character-string* is not complete in an extended script command. A paired double quotation mark or single quotation mark might be missing, or there might be no escape character following a backslash (\).

character-string

Specification in the extended script command

file-name

File name of the job definition script

line-number

Line number in the job definition script

If the error is on a continuation line, the message might display the number of the first line of the extended script command, not the line number of the continuation line.

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6315-E

The order for the command "*command-name*" is incorrect. filename="*file-name*" line=*line-number*

The cause might be one of the following:

- The location of an extended script command is invalid.
- The combination of an extended script command and another command is invalid.
- An extended script command is not specified at the beginning.

command-name

Name of the extended script command

file-name

File name of the job definition script

line-number

Line number in the job definition script

If the error is on a continuation line, the message might display the number of the first line of the extended script command, not the line number of the continuation line.

If the error was detected during execution, rather than during syntax analysis, `0` might be displayed as the line number. For example, this error might be detected during execution when an extended script command is used as a command substitution format.

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6316-E

An instruction exceeds the length limit. filename="*file-name*" line=*line-number*

The length of an extended script command exceeded the maximum.

file-name

File name of the job definition script

line-number

Line number in the job definition script

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6317-E

A continuation line is incorrectly specified. filename="*file-name*" line=*line-number*

The specification of a continuation line is invalid.

file-name

File name of the job definition script

line-number

Line number in the job definition script

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6318-E

An invalid instruction is specified. command="*command-name*" filename="*file-name*" line=*line-number*

An invalid extended script command name was specified.

command-name

Name of the extended script command

file-name

File name of the job definition script

line-number

Line number in the job definition script

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6319-E

> The number of "*item-name*" specified exceeds the upper limit. filename="*file-name*" line=*line-number*

The maximum number of instances of the item indicated by *item-name* has already been reached.

item-name

Extended script command name, option name, or argument

file-name

File name of the job definition script

line-number

Line number in the job definition script

If the error is on a continuation line, the message might display the number of the first line of the extended script command, not the line number of the continuation line.

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6320-E

> An error occurred in the script file. filename="*file-name*" function=*function-name* error="*error-details*"

An error occurred in a job definition script file.

file-name

File name of the job definition script

function-name

Name of the function where the error occurred

error-details

Details of the error. This is an error information string representation for `errno`.

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6321-E

> The attribute "-run {abnormal|always}" cannot be specified in this location. file name="*file-name*" line=*line-number*

A specified option is not permitted.

file-name

File name of the job definition script

line-number

Line number in the job definition script

If the error is on a continuation line, the message might display the number of the first line of the extended script command, not the line number of the continuation line.

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6323-E

The export parameter processing failed. line=*line-number*

A specified `export` parameter is invalid, or an environment variable cannot be specified.

line-number

Line number in the environment file

Following are the possible causes:

- The name or value of the environment variable exceeds the maximum length.

  If you add paths to the `PATH` environment variable, make sure that the maximum length is not exceeded as a result of paths being duplicated in child jobs.
- The name of the environment variable is invalid.
- A combination of `\`, `"`, or `'` symbols is invalid.

(S)

Terminates processing.

(O)

Correct the environment file.

KNAX6324-E

A file name could not be converted into an absolute path. initial filename="*file-name-before-conversion*" error="*error-details*" filename="*file-name*" line=*line-number*

The file indicated by *file-name-before-conversion* cannot be converted to an absolute path.

file-name-before-conversion

File name before conversion

error-details

Details of the error

file-name

File name of the job definition script

line-number

Line number in the job definition script

If the error is on a continuation line, the message might display the number of the first line of the extended script command, not the line number of the continuation line.

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6325-E

A block or function definition ended without finishing an internal job step. filename="*file-name*" line=*line-number*

The block or function definition ended before a job step within the block or function definition ended.

file-name

File name of the job definition script

line-number

Line number in the job definition script

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6326-E

The beginning of a job step is not defined. filename="*file-name*" line=*line-number*

There is no job step start definition.

file-name

File name of the job definition script

line-number

Line number in the job definition script

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6327-E

Nested definitions are used to define a job step. filename="*file-name*" line=*line-number*

Job step definitions are nested.

file-name

File name of the job definition script

line-number

Line number in the job definition script

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6328-E

Nested definitions are used to define the command "*command-name*". filename="*file-name*" line=*line-number*

command-name definitions are nested.

command-name

Name of the extended script command

file-name

File name of the job definition script

line-number

Line number in the job definition script

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6329-E

An instruction might have been altered partway. filename="*file-name*" line=*line-number*

An extended script command might have been changed during execution.

file-name

File name of the job definition script

line-number

Line number in the job definition script

(S)

Terminates processing.

(O)

Ensure that the job definition script will not change during execution, and then re-execute it.

KNAX6330-E

A script is called recursively. filename="*file-name*" line=*line-number*

A job definition script was called recursively.

file-name

File name of the job definition script

line-number

Line number in the job definition script

If the error is on a continuation line, the message might display the number of the first line of the extended script command, not the line number of the continuation line.

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6332-E

The script finished without closing a job step.

The job definition script ended, but there is a job step that has not been closed.

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6333-E

The existence of the file "*file-name*" could not be verified. error="*error-details*" filename="*job-definition-script-file-name*" line=*line-number*

The existence of *file-name* cannot be confirmed.

error-details

Details of the error

job-definition-script-file-name

File name of the job definition script

line-number

Line number in the job definition script

If the error is on a continuation line, the message might display the number of the first line of the extended script command, not the line number of the continuation line.

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6380-I

A job name will be added to the spool job directory of the root job. spool job directory="*spool-job-directory*"

A job name will be added to the root job's *spool-job-directory*.

spool-job-directory

Spool job directory name after the change

(S)

Resumes processing.

KNAX6381-E

Failed to change the name of the spool job directory. error="*error-details*" job ID="*job-ID*" jobname=*job-name*

Renaming of the spool job directory failed. The spool job directory still has the job ID. Possible causes are as follows:

- A spool job directory with the new name already exists.
- In Windows, if a job that executes an external command to create a child process is terminated forcibly, this message might be output with `Permission denied` displayed in place of *error-details*. This occurs when more than 255 non-child processes exist at the same time.

error-details

Details of the error

job-ID

Job ID

job-name

Job name

(S)

Resumes processing.

(O)

If you reference the spool job directory, reference the spool job directory indicated in the `KNAX6382-I` message that is output at the same time.

If you want to delete the spool job directory, delete it manually without using the `adshhk` command.

KNAX6382-I

The spool job directory "*spool-job-directory*" will be used for storage because an attempt to change its name failed.

The spool job directory was stored as is because renaming of *spool-job-directory* failed.

spool-job-directory

Name of the spool job directory before renaming

(S)

Resumes processing.

KNAX6383-E (UNIX only)

Failed to modify permissions for a spool job. (path="*path-name*" function="*function-name*" error details="*error-details*")

An attempt to modify the permission for the directory or file of the spool job indicated by *path-name* failed.

function-name

OS's API name

error-details

This is an error information string representation for `errno`.

(S)

Continues processing if *function-name* is `chmod`; otherwise, terminates processing.

(O)

Investigate the cause of the error and take the necessary corrective action. To modify the permission for the directory or file of this spool job change, execute the `chmod` command.

KNAX6385-E

This extended script command cannot be used with the current environment setting parameter. command name="*command-name*" parameter="*environment-setting-parameter*" filename="*file-name*" line=*line-number*

The indicated extended script command cannot be used with the current setting of the indicated environment setting parameter.

command-name

Name of the extended script command resulting in the error

environment-setting-parameter

Environment setting parameter and value that caused this error

file-name

Name of the job definition script file

line-number

Line number in the job definition script

If the error is on a continuation line, the message might display the number of the first line of the extended script command, not the line number of the continuation line.

(S)

Terminates processing.

(O)

Do not execute this command. If you want to execute this command, check and revise the specified environment setting parameter as necessary.

KNAX6399-E

A fatal error occurred. function=*function-name*, line=*line-number*

An unexpected error occurred.

function-name

Function name

line-number

Line number

(S)

Terminates processing.

(O)

Contact the system administrator.

KNAX6400-E

Failed to allocate the file "*file-environment-variable-definition*".

Allocation of the file indicated by *file-environment-variable-definition* failed.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the job definition script file.

KNAX6401-E

The file "*file-environment-variable-definition*" does not exist. file path=*file-path*

The file with the indicated *file-path* specified in *file-environment-variable-definition* does not exist.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the job definition script file.

KNAX6403-E

The temporary file ID "*temporary-file-ID*" is already defined.

A file has already been defined with the indicated *temporary-file-identifier*.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the job definition script file.

KNAX6404-E

Failed to create the environment variable "*file-environment-variable-definition*".

Creation of the environment variable indicated by *file-environment-variable-definition* failed.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the job definition script file.

KNAX6405-E

The file "*file-environment-variable-definition*" does not exist.

The file indicated by *file-environment-variable-definition* does not exist.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the job definition script file.

KNAX6406-E

Failed to verify the file defined by "*file-environment-variable-definition*". reason="*error-details*"

The error indicated by *error-details* occurred when an attempt was made to use the `stat` function to check the file indicated by *file-environment-variable-definition*.

error-details

Details of the error. This is an error information string representation for `errno`.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the job definition script file.

KNAX6407-E

The directory "*file-environment-variable-definition*" already exists.

The directory indicated by *file-environment-variable-definition* already exists.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the job definition script file.

KNAX6408-E

Failed to create the file "*file-environment-variable-definition*". reason="*error-details*"

The error indicated by *error-details* occurred when an attempt was made to create the file indicated by *file-environment-variable-definition*.

error-details

Details of the error. This is an error information string representation for `errno`.

(S)

Terminates processing.

(O)

Check and, if necessary, revise the job definition script file.

KNAX6409-I

The file *file-environment-variable-definition* was allocated as "*processing-value*". path=*file-path*[(*file-existence*)]

The file indicated by *file-path* was allocated according to *value-specified-for-processing* in the file definition indicated by *file-environment-variable-definition-name*.

processing-value

The following is displayed, depending on the command:

For the `#-adsh_file` or `#-adsh_file_temp` command

The value specified for `-chk`

For the `#-adsh_spoolfile` command

`spoolfile`

file-existence

This information is displayed only for a regular file. If the file exists, `exist` is displayed; if the file does not exist, `not exist` is displayed.

(S)

Resumes processing.

KNAX6410-I

The file *file-environment-variable-definition* was deallocated as "*processing-value*". path=*file-path*

The file indicated by *file-path* was released according to *processing-value* in the file definition indicated by *file-environment-variable-definition*.

processing-value

The following is displayed, depending on the command:

For the `#-adsh_file` or `#-adsh_file_temp` command

One of the following values is displayed as the specified postprocessing of files during normal times or abnormal times:

- `del`: Delete.
- `keep`: Do not delete.

For the `#-adsh_spoolfile` command

`spoolfile`

(S)

Resumes processing.

KNAX6411-E

Failed to set the shell variable "*file-environment-variable-definition*". details=*maintenance-information*

Specification of the shell variable indicated by *file-environment-variable-definition* failed.

(S)

Terminates processing.

(O)

The shell variable indicated by *file-environment-variable-definition-name* might have the read-only attribute. Check and, if necessary, revise the job definition script file. If there is no problem, contact the system administrator.

KNAX6412-E

Failed to get a shell variable. details=*maintenance-information*

Acquisition of a shell variable failed.

(S)

Terminates processing.

(O)

Contact the system administrator.

KNAX6413-E

The shell variable "*file-environment-variable-definition*" cannot be restored to a previous value. details=*maintenance-information*

The shell variable indicated by *file-environment-variable-definition* cannot be restored to its original value, or the current definition cannot be deleted because its original value was undefined.

(S)

Terminates processing.

(O)

Contact the system administrator.

KNAX6414-E

Failed to normalize the file name. API name=*API-name* reason="*cause*" details=*maintenance-information* filename="*script-file-name*" line=*line-number*

An error occurred while the file path specified by the `#adsh_file` command, which is an extended script command, was being normalized.

API-name

The name of the API in which the error occurred

cause

The following table explains the error details, the output contents, and their meanings:

No.	Output content	Meaning
1	Error information string representation for `errno`	See the applicable UNIX or Windows documentation.
2	`The path contains one or more invalid multibyte characters`	The path name contains an invalid multibyte character.
3	`The path contains too many components`	The path name contains too many components (more than 4,096).

line-number

The line number in the script

script-file-name

The file name of the script file

(S)

Terminates processing.

(O)

Eliminate the cause of the error based on *cause*. If the problem cannot be resolved, contact the system administrator.

KNAX6507-I

job-name.job-step-name step was skipped because of a run attribute.

The job step indicated by *job-step-name* that is defined in the batch job indicated by *job-name* was skipped because a condition specified in the `run` attribute in a job step definition command beginning with `#-adsh_step` was satisfied.

(S)

Resumes processing.

KNAX6508-I

job-name.*job-step-name* step was skipped because a previous step or command ended abnormally.

The job step indicated by *job-step-name* that is defined in the batch job indicated by *job-name* was skipped because the preceding job step or command terminated with an error.

(S)

Resumes processing.

KNAX6509-I

job-name.*job-step-name* step was not executed because of script context.

The job step indicated by *job-step-name* that is defined in the batch job indicated by *job-name* is not being executed due to job definition script control (such as an `if` control statement).

(S)

Resumes processing.

KNAX6510-W

The job controller cannot set the shell variable *shell-variable-name* because it is read-only. filename="*file-name*" line=*line-number*

An attempt was made by the job controller to set the shell variable indicated by *shell-variable-name* while the command indicated by *line-number* in the file indicated by *file-name* was running, but the attempt failed because the variable has the read-only attribute.

(S)

Resumes processing.

(O)

Check the job definition script and correct it if there is an error in the usage of the indicated shell variable.

KNAX6511-I

The value of the local shell variable PATH of the step will be passed in from outside the step. step=*job-step-name*

The `PATH` shell variable that is specified in the job step with the indicated *job-step-name* (and is in effect within the job step) inherits a value from a `PATH` shell variable defined outside the job step.

(S)

Resumes processing.

KNAX6512-I

The job controller was started by a custom job.

The job controller was started from a JP1/Advanced Shell custom job by JP1/AJS.

(S)

Resumes processing.

KNAX6521-E

The command *command-name* (line=*line-number*) failed. exit status=*exit-status-code* execution time=*execution-time-in-seconds* CPU time=*CPU-time-in-seconds*

The command terminated with an error.

command-name

Name of the command that executed. If the command name exceeds the maximum path length permitted by the OS, it is truncated at the maximum length.

line-number

Number of lines in the job definition script on which the command is specified

exit-status-code

Command's return code

execution-time-in-seconds

Command execution time. This is a reference value obtained by using the OS's API.

CPU-time-in-seconds

Command's CPU time. This is a reference value obtained by using the OS's API.

(S)

Resumes processing. If the erroneous command is in a job step, the system performs the following processing:

- If the job step's `onError` attribute is `stop`, the system executes the job step error block without executing the processing that follows the job step normal block.
- If the job step's `onError` attribute is `cont`, the system executes the processing that follows the job step normal block.

Note that if the job needs to be canceled on the basis of command execution results, the system issues the `KNAX6584-I` message and then terminates the processing without taking the above action.

(O)

Check the command's execution results, and then eliminate the cause of the error.

KNAX6522-E

The command *command-name* (line=*line-number*) ended abnormally because it received a signal. exit status=*exit-status-code* signal number=*signal-number* execution time=*execution-time-in-seconds* CPU time=*CPU-time-in-seconds*

The command terminated with an error by signal.

command-name

Name of the command that executed. If the command name exceeds the maximum path length permitted by the OS, it is truncated at the maximum length.

line-number

Number of lines in the job definition script on which the command is specified

exit-status-code

Command's return code

signal-number

Signal number received by the command

execution-time-in-seconds

Command execution time. This is a reference value obtained by using the OS's API.

CPU-time-in-seconds

Command's CPU time. This is a reference value obtained by using the OS's API.

(S)

Resumes processing. If the command that terminated with an error is in a job step, the system performs the following processing:

- If the job step's `onError` attribute is `stop`, the system executes the job step error block without executing the processing that follows the job step normal block.
- If the job step's `onError` attribute is `cont`, the system executes the processing that follows the job step normal block.

(O)

Check the command's execution results, and then eliminate the cause of the error.

KNAX6530-E

The specified combination of options is invalid. filename="*file-name*" line=*line-number*

There is an error in the combination of an option and the option's value.

file-name

Name of the job definition script file

line-number

Line number in the job definition script

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6531-E

The number of "*item-name*" exceeds the upper limit for the *scope*. filename="*file-name*" line=*line-number*

The number of instances of *item-name* exceeded the maximum.

item-name

Name of an extended script command

scope

`job` or `step`

file-name

Name of the job definition script file

line-number

Line number in the job definition script

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6540-I

A command group executed by another process (line=*line-number*) succeeded. exit status=*exit-status-code* execution time=*execution-time-in-seconds* CPU time=*CPU-time-in-seconds*

A job definition script that executed in another process, such as a command group, terminated normally.

lines-count

Number of lines in the job definition script that contains the job definition script executed in another process

exit-status-code

Return code of the job definition script that executed

execution-time-in-seconds

Execution time of the executed job definition script. This is a reference value obtained by using the OS's API.

CPU-time-in-seconds

CPU time of the executed job definition script. This is a reference value obtained by using the OS's API.

(S)

Resumes processing.

KNAX6541-E

A command group executed by another process (line=*line-number*) failed. exit status=*exit-status-code* execution time=*execution-time-in-seconds* CPU time=*CPU-time-in-seconds*

A job definition script that executed in another process, such as a command group, terminated with an error.

line-number

Number of lines in the script that contains the job definition script executed in another process

exit-status-code

Return code of the job definition script that executed in another process

execution-time-in-seconds

Execution time of the job definition script executed in another process. This is a reference value obtained by using the OS's API.

CPU-time-in-seconds

CPU time of the job definition script executed in another process. This is a reference value obtained by using the OS's API.

(S)

Resumes processing. If the job definition script that was executed in another process and that terminated with an error is in a job step, the system performs the following processing:

- If the job step's `onError` attribute is `stop`, the system executes the job step error block without executing the processing that follows the job step normal block.
- If the job step's `onError` attribute is `cont`, the system executes the processing that follows the job step normal block.

(O)

Check the execution results of the job definition script that was executed in another process and that terminated with an error, and then eliminate the cause of the error.

KNAX6542-E

A command group executed by another process (line=*line-number*) ended abnormally because it received a signal. exit status=*exit-status-code* signal number=*signal-number* execution time=*execution-time-in-seconds* CPU time=*CPU-time-in-seconds*

A job definition script that executed in another process, such as a command group, terminated with an error by signal.

line-number

Number of lines in the job definition script that contains the job definition script executed in another process

exit-status-code

Return code of the job definition script that executed in another process

signal-number

Signal number received by the job definition script that executed in another process

execution-time-in-seconds

Execution time of the job definition script executed in another process. This is a reference value obtained by using the OS's API.

CPU-time-in-seconds

CPU time of the job definition script executed in another process. This is a reference value obtained by using the OS's API.

(S)

Resumes processing. If the job definition script that was executed in another process and that terminated with an error is in a job step, the system performs the following processing:

- If the job step's `onError` attribute is `stop`, the system executes the job step error block without executing the processing that follows the job step normal block.
- If the job step's `onError` attribute is `cont`, the system executes the processing that follows the job step normal block.

(O)

Check the execution results of the job definition script that was executed in another process and that terminated with an error, and then eliminate the cause of the error.

KNAX6551-E

Execution of the command *command-name* for the function *function-name* failed. exit status=*exit-status-code* execution time=*execution-time-in-seconds* CPU time=*CPU-time-in-seconds*

A command executed by the function indicated by *function-name* terminated with an error.

command-name

Name of the command that executed. If the command name exceeds the maximum path length permitted by the OS, it is truncated at the maximum length.

function-name

Name of the function that executed the command. One of the following is displayed:

`command substitution`: Command substitution function

`trap action`: Action of the `trap` command

exit-status-code

Command's return code

execution-time-in-seconds

Command execution time. This is a reference value obtained by using the OS's API.

CPU-time-in-seconds

Command's CPU time. This is a reference value obtained by using the OS's API.

(S)

Resumes processing. If the erroneous command is in a job step, the system performs the following processing:

- If the job step's `onError` attribute is `stop`, the system executes the job step error block without executing the processing that follows the job step normal block.
- If the job step's `onError` attribute is `cont`, the system executes the processing that follows the job step normal block.

Note that if the batch job needs to be canceled on the basis of command execution results, the system issues the `KNAX6584-I` message and then terminates the processing without taking the above action.

(O)

Check the command's execution results, and then eliminate the cause of the error.

KNAX6552-E

The command *command-name* for the function *function-name* ended abnormally because it received a signal. exit status=*exit-status-code* signal number=*signal-number* execution time=*execution-time-in-seconds* CPU time=*CPU-time-in-seconds*

A command executed by the function indicated by *function-name* terminated with an error by signal.

command-name

Name of the command that executed. If the command name exceeds the maximum path length permitted by the OS, it is truncated at the maximum length.

function-name

Name of the function that executed the command. One of the following is displayed:

`command substitution`: Command substitution function

`trap action`: Action of the `trap` command

exit-status-code

Command's return code

signal-number

Signal number received by the command

execution-time-in-seconds

Command execution time. This is a reference value obtained by using the OS's API.

CPU-time-in-seconds

Command's CPU time. This is a reference value obtained by using the OS's API.

(S)

Resumes processing. If the command that terminated with an error is in a job step, the system performs the following processing:

- If the job step's `onError` attribute is `stop`, the system executes the job step error block without executing the processing that follows the job step normal block.
- If the job step's `onError` attribute is `cont`, the system executes the processing that follows the job step normal block.

(O)

Check the command's execution results, and then eliminate the cause of the error.

KNAX6560-I

A command group executed by another process for the function *function-name* succeeded. exit status=*exit-status-code* execution time=*execution-time-in-seconds* CPU time=*CPU-time-in-seconds*

The job definition script executed by the function indicated by *function-name* in another process, such as a command group, terminated normally.

function-name

Name of the function that executed the command. One of the following is displayed:

`command substitution`: Command substitution function

`trap action`: Action of the `trap` command

exit-status-code

Return code of the executed job definition script

execution-time-in-seconds

Execution time of the executed job definition script. This is a reference value obtained by using the OS's API.

CPU-time-in-seconds

CPU time of the executed job definition script. This is a reference value obtained by using the OS's API.

(S)

Resumes processing.

KNAX6561-E

A command group executed by another process for the function *function-name* failed. exit status=*exit-status-code* execution time=*execution-time-in-seconds* CPU time=*CPU-time-in-seconds*

The job definition script executed by the function indicated by *function-name* in another process, such as a command group, terminated with an error.

function-name

Name of the function that executed the command. One of the following is displayed:

`command substitution`: Command substitution function

`trap action`: Action of the `trap` command *exit-status-code*

Return code of the job definition script that executed in another process

execution-time-in-seconds

Execution time of the job definition script executed in another process. This is a reference value obtained by using the OS's API.

CPU-time-in-seconds

CPU time of the job definition script executed in another process. This is a reference value obtained by using the OS's API.

(S)

Resumes processing. If the job definition script that was executed in another process and that terminated with an error is in a job step, the system performs the following processing:

- If the job step's `onError` attribute is `stop`, the system executes the job step error block without executing the processing that follows the job step normal block.
- If the job step's `onError` attribute is `cont`, the system executes the processing that follows the job step normal block.

(O)

Check the execution results of the job definition script that was executed in another process and that terminated with an error, and then eliminate the cause of the error.

KNAX6562-E

A command group executed by another process for the function *function-name* ended abnormally because it received a signal. exit status=*exit-status-code* signal number=*signal-number* execution time=*execution-time-in-seconds* CPU time=*CPU-time-in-seconds*

The job definition script executed by the function indicated by *function-name* in another process, such as a command group, terminated with an error by signal reception.

function-name

Name of the function that executed the command. One of the following is displayed:

`command substitution`: Command substitution function

`trap action`: Action of the `trap` command

exit-status-code

Return code of the job definition script that executed in another process

signal-number

Signal number received by the job definition script that executed in another process

execution-time-in-seconds

Execution time of the job definition script executed in another process. This is a reference value obtained by using the OS's API.

CPU-time-in-seconds

CPU time of the job definition script executed in another process. This is a reference value obtained by using the OS's API.

(S)

Resumes processing. If the job definition script that was executed in another process and that terminated with an error is in a job step, the system performs the following processing:

- If the job step's `onError` attribute is `stop`, the system executes the job step error block without executing the processing that follows the job step normal block.
- If the job step's `onError` attribute is `cont`, the system executes the processing that follows the job step normal block.

(O)

Check the execution results of the job definition script that was executed in another process and that terminated with an error, and then eliminate the cause of the error.

KNAX6571-I

The child job child-job-name started. parent job=*parent-process-job-name* parent job ID=*parent-process-job-ID*

The child job indicated by *child-job-name* has been started from the job indicated by *parent-process-job-name* and *parent-process-job-ID*.

(S)

Resumes processing.

KNAX6572-I

The child job *child-job-name* will use the job environment file "*job-environment-file*".

The child job indicated by *child-job-name* is using the job environment file indicated by *job-environment-file*.

(S)

Resumes processing.

KNAX6578-I

The child job child-job-name ended. exit status=*exit-status-code* execution time=*execution-time-in-seconds* CPU time=*CPU-time-in-seconds*

The child job indicated by *child-job-name* has ended.

exit-status-code

Return code indicating the child job's execution results

For details about the return code, see the description of the return codes for the `adshexec` command.

If an error occurs during `adshexec` command postprocessing after this message was output, the return code indicated in this message might differ from the `adshexec` command's return code. The `adshexec` command's final return code is output to the parent job's `JOBLOG`.

execution-time-in-seconds

Total execution time from start to end of the child job (in seconds). This is a reference value obtained by using the OS's API.

CPU-time-in-seconds

Total CPU time from start to end of the child job (in seconds). This is a reference value obtained by using the OS's API.

(S)

Resumes processing.

KNAX6584-I

A job stopped because a command that terminates execution of the script was executed.

The batch job was stopped because the command for stopping execution of a job definition script was executed. No more job steps or job definition scripts will be executed. Even job steps whose `run` attribute is `abnormal` or `always` will not be executed.

Note that the specification in the `#-adsh_rc_ignore` command and the `successRC` attribute in the `#-adsh_step_start` command do not take effect on the executed command.

This message is displayed in the following cases:

- The job definition script was terminated immediately by execution of the `exit` command.
- The job definition script was terminated immediately by execution of the `return` command outside the function or the external script.
- The script was terminated by execution of the `exec` command with an executable command specified in its argument.
- An error occurred in a special built-in command, or a non-resumable error occurred in JP1/Advanced Shell processing (this does not include an error that occurs in the `typeset` command or an error that occurs in the `return` command executed in a function or external script).

(S)

Stops the batch job.

(O)

If the batch job was stopped due to an error, correct the error, and then re-execute the batch job.

KNAX6585-I

A job stopped because the exit status of a step met the conditions specified by the #-adsh_job_stop command.

The batch job was stopped because the return code of a job step satisfied a condition specified in the `#-adsh_job_stop` command.

(S)

Terminates processing.

(O)

If a problem has occurred, check the execution results of the job step, eliminate the cause of the error, and then re-execute the batch job.

KNAX6586-E

A job stopped because an error occurred that prevented the script from continuing.

The batch job was stopped due to a non-resumable error during execution of a job definition script. No more job steps or job definition scripts will be executed. Even job steps whose `run` attribute is `abnormal` or `always` will not be executed.

(S)

Cancels job execution.

(O)

Eliminate the cause of the error, and then re-execute the job.

KNAX6587-E

The number of child jobs exceeded the limit.

The number of child jobs that were started has exceeded the maximum permissible number.

(S)

Terminates processing.

(O)

Modify the job definition script file so that the number of child jobs started in a root job (including child jobs started from other child jobs) does not exceed 9,999,999.

KNAX6588-E

API error occurred. name=*API-name*, reason=*error-details*

The error indicated by *error-details* occurred in the API call indicated by *API-name*.

(S)

Terminates processing.

(O)

Eliminate the cause of the error by checking the indicated reason and other messages output together with this message. If the problem cannot be resolved, contact the system administrator.

KNAX6589-W

An API error occurred. Processing for the job will continue. API name=*API-name*, reason=*cause*, details=*maintenance-information*

The error indicated by *cause* occurred while the API indicated by *API-name* was being invoked, but the job will continue. Possible causes are as follows:

- If *API-name* is `CreateFile`, the `adshexec` command might have been executed in an environment in which the standard input, standard output, or standard error output cannot be used.
- If *API-name* is `isatty`, the `adshexec` command might have been executed in an environment in which the standard input cannot be used.

(S)

Resumes processing.

(O)

If no problem has occurred in the job operation, no action is required. If a problem has occurred in the job operation, eliminate the cause of the error, and then re-execute the batch job. If the problem cannot be resolved, contact the system administrator.

KNAX6590-E

An error occurred in the function *function-name*. details=*maintenance-information*

The function indicated by *function-name* failed.

function-name

Name of the function resulting in the error. One of the following is displayed:

`get variable`: Function related to acquisition of shell variables

`set variable`: Function related to setting shell variables

`unset variable`: Function related to deletion of shell variable settings

`run external script`: Function related to external scripts

`set rc`: Function related to job definition script return codes

`open file`: Function related to opening files

(S)

Resumes or terminates the processing depending on the function resulting in the error.

(O)

Contact the system administrator.

KNAX6591-E

> The job controller ended abnormally because it received a termination request signal. signal number=*signal-number*

The job controller was forcibly terminated because the termination request signal indicated by *signal-number* was received during job execution.

(S)

The job controller forcibly terminates any descendant process being started, performs required postprocessing (such as deleting files), and then terminates itself.

(O)

Check that the job was forcibly terminated.

KNAX6592-E

> The job controller ended abnormally because it received a termination request control signal. control type = *control-type*

The job controller was forcibly terminated because the control signal indicated by *control-type* was received during job execution. One of the following character strings is displayed for *control-type*:

- `CTRL + C`: A **CTRL** + **C** signal was received
- `CTRL + BREAK`: A **CTRL** + **BREAK** signal was received
- `CLOSE EVENT`: The user closed the console (**CTRL** + **C** was received)

(S)

The job controller forcibly terminates any descendant process being started, performs required postprocessing (such as deleting files), and then terminates itself.

(O)

Check that the job was forcibly terminated.

KNAX6593-E

The job controller received a termination request.

The job controller received a forced termination request.

(S)

The job controller forcibly terminates any descendant process being started, performs required postprocessing (such as deleting file), and then terminates itself.

(O)

Check that the batch job was forcibly terminated.

KNAX6594-E

The job controller ended abnormally because of the operation to terminate a process.

The job controller was forcibly terminated because of process termination processing (such as `TerminateProcess`).

(S)

The job controller forcibly terminates any descendant process being started, performs required postprocessing (such as deleting files), and then terminates itself.

(O)

Check that the batch job was forcibly terminated.

KNAX6596-E

job-name.job-step-name step failed. exit status=*exit-status-code* execution time=*execution-time-in-seconds* CPU time=*CPU-time-in-seconds*

The job step terminated with an error.

job-name

Job name

job-step-name

Job step name

exit-status-code

Job step's return code

execution-time-in-seconds

Job step's execution time. This is a reference value obtained by using the OS's API.

CPU-time-in-seconds

Job step's CPU time. This is a reference value obtained by using the OS's API.

(S)

Resumes the processing and executes only those subsequent job steps whose `run` attribute is `abnormal` or `always`. However, the system terminates the shell in the following cases:

- The `KNAX6584-I` message was issued because a command for terminating the job definition script was executed.
- Anther error resulting in termination of the job definition script occurred.

(O)

Check the execution results for each command in the job step, and then eliminate the cause of the error.

KNAX6597-I

> *job-name*.*job-step-name* step succeeded. exit status=*exit-status-code* execution time=*execution-time-in-seconds* CPU time=*CPU-time-in-seconds*

The job step terminated normally.

job-name

Job name

job-step-name

Job step name

exit-status-code

Job step's return code

execution-time-in-seconds

Job step's execution time. This is a reference value obtained by using the OS's API.

CPU-time-in-seconds

Job step's CPU time. This is a reference value obtained by using the OS's API.

(S)

Resumes processing.

KNAX6598-E

> An API error occurred. API name=*API-name*, reason=*cause*, details=*maintenance-information*

The error indicated by *causes* occurred in the API call indicated by *API-name*.

(S)

Terminates processing.

(O)

Eliminate the cause of the error by checking the indicated reason and other messages output together with this message. If the problem cannot be resolved, contact the system administrator.

KNAX6599-E

> An internal error occurred. reason=*cause*, details=*maintenance-information*

An internal error occurred due to the indicated *cause*.

(S)

Terminates processing.

(O)

Contact the system administrator.

KNAX6600-E

An error occurred in the function *function-name*([*argument*,...]). exit status=*exit-status-code*, error number=*error-number*, internal information=*maintenance-information*

An error occurred in the function indicated by *function-name*.

function-name

Function resulting in the error (function provided by the platform)

argument

Function argument. This information might not be displayed.

exit-status-code

Function's return value

error-number

Value of the `errno` global variable that indicates the nature of the error. The value is a decimal number.

(S)

Whether the processing is resumed depends on the status. See the message that is output following this message.

(O)

Determine the cause of the error based on the information provided in the messages, and then correct the error.

KNAX6601-E

An internal error occurred. details=*maintenance-information*

An internal conflict was detected during processing.

(S)

Terminates processing.

(O)

Contact the system administrator.

KNAX6602-E

An I/O error occurred during output to stdout.

An I/O error occurred during output to the `stdout` standard output.

(S)

Terminates processing.

(O)

Determine the cause of the I/O error at the standard output based on the messages output before and after this message, and then resolve the problem.

KNAX6603-E

Failed to get the current date.

The current date could not be obtained.

(S)

Terminates processing.

(O)

Determine the cause of the error based on the messages output before and after this message, and then resolve the problem.

KNAX6604-E (Windows only)

An error occurred in the function *function-name*([*argument*,...]). exit status=*exit-status-code*, last error code=*system-error-code*, internal information=*maintenance-information*

An error occurred in the function indicated by *function-name*.

function-name

Name of the function resulting in the error (function provided by the platform)

argument

Function argument and information needed for investigating the error

This information might not always be displayed. Some arguments might not be displayed.

exit-status-code

Function's return value

system-error-code

Return value of `LastError()` that indicates the nature of the error

(S)

The processing depends on the messages output before and after this message.

(O)

Determine the cause of the error based on the messages output before and after this message, and then resolve the problem.

KNAX6605-E

Conversion of the date and time failed. date and time=*date-and-time*

An error occurred when the date and time were converted to internal representation.

date-and-time: Date and time that could not be converted to internal representation

(S)

Terminates processing.

(O)

Specify for interpretation by the command a date and time that falls within the following range (based on UTC):

1970-01-01 at 00:00:00 through 2038-01-19 at 03:14:07

Note that a date and time that is within this range might still result in an error depending on the platform being used.

KNAX6610-E

The specified option "*option-name*" is invalid.

An unknown option was specified.

option-name

Unknown option name

(S)

Terminates processing.

(O)

Specify the correct option.

KNAX6611-E

No value is specified for the option "*option-name*".

No value was specified for the option indicated by *option-name*.

(S)

Terminates processing.

(O)

Specify a value for the displayed option.

KNAX6612-E

The value specified for the option "*option-value*" is invalid.

An option value is invalid.

option-value

Specified option value

(S)

Terminates processing.

(O)

Specify the correct value for the option.

KNAX6613-I

The start time is *YYYY-MM-DD hh*:*mm*:*ss*+*hhmm*.
OR
The start time is none.

YYYY−*MM*−*DD hh* **:** *mm* **:** *ss*+*hhmm* indicates the earliest job execution start date and time that was interpreted by the command. +*hhmm* is the adjustment in hours and minutes between UTC and local time.

`none` means that the start date and time have not been specified.

(S)

Resumes processing.

KNAX6614-I

The end time is *YYYY-MM-DD hh*:*mm*:*ss*+*hhmm*.
OR

The end time is none.

YYYY−*MM*−*DD hh*:*mm*:*ss*+*hhmm* indicates the latest job execution start date and time that was interpreted by the command. +*hhmm* is the adjustment in hours and minutes between UTC and local time.

`none` means that the end date and time have not been specified.

(S)

Resumes processing.

KNAX6615-E

The time range is invalid. The start time is later than the end time.

The earliest and latest job execution start dates/times are reversed.

(S)

Terminates processing.

(O)

Specify the latest job execution start date and time (in the `-e` option in the `adshevtout` command) and the earliest job execution start date and time (in the `-s` option in the `adshevtout` command) in such a manner that the former is after the latter.

KNAX6616-E

Too many operands are specified.

There are too many arguments in the command.

(S)

Terminates processing.

(O)

Specify the correct arguments in the command.

KNAX6632-E

The spool directory was not found. directory name="*spool-directory-name*"

The spool directory cannot be found.

spool-directory-name

Directory to be referenced

(S)

Terminates processing.

(O)

Check the spool directory specified in the environment file for any error.

KNAX6633-E

The spool directory could not be read. directory name="*spool-directory-name*"

The spool directory cannot be referenced.

spool-directory-name

Directory to be referenced

(S)

Terminates processing.

(O)

Check if the user executing the command has the permission to reference the spool directory.

KNAX6634-E

An error occurred during initialization of the adshevtout command.

An initialization error occurred in the `adshevtout` command.

(S)

Terminates processing.

(O)

Resolve the problem by referencing the message that was output before this message.

KNAX6635-E

Failed to lock the spool directory. directory name="*spool-directory-name*"

A lock error occurred on the spool directory.

spool-directory-name

Directory to be referenced

(S)

Terminates processing.

(O)

Check the following:

- The spool directory specified in the environment file is correct.
- The user has the access permission to reference or update the spool directory.
- A lock file can be created, referenced, or updated.

KNAX6636-E

The spool directory cannot be accessed because it is being accessed by another command. directory name="*spool-directory-name*"

The spool could not be referenced because it was being accessed by another command.

spool-directory-name

Directory to be referenced

(S)

Terminates processing.

(O)

Re-execute the command when the `adshhk` command is not executing.

KNAX6640-I

An event file was skipped because you do not have permission to access it. file name="*event-file-path-name*"

Referencing of an event file was skipped because the user executing the command did not have the permission to reference the event file.

event-file-path-name

Path name of the skipped event file

(S)

Resumes processing.

KNAX6644-E

An I/O error occurred during an attempt to read an event file. file name="*event-file-path-name*"

An I/O error occurred while reading an event file.

event-file-path-name

Path name of the event file resulting in the I/O error

(S)

Stops processing the event file resulting in the I/O error, and then processes another event file.

(O)

Determine the cause of the I/O error in the event file based on the messages output before and after this message, and then resolve the problem.

KNAX6645-W

This command does not support the event file version. file name="*event-file-path-name*", event file version=*version-number*

This command does not support the version of an event file.

event-file-path-name

Path name of the event file

version-number

Version number of the event file

(S)

Processes another event file without referencing the displayed event file.

(O)

Execute the command that supports the JP1/Advanced Shell version used to create the event file.

KNAX6646-E

The event file is corrupted. file name="*event-file-path-name*"

An event file is corrupted.

event-file-path-name

Path name of the event file that has been determined to be corrupted

(S)

Stops processing the event file determined to be corrupted, and then processes another event file.

(O)

Make sure that the event file was not updated illegally. If the cause cannot be determined, contact the system administrator.

KNAX6701-W

The job object is not available.

The job object cannot be used. The process created by a child process is not terminated by the `TerminateProcess` function when the batch job is forcibly terminated.

(S)

Continues processing.

KNAX6710-I

The built-in command "*command-name*" [with the option "*option-name*"] is not supported on the current platform. The command returned "*return-value*".[filename="*file-name*" line=*line-number*]

An attempt was made to execute a built-in command that is not supported by JP1/Advanced Shell on the current platform. Another possibility is that an attempt was made to execute the `trap` command in an environment in which `DISABLE` is specified in the `TRAP_ACTION_SIGTERM` environment setting parameter.

command-name

Name of the built-in command

option-name

Option name specified in the built-in command

return-value

Return value of the built-in command

file-name

Name of the job definition script file

line-number

Line number at which the attempt was made to execute the built-in command

(S)

Continues the processing using the indicated return code as the return code of the built-in command.

KNAX6711-E

The built-in command "*command-name*" [with the option "*option-name*"] is not supported on the current platform. The job failed.[filename="*file-name*" line=*line-number*]

An attempt was made to execute a built-in command that is not supported by JP1/Advanced Shell on the current platform. Or, an attempt was made to execute a built-in command that specified an option that is not supported by JP1/Advanced Shell on the current platform.

command-name

Name of the built-in command

option-name

Option name specified in the built-in command

file-name

Name of the job definition script file

line-number

Line number at which the attempt was made to execute the built-in command

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6712-E

The specified variable "*shell-variable-name*" cannot be exported because the current platform requires that its name be composed entirely of capital letters.[filename="*file-name*" line=*line-number*]

A variable whose name is not in all uppercase letters cannot be exported on the current platform.

shell-variable-name

Name of the shell variable whose export was attempted

file-name

Name of the job definition script file

line-number

Line number at which the attempt was made to execute the `export` command

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6713-E

The required directory "*directory-name*" does not exist.

A directory required in JP1/Advanced Shell on the current platform is missing.

directory-name

Name of the missing directory

(S)

Terminates processing.

(O)

Check and, if necessary, revise the setup procedure.

KNAX6714-E

Background execution is not supported.[filename="*file-name*" line=*line-number*]

Job definition scripts cannot be executed in the background on the current platform.

file-name

Name of the job definition script file

line-number

Line number at which the attempt was made to execute a job definition script in the background

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6715-E

Subshell execution is not supported.[filename="*file-name*" line=*line-number*]

Job definition scripts cannot be executed in a subshell on the current platform.

file-name

Name of the job definition script file

line-number

Line number at which the attempt was made to execute a job definition script in a subshell

(S)

Terminates processing.

(O)

Correct the job definition script.

KNAX6718-I

The trap action was not set because an unsupported termination method was specified in the trap command. [filename="*file-name*" line="*line-number*"]

The action was not set because an unsupported forced termination method was specified in the `trap` command.

file-name

Name of the job definition script file

line-number

Line number at which the attempt was made to execute the built-in command

(S)

Continues processing.

KNAX6750-E

The file or directory "*path-name*" does not exist.

The file or directory indicated by *path-name* does not exist.

(S)

Continues processing.

(O)

Make sure that the file or directory specified for *path-name* is correct.

KNAX6751-E

Failed to get ACL information. path="*path-name*", reason=*cause*, *details=details*

An attempt to acquire ACL information for the file or directory indicated by *path-name* failed.

(S)

Continues processing.

(O)

Make sure that the user who executes the `adshscripttool` command is the owner of the file or directory indicated by *path-name*. If the user is not the owner, execute the command as the owner.

KNAX6752-E

The format of the mode is invalid.

The format of the mode specified in the `-fmode` option of the `adshscripttool` command was not valid.

(S)

Continues processing.

(O)

Correct the mode specification.

KNAX6753-E

adshscripttool: A command line error occurred. details="*details*"

The command line specification is not valid. The following table lists the details displayed in the message and their meanings:

Details displayed	Meaning
`A required option is not specified`	A required option is not specified.
`The path name is not specified`	No path name is specified.
`The mode is not specified`	No mode is specified.
`No value is specified for the -s option`	No value is specified for the `-s` option.
`You cannot specify the -s option first`	The `-s` option was specified first (it cannot be specified first).
`An invalid value is specified for the -s option`	The value specified for the `-s` option is invalid.
`The path is specified more than once`	Multiple path names were specified (multiple path names cannot be specified).
`The mode is specified more than once`	Multiple modes were specified (multiple modes cannot be specified).

Details displayed	Meaning
`The specified path contains a null character`	An empty character string was specified for the path name.
`The specified mode contains a null character`	An empty character string was specified for the mode.
`The value specified for the -s option contains a null character`	An empty character string was specified as the value of the `-s` option.

(S)

Continues processing.

(O)

Correct the command specification, and then re-execute it.

KNAX6800-I

The path matched the conversion rule.[filename="*file-name*" line=*line-number*] path converted="*path-name-1*":"*path-name-2*"

The rule for converting *path-name-1* to *path-name-2* was satisfied.

file-name

Name of the job definition script file

line-number

Line number of the line satisfying the path conversion rule

(S)

Continues processing.

KNAX6801-I

The path matched the conversion rule.[filename="*file-name*" line=*line-number*] shell variable handling path="*shell-variable-name*"

The conversion rule using a shell variable that handles paths was satisfied.

shell-variable-name

Name of the shell variable that handles path names

file-name

Name of the job definition script file

line-number

Line number of the line satisfying the path conversion rule

(S)

Continues processing.

KNAX6803-I

The access path matched the conversion rule.[filename="*file-name*" line=*line-number*] path converted="*path-name-1*":"*path-name-2*"

The rule for converting *path-name-1* to *path-name-2* was satisfied.

file-name

Name of the job definition script file

line-number

Line number of the line satisfying the `PATH_CONV_ACCESS` parameter conversion rule

(S)

Continues processing.

KNAX6804-I

> The command argument matched the conversion rule.[filename="*file-name*" line=*line-number*] argument converted="*argument-1*":"*argument-2*"

The rule for converting *argument-1* to *argument-2* was satisfied.

file-name

Name of the job definition script file

line-number

Line number of the line satisfying the `COMMAND_CONV_ARG` parameter conversion rule

(S)

Continues processing.

KNAX6805-I

> The access path matched the conversion rule during execution of the function *function-name*. path converted="*path-name-1*":"*path-name-2*"

When the function indicated by *function-name* was executing, the rule for converting *path-name-1* to *path-name-2* was satisfied.

function-name

Name of the function satisfying the conversion rule. One of the following is displayed:

- `command substitution`: Command substitution function
- `trap action`: Action of the `trap` command

(S)

Continues processing.

KNAX6806-I

> The command argument matched the conversion rule during execution of the function *function-name*. argument converted="*argument-1*":"*argument-2*"

The rule for converting *argument-1* to *argument-2* was satisfied when the function indicated by *function-name* was executed.

function-name

Name of the function satisfying the conversion rule. One of the following is displayed:

- `command substitution`: Command substitution function

- `trap action`: Action of the `trap` command

(S)

Continues processing.

KNAX6810-E

The character specified to delimit directories is invalid.

The directory separator specified in the `PATH_CONV_ENABLE` parameter is invalid.

(S)

Terminates processing.

(O)

Correct the environment file.

KNAX6811-E

The character specified to delimit paths is invalid.

The path separator specified in the `PATH_CONV_ENABLE` parameter is invalid.

(S)

Terminates processing.

(O)

Correct the environment file.

KNAX6812-E

An invalid path is specified. path="*path-name*"

The path specified in the `PATH_CONV` parameter is invalid.

path-name

Path name specified in the `PATH_CONV` parameter

(S)

Terminates processing.

(O)

Correct the environment file.

KNAX6813-E

A line exceeds the size limit.[filename="*file-name*" line=*line-number*]

The length of a line exceeded the maximum permitted for job definition scripts.

file-name

Name of the job definition script file

line-number

Line number of the line whose length exceeds the maximum

(S)

Terminates processing.

(O)

Check and, if necessary, revise the job definition script.

KNAX6814-E

An invalid argument is specified. argument="*argument*"

An argument specified in the `COMMAND_CONV_ARG` or `PATH_CONV_ACCESS` parameter is invalid.

argument

Argument specified in the `COMMAND_CONV_ARG` or `PATH_CONV_ACCESS` parameter that is invalid

(S)

Terminates processing.

(O)

Correct the environment file.

KNAX6815-E

No argument is specified.

An argument is missing in the `COMMAND_CONV_ARG` or `PATH_CONV_ACCESS` parameter.

(S)

Terminates processing.

(O)

Correct the environment file.

KNAX6830-I

The command definition matched the rule specified by the environment settings parameter CHILDJOB_PGM. command line="*command-line*"

The rule for replacing a child job's execution specified in the `CHILDJOB_PGM` parameter was satisfied. Alternatively, a child job was executed by executing the `adshscripttool -exec` command.

command-line

Command line of the command to be replaced

(S)

Continues processing.

KNAX6831-I

The command definition matched the rule specified by the environment settings parameter CHILDJOB_SHEBANG. script="*script-name*" shebang="*shebang*"

The rule for replacing a child job's execution specified in the `CHILDJOB_SHEBANG` parameter was satisfied.

script-name

File name of the script file that is to be executed as a child job

shebang

First line beginning with `#!` of the script file that is to be executed as a child job

(S)

Continues processing.

KNAX6832-I

The command definition matched the rule specified by the environment settings parameter CHILDJOB_EXT. script="*script-name*"

The rule for replacing a child job's execution specified in the `CHILDJOB_EXT` parameter was satisfied.

script-name

File name of the script file that is to be executed as a child job

(S)

Continues processing.

KNAX6997-E

An API error occurred. API name=*API-name* reason=*cause*

The error indicated by *causes* occurred in the API call indicated by *API-name*.

If the API name is `AssignProcessToJobObject` and the cause is `Access is denied.`, the job controller was called from within the job definition script as a root job, not as a child job, or the job controller was called from a program that uses job objects other than JP1/Advanced Shell.

(S)

Terminates processing. However, if the API name is `AssignProcessToJobObject`, the system continues processing.

(O)

- When the job controller was called from within the job definition script as a root job, not as a child job
  Call the job controller as a child job.
- When the job controller was called from a program that uses job objects other than JP1/Advanced Shell
  When the job is terminated forcibly, the job controller terminates without forcibly terminating processes engaged in grandchild processing. If there is no problem in the operation during forced termination processing including the specifications of the program that calls the job controller, no action is needed. If there is a problem, contact the system administrator.
- Other than the above
  Eliminate the cause of the error by referencing the displayed cause. If the problem cannot be resolved, contact the system administrator.

KNAX6998-E

Memory cannot be allocated.[filename="*file-name*" line=*line-number*]

A memory shortage occurred.

file-name

Name of the job definition script file

line-number

Line number of the line where the error occurred

(S)

Terminates processing.

(O)

Contact the system administrator, who must check and, if necessary, revise the memory estimation.

KNAX6999-E

An internal error occurred. reason="*cause*" details="*maintenance-information*"

An internal error occurred.

(S)

Terminates processing.

(O)

Contact the system administrator.

KNAX7000-E

The input character string is too long.

The input character string is too long.

(S)

Resumes processing.

(O)

Check and, if necessary, revise the input character string, and then re-enter it.

KNAX7001-E

The specified command "*command-name*" does not exist.

The debugger command indicated by *command-name* does not exist.

(S)

Resumes processing.

(O)

Enter the correct command name.

KNAX7002-E

Too many arguments are specified.

Too many arguments are specified.

(S)

Resumes processing.

(O)

Specify the correct arguments, and then re-execute the command.

KNAX7003-E

The command "*command-name*" does not take any arguments.

The debugger command indicated by *command-name* does not take arguments.

(S)

Resumes processing.

(O)

Specify the command correctly, and then re-execute it.

KNAX7004-E

The command "*command-name*" requires the argument (*argument-details*).

The debugger command indicated by *command-name* requires the argument indicated by *argument-details*.

(S)

Resumes processing.

(O)

Specify the command correctly, and then re-execute it.

KNAX7006-W

Are you sure you want to restart the script from the beginning? (y or n)

This message asks whether the job definition script is to be re-executed from the beginning.

(S)

Resumes processing.

(O)

To re-execute the job definition script, enter `y`; otherwise, enter `n`.

KNAX7007-I

Execution of the following script will now start: *path-name-of-job-definition-script arguments*

Execution of the job definition script is starting.

(S)

Resumes processing.

KNAX7008-I

The script was not restarted.

The job definition script was not re-executed.

(S)

Resumes processing.

KNAX7009-I

> Are you sure you want to stop the script being debugged? (y or n)

This message asks whether the job definition script being debugged is to be terminated.

(S)

Resumes processing.

(O)

To terminate the job definition script, enter `y`; otherwise, enter `n`.

KNAX7010-E

> The script is not running.

The job definition script is not running.

(S)

Resumes processing.

(O)

Re-execute the command while the job definition script is running.

KNAX7011-I

> The command "*command-name*" was not executed.

The debugger command indicated by *command-name* was not executed.

(S)

Resumes processing.

KNAX7012-W

> The script is running. Are you sure you want to exit the debugger? (y or n)

This message asks whether the debugger is to be terminated while the job definition script is running.

(S)

Resumes processing.

(O)

To terminate the debugger, enter `y`; otherwise, enter `n`.

KNAX7013-E

> The file "*file-name*" is not parsed.

The job definition script file indicated by *file-name* has not been parsed.

(S)

Resumes processing.

(O)

Specify the correct file name, and then re-execute the command. Or, use the `#-adsh_script` command to call an external script within the job definition script file.

KNAX7014-E

The number of breakpoints and watchpoints exceeds the limit.

The number of breakpoints and watchpoints exceeded the maximum.

(S)

Resumes processing.

(O)

Restart the debugger, and then re-execute the command.

KNAX7015-W

The breakpoint cannot be set at line "*line-number*". The breakpoint will be set at the next available line.

A breakpoint cannot be set on the line indicated by *line-number*. A breakpoint will be set at the next line at which a breakpoint can be set.

(S)

Resumes processing.

KNAX7016-E

The breakpoint cannot be set at line "*line-number*". No next available line exists at which to set the breakpoint.

A breakpoint cannot be set on the line indicated by *line-number*. There are no more lines on which a breakpoint can be set.

(S)

Resumes processing.

(O)

Specify the correct line number, and then re-execute the command.

KNAX7017-E

The file "*file-name*" does not exist.

The file indicated by *file-name* does not exist.

(S)

Resumes processing.

(O)

Specify the correct file name, and then re-execute the command.

KNAX7018-I

Breakpoint "*breakpoint-number*": filename="*file-name*" line=*line-number*

This message displays information about a breakpoint. The displayed *file-name* is a base name.

(S)

Resumes processing.

KNAX7019-E

Line "*line-number*" does not exist.

The line indicated by *line-number* does not exist.

(S)

Resumes processing.

(O)

Specify the correct line number, and then re-execute the command. Or, move the line, and then re-execute the command.

KNAX7020-E

The specified function "*function-name*" is not defined.

The function indicated by *function-name* is undefined.

(S)

Resumes processing.

(O)

Specify the correct function name, and then re-execute the command.

KNAX7021-E

The specified job step "*job-step-name*" is not defined.

The job step indicated by *job-step-name* is undefined.

(S)

Resumes processing.

(O)

Specify the correct job step name, and then re-execute the command.

KNAX7022-E

The specified variable "*variable-name*" is invalid.

The variable indicated by *variable-name* is invalid.

(S)

Resumes processing.

(O)

Specify the correct variable name, and then re-execute the command.

KNAX7023-I

Watchpoint "*watchpoint-number*": variable="*variable-name*"

This message displays information about a watchpoint.

(S)

Resumes processing.

KNAX7024-E

The specified range "*number-range*" is invalid.

The range of breakpoint and watchpoint numbers indicated by *number-range* is invalid.

(S)

Resumes processing.

(O)

Specify a valid range of numbers, and then re-execute the command.

KNAX7025-I

Are you sure you want to delete all breakpoints and watchpoints? (y or n)

This message asks whether all breakpoints and watchpoints are to be deleted.

(S)

Resumes processing.

(O)

To delete all breakpoints and watchpoints, enter `y`; otherwise, enter `n`.

KNAX7026-E

No breakpoints or watchpoints are defined.

Breakpoints and watchpoints cannot be deleted because no breakpoints or watchpoints have been set.

(S)

Resumes processing.

(O)

If necessary, re-execute the command when breakpoints and watchpoints have been set.

KNAX7027-E

The breakpoint or watchpoint "*number*" is not defined.

The breakpoint or watchpoint indicated by *number* is not set.

(S)

Resumes processing.

(O)

Specify the correct number, and then re-execute the command.

KNAX7028-E

No breakpoints or watchpoints exist in the range "*number-range*".

There are no breakpoints or watchpoints in the range indicated by *number-range*.

(S)

Resumes processing.

(O)

Specify the correct range of numbers, and then re-execute the command.

KNAX7029-E

The specified argument "*argument*" is invalid.

The command argument indicated by *argument* is invalid.

(S)

Resumes processing.

(O)

Check and, if necessary, revise the specified argument, and then re-execute the command.

KNAX7032-I

The script "*job-definition-script-name*" stopped running.

Execution stopped in the job definition script indicated by *job-definition-script-name*.

(S)

Resumes processing.

KNAX7033-I

The external script "*job-definition-script-name*" stopped running.

Execution stopped in the external script indicated by *job-definition-script-name*.

(S)

Resumes processing.

KNAX7034-I

The script will continue.

Execution of the job definition script is underway.

(S)

Resumes processing.

KNAX7035-E

The debugger command "*command-name*" cannot be executed at the outermost level.

The debugger command indicated by *command-name* cannot be executed at the outermost location.

(S)

Resumes processing.

(O)

Re-execute the command while execution is stopped within the function.

KNAX7036-I

Execution will continue until the end of the current function.

Execution will continue until the end of the current function.

(S)

Resumes processing.

KNAX7037-I

Are you sure you want to exit the current function? (y or n)

This message asks whether the current function is to be terminated.

(S)

Resumes processing.

(O)

To terminate the current function, enter `y`; otherwise, enter `n`.

KNAX7038-I

The signal "*signal-name*" will be sent to the script.

The signal indicated by *signal-name* is being sent to the job definition script.

(S)

Resumes processing.

KNAX7039-E

The specified signal number "*signal-number*" does not exist.

The signal indicated by *signal-number* does not exist.

(S)

Resumes processing.

(O)

Specify the correct signal number, and then re-execute the command.

KNAX7040-E

The specified signal "*signal-name*" does not exist.

The signal indicated by *signal-name* does not exist.

(S)

Resumes processing.

(O)

Specify the correct signal name, and then re-execute the command.

KNAX7043-I

The script stopped because the signal "*signal-name*" (Stop=Yes) was received.

The job definition script was stopped because the signal indicated by *signal-name* that stops a job definition script was already received.

(S)

Resumes processing.

KNAX7044-E

The debugger command "*command-name*" requires a subcommand.

The debugger command indicated by *command-name* requires a subcommand.

(S)

Resumes processing.

(O)

Specify a subcommand, and then re-execute the command.

KNAX7045-E

"*command-name*": The subcommand "*subcommand-name*" is invalid.

The indicated *subcommand-name* of the debugger command indicated by *command-name* is invalid.

(S)

Resumes processing.

(O)

Specify the correct subcommand name, and then re-execute the command.

KNAX7046-E

The specified variable "*variable-name*" is not defined.

The variable indicated by *variable-name* is undefined.

(S)

Resumes processing.

(O)

Specify the correct variable name, and then re-execute the command.

KNAX7047-I

> No breakpoints or watchpoints are defined.

No breakpoints or watchpoints have been set.

(S)

Resumes processing.

(O)

If necessary, re-execute the command after breakpoints or watchpoints have been set.

KNAX7048-I

> Working directory: *directory-path*

The work directory was changed to the directory path shown as *directory-path*.

(S)

Resumes processing.

KNAX7049-E

> The working directory cannot be changed. (reason=*error-details*)

The work directory could not be changed for the reason indicated by *error-details*.

(S)

Resumes processing.

(O)

Eliminate the cause of the error, and then re-execute the command.

KNAX7050-E

> An ampersand "&" cannot be specified in arguments for the command "*command-name*".

The ampersand (`&`) is not allowed in an argument of the debugger command indicated by *command-name*.

(S)

Resumes processing.

(O)

To use an ampersand (`&`) for a purpose other than background execution, specify `\&`.

KNAX7052-E

> The type of the value to be assigned differs from that of the variable.

The type of variable differs from the type of value to be assigned.

(S)

Resumes processing.

(O)

Check and, if necessary, revise the types of the variable and the value to be assigned, and then re-execute the command.

KNAX7053-I

Usage: *command-name argument*

This message displays the usage of the debugger command indicated by *command-name*.

(S)

Resumes processing.

KNAX7054-E

The specified variable "*variable-name*" is read-only.

The variable indicated by *variable-name* is read-only.

(S)

Resumes processing.

(O)

Check and, if necessary, revise the attribute of the variable, and then re-execute the command.

KNAX7055-E

A breakpoint has already been set at line "*line-number*".

A breakpoint has already been set at the line indicated by *line-number*.

(S)

Resumes processing.

(O)

If necessary, delete the breakpoint, and then re-execute the command. Or, specify a different line number, and then re-execute the command.

KNAX7056-I

The value of the variable "*variable-name*" was changed to *numeric-value*.

The value of the variable indicated by *variable-name* was changed to *numeric-value*.

numeric-value

Numeric value or variable value (numeric value) specified as the right-hand term of the assignment expression in the `set` command

(S)

Resumes processing.

KNAX7057-I

> The value of the variable "*variable-name*" was changed to "*character-string*".

The value of the variable indicated by *variable-name* was changed to *character-string*.

character-string

Character string or variable value (character string) specified as the right-hand term of the assignment expression in the `set` command

(S)

Resumes processing.

KNAX7058-I

> Debugger received signal "*signal-name*".

The debugger received the signal indicated by *signal-name*.

(S)

Resumes processing.

KNAX7062-E

> No value is set for the specified variable "*variable-name*".

The variable indicated by *variable-name* has no value.

(S)

Resumes processing.

(O)

Specify a variable that has a value, and then re-execute the command.

KNAX7063-I

> A request from the command "*command-name*" was received while the script was running.

The request of the command indicated by *command-name* was accepted while the job definition script was running.

(S)

Cancels the batch job.

(O)

If necessary, re-execute the command.

KNAX7064-I

> A cancel request was received from the editor.

A cancellation request from the editor was accepted.

(S)

Cancels the batch job.

(O)

If necessary, re-execute the command.

KNAX7065-I

Job steps are not defined.

No job steps have been defined.

(S)

Resumes processing.

(O)

If necessary, define a job step in the job definition script, and then re-execute the command.

KNAX7066-I

Functions are not defined.

Functions are not defined.

(S)

Resumes processing.

(O)

If necessary, define a function in the job definition script, and then re-execute the command. Or, re-execute the command after function definitions have taken effect.

KNAX7067-I

Variables are not defined.

Variables are not defined.

(S)

Resumes processing.

(O)

If necessary, define a variable in the job definition script, and then re-execute the command. Or, re-execute the command after variable definitions have taken effect.

KNAX7068-I

Commands will be skipped until the end of the function.

The commands will be skipped until the end of the function.

(S)

Resumes processing.

KNAX7070-E

A watchpoint has already been set for the variable "*variable-name*".

No new watchpoint can be set for the variable indicated by *variable-name* because a watchpoint has already been set.

(S)

Resumes processing.

(O)

If necessary, delete the watchpoint, and then re-execute the command. Or, specify a different variable name, and then re-execute the command.

KNAX7071-E

A breakpoint cannot be set. reason=*error-cause*

No breakpoint can be set because execution of the job definition script was stopped by the debugger for the reason indicated by *cause-of-error*.

error-cause

One of the following is output:

- `trap action`: A `trap` action stop occurred during execution
- `EOF`: Execution stopped at the end of the file.

(S)

Resumes processing.

(O)

Eliminate the cause of the error, and then re-execute the command. Or, specify an appropriate argument, and then re-execute the command.

KNAX7072-E

A line cannot be listed. reason=*error-cause*

The contents of lines in the source file cannot be displayed because execution of the job definition script was stopped by the debugger for the reason indicated by *cause-of-error*.

error-cause

One of the following is output:

- `trap action`: A `trap` action stop occurred during execution
- `EOF`: Execution stopped at the end of the file.

(S)

Resumes processing.

(O)

Eliminate the cause of the error, and then re-execute the command. Or, specify an appropriate argument, and then re-execute the command.

KNAX7073-I

A request to execute a trap action was received from the editor. parameter value="<*parameter value*>"

A request to execute a trap action was received from the editor.

parameter value

Value specified in the `TRAP_ACTION_SIGTERM` environment setting parameter.

If `DISABLE` is specified in the `TRAP_ACTION_SIGTERM` environment setting parameter or no action by the `trap` command is defined, the `trap` command's action is not executed even when this message is issued.

(S)

Resumes processing.

KNAX7090-W

Information about one or more errors exceeds the limit and cannot be displayed.

One or more error information items cannot be displayed because the maximum number of items has been reached.

(S)

Resumes processing.

(O)

Specify a variable name that has a value, and then re-execute the command.

KNAX7099-E

The debugger ended abnormally.

The debugger terminated with an error.

(S)

Terminates processing.

(O)

Eliminate the cause of the error by referencing the other messages output together with this message, and then re-execute the command. If the problem cannot be resolved, contact the system administrator.

KNAX7101-E

Failed to create the debugger child process. (reason=*error-details*)

The creation of a debugger child process failed for the reason indicated by *error-details*.

(S)

Terminates processing.

(O)

Eliminate the cause of the error indicated by *error-details*, and then re-execute the command. If the problem cannot be resolved, contact the system administrator.

KNAX7102-I

The debugger child process was created. (PID=*process-ID*)

A debugger child process has been created.

(S)

Resumes processing.

KNAX7103-I

The debugger child process ended. (PID=*process-ID*)

A debugger child process ended.

(S)

Resumes processing.

KNAX7104-E

Failed to open a log file in the debugger.

An open error occurred in the log file while the job definition script was being run by the debugger.

(S)

Terminates processing.

(O)

Eliminate the cause of the error, and then re-execute the command. If the problem cannot be resolved, contact the system administrator.

KNAX7105-E

Failed to move to the debugger working directory. (reason=*error-details*)

The directory could not be changed to the debugger's work directory for the reason indicated by *error-details*.

(S)

Terminates processing.

(O)

Eliminate the cause of the error, and then re-execute the command.

KNAX7106-E

Failed to move to the script working directory. (reason=*error-details*)

The directory could not be changed to the work directory for the job definition script for the reason indicated by *error-details*.

(S)

Terminates processing.

(O)

Eliminate the cause of the error, and then re-execute the command.

KNAX7107-I

A signal was received during debugger command input.

A signal was received while a debugger command was being entered.

(S)

Terminates the processing if a signal for terminating the processing was received; continues the processing if a signal for continuing the processing was received.

(O)

If necessary, re-execute the command.

KNAX7108-E

Debugger command input failed.

An input error occurred in a debugger command.

(S)

Terminates processing.

(O)

Eliminate the cause of the error by referencing the other messages output together with this message, and then re-execute the command.

KNAX7109-I

An EOF was input by the debugger command.

EOF was entered by the debugger command.

(S)

Terminates processing.

KNAX7110-E

Failed to open the DBG file "*file-path*". (reason=*error-details*)

An open error occurred in the file indicated by *file-path* for the reason indicated by *error-details*.

(S)

Terminates processing.

(O)

Eliminate the cause of the error, and then re-execute the command.

KNAX7111-I

Parsing will now start for the DBG file "*file-path*".

Parsing of the file indicated by *file-path* is beginning.

(S)

Continues processing.

KNAX7112-E

The format of the DBG file "*file-path*" is invalid. details=*maintenance-information*

The format of the file indicated by *file-path* is invalid. The maintenance information provides the location of invalid internal data and the cause of the error.

(S)

Terminates processing.

(O)

Re-execute the command. If necessary, contact the system administrator.

KNAX7113-E

Processing for the "*API-name*" API failed. (reason=*error-details*)

The processing indicated by *OS's-API-name* failed for the reason indicated by *error-details*.

(S)

Terminates processing.

(O)

Eliminate the cause of the error indicated by *error-details*, and then re-execute the command. If the problem cannot be resolved, contact the system administrator.

KNAX7114-E

Failed to wait for the debugger child process. (reason=*error-details*)

Wait processing for a debugger child process failed for the reason indicated by *error-details*.

(S)

Terminates processing.

(O)

Eliminate the cause of the error indicated by *error-details*, and then re-execute the command. If the problem cannot be resolved, contact the system administrator.

KNAX7115-E

Failed to execute the exec command process. (reason=*error-details*)

Execution of the `exec` command process failed for the reason indicated by *error-details*.

(S)

Terminates processing.

(O)

Eliminate the cause of the error indicated by *error-details*, and then re-execute the command. If the problem cannot be resolved, contact the system administrator.

KNAX7116-E

Free disk space is low.

There is a shortage of free disk space.

(S)

Terminates processing.

(O)

Increase the available disk capacity, and then re-execute the command.

KNAX7117-I

Parsing of the DBG file "*file-path*" ended normally.

Parsing of the DBG file indicated by *file-path* has been completed.

(S)

Resumes processing.

KNAX7118-E

Failed to create the console.

Console creation failed.

(S)

Terminates processing.

(O)

Re-execute the command. If the same error recurs, contact the system administrator.

KNAX7119-E

Failed to duplicate the file descriptor.

Duplication of a file descriptor failed.

(S)

Terminates processing.

(O)

Re-execute the command. If the same error recurs, contact the system administrator.

KNAX7120-W

A parent process received SIGCHLD, but the child process had no change.

The parent process received `SIGCHLD`, but its child process's status remains unchanged.

(S)

Resumes processing.

KNAX7121-E

Failed to create a process by using the exec command. (reason=*error-details*)

Creation of an `exec` command process failed for the reason indicated by *error-details*.

(S)

Terminates processing.

(O)

Eliminate the cause of the error indicated by *error-details*, and then re-execute the command. If the problem cannot be resolved, contact the system administrator.

KNAX7122-E

Failed to wait for a process created by the exec command. (reason=*error-details*)

Wait processing for the `exec` command process failed for the reason indicated by *error-details*.

(S)

Terminates processing.

(O)

Eliminate the cause of the error indicated by *error-details*, and then re-execute the command. If the problem cannot be resolved, contact the system administrator.

KNAX7123-E

Failed to get the value of the variable "*variable-name*". details=*maintenance-information*

Acquisition of the value of the variable indicated by *variable-name* failed. If the attempt was to acquire the values of all variables, `<All variables>` is displayed for *variable-name*.

This message might be output because there is a shortage of system memory.

(S)

Terminates processing.

(O)

Contact the system administrator.

KNAX7124-E

Failed to set the value of the variable "*variable-name*". details=*maintenance-information*

An attempt to set a value in the variable indicated by *variable-name* failed.

(S)

Terminates processing.

(O)

Contact the system administrator.

KNAX7125-E

Failed to get information about the function "*function-name*". details=*maintenance-information*

Acquisition of information about the function indicated by *function-name* failed. If the attempt was to acquire information about all functions, `<All functions>` is displayed for *function-name*.

(S)

Terminates processing.

(O)

Contact the system administrator.

KNAX7126-I

Fault injection mode is set to {"on"|"off"}.

The fault injection mode was set.

- `on`

  The fault injection mode was enabled.

- `off`

  The fault injection mode was disabled.

(S)

Terminates processing.

KNAX7127-E

Fault injection mode could not be modified.

The attempt to reset the fault injection mode failed because the job definition script was resumed with the fault injection mode already enabled.

(S)

Resumes processing.

(O)

Execute the job definition script all the way to its end, or re-execute it.

KNAX7128-E

Failed to remove the DBG file. (filename=*file-path*, reason=*error-details*)

The debugger's DBG file cannot be deleted for the reason indicated by *error-details*.

file-path

File path of the DBG file.

error-details

Details of the error

(S)

Resumes processing.

(O)

Delete the unneeded file with a command such as `rm`.

KNAX7400-E

The number of response messages exceeds the limit.

The number of reply-request messages output by the `adshread` command exceeded the maximum.

(S)

Repeats processing.

(O)

Check and, if necessary, revise the `USERREPLY_WAIT_MAXCOUNT` parameter value.

KNAX7402-E

An error occurred during shared memory processing. (error information=*error-information*, function=*maintenance-information*)

An error occurred during shared memory manipulation while processing a reply-request message with the `adshread` command.

error-information

Status code output by the API resulting in the error

maintenance-information

Name of the API resulting in the error

(S)

Resumes processing.

(O)

Take appropriate action by checking the indicated *error-information*. For details about *error-information*, see *11.4.4 Handling Error Information Displayed in the User-Reply Functionality*.

KNAX7403-E

The number of parameters is invalid. filename="*file-name*" line=*line-number*

The number of arguments in the `adshecho` or `adshread` command is invalid.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Resumes processing.

(O)

Correct the job definition script file.

KNAX7404-E

The type of the variable "*variable-name*" is invalid. filename="*file-name*" line=*line-number*

The variable specified in the `adshread` command is the numeric type.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Resumes processing.

(O)

Correct the job definition script file.

KNAX7405-E

The specified message is too long. filename="*file-name*" line=*line-number*

The event notification message specified in the `adshecho` command or the reply-request message specified in the `adshread` command is too long.

file-name

Name of the job definition script file

line-number

Line number in the job definition script file where the error occurred

(S)

Resumes processing.

(O)

Correct the job definition script file.

KNAX7408-E

An internal error occurred. (details=*maintenance-information*)

An internal error occurred during `adshecho` or `adshread` command processing.

(S)

Terminates processing.

(O)

Contact the system administrator.

KNAX7420-E

The response contains a non-ASCII character. Re-enter the response.

A reply entered from the standard input contains non-ASCII characters.

(S)

Resumes processing.

(O)

Re-enter the reply using only ASCII characters.

KNAX7450-I

The job controller canceled the response request. (job ID=*job-ID*, line=*line-number*, host name=*host-name*)

The reply-request message handled by the `adshread` command was canceled because the job controller accepted a forced termination request.

job-ID

Job ID assigned to the batch job by JP1/Advanced Shell

line-number

Line number in the job definition script that issued the `adshread` command

host-name

Name of the host on which the service or daemon is running

(S)

Terminates processing.

KNAX7451-I

A response request was canceled. (job ID=*job-ID*, line=*line-number*, host name=*host-name*)

The reply-request message handled by the `adshread` command was canceled because the job controller accepted a termination request.

job-ID

Job ID assigned to the batch job by JP1/Advanced Shell

line-number

Line number in the job definition script where the `adshread` command was issued

host-name

Name of the host on which the service or daemon is running

(S)

Terminates processing.

KNAX7460-E

An error occurred during event processing. (error information=*error-information*, function=*maintenance-information*)

Issuance of a JP1 event failed.

error-information

Status code output by the API resulting in the error

maintenance-information

Name of the API resulting in the error

(S)

Resumes processing.

(O)

Take appropriate action by checking the indicated *error-information*. For details about *error-information*, see *11.4.4 Handling Error Information Displayed in the User-Reply Functionality*.

KNAX7461-E

An error occurred during event processing. (error information=*error-information*, function=open_sender, host=*host-name*)

Issuance of a JP1 event failed.

error-information

Status code output by the API resulting in the error

host-name

Host name of the batch operation server on which JP1/Advanced Shell is running

(S)

Resumes processing.

(O)

Take appropriate action by checking the indicated *error-information*. For details about *error-information*, see *11.4.4 Handling Error Information Displayed in the User-Reply Functionality*.

KNAX7462-E

An error occurred during event processing. (error information=*error-information*, function=*maintenance-information*, destination=*destination-host-name*, sequence no.=*source-serial-number*)

Issuance of a JP1 event failed.

error-information

Status code output by the API resulting in the error

maintenance-information

`event_send` (event transmission) or `check_arrival` (event arrival check)

destination-host-name,

Host name specified in the `HOSTNAME_JP1IM_MANAGER` parameter in the system environment file. If the `HOSTNAME_JP1IM_MANAGER` parameter is omitted, this is the name of the physical host of the batch operation server on which JP1/Advanced Shell is running.

source-serial-number

Sequence number in the source database in JP1/Base

(S)

Resumes processing.

(O)

Take appropriate action by checking the indicated *error-information*. For details about *error-information*, see *11.4.4 Handling Error Information Displayed in the User-Reply Functionality*.

Note that even though this message is displayed, the event might have arrived in JP1/IM - View. If a reply-waiting event has arrived, manually release the accumulation of reply-waiting events.

KNAX7464-E

Transmission of a event failed. (function=check_arrival, destination=*destination-host-name*, sequence no.=*source-serial-number*)

Transmission of a JP1 event from JP1/Base on the local host to JP1/Base on the host where JP1/IM - Manager is running failed.

destination-host-name

Host name specified in the `HOSTNAME_JP1IM_MANAGER` parameter in the system environment file

source-serial-number

Sequence number in the source event database in JP1/Base

(S)

Resumes processing.

(O)

Check the following:

- Whether JP1/Base is installed on the host where JP1/IM - Manager is running
- Whether JP1/Base's event server is running on the host on which JP1/IM - Manager is running
- Whether JP1/Base connection is established between the local host and the host where JP1/IM - Manager is running

Note that even though this message is displayed, the event might have arrived in JP1/IM - View. If a reply-waiting event has arrived, manually release the accumulation of reply-waiting events.

KNAX7465-W

The event is being transferred. (function=check_arrival, destination=*destination-host-name*, sequence no.=*source-serial-number*)

JP1/Base is transferring a JP1 event to the host specified in `HOSTNAME_JP1IM_MANAGER`.

destination-host-name

Host name specified in the `HOSTNAME_JP1IM_MANAGER` parameter in the system environment file

source-serial-number

Sequence number in the source event database in JP1/Base

(S)

Resumes processing.

KNAX7470-I

Data flow control will now be performed. (wait time=*wait-time*)

Data flow control is being performed for issuance of JP1 events.

wait-time

Amount of time (in milliseconds) by which issuance of JP1 events is being delayed in order to control data flow

(S)

Resumes processing.

KNAX7500-I

adshmd started.

The user-reply functionality's management daemon has started.

(S)

Resumes processing.

KNAX7501-I

adshmd stopped.

The user-reply functionality's management daemon has ended.

(S)

Resumes processing.

KNAX7502-E

An attempt to start adshmd failed.

Startup of the user-reply functionality's management daemon failed.

(S)

Terminates processing.

(O)

Eliminate the cause of the error, and then restart the user-reply functionality's management daemon.

KNAX7503-E

An error occurred in adshmd.

An error occurred in the user-reply functionality's management daemon.

(S)

Terminates processing.

(O)

Eliminate the cause of the error, and then restart the user-reply functionality's management daemon.

KNAX7508-I

program-name canceled the response request. (job ID=*job-ID*, line=*line-number*, host name=*host-name*)

A reply-request message handled by the `adshread` com was canceled.

program-name

Name of the service or daemon that canceled the reply-request message

job-ID

Job ID assigned to the batch job by JP1/Advanced Shell

line-number

Line number in the job definition script where the `adshread` command was issued

host-name

Name of the host on which the service or daemon is running

(S)

Resumes processing.

KNAX7509-I

The adshchmsg command canceled the response request. (job ID=*job-ID*, line=*line-number*, host name=*host-name*)

The reply-request message with the reply-request message number specified in the `-n` option of the `adshchmsg` command was canceled.

job-ID

Job ID assigned to the batch job by JP1/Advanced Shell

line-number

Line number in the job definition script where the `adshread` command was issued

host-name

Name of the host on which the service or daemon is running

(S)

Resumes processing.

KNAX7550-I

The service was installed successfully.

Registration of the `AdshmSvcD` or `AdshmSvcE` service was successful.

(S)

Resumes processing.

KNAX7551-E

Failed to install the service. (API=*maintenance-information*, error code=*error-code*)

Registration of the `AdshmSvcD` or `AdshmSvcE` service failed.

(S)

Cancels processing.

(O)

If the cause of this error is that the `AdshmSvcD` or `AdshmSvcE` service has already been registered, ignore this message.

If this message was issued at the time the `adshmsvcd` or `adshmsvce` command was executed, check and, if necessary, correct the options because an invalid option might have been specified.

If neither of the above applies, contact the system administrator.

KNAX7552-E

An error occurred during initialization of the [AdshmSvcD|AdshmSvcE] service. (API=*maintenance-information*, error code=*error-code*)

An error occurred while initializing the `AdshmSvcD` or `AdshmSvcE` service.

(S)

Cancels processing.

(O)

If the displayed maintenance information is `jhs_env_conf_readConfig`, check and, if necessary, revise the system environment file because its contents might be invalid.

If the displayed maintenance information is not `jhs_env_conf_readConfig` or if the contents of the system environment file are correct, contact the system administrator.

KNAX7553-E

The API "*API-name*" failed. (error code=*error-code*)

Creation or opening of Mutex failed during `AdshmSvcD` or `AdshmSvcE` service processing.

(S)

Cancels processing.

(O)

This error might occur because no user with an administrator role for the server is specified for the account used to start the `AdshmSvcD` or `AdshmSvcE` service.

If this error was caused by a specification error, correct the error, and then re-execute the command; otherwise, contact the system administrator.

KNAX7556-E

The logical host name is too long.

The logical host name specified for service registration is too long.

(S)

Cancels the service's registration processing.

(O)

Specify the appropriate logical host name, and then re-execute the command.

KNAX7560-I

The service was set up successfully.

Setup of the `AdshmSvcD` or `dshmSvcE` service was successful.

(S)

Resumes processing.

KNAX7561-E

Setup of the service failed. (API=*maintenance-information*, error code=*error-code*)

Setup of the `AdshmSvcD` or `AdshmSvcE` service failed. This message might be output because user-reply functionality settings are invalid.

(S)

Cancels processing.

(O)

Contact the system administrator, who must check and, if necessary, revise the settings of the user-reply functionality.

KNAX7600-E

Runtime environment for the custom job definition program is invalid.

The startup information for the custom job definition program started by JP1/AJS - View is invalid.

(S)

Terminates processing.

(O)

Take one of the following actions:

- Redefine the custom job because the custom job definition information might be invalid.
- Check that you are using the requisite version of JP1/AJS - View.

KNAX7601-E

The format of the startup data is invalid.

The startup information for the custom job definition program started by JP1/AJS - View is invalid.

(S)

Terminates processing.

(O)

Take one of the following actions:

- Redefine the custom job because the custom job definition information might be invalid.
- Check that you are using the requisite version of JP1/AJS - View.

KNAX7602-E

Ensure that the Advanced Shell custom job is registered correctly.

Information specified during custom job registration into JP1/AJS - View is invalid.

(S)

Terminates processing.

(O)

Take one of the following actions:

- Check the custom job registration information.
- Re-install JP1/Advanced Shell - Custom Job to use a new definition program.

KNAX7603-E

The definition program of Advanced Shell is invalid.

The definition program settings specified during custom job registration into JP1/AJS - View are invalid.

(S)

Terminates processing.

(O)

Take one of the following actions:

- Check the definition program settings specified during custom job registration into JP1/AJS - View.
- Reinstall JP1/Advanced Shell - Custom Job to use a new definition program.

KNAX7604-E

An invalid character was entered in the *definition-field-name* field.

An invalid character was entered in the field indicated by *definition-item-name* in the job definition window.

(S)

Cancels the processing and returns to the input window.

(O)

Delete the invalid character entered in the field indicated by *definition-item-name*.

KNAX7605-E

Enter a value in the *definition-field-name* field.

The required input field indicated by *definition-field-name* was omitted in the job definition window.

(S)

Cancels the processing and returns to the input window.

(O)

Enter a value in the field indicated by *definition-item-name*.

KNAX7606-E

The registered definition data is invalid.

The registered definition information is invalid.

(S)

Cancels processing. The job definition is unchanged.

(O)

Take one of the following actions:

- Re-create the job that was created in JP1/AJS - View's Jobnet Editor, and then re-execute the command.
- If you have used the `ajsdefine` command and JP1/AJS - Definition Assistant to define the job, check the permitted types of characters and lengths of character strings, re-create the job, and then re-execute the command.
- Check that you are using the requisite version of JP1/AJS - View.

KNAX7607-E

The definition data of Advanced Shell cannot be registered. (error code = *error-code*)

The job definition cannot be registered in JP1/AJS.

(S)

Cancels processing. The job definition is unchanged.

(O)

Take one of the following actions:

- Re-create the job that was created in JP1/AJS - View's Jobnet Editor, and then re-execute the command.
- Check that you are using the requisite version of JP1/AJS - View.

KNAX7608-E

The definition data of Advanced Shell cannot be registered. (error code = *error-code*, reason = *error-details*)

The job definition cannot be registered in JP1/AJS.

(S)

Cancels processing. The job definition is unchanged.

(O)

Take one of the following actions:

- Re-create the job that was created in JP1/AJS - View's Jobnet Editor, and then re-execute the command.
- Check that you are using the requisite version of JP1/AJS - View.

KNAX7609-E

The previously registered Advanced Shell definition data could not be collected. (error code = *error-code*)

The previous JP1/Advanced Shell definition information that was registered cannot be obtained.

(S)

Cancels processing. The job definition is unchanged.

(O)

Take one of the following actions:

- Re-create the job that was created in JP1/AJS - View's Jobnet Editor, and then re-execute the command.
- Check that you are using the requisite version of JP1/AJS - View.

KNAX7610-E

Failed to discard the input data. (reason = *error-details*)

Failed to discard the input data.

(S)

Cancels processing. The job definition has not been discarded.

(O)

Take one of the following actions:

- Re-create the job that was created in JP1/AJS - View's Jobnet Editor, and then re-execute the command.
- Check that you are using the requisite version of JP1/AJS - View.

KNAX7611-E

A logical error occurred. (function ID = *function-ID*)

An internal logical error occurred in JP1/Advanced Shell definitions.

(S)

Cancels processing.

(O)

Contact the system administrator.

KNAX7750-E

A logical error occurred. (function ID = *function-ID*, reason = *error-details*)

An internal logical error occurred in message output processing.

(S)

Cancels processing.

(O)

Contact the system administrator.

KNAX7770-E

Startup of the help file failed. (error code = *error-code*)

Startup of the Help file failed.

(S)

Returns to the initial screen.

(O)

Check the return code of the `ShellExecute` function indicated by *error-code*, and take appropriate action.

KNAX7771-E

The help file was not found. (file name = *file-name*)

The Help file cannot be found.

(S)

Returns to the initial screen.

(O)

Attempt to perform a recovery installation of the JP1/Advanced Shell - Custom Job.

KNAX7772-E

Startup of the help file failed. (reason = *error-details*)

Startup of the Help file failed.

(S)

Returns to the initial screen.

(O)

Attempt to perform a recovery installation of JP1/Advanced Shell - Custom Job.

KNAX7773-E

Startup of the help file failed.

Startup of the Help file failed.

(S)

Returns to the initial screen.

(O)

Attempt to perform a recovery installation of JP1/Advanced Shell - Custom Job.

KNAX7800-I

adshcollect:RAS completed collection of *file-name*
collected-file-name
collected-file-name
...

A `tar` file (*file-name*) containing the indicated collected files was created.

(S)

Terminates processing.

(O)

Give the created file to the system administrator.

KNAX7801-I

adshcollect:RAS completed collection of *file-name*
collected-file-name
collected-file-name
...

The file indicated by *file-name* containing the indicated collected files was created.

(S)

Terminates processing.

(O)

Compress the created file with the user's compression tool, and then provide it to the system administrator.

KNAX7802-E

Usage: adshcollect Directory [-f FileName] [-e FileName] [-h LogicalHostName]
Directory : Specify output directory
-f FileName : Specify a config file

-e FileName : Specify an environment file
-h LogicalHostName : Specify a logical host

Option settings are invalid.

(S)

Terminates processing.

(O)

Specify the correct options, and then re-execute the command.

KNAX7803-E

adshcollect:RAS error:*output-directory* (Permission denied).

There is no access permission for *output-directory*.

(S)

Terminates processing.

(O)

Grant access permission for *output-directory* or specify another directory, and then re-execute the command.

KNAX7804-E

adshcollect:RAS error:*output-directory* (not found or not a directory).

output-directory does not exist or it is not a directory.

(S)

Terminates processing.

(O)

Specify the correct output destination, and then re-execute the command.

KNAX7805-E

adshcollect:RAS error:*definition-file-name* (not found or not a file).

definition-file-name does not exist or it is not a file.

(S)

Terminates processing.

(O)

Specify the correct definition file, and then re-execute the command.

KNAX7806-E

adshcollect:RAS error:*definition-file-name* (Permission denied).

There is no access permission for *definition-file-name*.

(S)

Terminates processing.

(O)

Set the access permission for the definition file, and then re-execute the command.

KNAX7807-E

> adshcollect:RAS error:*environment-file-name* (not found or not a file).

environment-file-name does not exist or it is not a file.

(S)

Terminates processing.

(O)

Specify the correct environment file, and then re-execute the command.

KNAX7808-E

> adshcollect:RAS error:*environment-file-name* (Permission denied).

There is no access permission for *environment-file-name*.

(S)

Terminates processing.

(O)

Set the access permission for the environment file, and then re-execute the command.

KNAX7809-E

> adshcollect:RAS error:*definition-file-name* ("*keyword*" Syntax Error).

An invalid keyword was specified in *definition-file-name*.

(S)

Terminates processing.

(O)

Specify the definition file correctly, and then re-execute the command.

KNAX7810-E

> adshcollect:RAS error:*specified-value* (not found or not a file).

specified-value for keyword does not exist in the definition file, or it is not a file.

(S)

Terminates processing.

(O)

Specify the definition file correctly, and then re-execute the command.

KNAX7811-W

> adshcollect:RAS error:*specified-value* (Permission denied).

There is no access permission for *specified-value* for keyword in the definition file.

(S)

Continues processing.

(O)

Grant the access permission for the specified value or specify another value, and then re-execute the command.

KNAX7812-E

adshcollect:RAS error:*specified-value* (not found or not a directory).

specified-value does not exist in the definition file, or it is not a directory.

(S)

Terminates processing.

(O)

Specify the definition file correctly, and then re-execute the command.

KNAX7813-E

adshcollect:RAS error:*specified-value* (not found or not a directory).

specified-value for the required keyword does not exist in the environment file, or it is not a directory.

(S)

Terminates processing.

(O)

Specify the environment file correctly, and then re-execute the command.

KNAX7814-E

adshcollect:RAS error:*specified-value* (Permission denied).

There is no access permission for *specified-value* for keyword in the environment file.

(S)

Terminates processing.

(O)

Grant access permission for the specified value or specify another value, and then re-execute the command.

KNAX7880-E

Failed to "*API-name*". (reason=*error-details*)

OS's API processing failed for the reason indicated by *error-details*.

(S)

Terminates the processing if *OS's-API-name* is not `dladdr`; resumes the processing if it is `dladdr`.

(O)

If *OS's-API-name* is not `dladdr`, contact the system administrator. Eliminate the cause of the error, and then re-execute the command.

KNAX7892-I

adshexec received abnormal signal.

A program error notification signal for the job controller was received.

(S)

Resumes processing.

KNAX7893-I

adshexec received signal "*signal-name*".

The signal indicated by *signal-name* for the job controller was received. This message is output when JP1/Advanced Shell has received a signal required for execution control.

(S)

Resumes processing.

KNAX7894-E

adshexec is ended because of terminate request of second times.

The second `SIGTERM` signal for the job controller was received.

(S)

Terminates the job controller immediately. The system does not perform postprocessing, including deletion of temporary files and postprocessing of files.

(O)

If necessary, perform postprocessing on the resources created by the batch job.

KNAX7895-E

adshexec ended abnormally.

The job controller terminated with an error.

(S)

Terminates the job controller immediately. The system does not perform postprocessing, including deletion of temporary files and postprocessing of files.

(O)

If necessary, perform postprocessing on the resources created by the batch job.

KNAX7896-I

adshexec received terminate request.

A termination request for the job controller was received.

(S)

Performs postprocessing and then terminates the job controller. If this message was output while debug execution was stopped, the system re-executes the job definition script, performs postprocessing, and then terminates the job controller.

KNAX7897-E

Fatal error occurred in *maintenance-information*.

A fatal error occurred in the `adshexec` command.

(S)

Terminates processing.

(O)

Contact the system administrator.

KNAX7900-I

The manual has not been installed.

Copy the HTML files and image files from the manual installation media.

The manual has not been copied from the manual installation medium to the installation directory.

(S)

Keeps the web browser open until the **Close** button is clicked.

(O)

Click the **Close** button to close the web browser, and then copy the manual from the manual installation medium to the installation directory by following the procedure explained in the manual.

KNAX7901-I

The job controller will wait for all asynchronous processes at the end of the job.

The job controller places all asynchronous processes in wait status during job termination.

This message is not output for a child job that was started with `MERGE` specified for the `SPOOLJOB_CHILDJOB` parameter of the root job's environment file.

(S)

Resumes processing.

KNAX7902-I

The job controller will run in *input-mode*.

The job controller will run in the mode indicated by *input-mode*.

input-mode

Input mode of the job controller. One of the modes shown below is displayed. For details about the input modes, see *3.1.2(2) Job input modes*.

Displayed information	Meaning
`tty stdin mode`	Terminal input mode. The standard input is associated with the terminal.
`non-tty stdin mode`	Non-terminal input mode. The standard input is not associated with the terminal.

(S)

Resumes processing.

KNAX7999-I

> Advanced Shell ended. exit status=*exit-status-code*

The root job's job controller terminated the batch job with the return code indicated by *exit-status-code*.

(S)

Resumes processing.

KNAX9000-E

> The validity period for a product expired. program=*command-name*

The license has expired.

command-name

The name of the command that caused the error

(S)

Terminates processing.

(O)

To continue using the program product, install a commercial version.

KNAX9001-E

> Failed to authenticate the product. (*command-name*, *internal-information*)

License authentication failed.

command-name

The name of the command that caused the error

internal-information

Internal information that shows the error details

(S)

Terminates processing.

(O)

Contact the system administrator.

KNAX9002-E

> An error occurred. detail=*command-name*, adshhlicauth error, rc=*return-code*

An unexpected error occurred during license authentication.

command-name

The name of the command that caused the error

return-code

Internal information that shows the error details

(S)

Terminates processing.

(O)

Contact the system administrator.

11.4 Details of errors

This section explains the information that is displayed by *error-details* in Windows and UNIX message texts and that is specific to JP1/Advanced Shell.

11.4.1 Details of errors (Windows)

The messages issued by JP1/Advanced Shell might contain information about the C run-time function and Win32(R) API errors.

The table below lists and describes the causes of typical C run-time function errors that are most likely to occur in a JP1/Advanced Shell environment (Windows) and the actions to be taken. For information about other errors that are not listed in the table and Win32 API errors, see the documentation for the Windows being used.

Table 11–5: Causes of and actions in response to C run-time function errors (Windows)

Mnemonic	Error details	Cause	Action
`ENOENT`	`No such file or directory`	A file or directory cannot be found.	Check if the file exists.
`EIO`	`Input/Output error`	An input/output error occurred.	Take appropriate action according to the Windows or hardware information.
`ENXIO`	`No such device or address`	There is no access permission for the file.	Check if there is such a device or that the device is enabled. If the device is disabled, enable it. For other causes, check the documentation for the Windows being used.
`E2BIG`	`Arg list too long`	There is a shortage of area for the processing program's arguments or environment variables.	Check the processing program's arguments. Check and, if necessary, revise the environment variables specified in parameters such as `export` and the usage of extended script commands in the file management function, and then delete any unneeded environment variable settings.
`EAGAIN`	`Resource temporarily unavailable`	There are too many processes, or a temporary memory shortage has occurred.	If the error recurs when the command is re-executed, terminate unneeded processes.
`ENOMEM`	`Not enough space`	Possible causes are as follows: • A new process cannot be created due to a shortage of swap area or virtual memory. • There are too many processes, or some processes are using too much memory.	Take the following actions: • If there is a shortage of swap area or virtual memory, expand it. If the swap area or virtual memory cannot be expanded, terminate unneeded processes. • If some processes are using too much memory, evaluate whether they can be terminated.
`EACCES`	`Permission denied`	Possible causes are as follows: • The access permission is invalid. • A file must be specified in an argument of a JP1/Advanced Shell command, but a directory was specified.	Take the following actions: • Check if the file access permission is correct. • Check the argument of the JP1/Advanced Shell command to determine whether a directory is specified instead of a file. • If you specified the `CHILDJOB_SHEBANG` parameter, check the specified parameter value and

Mnemonic	Error details	Cause	Action
EACCES	Permission denied	Possible causes are as follows: - The access permission is invalid. - A file must be specified in an argument of a JP1/Advanced Shell command, but a directory was specified.	#! *executable-program-path* at the beginning of the file whose execution was attempted.
EFAULT	Bad address	An attempt was made to write data into an inaccessible area. The disk to which the data is to be written might have been disconnected.	If disks are being switched during system switchover, ignore this error message because there is no problem. If you disconnected the disk by mistake, restore the corresponding file from its backup or initialize the file first before using it. If neither of the above applies, contact the system administrator.
EEXIST	File exists	An attempt was made to create a file, but the file already exists.	Rename the file and re-execute the command. If the existing file is not needed, delete it and re-execute the command.
EINVAL	Invalid argument	An error was detected in memory management information.	Contact the system administrator.
ENFILE	Too many open files in system	The number of open files exceeded the maximum.	Check the total number of files in use in the system and close unneeded files.
EMFILE	Too many open files	Too many files are open in the corresponding process.	Contact the system administrator.
EFBIG	File too large	The file size exceeded the system limit.	Check and, if necessary, revise the size of a file to be used.
ENOSPC	No space left on device	There is not enough free space in the file system.	Allocate more free space.

11.4.2 Details of errors (UNIX)

The table below lists and describes the causes of typical errors that occur in a JP1/Advanced Shell environment and the actions to take. For information about other errors that are not listed in the table, see the documentation for the UNIX being used.

This subsection describes only the errors that are most likely to occur in a JP1/Advanced Shell environment. For details about other errors that are not described here, check the UNIX `errno` definition file (`errno.h`), which uses the mnemonic corresponding to the error number (`errno`) displayed in the messages.

Table 11–6: Causes of and actions in response to the error details (UNIX)

Mnemonic	Error details	Cause	Action
ENOENT	No such file or directory	A file or directory cannot be found.	Check if the file exists.
EIO	I/O error	An input/output error occurred.	Take appropriate action according to the UNIX or hardware information.
ENXIO	No such device or address	There is no access permission for the file.	Check if there is such a device or that the device is enabled. If the device is disabled, enable it. For other causes, check the documentation of the UNIX being used.

Mnemonic	Error details	Cause	Action
E2BIG	`Arg list too long`	There is a shortage of area for the processing program's arguments or environment variables.	Check the processing program's arguments. Check and, if necessary, revise the environment variables specified in parameters such as `export` and the usage of extended script commands in the file management function, and then delete unneeded environment variable settings.
EAGAIN	`Resource temporarily unavailable`	There are too many processes, or a temporary memory shortage has occurred.	If the error recurs when the command is re-executed, terminate unneeded processes.
ENOMEM	`Not enough space`	Possible causes are as follows: • A new process cannot be created due to a shortage of swap area or virtual memory. • There are too many processes or some processes are using too much memory.	Take the following actions: • If there is a shortage of swap area or virtual memory, expand it. If the swap area or virtual memory cannot be expanded, terminate unneeded processes. • If some processes are using too much memory, evaluate whether they can be terminated.
EACCES	`Permission denied`	The access permission is invalid.	Check if the file access permission is correct.
EFAULT	`Bad address`	An attempt was made to write data into an inaccessible area. The disk to which data is to be written might have been disconnected.	If disks are being switched during system switchover, ignore this error message because there is no problem. If you disconnected the disk by mistake, restore the corresponding file from its backup or initialize the file first before using it. If neither of the above applies, contact the system administrator.
EEXIST	`File exists`	An attempt was made to create a file, but the file already exists.	Rename the file, and then re-execute the command. If the existing file is not needed, delete it and re-execute the command.
EINVAL	`Invalid argument`	An error was detected in the memory management information.	Contact the system administrator.
ENFILE	`File table overflow`	The number of open files exceeded the maximum.	Increase in the UNIX kernel parameter the maximum number of files that can be open in the system (`maxuproc` × `nofiles`).
EMFILE	`Too many open files`	Too many files are open in the corresponding process.	Increase in the UNIX kernel parameter the maximum number of files that can be open in a process (`nofiles`).
EFBIG	`File too large`	The file size exceeded the system limit.	Check and, if necessary, revise the size of a file to be used.
ENOSPC	`No space left on device`	There is not enough free space in the file system.	Allocate more free space.
ENAMETOOLONG	`File name too long`	A file name is too long.	Check and, if necessary, revise the file name length.

11.4.3 Details of errors (specific to JP1/Advanced Shell)

The table below lists and describes the causes of and actions to be taken for the error details that are displayed specifically by JP1/Advanced Shell.

Table 11–7: Causes of and actions in response to the error details (specific to JP1/Advanced Shell)

Message ID	Error details	Cause	Action
KNAX4419-E	A line exceeds the maximum line size	The size of a line exceeds the maximum.	Check the size of the line and specify it to not exceed the maximum.
KNAX4420-E	The common application data folder cannot be found	A common application data folder cannot be found.	Check the execution environment for any problem.
	The shared documents folder cannot be found	A shared document folder cannot be found.	Check the execution environment for any problem.
KNAX6035-E	The file descriptor is incorrectly specified	A specified file descriptor is not a one-digit number.	Check and, if necessary, revise the specified file descriptor.
	The file descriptor cannot be used	A specified file descriptor is for a closed file or a file whose manipulation is prohibited by another process.	Check if the file descriptor is open. If it is open, check if its manipulation is being prohibited by a means such as locking by another process.
	The file descriptor is not open for writing	An attempt was made to write to the file descriptor of a closed file.	Check and, if necessary, revise the specified file descriptor.
	The file descriptor is not open for reading	An attempt was made to read the file descriptor of a closed file.	Check and, if necessary, revise the specified file descriptor.
	No background process exists	A file descriptor for a background process was specified, but the background process did not exist.	Check if the background process is running or has already been terminated.
KNAX6305-E	The file is not a regular file	A specified file is not a regular file.	Check and, if necessary, revise the specified file.
KNAX6333-E	The file is not a regular file	A specified file is not a regular file.	Check and, if necessary, revise the specified file.
KNAX6588-E	Error in signal handler	An error occurred during signal handler processing.	Check the execution environment for any error.

11.4.4 Handling Error Information Displayed in the User-Reply Functionality

This section explains the information output by the error messages listed below when the user-reply functionality is used and describes the appropriate actions to be taken in response to the messages. When the `adshcollect` command is used to collect user-reply functionality information, you must execute the command with the administrator permission.

- KNAX7402-E
- KNAX7460-E
- KNAX7461-E
- KNAX7462-E

(1) Error information displayed in the KNAX7402-E message and how to respond to it

Table 11–8: Error information displayed in the KNAX7402-E message and how to respond to it

Error number	Meaning	Corrective action
`1`	An unimplemented API call was made.	Contact a system administrator. If you are a system administrator, collect information in accordance with *10. Troubleshooting*, and then find a workaround or take corrective action.
`2`	An attempt to reference shared memory failed. The user-reply functionality's management daemon or service might not be running.	**Operator** Check that the job environment file settings are correct. The `SPOOL_DIR` parameter cannot be specified in a job environment file. In addition, contact a user with an administrator role on the machine and confirm the following: • The user-reply functionality's management daemon or service is running • The environment file settings are correct **User with an administrator role on the machine** Confirm the following: • The user-reply functionality's management daemon or service is running • The environment file settings are correct The `SPOOL_DIR` parameter can be specified only in the system environment file; it cannot be specified in a job environment file. • The user-reply functionality's management daemon or service was restarted after changes were made to the system environment file If the problem persists, contact a system administrator. If you are a system administrator, collect information in accordance with *10. Troubleshooting*, and then find a workaround or take corrective action.
`3`	The amount of unused memory space is insufficient.	Contact a system administrator. If you are a system administrator, re-estimate the amount of memory required.
`4`	An invalid argument was passed.	Contact a system administrator. If you are a system administrator, collect information in accordance with *10. Troubleshooting*, and then find a workaround or take corrective action.
`5`	A data inconsistency was detected.	Contact a system administrator. If you are a system administrator, collect information in accordance with *10. Troubleshooting*, and then find a workaround or take corrective action.
`6`	An unsupported character encoding was specified.	Contact a system administrator. If you are a system administrator, collect information in accordance with *10. Troubleshooting*, and then find a workaround or take corrective action.
`7`	An underflow occurred.	Contact a system administrator. If you are a system administrator, collect information in accordance with *10. Troubleshooting*, and then find a workaround or take corrective action.
`8`	An overflow occurred.	Contact a system administrator. If you are a system administrator, collect information in accordance with *10. Troubleshooting*, and then find a workaround or take corrective action.
`9`	The order of API calls was not valid.	Contact a system administrator. If you are a system administrator, collect information in accordance with *10. Troubleshooting*, and then find a workaround or take corrective action.
`10`	An inconsistency occurred in the status of an internal object.	Contact a system administrator. If you are a system administrator, collect information in accordance with *10. Troubleshooting*, and then find a workaround or take corrective action.

Error number	Meaning	Corrective action
11	Access to a resource was denied.	Check that you have write permission for the log directories specified in the environment file and the log files under them.
12	A specified file does not exist.	Check that you have write permission for the log directories specified in the environment file and the log files under them.
13	A file could not be opened.	Check that you have write permission for the log directories specified in the environment file and the log files under them.
14	Memory mapping could not be created for a file.	Check that you have write permission for the log directories specified in the environment file and the log files under them.
15	An error occurred in reading from a file.	Check that you have read permission for the log directories specified in the environment file and the log files under them.
16	An error occurred in writing to a file.	Check that you have write permission for the log directories specified in the environment file and the log files under them.
17	An error occurred in seek processing on a file.	Check that you have write permission for the log directories specified in the environment file and the log files under them.
18	An error occurred in flush processing on a file.	Check that sure you have write permission for the log directories specified in the environment file and the log files under them.
19	An error occurred in renaming a file.	Check that sure you have write permission for the log directories specified in the environment file and the log files under them.
20	An error occurred in copying a file.	Check that you have write permission for the log directories specified in the environment file and the log files under them.
21	An error occurred in deleting a file.	Check that you have write permission for the log directories specified in the environment file and the log files under them.
22	An attempt to create a directory failed.	Check that you have write permission for the log directories specified in the environment file and the log files under them.
23	An interprocess lock failed.	Contact a system administrator. If you are a system administrator, collect information in accordance with *10. Troubleshooting*, and then find a workaround or take corrective action.
24	An attempt was made to use a function that was disabled by the user program.	Contact a system administrator. If you are a system administrator, collect information in accordance with *10. Troubleshooting*, and then find a workaround or take corrective action.
25	An attempted overwrite was prohibited by the user program.	Contact a system administrator. If you are a system administrator, collect information in accordance with *10. Troubleshooting*, and then find a workaround or take corrective action.
400	A parameter was not valid.	Contact a system administrator. If you are a system administrator, collect information in accordance with *10. Troubleshooting*, and then find a workaround or take corrective action.
404	A reply-request message was deleted by the `adshchmsg` command, or shared memory could not be referenced.	If you are the operator, contact a user with an administrator role on the machine and check the following: • Whether the reply-request message was deleted by means of the `-d` option to the `adshchmsg` command • Whether the user-reply functionality's management daemon or service is running If you are a user with an administrator role on the machine, check that the user-reply functionality's management daemon or service is running. If the problem persists, contact a system administrator. If you are a system administrator, collect information in accordance with *10. Troubleshooting*, and then find a workaround or take corrective action.

Error number	Meaning	Corrective action
409	Reply-request messages from the same PID were output.	The reply-request message might have been output simultaneously in multiple threads. Check and, if necessary, revise the application.
503	The number of reply-waiting events exceeded the maximum.	Check and, if necessary, revise the value of the USERREPLY_WAIT_MAXCOUNT parameter.

(2) Error information displayed in the KNAX7460-E, KNAX7461-E, and KNAX7462-E messages and how to respond to it

Table 11–9: Error information displayed in the KNAX7460-E, KNAX7461-E, and KNAX7462-E messages and how to respond to it

Error number	Meaning	Corrective action
10	A parameter was not valid.	Contact a system administrator. If you are a system administrator, collect information in accordance with *10. Troubleshooting*, and then find a workaround or take corrective action.
11	The order is which functions were issued was not valid.	Contact a system administrator. If you are a system administrator, collect information in accordance with *10. Troubleshooting*, and then find a workaround or take corrective action.
30	A specified attribute was already registered.	Contact a system administrator. If you are a system administrator, collect information in accordance with *10. Troubleshooting*, and then find a workaround or take corrective action.
32	The maximum number of extended event attributes that can be registered has been exceeded.	Contact a system administrator. If you are a system administrator, collect information in accordance with *10. Troubleshooting*, and then find a workaround or take corrective action.
33	The total size of all extended event attributes that can be registered has been exceeded.	Contact a system administrator. If you are a system administrator, collect information in accordance with *10. Troubleshooting*, and then find a workaround or take corrective action.
40	A connection to the event service could not be established.	Check that the JP1/Base event service has started on the local host.
43	An input/output error occurred.	Contact a system administrator. If you are a system administrator, collect information in accordance with *10. Troubleshooting*, and then find a workaround or take corrective action.
50	A JP1/Base library could not be found.	Install JP1/Base and re-execute the job.
51	There is not enough memory.	Contact a system administrator. If you are a system administrator, re-estimate the amount of memory required.
52	The number of open files has reached the maximum.	Contact a system administrator. If you are a system administrator, re-estimate the number of FDs that can be opened.
60	A JP1 event has not been initialized.	Contact a system administrator. If you are a system administrator, collect information in accordance with *10. Troubleshooting*, and then find a workaround or take corrective action.
70	A system error occurred.	Contact a system administrator.

Error number	Meaning	Corrective action
`70`	A system error occurred.	If you are a system administrator, collect information in accordance with *10. Troubleshooting*, and then find a workaround or take corrective action.

Appendixes

A. Coverage Information That Is Acquired

Coverage information includes C0 information and C1 information. The items for which coverage information is acquired are described in the sections below.

Note that coverage information is not acquired for the following items:

- Conditional expressions
- Arithmetic operations
- Variables

A.1 Commands for which coverage information is acquired

The tables in this section show for the following types of commands the individual commands for which coverage information is acquired:

- Standard shell commands
- Extended shell commands
- Extended script commands
- Commands other than the above

(1) Standard shell commands

(a) Special built-in commands

Table A–1: Special built-in commands for which coverage information is acquired

Item	C0	C1
`.` (dot) command	Y	N
`:` (colon) command	Y	N
`break` command	Y	N
`continue` command	Y	N
`eval` command	Y	N
`exec` command	Y	N
`exit` command	Y	N
`export` command	Y	N
`readonly` command	Y	N
`return` command	Y	N
`set` command	Y	N
`shift` command	Y	N
`trap` command	Y	N
`typeset` command	Y	N

Item	C0	C1
unset command	Y	N

Legend:

Y: Coverage information is acquired and displayed.

N: No coverage information is acquired.

(b) Regular built-in commands

Table A–2: Regular built-in commands for which coverage information is acquired

Item	C0	C1
alias command	Y	N
builtin command	Y	N
cd command	Y	N
command	Y	N
echo command	Y	N
false command	Y	N
getopts command	Y	N
kill command	Y	N
let command	Y	N
print command	Y	N
pwd command	Y	N
read command	Y	N
test command	Y	N
times command	Y	N
true command	Y	N
ulimit command	Y	N
umask command	Y	N
unalias command	Y	N
wait command	Y	N
whence command	Y	N

Legend:

Y: Coverage information is acquired and displayed.

N: No coverage information is acquired.

(2) Extended shell commands

Table A–3: Extended shell commands for which coverage information is acquired

Item	C0	C1
adshecho command	Y	N

Item	C0	C1
adshread command	Y	N
adshscripttool command	Y	N

Legend:

Y: Coverage information is acquired and displayed.

N: No coverage information is acquired.

(3) Extended script commands

Table A–4: Extended script commands for which coverage information is acquired

Item	C0	C1
#-adsh_file command	Y	N
#-adsh_file_temp command	Y	N
#-adsh_job command	Y	N
#-adsh_job_stop command	Y	N
#-adsh_path_var command	Y	N
#-adsh_rc_ignore command	Y	N
#-adsh_script command	Y	N
#-adsh_spoolfile command	Y	N
#-adsh_step_start command	Y	Y[#1]
#-adsh_step_error command	Y	Y[#2]
#-adsh_step_end command	Y	N

Legend:

Y: Coverage information is acquired and displayed.

N: No coverage information is acquired.

#1

For details about the C1 information that is displayed, see *3.9.4(3)(e) #-adsh_step_start command*.

#2

For details about the C1 information that is displayed, see *3.9.4(3)(f) #-adsh_step_error command*.

(4) Commands other than the above

The following table indicates whether coverage information is acquired for commands outside of JP1/Advanced Shell (including OS commands and user-created commands).

Table A–5: Commands other than the above for which coverage information is acquired

Item	C0	C1
Commands other than the above	Y	N

Legend:

Y: Coverage information is acquired and displayed.

N: No coverage information is acquired.

A.2 Control statements for which coverage information is acquired

The following table indicates for each control statement whether coverage information is acquired.

Table A–6: Control statements for which coverage information is acquired

Item	C0	C1
`if`	N	Y
`if` condition	Y	N
`then`	N	N
`elif`	N	Y
`elif` condition	Y	N
`else`	N	Y
`fi`	N	C
`for`	N	Y
variable	N	N
`in`	N	N
wordlist	N	N
`do`	N	N
`done`	N	Y
`while`	N	Y
`while` condition	Y	N
`until`	N	Y
`until` condition	Y	N
`case`	N	N
expression	N	N
pattern`)`	N	Y
`*)`	N	Y
`;;`	N	N
`esac`	N	C

Legend:

Y: Coverage information is acquired and displayed.

C: Coverage information is acquired and displayed in the following cases:

`fi`: When the `else` clause is not specified

`esac`: When the `*` pattern is not specified. The `*` pattern is the default matching pattern for when none of the other patterns in the `case` statement is matched.

N: No coverage information is acquired.

A.3 Functions for which coverage information is acquired

The following table indicates whether coverage information is acquired when a function is called. No coverage information is acquired when a function is defined.

Table A–7: Function calls for which coverage information is acquired

Item	C0	C1
Call of the function name	Y	N
Execution of the `function`	N	N
Execution of the function name	N	N
Execution of the `( )` portion	N	N
Execution of processing starting with `{`	N	N
Execution of commands and control statements	Y	C
Execution of processing ending with `}`	N	N

Legend:

Y: Coverage information is acquired and displayed.

C: Coverage information is acquired and displayed when there is C1 information in the control statements that are executed.

N: No coverage information is acquired.

A.4 Metacharacters for which coverage information is acquired

Coverage information is acquired only for command separators, not for other uses of metacharacters. The following table indicates the command separators for which coverage information is acquired.

Table A–8: Command separators for which coverage information is acquired

Item	C0	C1
`cmd_1;cmd_2`	Y	N
`cmd_1&&cmd_2`	Y	N
`cmd_1||cmd_2`	Y	N

Legend:

Y: Coverage information is acquired and displayed.

N: No coverage information is acquired.

Note that no coverage information is acquired for metacharacters that are used for the following purposes:

- Comments
- Line continuations
- Variable substitutions
- Command substitutions
- File name substitutions
- Redirects
- Here documents

- Command groupings
- Other metacharacters

A.5 Shell variable operations for which coverage information is acquired

The following table shows the coverage information acquired when you assign a value to a shell variable.

Table A–9: Shell variable operations for which coverage information is acquired

Item	C0	C1
shell-variable=*value*	Y	N

Legend:

Y: Coverage information is acquired and displayed.

N: No coverage information is acquired.

B. Reference Material for This Manual

This appendix provides reference information for reading the manual.

B.1 Related publications

This manual is part of a related set of manuals. The manuals in the set are listed below (with the manual numbers):

JP1/AJS

- *Job Management Partner 1 Version 10 Job Management Partner 1/Automatic Job Management System 3 Overview* (3021-3-318(E))
- *Job Management Partner 1 Version 10 Job Management Partner 1/Automatic Job Management System 3 System Design (Configuration) Guide* (3021-3-319(E))
- *Job Management Partner 1 Version 10 Job Management Partner 1/Automatic Job Management System 3 System Design (Configuration) Guide* (3021-3-319(E))
- *Job Management Partner 1 Version 10Job Management Partner 1/Automatic Job Management System 3 System Design (Work Tasks) Guide* (3021-3-320(E))
- *Job Management Partner 1 Version 10 Job Management Partner 1/Automatic Job Management System 3 Configuration Guide 1* (3021-3-321(E))
- *Job Management Partner 1 Version 10 Job Management Partner 1/Automatic Job Management System 3 Configuration Guide 2* (3021-3-322(E))
- *Job Management Partner 1 Version 10 Job Management Partner 1/Automatic Job Management System 3 Administration Guide* (3021-3-323(E))
- *Job Management Partner 1 Version 10 Job Management Partner 1/Automatic Job Management System 3 Troubleshooting* (3021-3-324(E))
- *Job Management Partner 1 Version 10 Job Management Partner 1/Automatic Job Management System 3 Operator's Guide* (3021-3-325(E))
- *Job Management Partner 1 Version 10 Job Management Partner 1/Automatic Job Management System 3 Command Reference 1* (3021-3-326(E))
- *Job Management Partner 1 Version 10 Job Management Partner 1/Automatic Job Management System 3 Command Reference 2* (3021-3-327(E))
- *Job Management Partner 1 Version 10 Job Management Partner 1/Automatic Job Management System 3 Linkage Guide* (3021-3-328(E))
- *Job Management Partner 1 Version 10 Job Management Partner 1/Automatic Job Management System 3 Messages 1* (3021-3-329(E))
- *Job Management Partner 1 Version 10 Job Management Partner 1/Automatic Job Management System 3 Messages 2* (3021-3-330(E))

JP1/NETM/DM

- *Job Management Partner 1 Version 9 Job Management Partner 1/Software Distribution Description and Planning Guide* (3020-3-S79(E)), for Windows systems
- *Job Management Partner 1 Version 9 Job Management Partner 1/Software Distribution Administrator's Guide Volume 1* (3020-3-S81(E)), for Windows systems
- *Job Management Partner 1 Version 8 Job Management Partner 1/Software Distribution SubManager* (3020-3-L42(E)), for UNIX systems

JP1/Base

- *Job Management Partner 1 Version 10 Job Management Partner 1/Base User's Guide* (3021-3-301(E))
- *Job Management Partner 1 Version 9 Job Management Partner 1/Base User's Guide* (3020-3-R71)

JP1/IM

- *Job Management Partner 1 Version 10 Job Management Partner 1/Integrated Management - Manager Configuration Guide* (3021-3-306(E))
- *Job Management Partner 1 Version 10 Job Management Partner 1/Integrated Management - Manager Administration Guide* (3021-3-307(E))

B.2 Conventions: abbreviations

This manual uses the following abbreviations for product names:

Abbreviation		Full name or meaning
JP1/Advanced Shell		Job Management Partner 1/Advanced Shell
		Job Management Partner 1/Advanced Shell - Developer
JP1/AJS	JP1/AJS3	Job Management Partner 1/Automatic Job Management System 3 - Agent Job Management Partner 1/Automatic Job Management System 3 - Manager Job Management Partner 1/Automatic Job Management System 3 - View
JP1/AJS - Agent	JP1/AJS3 - Agent	Job Management Partner 1/Automatic Job Management System 3 - Agent
JP1/AJS - Manager	JP1/AJS3 - Manager	Job Management Partner 1/Automatic Job Management System 3 - Manager
JP1/AJS - View	JP1/AJS3 - View	Job Management Partner 1/Automatic Job Management System 3 - View
JP1/IM	JP1/IM - Manager	Job Management Partner 1/Integrated Management - Manager
	JP1/IM - View	Job Management Partner 1/Integrated Management - View
UNIX	Linux	Red Hat Enterprise Linux 5(AMD/Intel 64) Red Hat Enterprise Linux 5(x86) Red Hat Enterprise Linux 5 Advanced Platform(AMD/Intel 64) Red Hat Enterprise Linux 5 Advanced Platform(x86) Red Hat Enterprise Linux Server 6 (64-bit x86_64) Red Hat Enterprise Linux Server 6 (32-bit x86)
	AIX	AIX V6.1 AIX V7.1
	HP-UX	HP-UX 11i V3 (IPF)
	Solaris	Solaris 10 (SPARC) Solaris 11 (SPARC)

B.3 Conventions: directory names

This manual uses the term *directory* wherever possible as a generic term for what Windows calls a *folder* and UNIX calls a *directory*.

In connection with this convention, this manual uses / as the directory delimiter. In Windows-specific cases, \ is used as the folder delimiter.

B.4 Conventions: KB, MB, GB, and TB

This manual uses the following conventions:

- 1 KB (kilobyte) is 1,024 bytes.
- 1 MB (megabyte) is $1{,}024^2$ bytes
- 1 GB (gigabyte) is $1{,}024^3$ bytes.
- 1 TB (terabyte) is $1{,}024^4$ bytes.

C. Glossary

This glossary defines the terminology used in this manual.

.env file

A file in which are set the path names to the `ENV` environment variables and that is loaded when the shell starts. You can use the `KSH_ENV_READ` environment setting parameter to specify whether this file is to be loaded.

argument

A generic term for an item that is specified following a command name. Multiple arguments are separated by a delimiter on the command line or in a job definition script.

arithmetic operation

Any of the calculations performed using arithmetic operators in a job definition script. In an arithmetic operation, the values assigned to variables are handled as numeric values.

base name

The portion of a file name excluding the `.`*extension* portion. For example, the base name of `adshexec.exe`, the program for executing batch jobs, is `adshexec`.

batch job

A job executed by batch processing.

batch operation server

A server on which JP1/Advanced Shell is installed that is used to execute batch jobs. JP1/AJS - Agent or JP1/AJS - Manager must be installed when JP1/AJS is used.

batch processing

The process of gathering collected data and transactions and processing them in bulk on a regular schedule, such as every day, week, or month.

breakpoint

Coding that forces execution to stop and that is inserted into a job definition script in order to pause the processing during development in order to check the operational status of the job definition script. The debugger interrupts the processing at a breakpoint so that the developer can check the values of variables and registers at the time of the interruption.

built-in command

A command that is included as part of the shell, and thus requires only the shell itself to be executed. Built-in commands can be used in job definition scripts, and they can be executed from the shell or from the command prompt. JP1/Advanced Shell supports regular built-in commands and special built-in commands.

child job

A job whose job definition script is executed as a descendant process of the root job. Child jobs are executed in accordance with one of the following parameter specifications or their default settings:

- `CHILDJOB_EXT` parameter

- CHILDJOB_PGM parameter
- CHILDJOB_SHEBANG parameter

child job execution log output file

The output file for the job execution log of a child job, which is created by the child job and output in the spool job directory of the root job.

command

Generic name for any instruction that can be used in JP1/Advanced Shell. Commands are executed from the shell or the command prompt, as well as from job definition scripts.

command grouping

Facility for executing multiple commands as a unit in JP1/Advanced Shell.

command line

The line displayed to the user for entering commands. In the Windows command prompt, input is entered after the > on the command line. In the UNIX shell, input is entered after the % on the command line.

command prompt

The window in a Windows environment that requests that a command be entered.

command separator

The functionality that allows developers to write more than one command on a single line of a job definition script in JP1/Advanced Shell.

conditional

A test that controls the processing branch that is to execute based on the results of a conditional expression in a control statement in a job definition script.

conditional expression

A formula used in a job definition script that expresses a calculation using numeric comparisons, string comparisons, file attributes, logical operators, and the ternary operator.

conditional parameter

Any of the parameters that are set in an environment file and that are specified in order to configure the environment setting parameters and export parameters that are valid only in the physical host or in a specific logical host.

console

The terminal screen.

control statement

Same meaning as *script control statement*.

core dump

A source of maintenance information collected by a trace program and consisting of core files and dump files. When a problem occurs, the contents of memory are saved to a file that can be used to assist with troubleshooting.

coverage information

Information that provides measurements in tests of the extent of coverage. The two types of coverage information are C0, which is statement coverage information, and C1, which is branch coverage information. C0 measures the ratio (%) of commands in a job definition script that execute, while C1 measures the ratio (%) of branches in a job definition script that execute.

custom job

A predefined job for executing a task with a specific purpose in JP1/AJS. The custom job component for JP1/Advanced Shell is required in order to take advantage of JP1/AJS's custom job functionality in JP1/Advanced Shell.

debug

The process of testing a job definition script created in the development environment or of investigating errors in a script. To debug, you must launch the debugger.

debugger

A program for testing a job definition script created in the development environment and for investigating errors in a script. In the Windows environment, the debugging functions of the JP1/Advanced Shell editor are used. In the UNIX environment, the debugger is started by specifying the `-d` option in the `adshexec` command.

definition file

A file that defines the directories into which data for troubleshooting is to be collected.

development environment

An environment provided by JP1/Advanced Shell - Developer that supports development of job definition scripts for batch processing.

dialog box

A window that asks the user to enter a response.

editor

A program for creating job definition scripts efficiently by taking advantage of a variety of features provided in the development environment.

environment file

A file that contains environment information.

environment information

Information, such as environment variables and environment file parameters, that must be set before JP1/Advanced Shell starts.

environment setting parameter

Any of the parameters that are set in an environment file for the purpose of defining the JP1/Advanced Shell execution environment. These parameters are specified in the format `#-adsh_conf` *parameter-value*.

environment variable

Any of the variables that contain various system settings that can be set by the user.

execution environment

The environment provided by JP1/Advanced Shell for execution of batch operations. JP1/Advanced Shell refers to the execution environment in its narrow sense.

export parameter

A parameter that is set in the environment file and whose function is to set an environment variable when a command starts.

extended script command

A command that is executed in a job definition script. Compared to normal shell script commands, these commands offer the additional capability to control batch job execution. They are also referred to as job execution control commands. In JP1/Advanced Shell, these commands start with `#-adsh`.

extended shell command

A built-in command that is internal to the shell and executed by the shell itself. Extended shell commands can be used in job definition scripts.

extended shell variable

A shell variable with a special meaning that is provided by JP1/Advanced Shell.

external command

Any of the UNIX-compatible commands, OS-provided commands, user-created executable files, and other programs that are not shell built-in commands.

fault injection mode

A mode used during debugging to simulate the occurrence of an error.

In UNIX, you enable or disable the fault injection mode with the `joberrmode` command. In Windows, you choose the **Fault Injection Mode** menu item in the JP1/Advanced Shell editor.

file allocation

In JP1/Advanced Shell, such operations as registering postprocessing of files are referred to as *file allocation*.

file descriptor

A numeric identifier for distinguishing the different types of input and output in JP1/Advanced Shell. In JP1/Advanced Shell, the standard output is assigned `1`, the standard error output is assigned `2`, and `3` through `9` can be allocated and used for other purposes.

flow control

Functionality for controlling the event-issuance interval for JP1 events that are issued during execution of the `adshread` and `adshecho` commands.

here document

A redirection functionality used in a job definition script by which standard input is generated with the job definition script.

job controller

A program for controlling a job while the job is running. The `adshexec` command is the job controller.

job definition script file

A program file that defines a job that has been prepared as a job definition script.

job execution log

A collection of messages output by a job, including the start and end messages for the job and job steps. At the end of a job, the contents of the job execution log are sent to the standard error output by the job controller.

job ID

An identification number (sequentially generated between `000001` and `999999`) that is assigned to a job by JP1/Advanced Shell at the time the job is executed. Each job is assigned a unique identifier, so that each job can be identified individually on the basis of its job ID. Once job ID `999999` has been assigned, the next job is assigned job ID `000001`.

job information

Information associated with a job, such as the job name, job ID, and job step names.

jobnet

A set of jobs whose execution order is defined. Jobs within the jobnet execute automatically in the predefined order. The jobnet is a functionality provided by JP1/AJS.

job scheduler

A product that performs job scheduling. It is part of a suite of products in JP1/Advanced Shell used to link to JP1/AJS.

job step

A range of processing within a job defined in a job definition script that demarcates a unit of specific processing. The job step is the smallest unit for performing a specific operation (task) in JP1/Advanced Shell. A job is made up of a collection of job steps. Job steps are defined with the `#-adsh_step_start`, `#-adsh_step_error` (optional), and `#-adsh_step_end` commands.

JP1/Advanced Shell

A product used to create and execute batch jobs from job definition scripts. JP1/Advanced Shell can be divided into JP1/Advanced Shell and JP1/Advanced Shell - Developer. In this narrow sense, JP1/Advanced Shell refers to the execution environment in which batch jobs are executed from job definition scripts. Batch jobs in both Windows and UNIX can be run from the same job definition script.

JP1/Advanced Shell - Custom Job

A program for creating jobs that are custom-defined in the operation management console in JP1/Advanced Shell.

JP1/Advanced Shell - Developer

A product used for developing job definition scripts for batch jobs. This term also refers to the development environment in which job definition scripts are developed.

JP1/AJS3

Abbreviation for JP1/Automatic Job Management System 3, which is the successor product to JP1/AJS2. By linking JP1/Advanced Shell to JP1/AJS3, you can achieve distributed processing among multiple PCs.

log

Historical information that is output by the computer. Timestamps, messages, and similar items are output as logs.

long option

A type of option specified in command arguments. A long option begins with two consecutive hyphens (`--`) followed by a character string.

metacharacter

A character (or character string) that has a special meaning in a job definition script.

operand

A type of command argument specified on the command line. An operand is a default command argument that is specified in addition to option names and option values. Parameter values are also called operands.

option

In general, a pre-selected capability that is added to the instructions provided by a computer input device.

In JP1/Advanced Shell, a command argument consisting of one hyphen (`-`) followed by one character is called a short option, and a command argument consisting of two consecutive hyphens (`--`) followed by a character string is called a long option.

An argument specified immediately following an option is the option's *value*.

pipe

A functionality for linking the standard output of a previous command to the standard input of a subsequent command.

program output data file

A file to which a user program can output its execution results. JP1/Advanced Shell creates the file name automatically in order to consolidate the user program's output results with the system execution log.

quotation

Either the single quotation mark (`'`) or the double quotation mark (`"`).

redirection

Capability before a command in a job definition script is executed to change the input source for the information needed to execute the command or the output destination for the execution results. Typically, the keyboard is assigned as the standard input and the screen is assigned as the standard output, but redirection enables these assignments to be changed.

regular built-in command

Any of a set of the built-in commands among the standard shell commands. In the case of a regular built-in command, even if its command syntax is invalid, it does not exit the shell that is executing the command (see also *special built-in command*).

regular file

A file used for input or output by a job definition script. Regular files might remain after the job finishes, or regular files might be deleted during execution of the job. Regular files can be defined with the `#-adsh_file` command or the `adshfile` command.

reply-request message

A message that asks the operator to enter a reply.

reply-waiting event

A JP1 event that provides notification of a reply-request message.

reserved script command

A command that can be used as a reserved word in a job definition script. An example is the `time` command.

return code

A code that is returned to report the execution result of a job definition script or a command.

root job

A job executed from JP1/AJS or a login shell that is not a child job.

script

A text file into which is assembled a series of commands that can be executed sequentially from the shell. A script in JP1/Advanced Shell is called a *job definition script*, and they can be executed in both the Windows and UNIX environments.

script control statement

A statement for managing commands in a job definition script. Examples include the `if`, `for`, `while`, `until`, and `case` statements.

script file

A file in which a script that has been created is saved.

shell

A program that interprets instructions provided by a computer input device and passes them to the OS.

shell command

Generic name for any command used in JP1/Advanced Shell that is executed in the shell or from the command prompt.

shell operation command

A command that is provided as an executable binary file or a shell script. The two types of shell operation commands are those that can be used only in job definition scripts and those that can be used not only in job definition scripts but from OS shells and the command prompt. The shell operation commands include the `adshexec` command (executes batch jobs).

shell option

Any of the pre-selected capabilities that are added to the instructions provided by a computer input device to the shell.

shell script

A text file into which you assemble a series of commands so that you can then execute those commands sequentially from the shell. A shell script in JP1/Advanced Shell is referred to as a *job definition script*, and it can be executed in both the Windows and UNIX environments.

shell variable

An area of memory assigned as a value in a job definition script. You can reference the value of a created variable.

short option

A type of option specified in command arguments. A short option begins with a hyphen (-) followed by one character.

signal

A mechanism in UNIX by which processes report to each other the occurrence of asynchronous events. For example, a signal is sent when a job is forcibly terminated in JP1/Advanced Shell.

special built-in command

Any of a set of the built-in commands among the standard shell commands. In the case of a special built-in command, if its command syntax is invalid, it exits the shell that is executing the command (see also *regular built-in command*).

spool

The location where JP1/Advanced Shell stores the execution results of jobs and job execution logs.

spool job

The execution results for each job created in the spool directory.

standard error output (stderr)

A stream to which a program outputs its error messages and other messages.

standard input (stdin)

A stream from which a program receives its input data.

standard output (stdout)

A stream to which a program outputs its data.

standard shell command

A built-in command that is internal to the shell and executed in a process in the shell itself. Standard shell commands can be used in job definition scripts.

subshell

In a UNIX environment, a child process that has the same name as the job controller, is neither a root job nor a child job, and is created temporarily automatically when an external command or a specific syntax is executed in a job definition script.

symbolic link

A link that is implemented as a file that contains the actual file path.

system execution log

A log output by a job controller in JP1/Advanced Shell in order to facilitate integrated management of job execution status by the system administrators. Log information from multiple job controllers can be output to a single log.

temporary file

A file whose use is transient during job execution. Temporary files are created by a job or job step, and they are deleted automatically when the job terminates. Temporary files are defined with the `#-adsh_file_temp` command.

trace log

Information collected to assist in investigating and resolving problems that occur in JP1/Advanced Shell.

trap action

An action that is defined in the `trap` command's *action* argument.

UNIX-compatible command

Any of the standard UNIX commands, such as the `ls` command, that can be used in JP1/Advanced Shell. These commands can also be used in a Windows environment, which facilitates interoperability between UNIX and Windows.

variable

A location or array in memory that is used to handle values in a job definition script. Examples of variables include shell variables and environment variables.

watchpoint

A special breakpoint that stops a job definition script when the value of a variable or expression changes. A watchpoint can be managed in the same way as any other type of breakpoint.

wildcard

A character, such as the asterisk (`*`) or the question mark (`?`), that can be specified as a stand-in for any character or character string. The asterisk (`*`) represents any character string, and the question mark (`?`) represents any single character.

In addition, you can use square brackets (`[]`) to obtain a match with any of the characters in the character string enclosed in the square brackets. You can also use the hyphen (`-`) to separate values constituting a range, or the exclamation mark (`!`) for a condition to be true when none of the characters enclosed in square brackets results in a match. You can also use the comma (`,`) to assemble a comma-separated list of character strings, any one of which can be selected.

Index