HITACHI
Inspire the Next

Job Management Partner 1 Version 10

Job Management Partner 1/IT Desktop Management 2 Automatic Installation Tool Administration Guide

3021-3-374(E)

■ Relevant program products

Job Management Partner 1/IT Desktop Management 2 - Manager

P-2642-78AL Job Management Partner 1/IT Desktop Management 2 - Manager version 10-50

The above product includes the following:

P-CC2642-7AAL Job Management Partner 1/IT Desktop Management 2 - Manager version 10-50 (for Windows Server 2012, Windows Server 2008 Enterprise, Windows Server 2008 Standard, Windows Server 2008 R2, Windows Server 2003 Enterprise, Windows Server 2003 Standard, Windows Server 2003 Enterprise (x64), and Windows Server 2003 Standard (x64))

P-CC2642-7BAL Job Management Partner 1/IT Desktop Management 2 - Agent version 10-50 (for Windows 8.1, Windows 8, Windows Server 2012, Windows 7 Enterprise, Windows 7 Professional, Windows 7 Ultimate, Windows 7 Home Premium, Windows 7 Starter, Windows Server 2008 Enterprise, Windows Server 2008 Standard, Windows Server 2008 R2, Windows Vista, Windows Server 2003 Enterprise, Windows Server 2003 Standard, Windows Server 2003 Enterprise (x64), Windows Server 2003 Standard (x64), and Windows XP)

P-CC2642-7CAL Job Management Partner 1/IT Desktop Management 2 - Network Monitor version 10-50 (for Windows 8.1 Enterprise, Windows 8.1 Pro, Windows 8 Enterprise, Windows 8 Pro, Windows Server 2012, Windows 7 Enterprise, Windows 7 Professional, Windows 7 Ultimate, Windows Server 2008 Enterprise, Windows Server 2008 Standard, Windows Server 2008 R2, Windows Server 2003 Enterprise, and Windows Server 2003 Standard)

P-CC2642-7DAL Job Management Partner 1/IT Desktop Management 2 - Asset Console version 10-50 (for Windows Server 2012, Windows Server 2008 Enterprise, Windows Server 2008 Standard, Windows Server 2008 R2, Windows Server 2003 Enterprise, Windows Server 2003 Standard, Windows Server 2003 Enterprise (x64), and Windows Server 2003 Standard (x64))

■ Trademarks

Acrobat is either a registered trademark or a trademark of Adobe Systems Incorporated in the United States and/or other countries.

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

Adobe and Flash Player are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Adobe, and Reader are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

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

Itanium is a trademark of Intel Corporation in the United States and other countries.

McAfee is a trademarks or a registered trademark of McAfee, Inc. in the United States and other countries.

Microsoft and Hyper-V are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

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

Norton AntiVirus is a trademark or registered trademark of Symantec Corporation or its affiliates in the U.S. and other countries.

ODBC is Microsoft's strategic interface for accessing databases.

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

RSA and BSAFE are either registered trademarks or trademarks of EMC 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.

This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

This product includes software developed by Ben Laurie for use in the Apache-SSL HTTP server project.

This product includes software developed by Daisuke Okajima and Kohsuke Kawaguchi (http://relaxngcc.sf.net/).

This product includes software developed by IAIK of Graz University of Technology.

Portions of this software were developed at the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign.

This product includes software developed by the University of California, Berkeley and its contributors.

This software contains code derived from the RSA Data Security Inc. MD5 Message-Digest Algorithm, including various modifications by Spyglass Inc., Carnegie Mellon University, and Bell Communications Research, Inc (Bellcore).

Regular expression support is provided by the PCRE library package, which is open source software, written by Philip Hazel, and copyright by the University of Cambridge, England. The original software is available from ftp://ftp.csx.cam.ac.uk/pub/software/programming/pcre/

This product includes software developed by Ralf S. Engelschall <rse@engelschall.com> for use in the mod_ssl project (http://www.modssl.org/).

This product includes RSA BSAFE(R) Cryptographic software of EMC Corporation.

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 screen shots

Microsoft product screen shots reprinted with permission from Microsoft Corporation.

■ 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

April 2015: 3021-3-374(E)

■ Copyright

All Rights Reserved. Copyright (C) 2015, Hitachi, Ltd.

Copyright, patent, trademark, and other intellectual property rights related to the "TMEng.dll" file are owned exclusively by Trend Micro Incorporated.

Preface

This manual describes in detail how to use Automatic Installation Tool, which is a component of Job Management Partner 1/IT Desktop Management 2 - Manager (abbreviated hereafter to *JP1/IT Desktop Management 2*). Read this manual when creating AIT (Automatic Installation Tool) files to be used to automatically install software.

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

Intended readers

This manual is intended for administrators who wish to distribute software by using JP1/IT Desktop Management 2.

Organization of this manual

This manual is organized into the following chapters and appendixes:

1. Remote Installation Using an AIT File

Chapter 1 describes the AIT file and how to perform remote installation by using the AIT file. The chapter also gives some cautionary notes on creating and using AIT files.

2. Creating an AIT File

Chapter 2 describes how to use the Automatic Installation Tool to create an AIT file.

3. AIT Language Reference

Chapter 3 describes the format of AIT files and the syntax of the AIT language.

4. API Function Reference

Chapter 4 describes the API functions that can be used in the AIT language.

5. Troubleshooting

Chapter 5 describes the actions to be taken when a problem occurs in the Automatic Installation Tool. This chapter also describes messages that might be output by the Automatic Installation Tool.

A. Menus

Appendix A lists the menus available in the Automatic Installation Tool window.

B. Editing a Program Product ID File

Appendix B describes how to edit the program product ID file that is generated when an AIT file is created.

C. Reference Material for This Manual

Appendix C provides reference information, including various conventions, for this manual.

D. Glossary

Appendix D defines terms used in this manual.

Contents

Preface i

Intended readers i
Organization of this manual i

1. Remote Installation Using an AIT File 1

1.1 What is the AIT file? 2
1.2 Procedure for remote installation using an AIT file 4
1.2.1 Creating an AIT file and a program product ID file 4
1.2.2 Storing the created files 4
1.2.3 Packaging 4
1.2.4 Executing remote installation 6
1.3 Notes on creating and using AIT files 8

2. Creating an AIT File 9

2.1 Overview of the Automatic Installation Tool 10
2.1.1 Functionalities of the Automatic Installation Tool 10
2.1.2 Starting and terminating the Automatic Installation Tool 10
2.2 Structure of an AIT file, and procedure for creating it 12
2.3 Checking the sequence and properties of installer windows 15
2.3.1 Items you should check 15
2.3.2 Acquiring properties of installer windows 17
2.4 Recording installation operations 20
2.4.1 Procedure for recording your installation operations 23
2.4.2 Pausing and resuming recording 26
2.4.3 Recording installation operations that request the OS to be restarted 26
2.5 Generating the PACKAGE_INFO section 28
2.5.1 Procedure for generating the PACKAGE_INFO section and a program product ID file 28
2.5.2 Specifying the installer and the files for identifying the software to be distributed 31
2.6 Editing an AIT file 34
2.6.1 Window processing 34
2.6.2 Automatically generated flags 40
2.6.3 Checking and modifying an automatically generated AIT file 42
2.6.4 Linkage with the Packager and Remote Installation Manager 46
2.6.5 Adding the coding for error handling and setting a return code 48
2.6.6 Example of a completed AIT file 51
2.7 Debugging an AIT file 58

2.7.1 Syntax check and execution 59
2.7.2 Debugging 60

3. AIT Language Reference 69

3.1 Format of AIT files 70
3.2 Sections 71
3.2.1 PACKAGE_INFO 71
3.2.2 DEFINE 72
3.2.3 MAIN 73
3.2.4 ERROR 74
3.3 Data types 75
3.3.1 integer 75
3.3.2 float 75
3.3.3 bool 76
3.3.4 string 77
3.4 Operators 80
3.4.1 Assignment 80
3.4.2 Unary plus 82
3.4.3 Unary minus 83
3.4.4 Unary not 83
3.4.5 Adding operators 84
3.4.6 Multiplying operators 84
3.4.7 Comparison operators 86
3.4.8 Bitwise operators 87
3.4.9 Logical operators 88
3.4.10 Priority of operators 90
3.5 Variables and constants 92
3.5.1 Example of variables 92
3.5.2 Example of constants 92
3.6 Program flow control 93
3.6.1 goto 93
3.6.2 Label 94
3.6.3 if-else-endif 94
3.6.4 while-loop 95
3.6.5 do-while 96
3.6.6 for-next 97
3.6.7 continue 99
3.6.8 break 100
3.6.9 switch-endswitch 100
3.7 Function calls 103
3.7.1 Format 103
3.7.2 Example of coding 104
3.8 Keywords 105
3.9 Macros 107

3.9.1 Macros for window and check operations 107
3.9.2 Macros for message operations 109
3.9.3 Macros for file operations 109
3.9.4 Macros for IME operations 109
3.9.5 Macros for utility operations 110
3.9.6 Macros for registry operations 110
3.9.7 Macros for directory operations 111
3.9.8 Macros for error logging 111

4. API Function Reference 113

4.1 API functions 114
4.1.1 Window operations 114
4.1.2 Check operations 116
4.1.3 Resolution check 117
4.1.4 Date/time operations 117
4.1.5 IME operations 117
4.1.6 Character string operations 118
4.1.7 Message operations 119
4.1.8 Registry operations 119
4.1.9 Redirect operations 119
4.1.10 Directory operations 120
4.1.11 File operations 120
4.1.12 INI file operations 121
4.1.13 Recorder operations 121
4.1.14 Taskbar operations 121
4.1.15 Utility operations 122
4.1.16 Interfacing with JP1/IT Desktop Management 2 122
4.2 Details about the API functions 123
4.3 Examples of using API functions 303
4.3.1 Deleting carriage return and linefeed characters 303
4.3.2 Extracting characters 303
4.3.3 Manipulating the HKEY_CURRENT_USER registry key during remote installation 304

5. Troubleshooting 307

5.1 Checking messages 308
5.1.1 Message output destination 308
5.1.2 Format of message explanations 308
5.2 Messages that may be displayed during editing 311
5.3 Messages that may be displayed during execution and parsing of AIT files 324

Appendixes 365

A. Menus 366
B. Editing a Program Product ID File 369

C. Reference Material for This Manual 371
C.1 Related publications 371
C.2 Conventions: Abbreviations for product names 371
C.3 Conventions: Acronyms 374
C.4 Conventions: Fonts and symbols 375
C.5 Conventions: Version numbers 376
C.6 About online help 377
C.7 Conventions: KB, MB, GB, and TB 377
D. Glossary 378
Index 381

Chapter

1. Remote Installation Using an AIT File

When JP1/IT Desktop Management 2 remotely installs software onto computers, user response to the installer can be automated by using an AIT file. This chapter describes the AIT file and how to perform remote installation by using the AIT file. The chapter also gives some cautionary notes on creating and using AIT files.

1.1 What is the AIT file?
1.2 Procedure for remote installation using an AIT file
1.3 Notes on creating and using AIT files

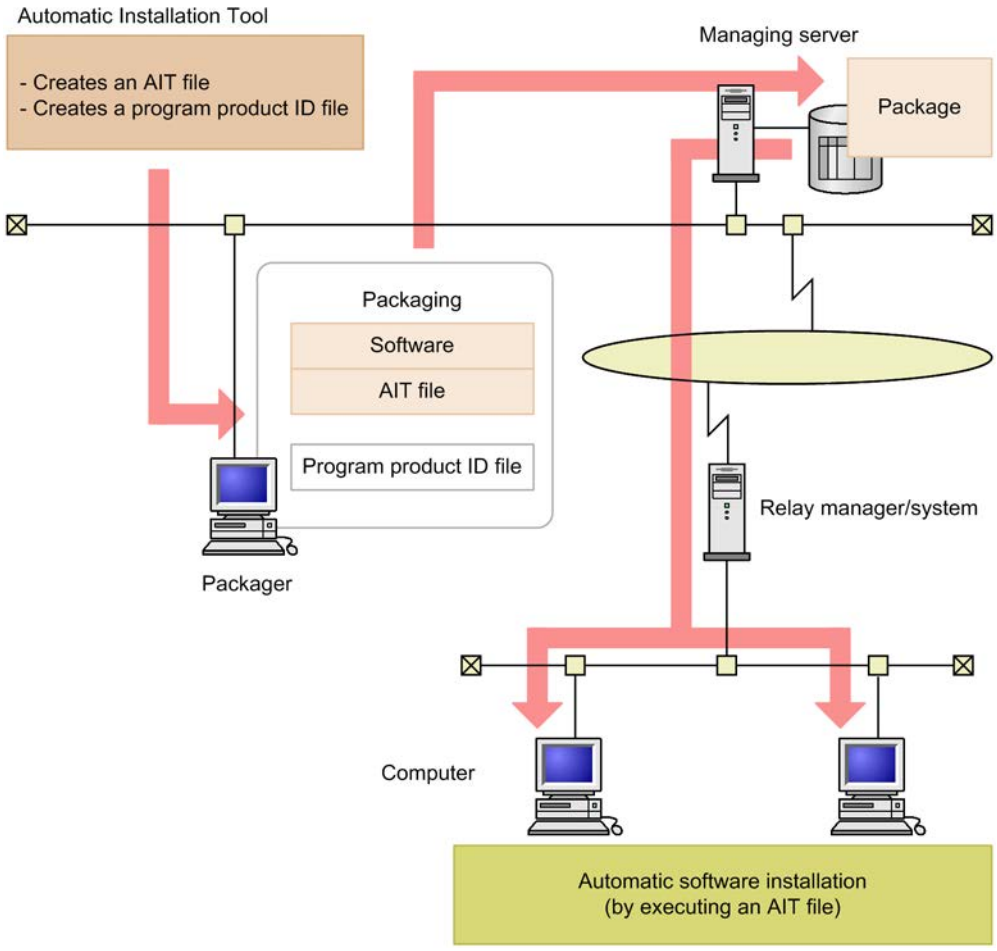
1.1 What is the AIT file?

An AIT file contains a script that automatically responds to a software installer. If you package an AIT file together with the software you want to distribute, and then perform remote installation of the package, the software will automatically be installed on the destination computers. Users at the destination computers do not need to respond to the installer.

An AIT file is needed when you want to remotely install third party's software or a user program that requests the users at the clients to respond to the installer. You can create an AIT file by using the Automatic Installation Tool, which is a component of JP1/IT Desktop Management 2.

The following figure shows an overview of the remote installation by using an AIT file.

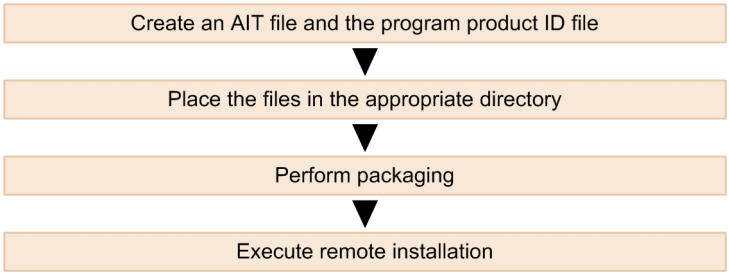
Figure 1-1: Remote installation using an AIT file



1.2 Procedure for remote installation using an AIT file

The following figure shows the procedure for using an AIT file for remote installation.

Figure 1-2: Procedure for remote installation using an AIT file



1.2.1 Creating an AIT file and a program product ID file

To create the AIT file for the software you want to distribute, you need to check the procedure for installing the software. Based on the procedure you checked, create the AIT file that has the system automatically responding to the installer. AIT files are created in the AIT language specific to the Automatic Installation Tool.

When you create an AIT file, you must also create a *program product ID file*, which contains information that associates the AIT file with the software to be distributed. The program product ID file must be named `PPDEFAIT.DMP` and must be stored in the predefined location.

For details about how to create an AIT file and a program product ID file, see *2. Creating an AIT File* and subsequent chapters.

1.2.2 Storing the created files

Place the program product ID file in the following directory at the Packager PC:

Packager-installation-directory`\DMPRM\PPDEFAIT.DMP`

If using a user-created AIT file, place it in the directory at the Packager PC specified when setting the program product ID file.

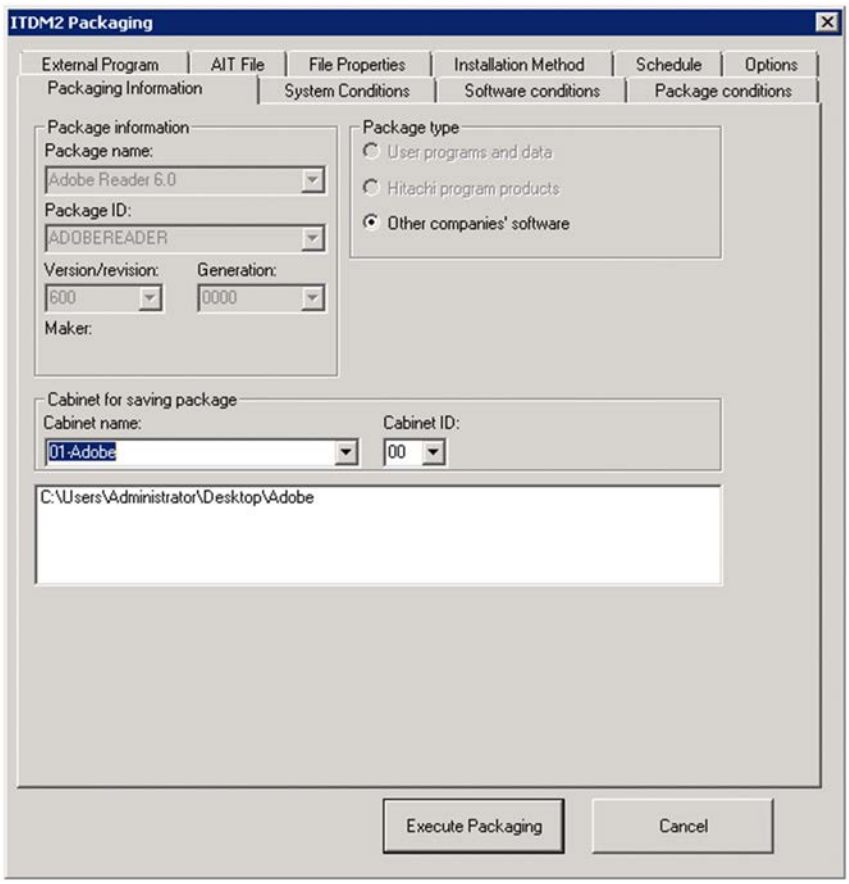
1.2.3 Packaging

After placing the AIT file and program product ID file in the appropriate locations, use the Packager to package the software that you want to distribute.

During packaging, the values defined in the AIT file and program product ID file are displayed for **Package ID**, **Version**, and **Product** on the **Packaging Information** page in the JP1/ITDM2 Packaging dialog box. You cannot change the values of these items

in this dialog box.

Figure 1-3: Packaging Information page



The JP1/ITDM2 Packaging dialog box also displays other information defined in the AIT file as shown below. You can change the values of this information during packaging or remote installation.

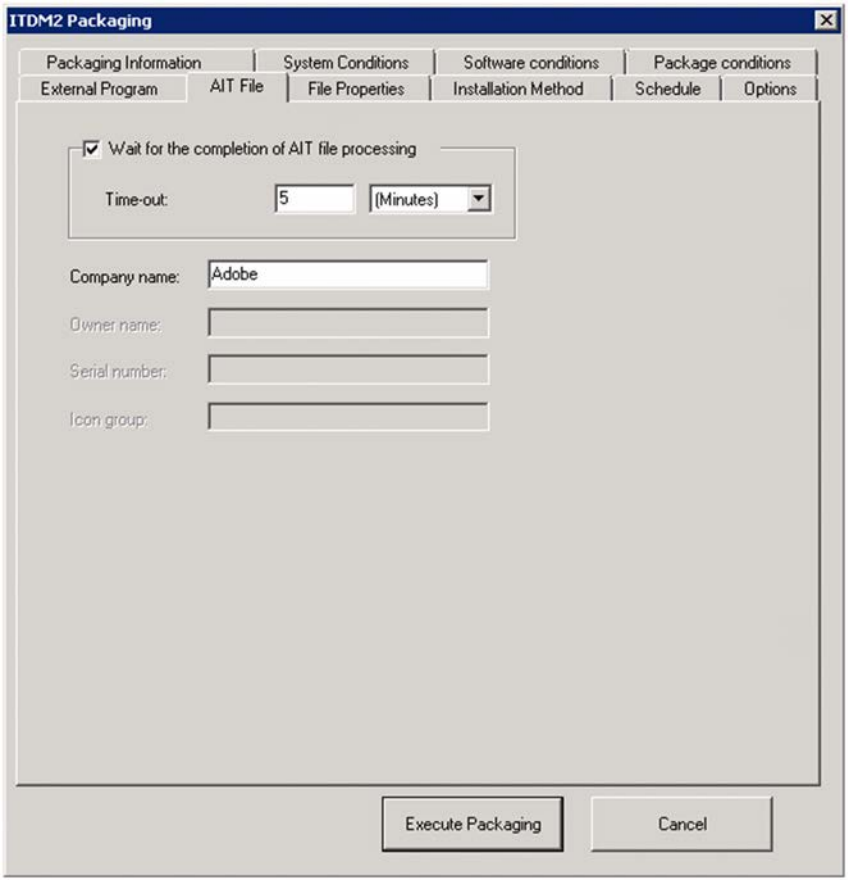
- Installation directory
- Organization
- User name
- Serial number
- Icon group

If the user at a target computer responds to the dialog box, or uses the keyboard or mouse during installation, the information displayed in the installer window might

differ from the information expected by the AIT file. As a result, remote installation based on the AIT file might stop. You can set a time-out to cancel the remote installation forcibly if a length of time elapses after the installer begins to wait for a user response.

Typically, you should set the time period about three times longer than that required to install software to be distributed.

Figure 1-4: AIT File Settings page



For details about packaging, see the *Job Management Partner 1 Version 10 Job Management Partner 1/IT Desktop Management 2 Distribution Function Administration Guide*.

1.2.4 Executing remote installation

Use the Remote Installation Manager to create and execute a job to distribute software. For details about how to use the Remote Installation Manager, see the *Job*

Management Partner 1 Version 10 Job Management Partner 1/IT Desktop Management 2 Distribution Function Administration Guide.

1.3 Notes on creating and using AIT files

This section gives notes on creating and using AIT files.

- AIT files do not support Web pages and software created in Java or ActiveX.
- If API functions that manipulate the registry key `HKEY_CURRENT_USER` are defined in an AIT file, the user of each computer on which remote installation is to be performed must have logged on as a member of the Administrators group.

  Such a user only can access `HKEY_CURRENT_USER`. If the client user is not a member of the Administrators group, the target registry key of the API functions is automatically changed to `HKEY_USERS\.DEFAULT`. This change may cause installation to fail.

  However, if the change of the target registry key does not affect any operations of the software being installed, the installation will continue.
- You might want to use an AIT file to remotely install a package that requires restarting the target computer after installation. In this case, use an AIT file that is set to terminate the installer without restarting the target computer. For remote installation to end successfully, the target computer must be restarted by the package settings.
- Be careful when executing an API function for 64-bit-related data (registry, folder, or file) in any of the following operation systems: the 64-bit edition of Windows 8.1, Windows 8, Windows Server 2012, the 64-bit edition of Windows 7, the 64-bit edition of Windows Server 2008, the 64-bit edition of Windows Vista, and Windows Server 2003 (x64). If the target data is specified as an argument in an API function that belongs to one of the following categories, the target data might be changed (redirected) to the corresponding 32-bit-related data:
  - Registry operations
  - Directory operations
  - File operations
  - INI file operations

Chapter

2. Creating an AIT File

The Automatic Installation Tool provides an integrated environment for creating an AIT file. This chapter describes how to use the Automatic Installation Tool to create an AIT file.

2.1 Overview of the Automatic Installation Tool
2.2 Structure of an AIT file, and procedure for creating it
2.3 Checking the sequence and properties of installer windows
2.4 Recording installation operations
2.5 Generating the PACKAGE_INFO section
2.6 Editing an AIT file
2.7 Debugging an AIT file

2.1 Overview of the Automatic Installation Tool

This section gives an overview of the functionalities of the Automatic Installation Tool, and explains how to start and terminate it.

2.1.1 Functionalities of the Automatic Installation Tool

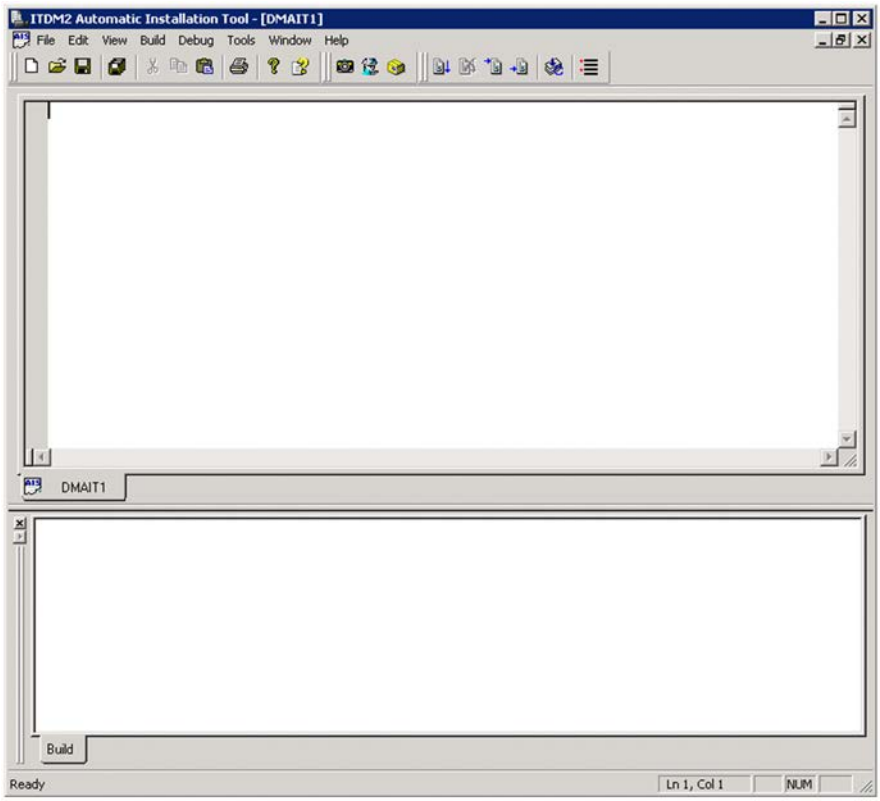
The Automatic Installation Tool provides the following functionalities for creating AIT files:

- The Edit window for creating and editing an AIT file. This window provides cut, copy, paste, indentation, and other text editing functionalities.
- The **Window Properties tool** for acquiring the properties of installer windows. You can copy the acquired properties on the Clipboard, and can use them as arguments for the API functions used in the AIT file.
- The **Recorder** functionality for recording the installation operations you actually performed. This functionality lets you automatically generate the AIT file that simulates your operations on the installer.
- The **Syntax Check** functionality for checking whether the AIT file meets the AIT language specifications. If this functionality detects errors, it displays them on a window.
- The **Execute** functionality for replaying the script in the AIT file after conducting a syntax check.
- The **Debug** functionality for helping detect and correct syntax errors. In the debug mode, you can stop executing the script in the AIT file at each breakpoint you set or at each statement. Moreover, you can check the values set in variables and update them during execution of the AIT file.
- The API (Application Programming Interface) specific to the Automatic Installation Tool. This API allows you to perform various operations such as replay of the installation script, and operations on the registry, files, and text.
- Functionality for creating a program product ID file, which associates an AIT file with the software to be distributed.

2.1.2 Starting and terminating the Automatic Installation Tool

To start the Automatic Installation Tool, from the **Start** menu, select **JP1_IT Desktop Management 2 - Agent**, **Administrator Tool**, and then **Automatic Installation Tool**. The following JP1/ITDM2 Automatic Installation Tool window appears.

Figure 2-1: JP1/ITDM2 Automatic Installation Tool window



To terminate the Automatic Installation Tool, from the **File** menu, choose **Exit**.

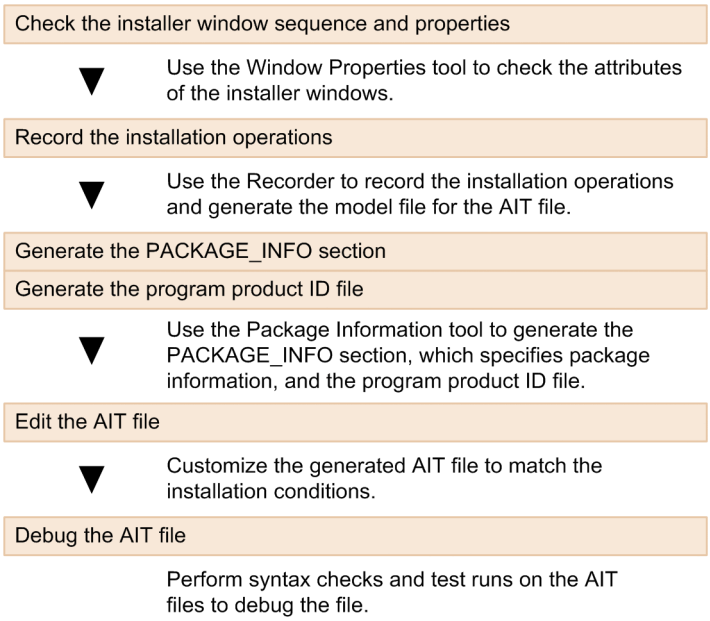
Note that, to perform recording in certain versions of JP1/IT Desktop Management 2 - Agent, you must open the JP1/ITDM2 Automatic Installation Tool window as a user who has the necessary permissions to execute the program. This restriction applies to JP1/IT Desktop Management 2 - Agent for any of the following OSs: Windows 8.1, Windows 8, Windows Server 2012, Windows 7, Windows Server 2008, and Windows Vista.

Furthermore, for these versions of JP1/IT Desktop Management 2 - Agent, you can open multiple JP1/ITDM2 Automatic Installation Tool windows at the same time. However, when recording or debugging, use a single window.

2.2 Structure of an AIT file, and procedure for creating it

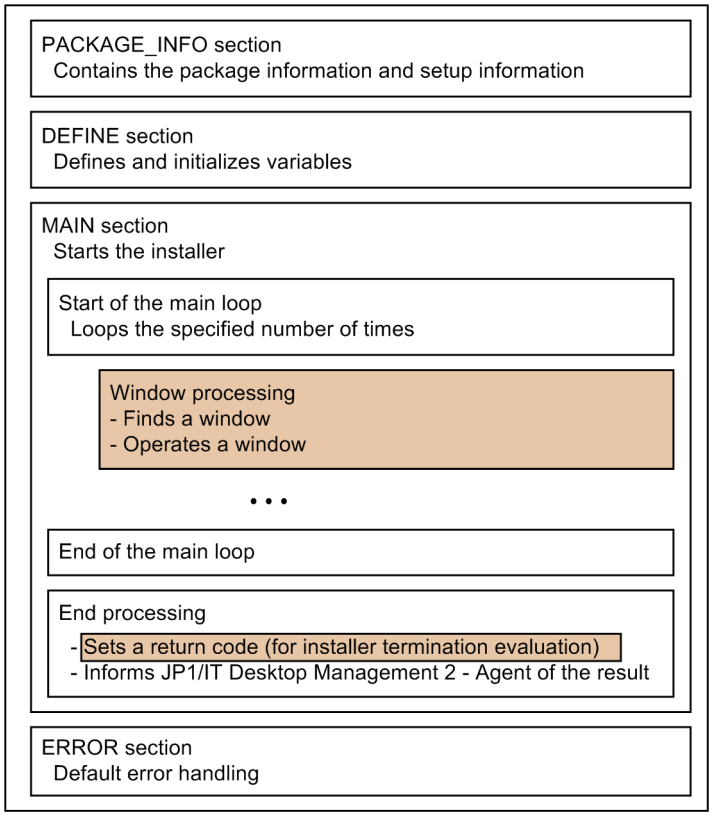
Normally, you use the following procedure to create an AIT file. In most cases, you will need to repeat this procedure several times to complete the desired AIT file.

Figure 2-2: Procedure for creating an AIT file



After you use the Recorder to record installation operations, if you use the Package Information tool to generate the PACKAGE_INFO section, an AIT file with the structure shown in the following figure is automatically generated. You can use this file as a model file to create a new AIT file. Normally, you modify the shaded portions in this file to complete the AIT file.

Figure 2-3: Structure of an AIT file



As shown in Figure 2-3, the AIT file consists of four sections. All the sections are required. You cannot change the order of the sections. The following gives an overview of these sections.

- PACKAGE_INFO section

  This section contains the package information and setup information for the software to be distributed. You can manually complete this section, but you can also use the Package Information tool that automatically generates this section.

- DEFINE section

  This section defines and initializes the variables that may be used in the MAIN and ERROR sections. No variables can be defined in sections other than the DEFINE section. If you want to add variables in the MAIN and ERROR sections or want to change initial values of variables in the MAIN and ERROR sections, you have to modify this section.

- MAIN section

This section contains the operations on the windows output by the installer. You must manually modify this section automatically generated by the Recorder to code the operations on all the installer windows. You can also set return codes for the results of installation.

- ERROR section

  If an internal error occurs during execution of the AIT file, control over the execution moves to this section. If you want to change the behavior when an error has occurred, modify this section.

You can write comments in the AIT file. The AIT language is not case sensitive. For details about AIT files, see *3. AIT Language Reference*.

Although you can manually create a program product ID file, you can also use the Package Information tool to conveniently create the program product ID file. This is possible because the tool can generate the file when the PACKAGE_INFO section is generated.

The program product ID file generated by the Package Information tool is assigned the name `PPDEFAIT.DMP` and stored in *JP1/IT-Desktop-Management-2-installation-folder*`\DMPRM`.

2.3 Checking the sequence and properties of installer windows

Start the installer of the software you want to distribute to check the installation procedure. You have to check the process of installation in each OS for the following items, and record the installation procedures on paper:

- Sequence and properties of installer windows
- Properties of dialog boxes

To view those attributes, you can conveniently use the *Window Properties tool* of the Automatic Installation Tool. The Window Properties tool allows you to obtain the following GUI properties of windows and controls:

Window property	Description
Window text	Caption of the window or control
Class name	Class name of the window or control
Module name	Application that opened the window
Control ID	ID of the window or control
Control type	Type of the control such as *window* and *button*
Associated label	The associated label of a control is the text that immediately precedes that control in the tab order. If no such text exists, the control does not have an associated label.
Enabled	Identifies whether the window or control is enabled.
Visible	Identifies whether the window or control is visible.

You can pass these properties to the API functions used in the AIT file in order to identify windows and controls.

2.3.1 Items you should check

Check the operations that the installer requests you to perform, including the sequence of them. Manually install the software to identify the installation procedure, and create a list of the items shown below.

Note that the installation operations may differ depending on the installation method and the status of the target PC (such as the OS type, free hard disk space, available memory, and installed software). Conduct checks carefully.

There are some programs that require the user to restart the operating system during installation. However, the AIT file does not support operations after a system restart. Therefore, you only have to check the installation operations before the operating

system is restarted.

- Window text

  Write down the *window text* you checked using the Window Properties tool.

- Class name

  Write down the *class name* you checked using the Window Properties tool.

- Control ID

  Write down the *control ID* you checked using the Window Properties tool.

- Control type

  Write down the *control type* you checked using the Window Properties tool.

- Operations

  Write down the operations (including clicking the **OK** button) on the dialog box.

- Note

  Write down a comment or note.

The following is an example of the results of checking the procedure for installing Adobe Reader 6.0. N/A in the *Control ID* column indicates that no control ID is used.

Table 2-1: Results of checking the Adobe Reader 6.0 installation procedure

#	Window text	Class name	Control ID	Control type	Operations	Note
1	Adobe Reader 6.0 - Setup	`MsiDialogNoCloseClass`	N/A	Window	• The **Next** button is clicked.	When the **Next** button exists.
2	Adobe Reader 6.0 - Setup	`MsiDialogNoCloseClass`	N/A	Window	• The **Install** button is clicked.	When the **Install** button exists.

#	Window text	Class name	Control ID	Control type	Operations	Note
3	Adobe Reader 6.0 - Setup	`MsiDialogNoCloseClass`	N/A	Window	• When the window appears for the first time, the **Browse** button is clicked. • When the window appears next, the **Next** button is clicked.	When the **Change Destination Folder** button exists.
4	Adobe Reader 6.0 Installer Information	`MsiDialogNoCloseClass`	N/A	Window	• The **No** button is clicked on the dialog box for restart confirmation.	-

Legend:

-: Not applicable

2.3.2 Acquiring properties of installer windows

The following explains how to use the Window Properties tool to acquire the properties of the installer windows.

(1) Acquiring the properties of a window or control

1. From the **Tools** menu, choose **Window Properties**.

   The Window Properties dialog box appears.

2. Activate the software for which you want to acquire the properties.

   Make sure that both the window you want to check and the Window Properties dialog box are displayed on the desktop.

3. Drag and drop the Finder icon () onto the window or control you want to check.

   The Window Properties dialog box displays the properties of the window or control.

Figure 2-4: Window Properties dialog box

You can copy the properties displayed in the Window Properties tool to the Clipboard.

To copy them, click  in the Window Properties dialog box. The attributes are copied as follows:

```
Window text: Adobe Reader 6.0 - Setup
Class name: MsiDialogNoCloseClass
Module name: _MSIEXEC.EXE
Control ID: N/A
Control type: Window
Associated label: N/A
Enabled: Yes
Visible: Yes
```

(2) Displaying properties of associated windows and controls

You can use toolbar buttons on the Window Properties dialog box to display properties of the parent window and its first child window, and of the foreground and background windows. You can also change the display mode of the Window Properties dialog box. The following gives the toolbar buttons on the Window Properties dialog box.

Toolbar button	Description
	**Jump to Parent Window**: Displays the properties of the parent window of the window or control for which you acquired the properties.
	**Jump to First Child Window**: Displays the properties of the first child window of the window or control for which you acquired the properties.
	**Jump to Preceding Window**: Displays the properties of the previous window of the window or control for which you acquired the properties.

Toolbar button	Description
	**Jump to Next Window**: Displays the properties of the window following the window or control for which you acquired the properties.
	**Show/Hide during Find**: Hides the Windows Properties dialog box while the Finder icon is being dragged and then displays the dialog box when the icon is dropped. If the dialog box is hidden when you drop the Finder icon, you can select the target control easier.
	**Display Always in Foreground**: Always displays the Window Properties dialog box in the foreground.

2.4 Recording installation operations

Use the *Recorder* of the Automatic Installation Tool to record installation operations you actually perform. The Recorder records the events issued when you performed installation operations such as pressing keys, clicking mouse buttons, and operations on controls. The Recorder then automatically creates an initial AIT file for simulating user operations. By modifying the initial AIT file, you can create the desired AIT file.

The Recorder automatically generates all the sections except PACKAGE_INFO. The following figure shows an example of an AIT file that is generated when the installation operations for Adobe Reader 6.0 are recorded.

Figure 2-5: AIT file generated automatically through recording installation operations for Adobe Reader 6.0 (1/3)

```
DEFINE
{
    integer iLoopCount = 0;
    integer iLoopMax = 60;
    integer DM_RTN;
    integer WINH;
    integer iCapsLockState;
    integer iNumLockState;
    integer iScrollLockState;
    integer iInsertLockState;
    integer AITIGNORE = 0;
    integer AITFLAG1=1;
    integer AITFLAG2=1;
    integer AITEVENTFLAG1=1;
    bool bRtn;
    const integer OK_END = 0;
    const integer NG_END = -1;
    float SLEEP_TIME = 1.0;
    float SLEEP_TIME_RESTART = 10.0;
    float SLEEP_TIME_EVENTS = 0.5;
}
MAIN
{
    AIT_SetDefaultWaitTimeout(1.0);
    AIT_DMPSTRC();
    DM_RTN = NG_END;
    iCapsLockState = AIT_GetKeyState(CAPSLOCK);
    iNumLockState = AIT_GetKeyState(NUMLOCK);
    iScrollLockState = AIT_GetKeyState(SCROLLLOCK);
    iInsertLockState = AIT_GetKeyState(INSERTLOCK);
    bRtn= AIT_Exec(InstallerName,SW_SHOWNORMAL);
    if(bRtn == false)
        iLoopCount = iLoopMax;
    Endif;
```

Starts the installer.

Figure 2-6: AIT file generated automatically through recording installation operations for Adobe Reader 6.0 (2/3)

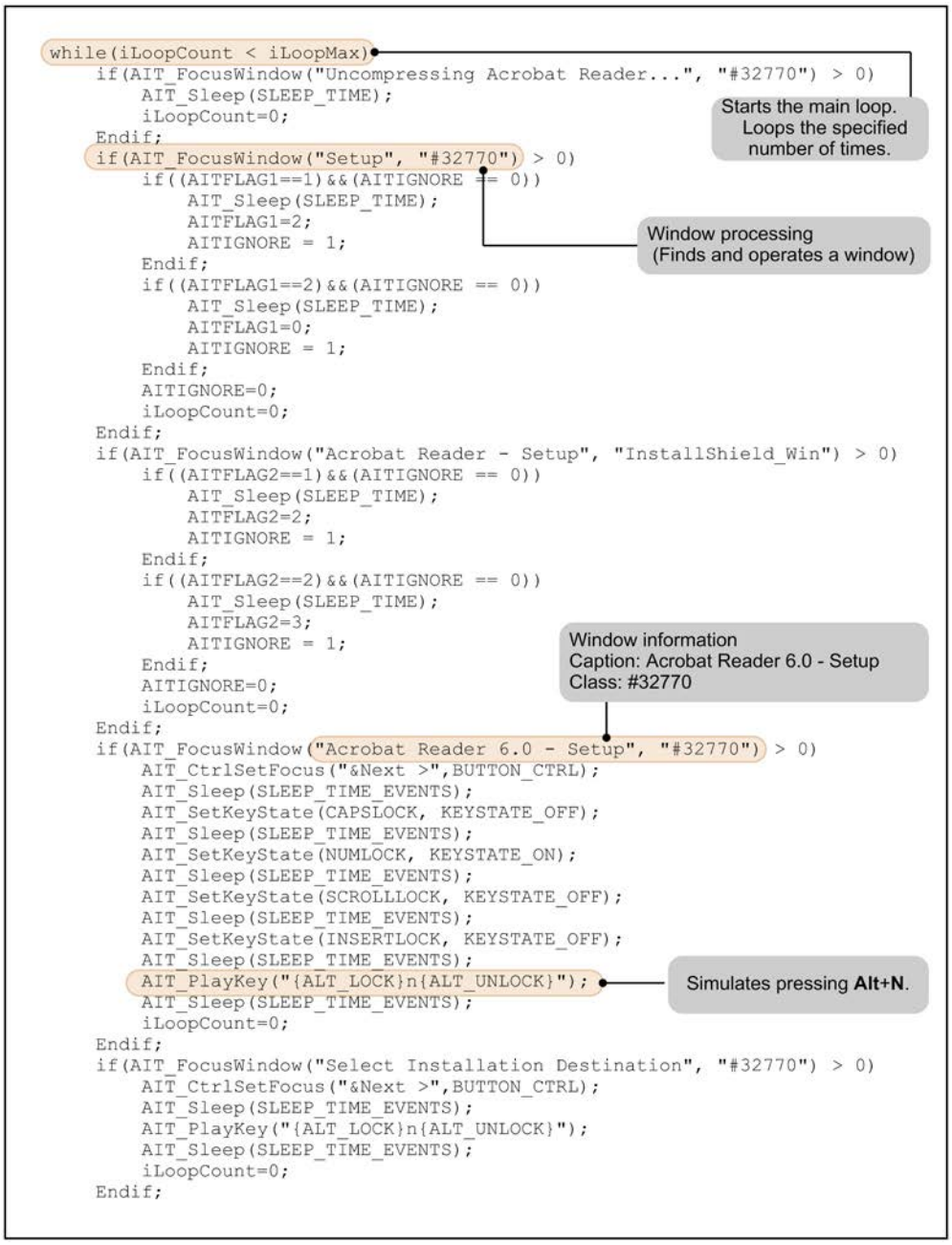
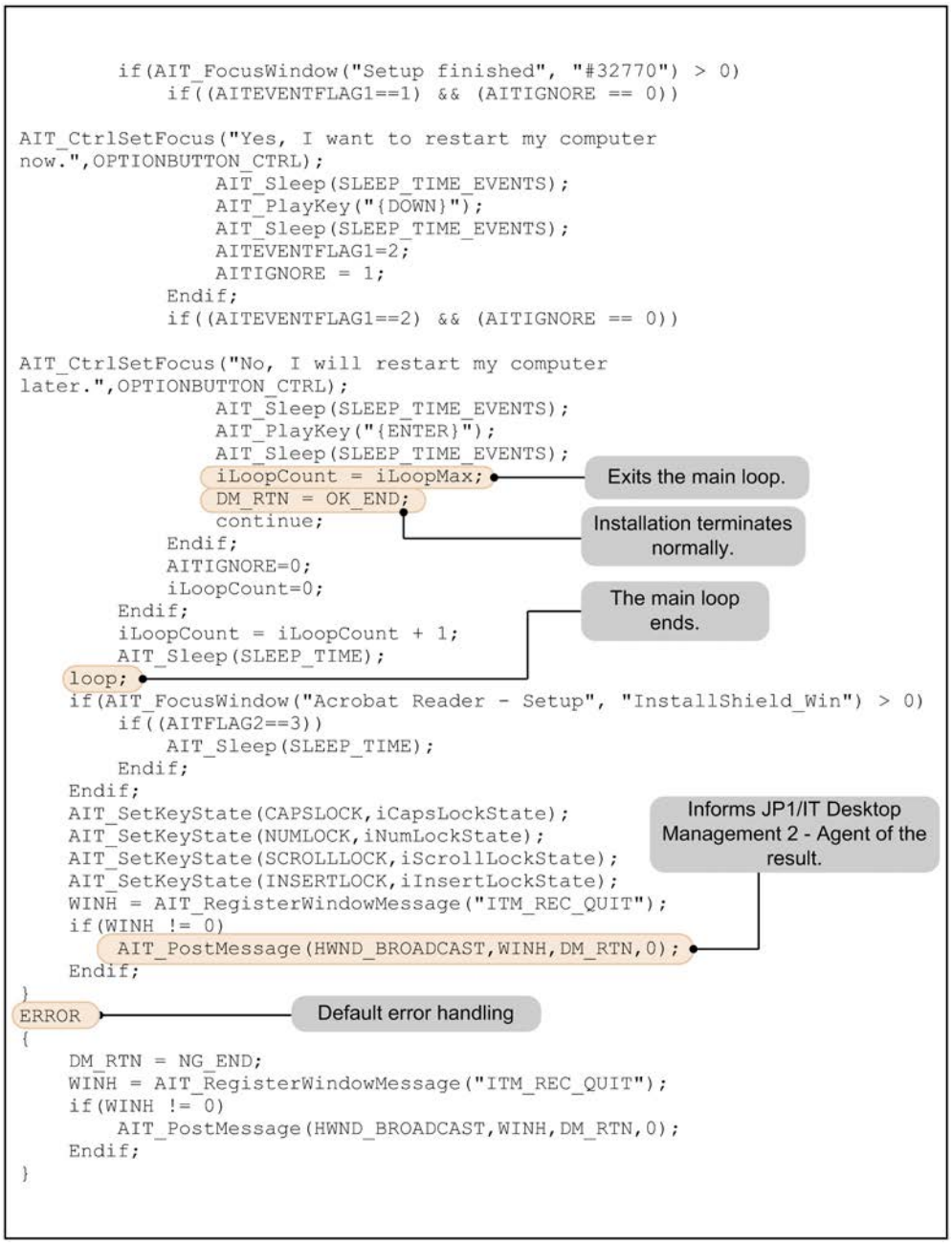


Figure 2-7: AIT file generated automatically through recording installation operations for Adobe Reader 6.0 (3/3)



2.4.1 Procedure for recording your installation operations

This subsection explains how to use the Recorder to record the installation operations you actually perform. This recording automatically generates an initial AIT file that you will modify to create the desired AIT file that simulates the user operations.

When you use the Recorder to record installation operations, you can add the `AIT_LogMesssage` statement in the AIT file. This statement enables the automatic logging functionality that logs messages when recorded operations (events) are replayed. The logged messages will be helpful when you test the AIT file you created. You can test the AIT file by choosing **Execute** from the **Build** menu.

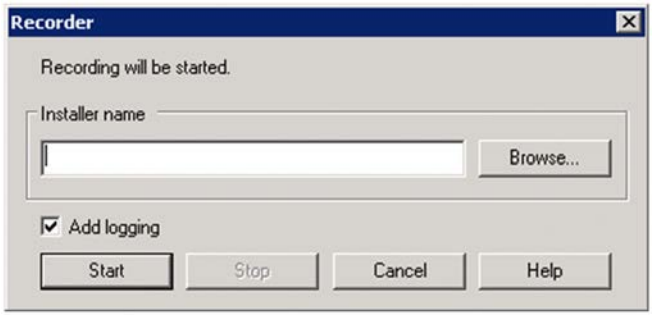
While you are recording your installation operations, do not use the mouse whenever possible. Since mouse operations depend on screen coordinates, the simulation may be unable to securely respond to the installer. The Recorder may fail to record operations using the mouse wheel. You should record events by keying-in instead of mouse operations.

Before recording, terminate all the applications other than the Automatic Installation Tool and the installer. Even if you do not carry out operations for other applications, the Recorder records all the displayed windows.

1. From the **Tools** menu, choose **Recorder**.

   The Recorder dialog box appears.

   Figure 2-8: Recorder dialog box (start recording)



2. Select the **Add logging** check box. When the installer is not activated yet, specify the executable file of the installer in **Installer name**.

   Installer name

   In the **Installer name** text box, you can specify the executable file of the installer of the software for which you want to record the installation operations. When you leave this text box blank, you must activate the installer in advance.

   Add logging

When this check box is selected, the `AIT_LogMesssage` statement is added to the AIT file. When this statement is added, the Automatic Installation Tool logs errors and informational messages while simulating the user operations based on the AIT file.

When this check box is not selected, the `AIT_LogMesssage` statement is not added to the generated AIT file.

3. Click the **Start** button.

   The subsequent user operations will be recorded.

   When you have specified the installer in **Installer name** in Step 2, the specified installer is activated.

4. Carry out actual software installation operations.

   The system creates a recording sequence for simulating user operations.

   While the JP1/ITDM2 Automatic Installation Tool is recording your operations, its icon is displayed in the Windows taskbar.

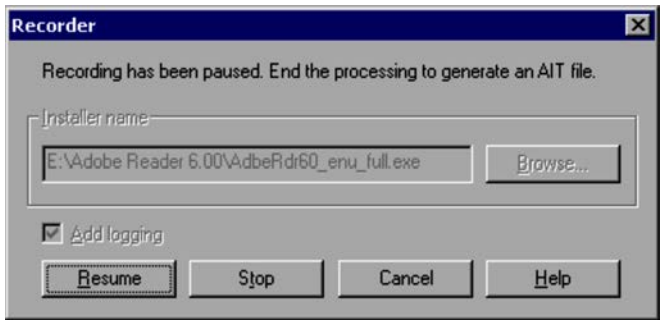
5. When you have finished performing the operations for software installation, click the **JP1/ITDM2 Automatic Installation Tool** icon on the Windows taskbar.

   In general, always use the left mouse button to stop the recording.

   In Windows 8.1, Windows 8, Windows Server 2012, Windows 7, or Windows Server 2008 R2, right-clicking the icon displays the Windows Jump List. To prevent the display of the Windows Jump List from being recorded as an installation operation, always use the left mouse button to click the Automatic Installation Tool icon.

   The following Recorder dialog box appears. At this point of time, the recording is paused.

   Figure 2-9: Recorder dialog box (pause recording)



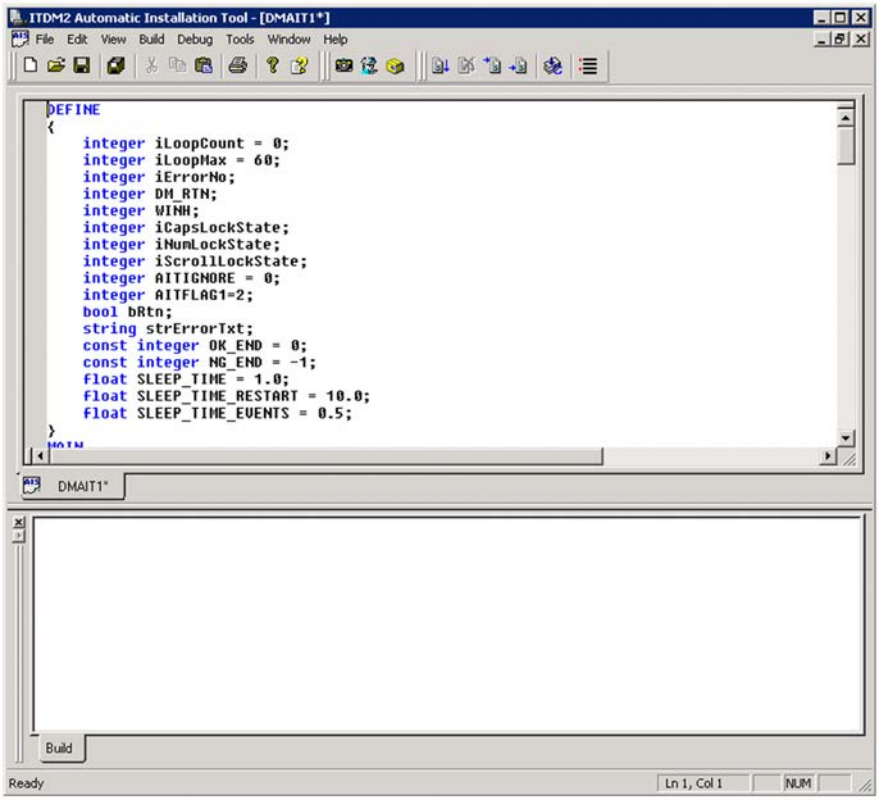
6. Click the **Stop** button.

The recording ends. A message box appears to confirm whether to update the package information.

If you click **Yes** in this message box, the Package Information dialog box appears. You can then generate the `PACKAGE_INFO` section. For details about the Package Information dialog box, see *2.5 Generating the PACKAGE_INFO section*.

If you click **No**, the JP1/ITDM2 Automatic Installation Tool window opens, displaying the contents of the generated AIT file.

Figure 2-10: JP1/ITDM2 Automatic Installation Tool window displaying the contents of an AIT file



7. From the **File** menu, choose **Save As** to save the automatically generated AIT file with any name you like.

   The AIT file is saved with the extension `.ais`. This is an initial AIT file that you will modify as required to create the desired AIT file.

If a restart event occurs during recording, operations carried out by the user during the recording will be held by the Automatic Installation Tool. When you activate the

Automatic Installation Tool after the PC is restarted, the Automatic Installation Tool displays a message box to confirm whether to generate the AIT file. Click **Yes** to generate the AIT file.

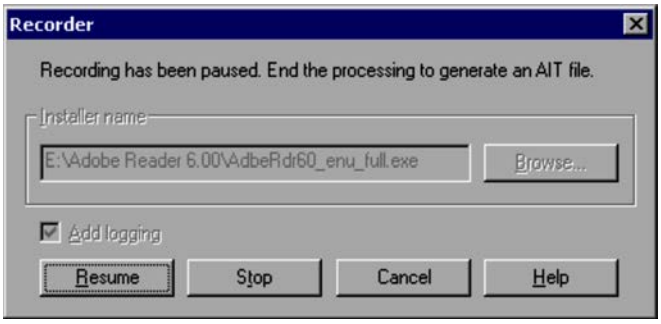
2.4.2 Pausing and resuming recording

You can pause and resume recording of installation operations.

1. From the Windows taskbar, select the JP1/ITDM2 Automatic Installation Tool icon.

   You will see the displayed Recorder dialog box, with recording paused.

   Figure 2-11: Recorder dialog box (pause recording)



2. With recording paused, click the **Resume** button.

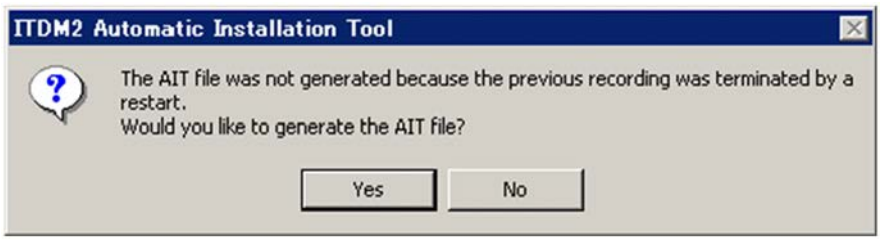
   The recording will be resumed. Carry on installation operations.

2.4.3 Recording installation operations that request the OS to be restarted

The recording of a sequence of installation operations that request the OS to be restarted finishes when the OS is restarted.

To display the generated AIT file, activate the Automatic Installation Tool after the OS is restarted. When the Automatic Installation Tool starts, it displays the message dialog box for selecting whether to enable the recording that was performed before the OS is restarted.

Figure 2-12: Message dialog box for selecting whether to enable or disable the recording



When you click **Yes**, the Automatic Installation Tool displays the AIT file that contains the operations recorded before the OS was restarted.

2.5 Generating the PACKAGE_INFO section

Any AIT file requires the PACKAGE_INFO section in which to specify package information on software to be distributed and information necessary for setup. You can conveniently use the *Package Information tool* to create and verify the PACKAGE_INFO section.

An AIT file also requires a program product ID file, which associates the AIT file with the software to be distributed. You can use the Package Information tool to generate the program product ID file automatically. For details about how to edit the generated program product ID file, see *B. Editing a Program Product ID File*.

This section describes how to use the Package Information tool to generate the PACKAGE_INFO section and a program product ID file.

2.5.1 Procedure for generating the PACKAGE_INFO section and a program product ID file

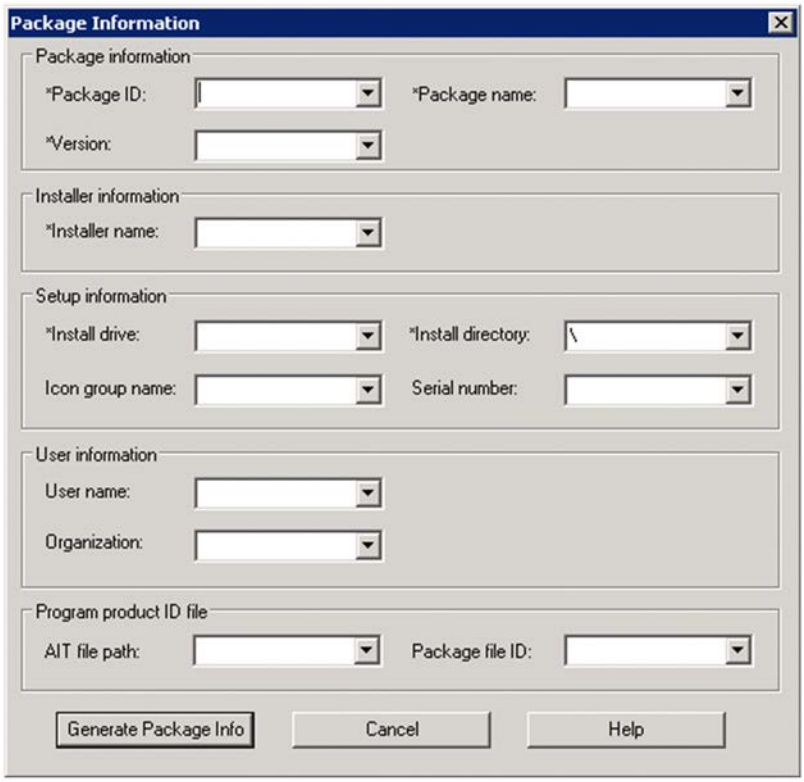
This subsection explains how to generate the PACKAGE_INFO section and a program product ID file. For details about the PACKAGE_INFO section, see *3.2.1 PACKAGE_INFO*.

To generate the PACKAGE_INFO section and a program product ID file:

1. From the **File** menu, choose **Open** to open the AIT file for which you want to generate the package information.
2. From the **Tools** menu, choose **Package Information**.

   The Package Information dialog box appears.

Figure 2-13: Package Information dialog box



If the AIT file already contains the PACKAGE_INFO section, the Package Information dialog box displays the existing values in the PACKAGE_INFO section.

3. Enter a value for each item.

   For any items, you cannot use `\n`, `\r`, `\t`, and other character strings that have special meanings. In the dialog box, you must specify a value for the items marked with an asterisk (`*`). The following gives the meaning of each item.

Item	Description
Package ID	Specify the package ID with a value from 1 to 44 bytes. You can use upper-case alphanumeric characters, a hyphen (`-`) and an underscore (_).
Product	Specify the package name with a value from 1 to 50 bytes. You cannot use a backslash (\) and a semicolon (`;`).
Version	Specify the version or revision of the software with a value from 1 to 6 bytes. You can use upper-case letters, numbers, and a slash (`/`).

Item	Description
Installer name	Specify the name of the installer to be used to install the software with a value from 1 to 256 bytes. You cannot use the following symbols: * " : | < > ?
Install drive	Specify the drive in which to install the software by using two characters: an alphanumeric character and a colon (:)
Install directory	Specify the path name of the directory in which to install the software. Specify the path name with a value from 1 to 128 bytes. The name must start with a backslash (\).
Icon group name	Specify the icon group name of the software with a value from 1 to 40 bytes.
Serial number	Specify the serial number of the software to be installed with a value from 1 to 64 bytes. For software for which a CD key is required during installation, enter the CD key.
English	Select this option button if the software is the English version. Specify the user name and the organization below.
User name	Specify the software owner name with a value from 1 to 40 bytes.
Organization	Specify the company name of the software owner with a value from 1 to 80 bytes.
AIT file path	Specify the full-path name, including the drive name, of the AIT file to be generated with a value from 1 to 256 bytes. You cannot use a semicolon (;).
Package file ID	Specify one or more files that constitute the software to be distributed so that the software can be uniquely identified. Specify a file name that uniquely identifies the software to be distributed with a value from 1 to 477 bytes. When specifying two or more file names, use a semicolon (;) to separate each file name. When all of the specified files exist during packaging, the Package Information tool assumes that the software that includes the files is to be distributed by using the AIT file.

Note that if you generate the PACKAGE_INFO section without specifying either **AIT file path** or **Package file ID**, the Package Information tool does not generate a program product ID file.

For details about how to specify **Installer name** and **Package file ID**, see *2.5.2 Specifying the installer and the files for identifying the software to be distributed*.

4. Click the **Generate Package Info** button.

   The Package Information tool checks the length, characters, and other items to validate the specified information in the same way as the Packager. The PACKAGE_INFO section is generated or updated in the AIT file.

```
PACKAGE_INFO
{
    PackageID         = "ADOBEREADER";
    Product           = "Adobe Reader 6.0";
    Version           = "0600";
    InstallerName     = "AdbeRdr60_enu_full.exe";
    InstallDrive      = "C:";
    InstallDirectory  = "'\Program Files'\Adobe'\Acrobat 6.0";
}
```

In addition, a program product ID file is generated with the information specified in the Package Information dialog box. If a program product ID file already exists, the specified information is added to the existing file.

The generated program product ID file is assigned the file name `PPDEFAIT.DMP` and stored in *JP1/IT-Desktop-Management-2-installation-folder*`\DMPRM`.

5. From the **File** menu, choose **Save** to save the AIT file with the PACKAGE_INFO section generated or updated.

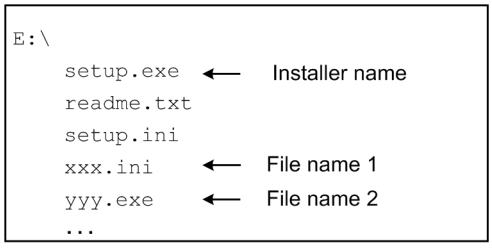
The values entered in the Package Information dialog box are displayed in the JP1/ITDM2 Packaging dialog box during packaging. Among the items displayed in this dialog box, you cannot change the values of the **Package ID**, **Product**, and **Version**. You can change the values of other items during packaging or remote installation.

2.5.2 Specifying the installer and the files for identifying the software to be distributed

The installer name in the AIT file and the file names (for identification) in the program product ID file must be specified with the relative paths from the packaging directory.

The following examples show how to specify the above file names in the AIT file and the program product ID file. In the following examples, suppose the files to be packaged exist at the CD-ROM drive (`E:`).

- When the packaging directory contains no subdirectories



Packaging directory:

```
E:\
```

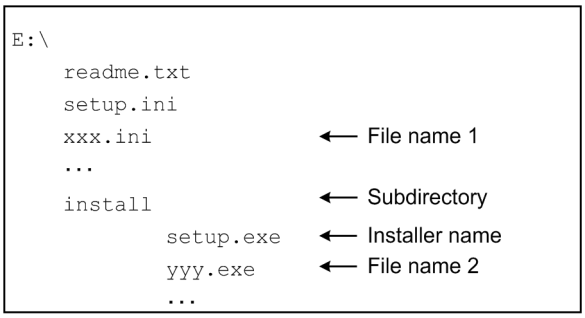
Installation program name you specify as package information in the AIT file:

```
setup.exe
```

File names you specify in the program product ID file:

```
xxx.ini;yyy.exe
```

- When the packaging directory contains a subdirectory



Packaging directory:

```
E:\
```

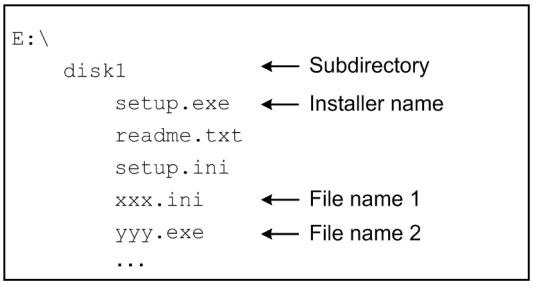
Installer name you specify as package information in the AIT file:

```
install\setup.exe
```

File names you specify in the program product ID file:

```
xxx.ini;install\yyy.exe
```

- When the packaging directory is a subdirectory



Packaging directory:

```
E:\disk1
```

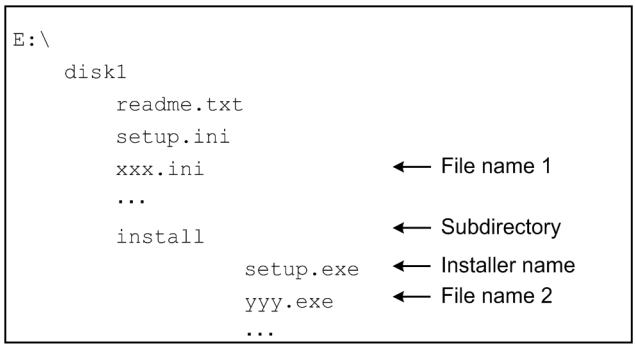
Installer name you specify as package information in the AIT file:

```
setup.exe
```

File names you specify in the program product ID file:

```
xxx.ini;yyy.exe
```

- When the packaging directory is a subdirectory that contains a subordinate directory



Packaging directory:

```
E:\disk1
```

Installer name you specify as package information in the AIT file:

```
install\setup.exe
```

File names you specify in the program product ID file:

```
xxx.ini;install\yyy.exe
```

2.6 Editing an AIT file

After an initial AIT file has been generated automatically by recording, you need to edit it. To edit an AIT file, you have to understand the API functions for window processing that are frequently used in AIT files.

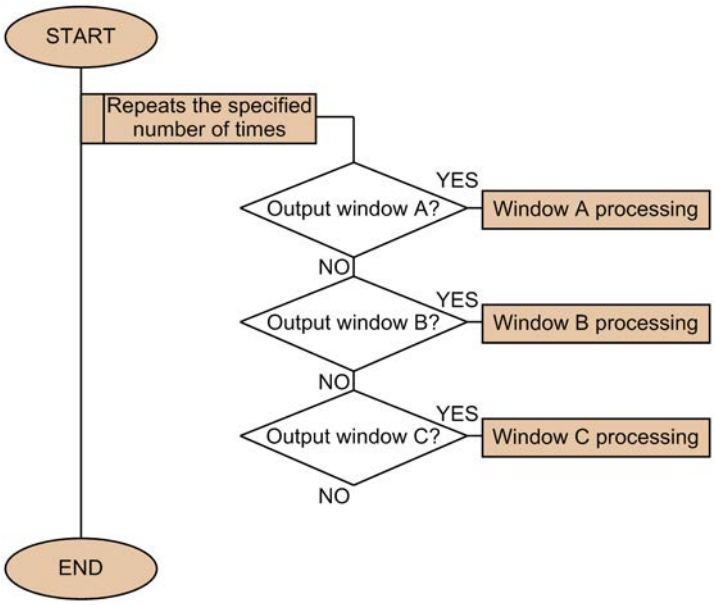
This section describes the API functions for window processing, then explains required manual modifications.

2.6.1 Window processing

During remote installation, the user at a destination computer might respond to dialog boxes or use the keyboard or mouse by mistake. If so, remote installation might stop. In the MAIN section, you must code the main processing, while taking possible interruptions into consideration.

The following figure shows the window processing in an AIT file:

Figure 2-14: Window processing in an AIT file



The AIT file is coded to repeat a loop that finds and operates windows. Therefore, if the user at the computer responds to Window A by mistake and Window B appears as a result, the Automatic Installation Tool might skip the processing for Window A and execute the processing for Window B instead.

When creating an AIT file, you check the windows displayed during installation, and

list the operations on the windows. These operations are coded in a loop. This processing structure allows you to complete the processing regardless of the order of the output windows and user operations.

In an AIT file, the processing for windows must be sequentially coded in the loop. The processing of each window is coded as a set of the following processes:

- Finding a window
- Operating the found window

These processes are coded using the API functions listed in Tables 2-2 and 2-3.

Table 2-2: API functions for finding a window

API	Description
`AIT_FocusWindow`	Finds a window, and sets the focus on the window.
`AIT_CtrlSetFocus`	Sets the focus on a specific control.

Table 2-3: API functions for operating a window

API	Description
`AIT_VerifyExistence`	Checks whether the window contains controls such as buttons and check boxes.
`AIT_VerifyEnabled`	Checks whether the control is enabled.
`AIT_VerifyPos`	Identifies the position of the control.
`AIT_PlayKey`	Simulates keyboard operations such as pressing **Enter**.

Note on window processing

This note applies when you use an AIT file to remotely install software in any of the following OS environments: Windows 8.1, Windows 8, Windows Server 2012, Windows 7, Windows Server 2008, Windows Vista, Windows Server 2003, or Windows XP. In this case, the API functions for window operations (`AIT_FocusWindow` and `AIT_CtrlSetFocus`) might not be able to set the focus in the application. In such cases, execute either of the API functions `AIT_Exec` and `AIT_ExecCommand`, which are used for recorder operations, and then use `AIT_PlayKey` to simulate the pressing of the **Alt** + **Tab** keys to move the focus from the desktop.

The following describes the roles of these API functions, and explains how to use them. For details about the parameters and return values for the API functions, see *4. API Function Reference*.

(1) AIT_FocusWindow

`AIT_FocusWindow` finds a window, and sets the focus on the window. Before starting the operations on the target window, you must find it and set the focus on it.

When you find a window, you need to specify the window text and class name of the window as parameters. To identify the window text and the class name, you can conveniently use the Window Properties tool. For this purpose, you can also use the codes in the AIT file generated automatically by the Recorder.

If the target window is found, the API function returns the handle to the window; otherwise, it returns 0.

The following gives an example of using `AIT_FocusWindow`.

Figure 2-15: Example of using AIT_FocusWindow

```
if(AIT_FocusWindow("Select Directory", "#32770") > 0)
     AIT_PlayKey("%(p)");
     AIT_PlayKey(InstallPoint);
     AIT_PlayKey("{ENTER}");
     AIT_Sleep(SLEEP_TIME_EVENTS);
     iLoopCount = 0;
Endif;
```

(2) AIT_CtrlSetFocus

`AIT_CtrlSetFocus` sets the focus on a specific control. If the window contains multiple controls, use `AIT_CtrlSetFocus` to set the focus on the target control, then simulate user operations.

The target control is specified with its caption or control ID, as well as the control type (such as button or list box).

The following gives an example of using `AIT_CtrlSetFocus` with the caption of a control specified.

Figure 2-16: Example of using AIT_CtrlSetFocus

```
if(AIT_FocusWindow("Select Installation Directory", "#32770") > 0)
   AIT_CtrlSetFocus("&Next >",BUTTON_CTRL);
    AIT_Sleep(SLEEP_TIME_EVENTS);
    AIT_PlayKey("{ENTER}");
    AIT_Sleep(SLEEP_TIME_EVENTS);
    iLoopCount = 0;
Endif;
```

(3) AIT_VerifyExistence

AIT_VerifyExistence checks whether the control exists on the window.

For example, if there are windows that have the same caption, the window found by AIT_VerifyExistence is not always the target window that you want to operate. In such a case, you can use AIT_VerifyExistence to check whether the window contains the desired control, and to determine whether the window is the target to be operated.

The target control is specified with its caption or control ID, as well as the control type (such as button or list box). If the target control exists, this API function returns 1; otherwise it returns 0.

The following gives an example of using AIT_VerifyExistence with the caption of a control specified.

Figure 2-17: Example of using AIT_VerifyExistence

```
if(AIT_FocusWindow("Setup", "#32770") > 0)
    if(AIT_VerifyExistence("&Finish",BUTTON_CTRL)==1)
        AIT_CtrlSetFocus("&Finish",BUTTON_CTRL);
        AIT_Sleep(SLEEP_TIME_EVENTS);
        AIT_PlayKey("{ENTER}");
        AIT_Sleep(SLEEP_TIME_EVENTS);
        iLoopCount = 0;
    Endif;
Endif;
```

(4) AIT_VerifyEnabled

AIT_VerifyEnabled checks whether the control is enabled.

For example, suppose buttons are associated with a check box, and they are enabled when the check box is selected and are disabled when it is not selected. In such a case, you can use AIT_VerifyEnabled to check whether the button is enabled. If the button is enabled, a simulation of the actions that occur when a user clicks the button is performed.

AIT_VerifyEnabled is used in combination with AIT_VerifyExistence or AIT_VerifyPos. After the existence of the target control is verified with AIT_VerifyExistence or AIT_VerifyPos, AIT_VerifyEnabled is used to check whether the control is enabled.

The target control is specified with its caption or control ID, as well as the control type (such as button or list box). If the target control is enabled, this API function returns 1; otherwise it returns 0.

The following gives an example of using AIT_VerifyEnabled with the caption of a control specified.

Figure 2-18: Example of using AIT_VerifyEnabled

```
if(AIT_FocusWindow("Setup", "#32770") > 0)
    if ((AIT_VerifyExistence("&Finish",BUTTON_CTRL)==1)
    && ((AIT_VerifyEnabled("&Finish",BUTTON_CTRL)==1)
        AIT_CtrlSetFocus("&Finish",BUTTON_CTRL);
        AIT_Sleep(SLEEP_TIME_EVENTS);
        AIT_PlayKey("{ENTER}");
        AIT_Sleep(SLEEP_TIME_EVENTS);
        iLoopCount = 0;
    Endif;
Endif;
```

(5) AIT_VerifyPos

`AIT_VerifyPos` checks the tab order of the control. Similar to `AIT_VerifyExistence`, `AIT_VerifyPos` is also used to check whether the target control exists. However, if there are controls that have the same caption in the window, the target control cannot be identified by the caption only. In such a case, the target control can be identified by using `AIT_VerifyPos` with the tab order of the target control specified.

The target control is specified with its caption or control ID, as well as the control type (such as button or list box) and tab order. If the specified tab order matches that of the control, this API function returns 1; otherwise it returns 0.

The following gives an example of using AIT_VerifyPos with the caption of a control specified.

Figure 2-19: Example of using AIT_VerifyPos

```
if(AIT_FocusWindow("Setup", "#32770") > 0)
    if(AIT_VerifyPos("&Finish",BUTTON_CTRL,6)==1)
        AIT_CtrlSetFocus("&Finish",BUTTON_CTRL);
        AIT_Sleep(SLEEP_TIME_EVENTS);
        AIT_PlayKey("{ENTER}");
        AIT_Sleep(SLEEP_TIME_EVENTS);
        iLoopCount = 0;
    Endif;
Endif;
```

(6) AIT_PlayKey

`AIT_PlayKey` simulates a keyboard operation.

In this API function, you can specify characters to be entered or keys to be pressed. For example, when the character string `abcd` is specified, this API function enters it as if you typed. When `{ESC}` is specified, this API function simulates pressing **Esc**, and when `%(F)` is specified, it simulates pressing the **Alt**+**F** shortcut key.

The following gives an example of using `AIT_PlayKey`.

Figure 2-20: Example of using AIT_PlayKey

```
if(AIT_FocusWindow("Setup", "#32770") > 0)
    if(AIT_VerifyPos("Accoun&t",BUTTON_CTRL,6)==1)
       AIT_PlayKey("administrator");
       AIT_Sleep(SLEEP_TIME_EVENTS);
       AIT_PlayKey("{ENTER}");
       iLoopCount = 0;
    Endif;
Endif;
```

(7) Example of automatically generated codes for window processing

In an AIT file automatically generated by the Recorder, window information and your operations on windows are recorded. The following figure shows an example of automatically generated code for window processing.

Figure 2-21: Example of automatically generated codes for window processing

```
...
MAIN
{
    AIT_SetDefaultWaitTimeout(1.0);
    AIT_DMPSTRC();
    DM_RTN = NG_END;
    iCapsLockState = AIT_GetKeyState(CAPSLOCK);
    iNumLockState = AIT_GetKeyState(NUMLOCK);
    iScrollLockState = AIT_GetKeyState(SCROLLLOCK);
    iInsertLockState = AIT_GetKeyState(INSERTLOCK);
    bRtn= AIT_Exec(InstallerName,SW_SHOWNORMAL);
    if(bRtn == false)
        iLoopCount = iLoopMax;
    Endif;
    while(iLoopCount < iLoopMax)
        if(AIT_FocusWindow("Uncompressing Acrobat Reader...", "#32770") > 0)
            AIT_Sleep(SLEEP_TIME);
            iLoopCount=0;
        Endif;
        if(AIT_FocusWindow("Acrobat Reader 6.0 - Setup", "#32770") > 0)
            AIT_CtrlSetFocus("&Next >",BUTTON_CTRL);
            AIT_Sleep(SLEEP_TIME_EVENTS);
            AIT_PlayKey("{ALT_LOCK}n{ALT_UNLOCK}");
            AIT_Sleep(SLEEP_TIME_EVENTS);
            iLoopCount=0;
        Endif;
...
```

Finds a window

Sets the focus on the control

Simulates pressing **Alt+N**

2.6.2 Automatically generated flags

Operations on different windows that have the same caption are recorded in an AIT file as operations on the same window. In the generated AIT file, the following flags are used to script the operations on windows:

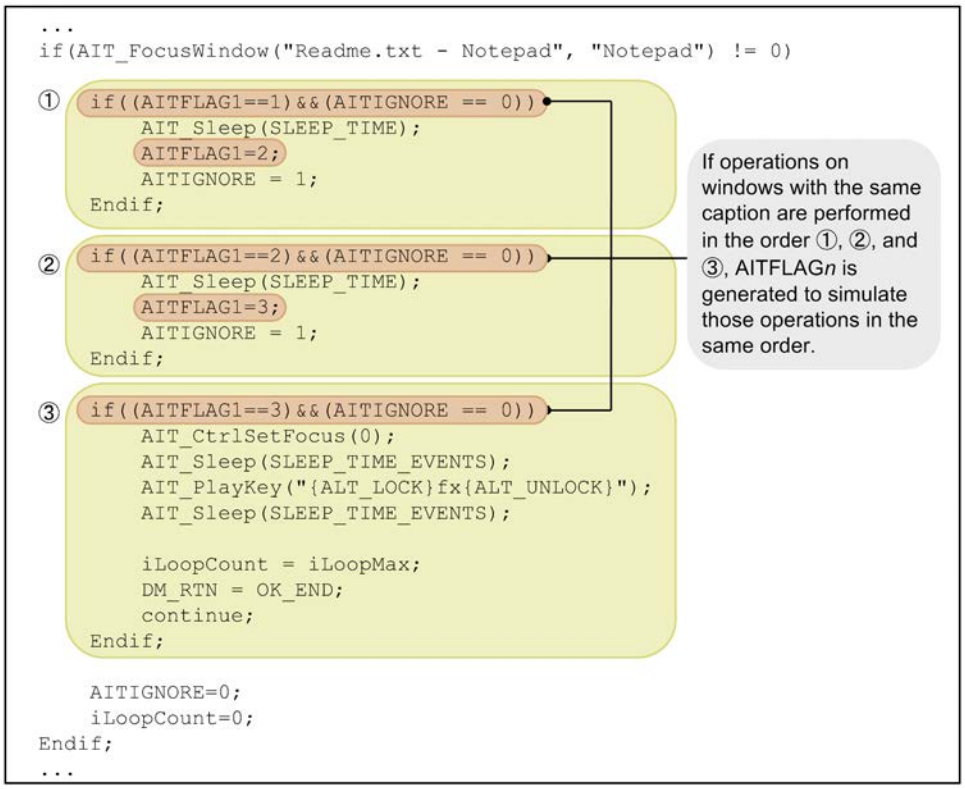
- Window flag `AITFLAG`*n* (*n*: 1 or a greater integer)
- Event flag `AITEVENTFLAG`*n* (*n*: 1 or a greater integer)
- Invalid flag `AITIGNORE`

The following describes these flags.

(1) Window flag (AITFLAG1n)

Window flags specify the order in which windows will be displayed. The following figure gives an example of generated window flags.

Figure 2-22: Example of generated window flags



(2) Event flag (AITEVENTFLAGn)

Event flags specify the order in which window operations will be performed. The following figure gives an example of generated event flags.

Figure 2-23: Example of generated event flags

(3) Invalid flag (AITIGNORE)

The invalid flag suppresses execution of the next operation until a window closes. The following figure gives an example of generated invalid flags.

Figure 2-24: Example of generated invalid flags

2.6.3 Checking and modifying an automatically generated AIT file

After an AIT file has been generated automatically by the Recorder, you need to check and modify the generated codes in light of the following concerns:

(1) Repeat recording under changed installation conditions

The windows output by the installer vary with the hard disk's free space and OS installation conditions. All windows and events cannot be recorded by one recording cycle. To simulate operations for all windows, you have to repeat recording under changed installation conditions to generate multiple AIT files. The following gives possible installation conditions.

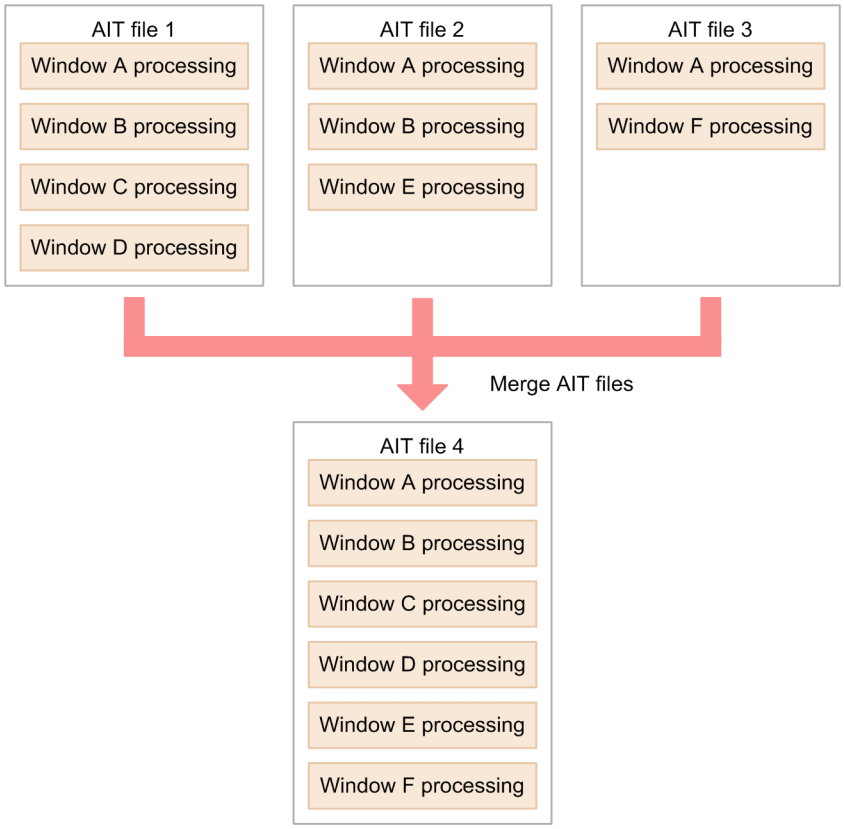
Installation conditions	Description
OS	The windows output by the installer may vary with the installed OS.

Installation conditions	Description
PC environment	The windows output by the installer may depend on the hard disk's free space, whether the prerequisite programs have been installed, whether the software is installed for the first time or for upgrading the version, and other target PC environment factors.
User operations	The windows output by the installer may vary depending on the user's operation such as changing the installation directory and canceling installation.

After multiple AIT files are generated according to different installation conditions, extract necessary codes from the AIT files, and combine the codes into one AIT file. Here, the necessary codes mean the codes for simulating operations on different windows output according to installation conditions.

The following figure shows an example of combining multiple AIT files.

Figure 2-25: Example of combining multiple AIT files



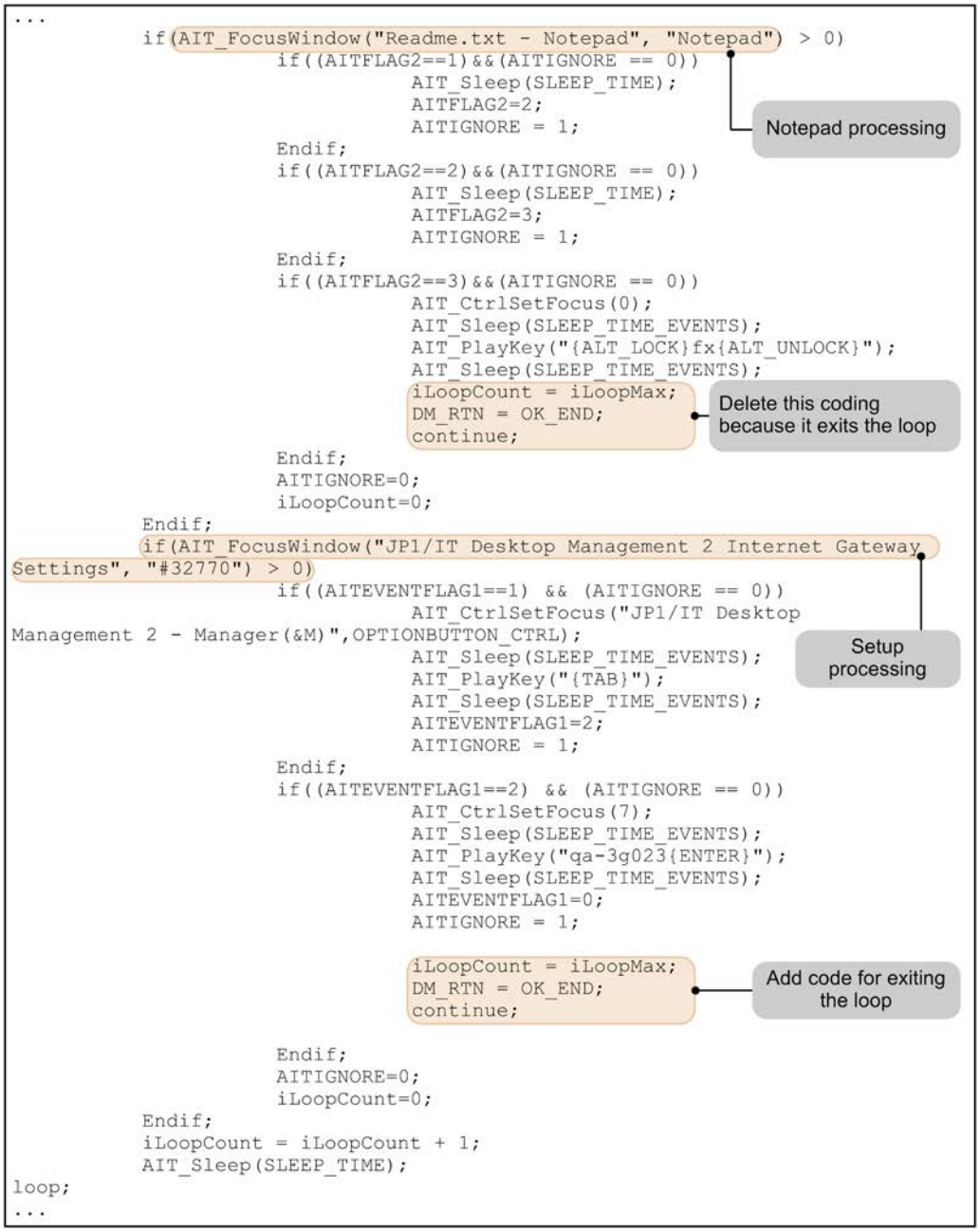
(2) Correctly identify the end of installation operations

The Recorder automatically generates the code of normally terminating installation for the window you operated last during recording. However, the window you operated last is not always the one output at the end of installation.

Suppose that the installer opens a Readme file with Notepad after the installation. In this case, you may close the Readme file at the end of the recording. If you do this, the processing for terminating installation is generated for the operation for closing Notepad. During installation by using an AIT file generated in this way, if you close Notepad before the installation ends, the ongoing installation is terminated.

To avoid this, do not record any operations on windows that appear after the end of the installation. Alternatively, manually modify the AIT file so that Notepad is not closed until the end of the installation. The following gives an example of modifying an AIT file.

Figure 2-26: Example of modifying an AIT file



If a dialog box that appears during installation and the dialog box that appears at the

end of installation have the same caption, the Automatic Installation Tool cannot identify the end of installation correctly. In this case, additionally specify a label or button to differentiate the dialog boxes.

(3) Do not use variable text to identify a window

For identification of a window, you can use not only window text, but also the text of controls on the window as conditions for identifying. However, do not use variable text as conditions for identifying a window.

For example, suppose that a control displays the PC's free disk space in the `Available disk space: 2252195 K` format. As each PC has a different amount of free disk space, you cannot use the string `2252195 K` to identify the window.

In an automatically generated AIT file, if text that varies depending on the PC environment or OS is used to identify a window, delete the text. Alternatively, you should only use constant text (such as `Available disk space:`) for window identification.

(4) Delete codes for unrelated windows

If applications other than the installer are activated during recording, the Recorder may have recorded codes for windows that are not related to the installer. If such codes have been recorded, delete them.

2.6.4 Linkage with the Packager and Remote Installation Manager

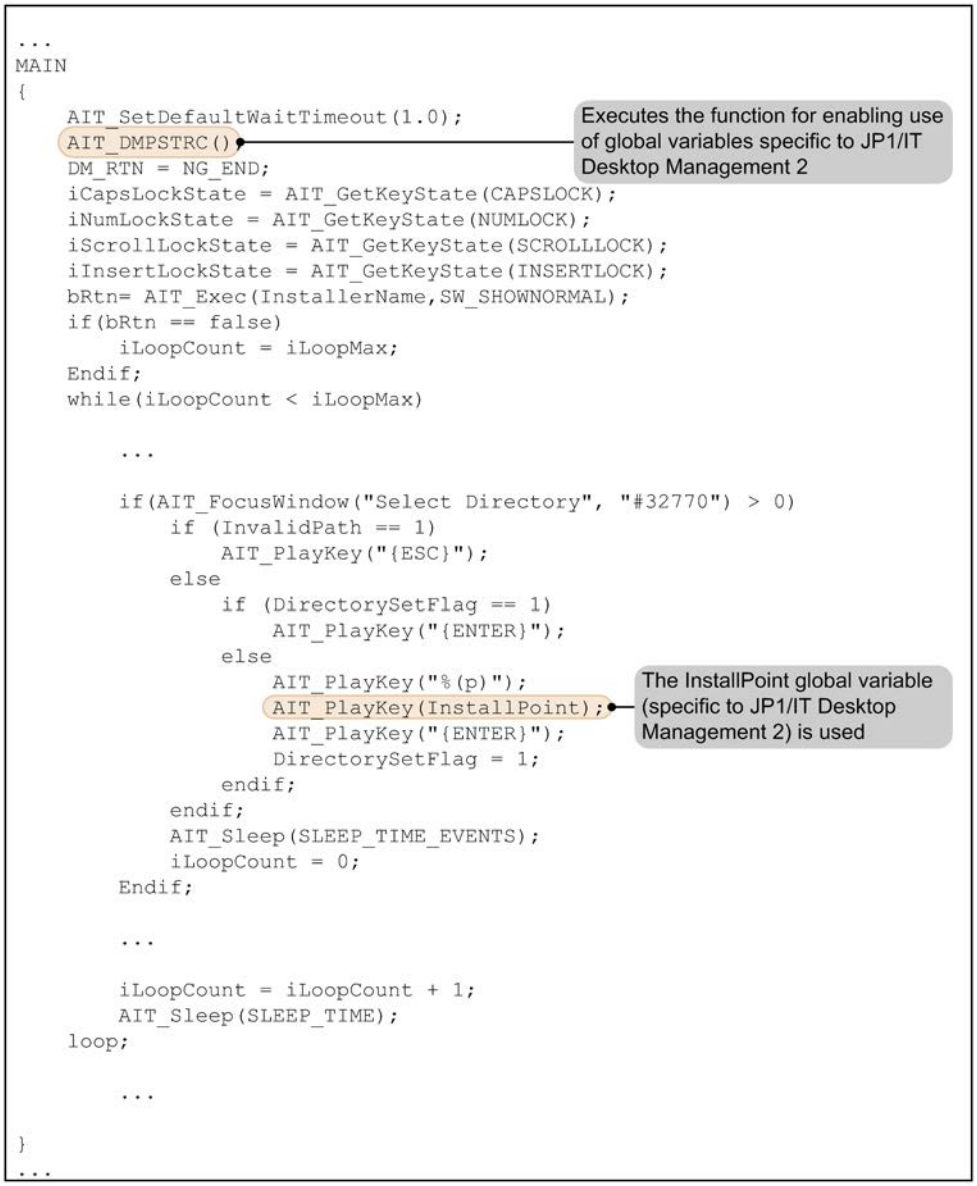
During packaging or remote installation, if you change the package information such as *Install directory* or *User name*, the PACKAGE_INFO section in the AIT file is updated automatically. However, such information in the MAIN section is not updated. To prevent inconsistency between these sections, instead of hard-coding such information, use the global variables provided by JP1/IT Desktop Management 2 as needed. The following table describes the available global variables for JP1/IT Desktop Management 2.

Global variable for JP1/IT Desktop Management 2	Description
`InstallerName`	This is the installer name specified in the PACKAGE_INFO section. You cannot change this value during packaging or remote installation. Note that, during remote installation, this variable contains the installer name including the path name of the installation work directory on the computer. Therefore, the value during remote installation differs from the value during AIT file debugging.
`InstallDrive`	This is replaced with the installation drive name specified during packaging or remote installation. The default is the installation drive name specified in the PACKAGE_INFO section.

Global variable for JP1/IT Desktop Management 2	Description
`InstallDirectory`	This is replaced with the installation directory name specified during packaging or remote installation. The default is the installation directory name specified in the PACKAGE_INFO section.
`InstallPoint`	This is replaced with a combination of `InstallDrive` and `InstallDirectory`, that is, the installation path directory including the drive name.
`JUser`	This is replaced with the user name specified during packaging or remote installation.
`JCompany`	This is replaced with the company name specified during packaging or remote installation.
`SerialNumber`	This is replaced with the serial number specified during packaging or remote installation. It is used to type in the license key.
`IconGroupName`	This is replaced with the icon group name specified during packaging or remote installation. It is used to change the icon group.

The following figure shows an example of using global variables.

Figure 2-27: Example of using global variables



2.6.5 Adding the coding for error handling and setting a return code

You need to add error handling to support windows output if an error occurs. Determine whether to continue or abort the installation if an error occurs, and code the error handling according to the determination.

Also, you need to add the coding for setting a return code to indicate whether or not the installation normally terminated. This coding is necessary for checking errors at the managing server. The return code is a two-digit value displayed as the ninth and tenth (from the left) digits of the maintenance code in the Details dialog box.

The following figure shows an example of coding for error handling and setting a return code.

Figure 2-28: Example of coding for error handling and setting a return code (1/2)

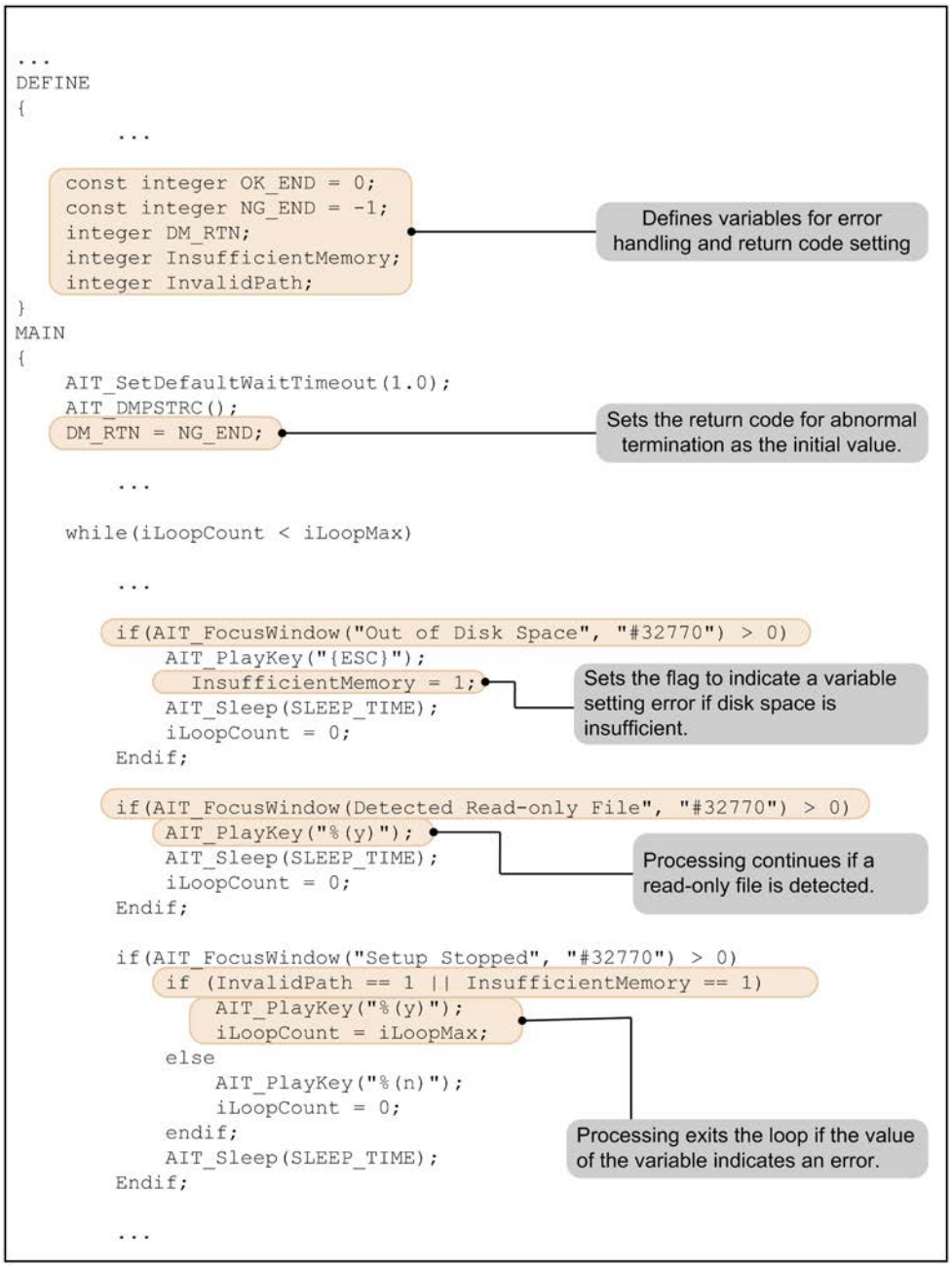


Figure 2-29: Example of coding for error handling and setting a return code (2/2)

2.6.6 Example of a completed AIT file

The following gives an example of a completed AIT file for remotely installing Acrobat Reader 6.0:

```
PACKAGE_INFO
{
  PackageID        = "ADOBEREADER";
  Product          = "Adobe Reader 6.0";
  Version          = "0600";
  InstallerName    = "AdbeRdr60_enu_full.exe";
  InstallDrive     = "C:";
  InstallDirectory = "'\Program Files'\Adobe'\Acrobat 6.0";
}
DEFINE
{
  integer iLoopCount = 0;
  integer iLoopMax = 60;
  integer DM_RTN;
```

```
  integer WINH;
  integer iCapsLockState;
  integer iNumLockState;
  integer iScrollLockState;
  bool bRtn;
  const integer OK_END = 0;
  const integer NG_END = -1;
  float SLEEP_TIME = 1.0;
  float SLEEP_TIME_RESTART = 10.0;
  float SLEEP_TIME_EVENTS = 0.5;
  integer DirectorySetFlag=0;
  integer InsufficientMemory=0;
  integer InvalidPath=0;
}
MAIN
{
  AIT_SetDefaultWaitTimeout(1.0);
  AIT_DMPSTRC();
  DM_RTN = NG_END;
  iCapsLockState = AIT_GetKeyState(CAPSLOCK);
  iNumLockState = AIT_GetKeyState(NUMLOCK);
  iScrollLockState = AIT_GetKeyState(SCROLLLOCK);
  bRtn= AIT_Exec(InstallerName,SW_SHOWNORMAL);
  if(bRtn == false)
      iLoopCount = iLoopMax;
  Endif;
  while(iLoopCount < iLoopMax)
      if(AIT_FocusWindow("Netopsystems FEAD Optimizer",
"#32770") != 0)
          AIT_Sleep(SLEEP_TIME);
          iLoopCount=0;
      Endif;
     if(AIT_FocusWindow("InstallShield Wizard", "#32770") != 0)
          AIT_Sleep(SLEEP_TIME);
          iLoopCount=0;
      Endif;
      if(AIT_FocusWindow("Windows Installer", "#32770") != 0)
          AIT_Sleep(SLEEP_TIME);
          iLoopCount=0;
      Endif;
      if(AIT_FocusWindow("Adobe Reader 6.0 - Setup",
"MsiDialogCloseClass") != 0)
          if((AIT_VerifyExistence("&Next >",BUTTON_CTRL)==1)
          && (AIT_VerifyEnabled("&Next >",BUTTON_CTRL)==1))
              if (InvalidPath == 1 || InsufficientMemory == 1)
                  AIT_CtrlSetFocus("&Next >",BUTTON_CTRL);
                  AIT_Sleep(SLEEP_TIME_EVENTS);
                  AIT_PlayKey("{ESC}");
```

```
                    AIT_Sleep(SLEEP_TIME_EVENTS);
                else
                    if (DirectorySetFlag == 0)
                    if((AIT_VerifyExistence("Change &Destination
Folder...",BUTTON_CTRL)==1)
                      && (AIT_VerifyEnabled("Change &Destination
Folder...",BUTTON_CTRL)==1))
                        AIT_CtrlSetFocus("Change &Destination
Folder...",BUTTON_CTRL);
                          AIT_Sleep(SLEEP_TIME_EVENTS);
                          AIT_PlayKey("%(d)");
                          AIT_Sleep(SLEEP_TIME_EVENTS);
                       else

if((AIT_VerifyExistence("Re&pair",OPTIONBUTTON_CTRL)==1)
                          &&
(AIT_VerifyEnabled("Re&pair",OPTIONBUTTON_CTRL)==1))

AIT_CtrlSetFocus("Re&pair",OPTIONBUTTON_CTRL);
                              AIT_Sleep(SLEEP_TIME_EVENTS);
                              AIT_PlayKey("{ENTER}");
                              AIT_Sleep(SLEEP_TIME_EVENTS);
                          else
                           AIT_CtrlSetFocus("&Next >",BUTTON_CTRL);
                              AIT_Sleep(SLEEP_TIME_EVENTS);
                              AIT_PlayKey("%(n)");
                              AIT_Sleep(SLEEP_TIME_EVENTS);
                          Endif;
                       Endif;
                    else
                       AIT_CtrlSetFocus("&Next >",BUTTON_CTRL);
                       AIT_Sleep(SLEEP_TIME_EVENTS);
                       AIT_PlayKey("%(n)");
                       AIT_Sleep(SLEEP_TIME_EVENTS);
                    Endif;
                Endif;
            Endif;
            if((AIT_VerifyExistence("&Install",BUTTON_CTRL)==1)
            && (AIT_VerifyEnabled("&Install",BUTTON_CTRL)==1))
                AIT_CtrlSetFocus("&Install",BUTTON_CTRL);
                AIT_Sleep(SLEEP_TIME_EVENTS);
                AIT_PlayKey("%(i)");
                AIT_Sleep(SLEEP_TIME_EVENTS);
            Endif;
            if((AIT_VerifyExistence("&Finish",BUTTON_CTRL)==1)
            && (AIT_VerifyEnabled("&Finish",BUTTON_CTRL)==1))
                AIT_CtrlSetFocus("&Finish",BUTTON_CTRL);
                AIT_Sleep(SLEEP_TIME_EVENTS);
```

```
                AIT_PlayKey("%(f)");
                AIT_Sleep(SLEEP_TIME_EVENTS);
                iLoopCount = iLoopMax;
                if (InvalidPath == 0 && InsufficientMemory == 0)
                    DM_RTN = OK_END;
                Endif;
                continue;
            Endif;
            if((AIT_VerifyExistence("&No",BUTTON_CTRL)==1)
            && (AIT_VerifyEnabled("&No",BUTTON_CTRL)==1)
            && (AIT_VerifyExistence("&Yes",BUTTON_CTRL)==1)
            && (AIT_VerifyEnabled("&Yes",BUTTON_CTRL)==1)
          && (AIT_VerifyExistence("Are you sure you want to cancel
Adobe Reader 6.0 installation?",STATIC_CTRL)==1)
           && (AIT_VerifyEnabled("Are you sure you want to cancel
Adobe Reader 6.0 installation?",STATIC_CTRL) == 1))
                if (InvalidPath == 1 || InsufficientMemory == 1)
                    AIT_CtrlSetFocus("&Yes",BUTTON_CTRL);
                    AIT_Sleep(SLEEP_TIME_EVENTS);
                    AIT_PlayKey("y");
                    AIT_Sleep(SLEEP_TIME_EVENTS);
                else
                    AIT_CtrlSetFocus("&No",BUTTON_CTRL);
                    AIT_Sleep(SLEEP_TIME_EVENTS);
                    AIT_PlayKey("n");
                    AIT_Sleep(SLEEP_TIME_EVENTS);
                Endif;
            Endif;
            if((AIT_VerifyExistence("OK",BUTTON_CTRL)==1)
            && (AIT_VerifyEnabled("OK",BUTTON_CTRL)==1)
            && (AIT_VerifyExistence("Out of Disk
Space",STATIC_CTRL)==1)
            && (AIT_VerifyEnabled("Out of Disk
Space",STATIC_CTRL)==1))
                AIT_CtrlSetFocus("OK",BUTTON_CTRL);
                AIT_Sleep(SLEEP_TIME_EVENTS);
                AIT_PlayKey("{ENTER}");
                AIT_Sleep(SLEEP_TIME_EVENTS);
                InsufficientMemory = 1;
            Endif;
            if((AIT_VerifyPos("&Change...",BUTTON_CTRL,1)==1)
          && (AIT_VerifyEnabled("&Change...",BUTTON_CTRL, 1) == 1)
            && (AIT_VerifyPos("&Help",BUTTON_CTRL,2)==1)
            && (AIT_VerifyEnabled("&Help",BUTTON_CTRL, 2) == 1)
            && (AIT_VerifyPos("&Space",BUTTON_CTRL,3)==1)
            && (AIT_VerifyEnabled("&Space",BUTTON_CTRL, 3) == 1)
            && (AIT_VerifyPos("< &Back",BUTTON_CTRL,4)==1)
           && (AIT_VerifyEnabled("< &Back",BUTTON_CTRL, 4) == 1)
```

```
          && (AIT_VerifyPos("&Next >",BUTTON_CTRL,5)==1)
          && (AIT_VerifyEnabled("&Next >",BUTTON_CTRL, 5) == 1)
          && (AIT_VerifyPos("Cancel",BUTTON_CTRL,6)==1)
          && (AIT_VerifyEnabled("Cancel",BUTTON_CTRL, 6) == 1)
          && (AIT_VerifyPos("Custom Setup",STATIC_CTRL,9)==1)
          && (AIT_VerifyEnabled("Custom Setup",STATIC_CTRL, 9)
== 1)
          && (AIT_VerifyPos("Feature
Description",BUTTON_CTRL,18)==1)
          && (AIT_VerifyEnabled("Feature
Description",BUTTON_CTRL, 18) == 0))
              AIT_CtrlSetFocus(0);
              AIT_Sleep(SLEEP_TIME_EVENTS);
              AIT_IMESetOpenStatus(0, false);
              AIT_Sleep(SLEEP_TIME_EVENTS);
              AIT_PlayKey("{ESC}");
              AIT_Sleep(SLEEP_TIME_EVENTS);
          Endif;
          iLoopCount=0;
      Endif;
      if(AIT_FocusWindow("Adobe Reader 6.0 - Setup",
"MsiDialogNoCloseClass") != 0)
          if((AIT_VerifyExistence("&Next >",BUTTON_CTRL)==1)
          && (AIT_VerifyEnabled("&Next >",BUTTON_CTRL)==1))
              AIT_CtrlSetFocus("&Next >",BUTTON_CTRL);
              AIT_Sleep(SLEEP_TIME_EVENTS);
              AIT_PlayKey("%(n)");
              AIT_Sleep(SLEEP_TIME_EVENTS);
          Endif;
          iLoopCount=0;
      Endif;
      if(AIT_FocusWindow("Adobe Reader 6.0 Setup",
"MsiDialogCloseClass") != 0)
          if((AIT_VerifyPos("&OK",BUTTON_CTRL,1)==1)
          && (AIT_VerifyEnabled("&OK",BUTTON_CTRL, 1) == 1)
          && (AIT_VerifyPos("&Cancel",BUTTON_CTRL,2)==1)
          && (AIT_VerifyEnabled("&Cancel",BUTTON_CTRL, 2) == 1)
          && (AIT_VerifyPos("&Look in:",STATIC_CTRL,3)==1)
        && (AIT_VerifyEnabled("&Look in:",STATIC_CTRL, 3) == 1)
          && (AIT_VerifyPos("&Look in:",COMBO_CTRL,4)==1)
         && (AIT_VerifyEnabled("&Look in:",COMBO_CTRL, 4) == 1)
          && (AIT_VerifyPos("Up One Level",BUTTON_CTRL,5)==1)
          && (AIT_VerifyEnabled("Up One Level",BUTTON_CTRL, 5)
== 1)
          && (AIT_VerifyPos("Create New
Folder",BUTTON_CTRL,6)==1)
        && (AIT_VerifyEnabled("Create New Folder",BUTTON_CTRL,
6) == 1)
```

```
          && (AIT_VerifyPos("&Folder name:",STATIC_CTRL,8)==1)
         && (AIT_VerifyEnabled("&Folder name:",STATIC_CTRL, 8)
== 1)
          && (AIT_VerifyPos("Browse to the destination
folder.",STATIC_CTRL,9)==1)
          && (AIT_VerifyEnabled("Browse to the destination
folder.",STATIC_CTRL, 9) == 1)
          && (AIT_VerifyPos("Change Current Destination
Folder",STATIC_CTRL,10)==1)
          && (AIT_VerifyEnabled("Change Current Destination
Folder",STATIC_CTRL, 10) == 1)
          && (AIT_VerifyPos("InstallShield",STATIC_CTRL,12)==1)
         && (AIT_VerifyEnabled("InstallShield",STATIC_CTRL, 12)
== 1)
          && (AIT_VerifyPos("InstallShield",STATIC_CTRL,13)==1)
         && (AIT_VerifyEnabled("InstallShield",STATIC_CTRL, 13)
== 0))
              if (InvalidPath == 0)
                  AIT_CtrlSetFocus(0);
                  AIT_Sleep(SLEEP_TIME_EVENTS);
                  AIT_IMESetOpenStatus(0, false);
                  AIT_PlayKey(InstallPoint);
                  AIT_PlayKey("{ENTER}");
                  AIT_Sleep(SLEEP_TIME_EVENTS);
                  DirectorySetFlag = 1;
              else
                  AIT_CtrlSetFocus("&Cancel", BUTTON_CTRL);
                  AIT_Sleep(SLEEP_TIME_EVENTS);
                  AIT_PlayKey("{ENTER}");
                  AIT_Sleep(SLEEP_TIME_EVENTS);
              Endif;
          Endif;
          iLoopCount=0;
      Endif;
      if(AIT_FocusWindow("Adobe Reader 6.0 Installer
Information", "MsiDialogCloseClass") != 0)
          if(AIT_VerifyPos("~Error 1314.The specified
path",STATIC_CTRL,1)==1)
              InvalidPath = 1;
              if((AIT_VerifyExistence("&OK",BUTTON_CTRL)==1)
              && (AIT_VerifyEnabled("&OK",BUTTON_CTRL)==1))
                  AIT_CtrlSetFocus("&OK",BUTTON_CTRL);
                  AIT_Sleep(SLEEP_TIME_EVENTS);
                  AIT_PlayKey("{ENTER}");
                  AIT_Sleep(SLEEP_TIME_EVENTS);
              Endif;
          Endif;
          if(AIT_VerifyExistence("~You must
```

```
restart",STATIC_CTRL)==1)
              if((AIT_VerifyExistence("&No",BUTTON_CTRL)==1)
              && (AIT_VerifyEnabled("&No",BUTTON_CTRL)==1))
                  AIT_CtrlSetFocus("&No",BUTTON_CTRL);
                  AIT_Sleep(SLEEP_TIME_EVENTS);
                  AIT_PlayKey("n");
                  AIT_Sleep(SLEEP_TIME_EVENTS);
              Endif;
          Endif;
          iLoopCount=0;
      Endif;
      iLoopCount = iLoopCount + 1;
      AIT_Sleep(SLEEP_TIME);
  loop;
  WINH = AIT_RegisterWindowMessage("ITM_REC_QUIT");
  if(WINH != 0)
      AIT_PostMessage(HWND_BROADCAST,WINH,DM_RTN,0);
  Endif;
}
ERROR
{
  DM_RTN = NG_END;
  WINH = AIT_RegisterWindowMessage("ITM_REC_QUIT");
  if(WINH != 0)
      AIT_PostMessage(HWND_BROADCAST,WINH,DM_RTN,0);
  Endif;
}
```

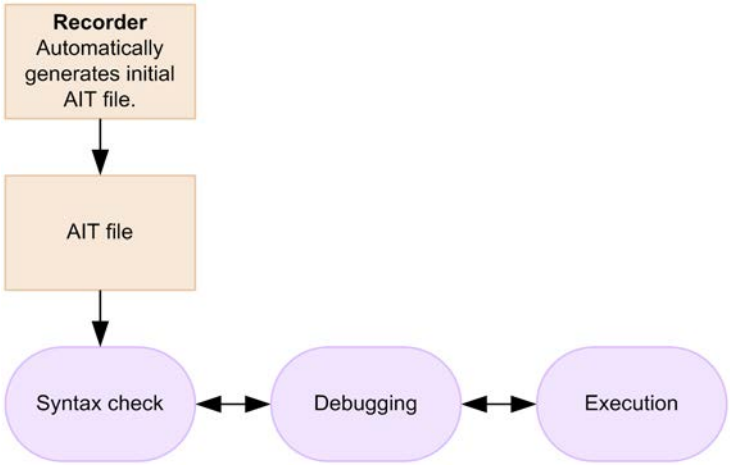
2.7 Debugging an AIT file

Once you created an AIT file, repeat syntax checks, execution, and debugging to finish a complete AIT file that can simulate user operations correctly. You do not need to compile AIT files.

Note that, depending on the JP1/IT Desktop Management 2 - Agent version, you must open the JP1/ITDM2 Automatic Installation Tool window with Administrator permissions when debugging an AIT file for a program that requires Administrator permissions. This restriction applies to JP1/IT Desktop Management 2 - Agent for any of the following OSs: Windows 8.1, Windows 8, Windows Server 2012, Windows 7, Windows Server 2008, and Windows Vista. If the program can be executed with non-Administrator user permissions, you can debug the AIT file with either type of permissions.

The following figure shows the flow of debugging an AIT file:

Figure 2-30: Flow of debugging an AIT file



When you open an AIT file with the Automatic Installation Tool, the **Build** and **Debug** menus for debugging are enabled.

The **Build** menu allows you to detect syntax errors in the AIT file and to execute the active (currently opened) AIT file.

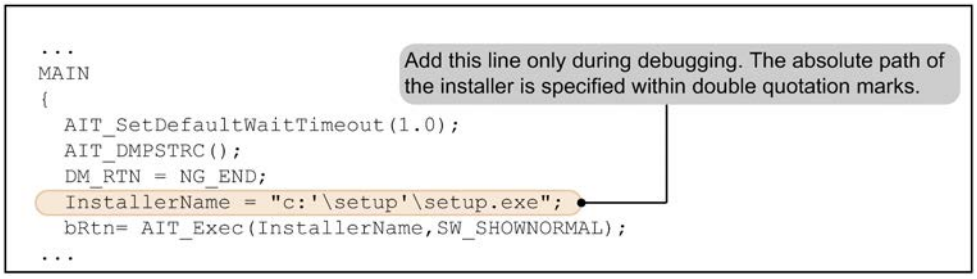
The **Debug** menu allows you to set breakpoints at which you can stop execution of the AIT file. You can use the Watch window to view and update variable values.

These facilities can be used only for the AIT files that have the extension `.ais`.

Notes on debugging

In the PACKAGE_INFO section, the installer name is specified in `InstallerName` with a relative path from the packaging directory. During debugging, in the MAIN section, you have to replace the relative path temporarily with the absolute path. After completion of debugging, you must set the original relative path. During syntax checks, however, you do not need to replace the relative path with the absolute path. The following gives an example of replacing the value of `InstallerName` with the absolute path.

Figure 2-31: Example of replacing the value of InstallerName with the absolute path



2.7.1 Syntax check and execution

The Automatic Installation Tool provides the **Build** menu that lets you conduct a syntax check on, or execute, an AIT file.

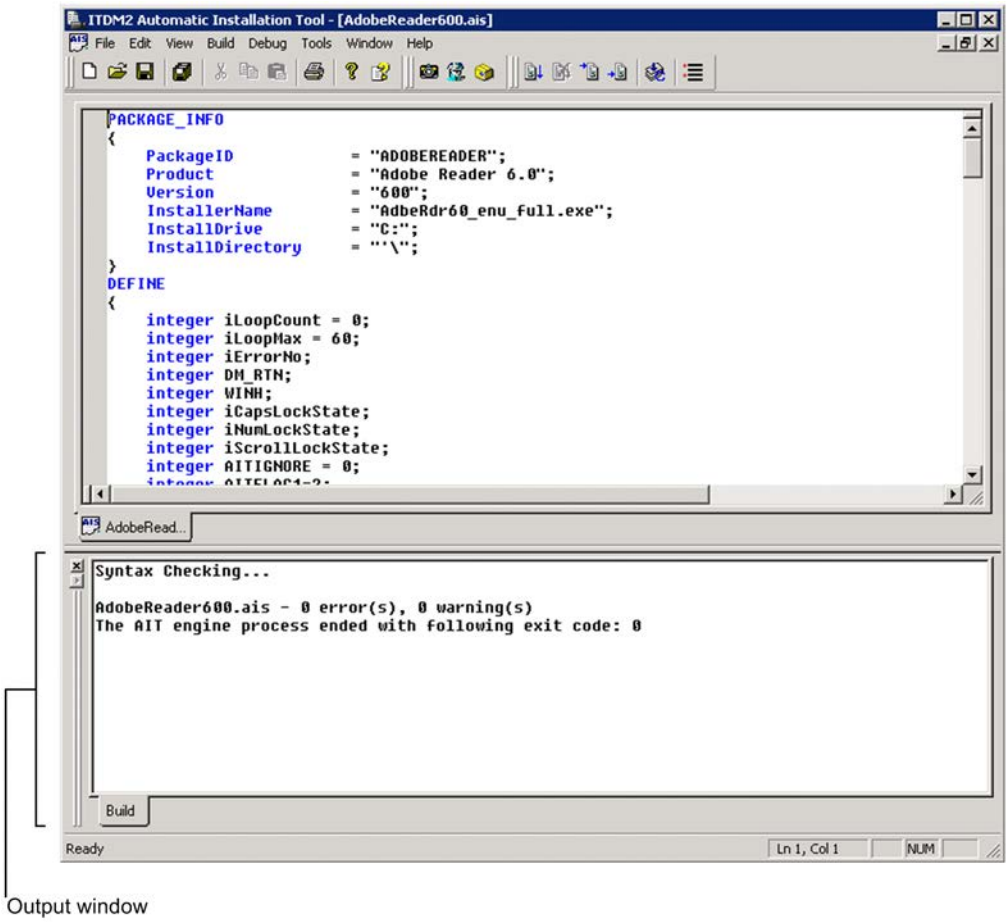
- **Syntax Check**

  This functionality lets you check the syntax of the active (currently opened) AIT file. At the bottom of the Automatic Installation Tool window, a window called the *output window* appears to display any syntax errors and warning messages.

- **Execute**

  This functionality lets you execute the active AIT file. When no syntax errors are detected in the active AIT file, execute it. After the AIT file is executed, the output window displays the exit code. If there were errors during execution, the output window displays the errors and warning messages.

Figure 2-32: Output window



You can select whether to show or hide the output window by choosing **Output** in the **View** menu.

2.7.2 Debugging

To facilitate debugging an AIT file, you can stop execution of the AIT file at specific points. When execution of an AIT file is stopped, you can reference or update the values of variables in the Watch window.

You can stop execute of an AIT file:

- at a breakpoint you set;
- before the cursor line; or
- in units of statements.

(1) Setting breakpoints

Breakpoints are the points at which the debugging process stops. You can set breakpoints in any lines, and can enable or disable them. Breakpoints are marked with circles displayed on the left of the Edit window. Colored circles indicate enabled breakpoints and white circles indicate disabled breakpoints. You can remove unnecessary breakpoints. All breakpoints you set will be cleared when you close the AIT file.

You can add, remove, enable or disable breakpoints by using the **Breakpoints Setup** or **Add/Remove Breakpoint** menu.

(a) Using the Breakpoints Setup dialog box to set breakpoints

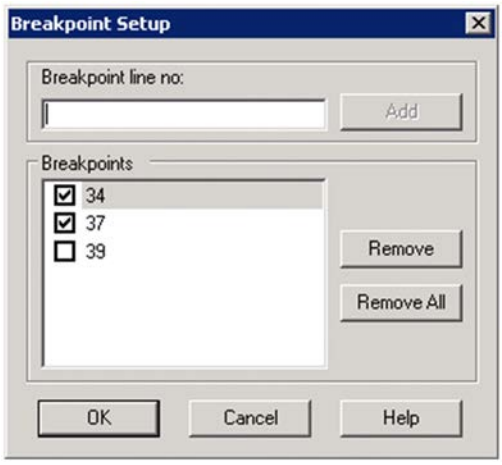
The following procedure shows how to use the Breakpoints Setup dialog box to add, remove, enable or disable breakpoints.

To use the Breakpoints Setup dialog box to set breakpoints:

1. Open the AIT file. Then, from the **Debug** menu, choose **Breakpoints Setup**.

   The Breakpoints Setup dialog box appears.

   Figure 2-33: Breakpoints Setup dialog box



Breakpoint line no.

In this text box, type in the line number of a line where you want to add a breakpoint, and click the **Add** button. A breakpoint is set on the specified line.

Breakpoints

This list box lists all the breakpoints set in the active AIT file. Breakpoints are indicated with line numbers. When the check box of a breakpoint is

selected, the breakpoint is enabled. When the check box of a breakpoint is not selected, the breakpoint is disabled.

Remove button

Clicking this button removes the breakpoints at the lines selected in the **Breakpoints** list box.

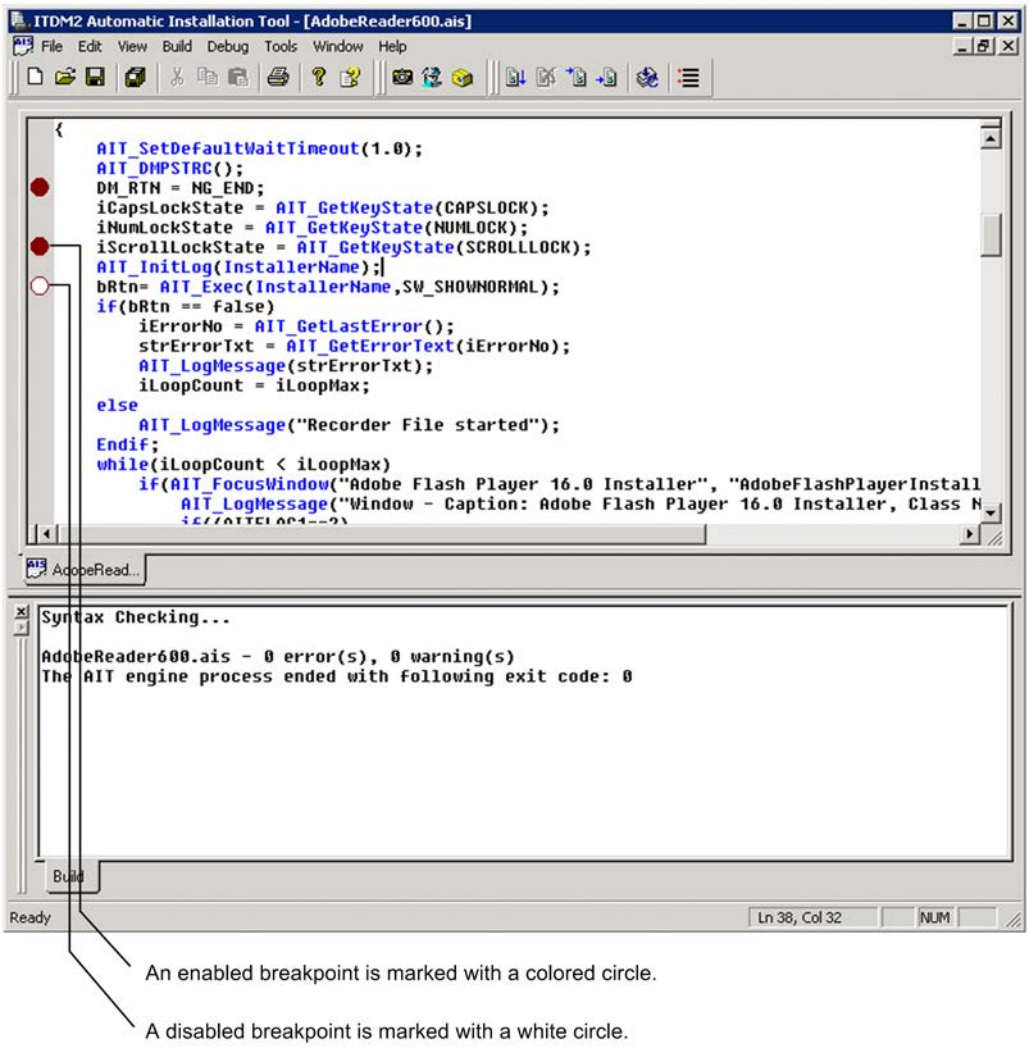
Remove All button

Clicking this button removes all breakpoints you set.

2. Set breakpoints, then click the **OK** button.

Settings in the Breakpoints Setup dialog box are applied to the Edit window. The following figure shows the Edit window.

Figure 2-34: Edit window



(b) Using Add/Remove Breakpoint to set a breakpoint

The following procedure shows how to add, remove or enable a breakpoint in the Edit window without using the Breakpoints Setup dialog box.

To use **Add/Remove Breakpoint** to set a breakpoint:

1. Open the AIT file. Then, position the cursor at a line at which you want to stop the debugging process.
2. From the **Debug** menu, choose **Add/Remove Breakpoint**.

The operation differs depending on the selected line.

- Line without a breakpoint:

  An enabled breakpoint is added.

- Line with an enabled breakpoint:

  The breakpoint is removed.

- Line with a disabled breakpoint:

  The breakpoint is enabled.

You can also press **F9** as a shortcut, instead of choosing **Add/Remove Breakpoint**.

(2) Stopping execution of the AIT file at a specific point

You can execute the AIT file from the current point and stop at a specific point.

You can stop execution of the AIT file by:

- setting a breakpoint and choosing **Go** from the **Debug** menu;
- moving the cursor to a certain position, and choosing **Run to Cursor** from the **Debug** menu; or
- choosing **Step by Step** in the **Debug** menu for execution in units of statements.

To terminate debugging, from the **Debug** menu, choose **Stop Debugging**.

(a) Stopping execution of the AIT file at breakpoints

You can stop execution of the AIT file at the breakpoints you set. If breakpoints are set on blank, comment, and other non-executable lines, the breakpoints will be moved forward to the nearest executable lines when the debugging starts.

The following procedure shows how to execute the AIT file and stop the execution at breakpoints.

To execute the AIT file, and stop the execution at breakpoints:

1. Set breakpoints at certain positions in the AIT file.

   For details on how to set breakpoints, see *(1)* above.

2. From the **Debug** menu, choose **Go**.
3. The statement is executed to the first breakpoint, with the debug cursor moved to the breakpoint line.
4. Repeat step 2 to execute the AIT file to the next breakpoint, or select another item in the **Debug** menu.

(b) Stopping execution of the AIT file before the cursor line

You can stop execution of the AIT file before the cursor line.

To stop execution of the AIT file before the cursor line:

1. In the AIT file, move the cursor to a position at which you want to stop execution of the file.
2. From the **Debug** menu, choose **Run to Cursor**.

   The statements before the cursor line are executed, with the debug cursor moved to the position specified in step 1.
3. Repeat steps 1 and 2 to execute the AIT file to the next cursor, or select another item in the **Debug** menu.

(c) Stopping execution of the script in units of statements

You can stop execution of the script in units of statements to the end of the AIT file or until the debug process is stopped by choosing **Stop Debug**.

To stop execution of the script in units of statements:

1. Open the AIT file.
2. From the **Debug** menu, choose **Step by Step**.

   The current statement line is executed, with the debug cursor moved to the next statement line.
3. Repeat step 2 to execute the current statement line, or select another item in the **Debug** menu.

(3) Monitoring and changing values of variables

During debugging of an AIT file, you can use the Watch window to monitor the values of the specified variables. When execution of the AIT file stops at a specific position, the Watch window displays the current values of the variables. You can also use the Watch window to change variable values.

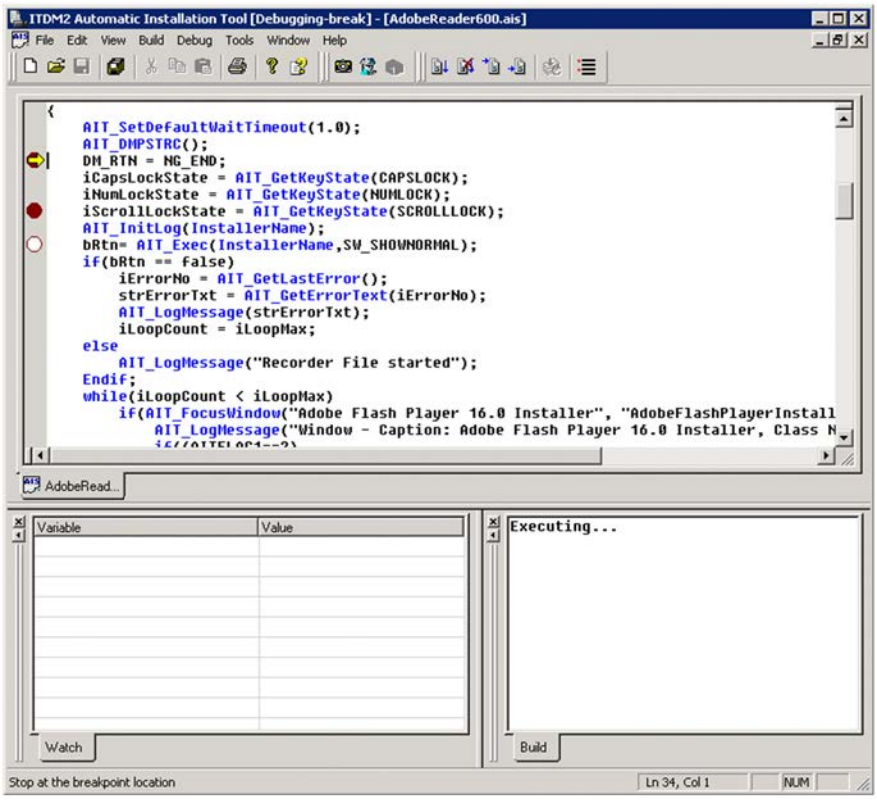
The following procedure shows how to monitor the specified variables in the Watch window.

To monitor variables in the Watch window:

1. During debugging of the AIT file, choose **Display** and then **Watch**.

   The Watch window appears.

Figure 2-35: Watch window



2. In the **Variable** column, specify a variable name, and press **Enter**.

   You can also enter a variable name by dragging and dropping it from the Edit window to the Watch window.

   The Watch window displays the values of the variables you entered. Numeric data displayed in decimal by default. This data can also be displayed in hexadecimal if you select **Hexadecimal** in the menu appearing by right-clicking. String data is displayed as string constants.

   If you specify a variable name not defined in the AIT file, an error message appears.

3. Execute the AIT file to a specific position.

   The current values of the variables are displayed in the Watch window.

4. If you want to change the value of a variable, in the Watch window, enter a new value, and press **Enter**.

The new value is applied to the script in the AIT file.

You can remove the name of a variable from the Watch window if you do not want to monitor it any longer. To remove a variable name, in the Watch window, select the variable name, and press **Delete**.

Chapter

3. AIT Language Reference

This chapter describes the format of AIT files and the syntax of the AIT language.

3.1 Format of AIT files
3.2 Sections
3.3 Data types
3.4 Operators
3.5 Variables and constants
3.6 Program flow control
3.7 Function calls
3.8 Keywords
3.9 Macros

3.1 Format of AIT files

Each AIT file consists of the following four sections:

- PACKAGE_INFO
- DEFINE
- MAIN
- ERROR

All of these sections are required. You cannot change the order of the sections.

The following shows the format of AIT files. In the AIT language, alphabetic letters are not case sensitive. You can add comments in the script.

Format of AIT files

```
PACKAGE_INFO
{
   // Package information
}

DEFINE
{
   // Defines and initializes variables and constants
}

MAIN
{
   // Operations on windows
}

ERROR
{
   // Error handling
}
```

3.2 Sections

This section explains the sections in an AIT file.

3.2.1 PACKAGE_INFO

The PACKAGE_INFO section contains the package information on the software you want to distribute, and information necessary to set up the software.

The following explains the items you can specify in this section.

(1) Format

You can specify only the following items:

```
PackageID = "package-ID";
Product  =  "product";
Version = "version";
InstallerName = "installer-name";
InstallDrive = "installation-drive";
InstallDirectory = "installation-directory";
IconGroupName = "icon-group-name";
SerialNumber = "serial-number";
JUser = "user-name";
JCompany = "company-name";
ScriptFileVersion = "AIT-file-version";
```

(2) Items to be described

The following table describes the items you can specify in the PACKAGE_INFO section. You cannot use such characters having special meanings as `\n`, `\r`, and `\t`.

Item	Description	Type
`PackageID`	Specify the package ID with a value from 1 to 44 bytes. You can use upper-case letters, numbers, hyphens (`-`), and underscores (_).	Required
`Product`	Specify the package name with a value from 1 to 50 bytes. You cannot use \.	Required
`Version`	Specify the version or revision of the software with a value from 1 to 6 bytes. You can use upper-case letters, numbers, and a slash (`/`).	Required
`InstallerName`	Specify the name of the installer to be used to install the software with a value from 1 to 256 bytes. You cannot use the following characters: `* " : | < > ?`	Required
`InstallDrive`	Specify the drive in which to install the software by using two characters: an alphanumeric character and a colon (`:`)	Required

Item	Description	Type
`InstallDirectory`	Specify the path name of the directory in which to install the software. Specify the path name with a value from 1 to 128 bytes. The name must start with a backslash (\).	Required
`IconGroupName`	Specify the icon group of the software with a value from 1 to 40 bytes.	Optional
`SerialNumber`	Specify the serial number of the software to be installed with a value from 1 to 64 bytes. For software which requires the CD key for its installation, enter the CD key.	Optional
`JUser`	Specify a value from 1 to 40 bytes for the software owner name.	Optional
`JCompany`	Specify a value from 1 to 80 bytes for the company name of the software owner.	Optional
`ScriptFileVersion`	Specify the AIT file version in the *n*`.`*n*`.`*n*`.`*n* format using numeric characters. You cannot specify a series of dots (`.`). You must specify all four *n* values delimited by a dot (`.`). If the fourth *n* is not specified, it is assumed to be 0. For example, `1.0.0.` is assumed to be `1.0.0.0`. This information prevents the Automatic Installation Tool from executing an AIT file created by an old engine of an earlier version of the Automatic Installation Tool. If the version of the AIT file is not specified in the PACKAGE_INFO section, the DLL version of the active engine is acquired as the AIT file version.	Optional

(3) Example of coding

```
PACKAGE_INFO
{
   PackageID = "D";
   Version = "1";
   Product =  "product";
   InstallerName = "installer-name";
   InstallDrive = "D:";
   InstallDirectory = "\Plan14.1";
   JUser = "package-user";
   JCompany = "package-user-company";
   SerialNumber = "package-serial-number";
   IconGroupName= "icon-group-name";
   ScriptFileVersion = "1.0.0.0";
}
```

3.2.2 DEFINE

The DEFINE section defines all variables and constants used in the AIT file. This section can contain only the codes for declaring and initializing variables and constants.

(1) Example of coding

```
DEFINE
{
    const integer OK_END = 0, NG_END = -1 ;
    string sMsgText;
    float TimeOut;
    bool sInvalidPathFlag = false;
}
```

(2) Notes

- You cannot redefine variables.
- In the AIT file, if you use a variable not declared in the DEFINE section, the output window displays a warning message upon a syntax check.

3.2.3 MAIN

The MAIN section contains the codes for installing software automatically. In this section, you can use all statements other than data type declarations.

You can also code this section to notify the agent of the return code for the result of installation.

(1) Example of coding

```
MAIN
{
    AIT_SetDefaultWaitTimeout(1.0);
    AIT_DMPSTRC();
    DM_RTN = NG_END;
    AIT_InitLog(InstallerName);
    bRtn= AIT_Exec(InstallerName,SW_SHOWNORMAL);
    if(bRtn == false)
        iErrorNo = AIT_GetLastError();
        strErrorTxt = AIT_GetErrorText(iErrorNo);
        AIT_LogMessage(strErrorTxt);
        iLoopCount = iLoopMax;
    else
        AIT_LogMessage("Recorder File started");
    Endif;
    while(iLoopCount < iLoopMax)
        if(AIT_FocusWindow("check", "#32770") > 0)
          AIT_LogMessage("Window - Caption: check, Class Name:
#32770");
            AIT_CtrlSetFocus("check box(&C)",CHECKBOX_CTRL);
            AIT_Sleep(SLEEP_TIME_EVENTS);
            AIT_LogMessage("AIT_CtrlSetFocus('"check
box(&C)'",CHECKBOX_CTRL);");
            AIT_PlayKey("{ENTER}");
            AIT_Sleep(SLEEP_TIME_EVENTS);
```

```
                AIT_LogMessage("AIT_PlayKey('"{ENTER}'");");
                iLoopCount = iLoopMax;
                DM_RTN = OK_END;
                continue;
            Endif;
            iLoopCount = iLoopCount + 1;
            AIT_Sleep(SLEEP_TIME);
        loop;
        if(DM_RTN == OK_END)
            AIT_LogMessage("Recorder File ended normally");
        else
            AIT_LogMessage("Recorder File ended Abnormally");
        Endif;
        WINH = AIT_RegisterWindowMessage("ITM_REC_QUIT");
        if(WINH != 0)
            AIT_PostMessage(HWND_BROADCAST,WINH,DM_RTN,0);
        Endif;
}
```

3.2.4 ERROR

The ERROR section contains the statements to be executed if the installation abnormally terminates. You can code this section to notify the agent of the return code for the result of installation.

If an error occurs during execution of an AIT file, control moves to this section.

(1) Example of coding

```
ERROR
{
   WINH = AIT_RegisterWindowMessage("ITM_REC_QUIT");
     if  (WINH != 0)
       AIT_PostMessage (HWND_BROADCAST,WINH,-1,0);
     endif;
}
```

3.3 Data types

This section explains the data types that can be used in AIT files. This system supports the basic data types shown below.

3.3.1 integer

The `integer`-type is a basic data type that contains a numeric character in the range from -2,147,483,648 to +2,147,483,647. Data of this type cannot include any decimals and exponents.

The keyword `integer` allows you to declare `integer`-type variables and initialize variables and constants. You can use this keyword only in the DEFINE section. When you specify two or more variables after this keyword, use a comma (`,`) to delimit them.

You can declare a constant by using the keyword `const` in combination with the keyword `integer`. The values of the constants declared in the DEFINE section can only be referenced, and cannot be changed in the MAIN or ERROR section.

(1) Format

```
DEFINE
{
   [const] integer variable_name1 [= integer_constant1] [,
variable_name2 [= integer_constant2] ];
}
```

(2) Example of coding

```
DEFINE
{
    const integer OK_END = 0, NG_END = -1 ; // Enabled
    integer end_status, return_code;        // Initialized at 0
}
```

(3) Notes

- When you initialize variables or constants, you can specify only decimal values.
- An `integer`-type variable can be assigned only a value of type `integer`, `bool`, or `float`. However, if a value of type `float` is assigned, because the value may become inaccurate, a warning message appears in the output window upon a syntax check.
- All variables are initialized at 0 by default. In the example shown in (2) above, `end_status` and `return_code` are initialized at 0.

3.3.2 float

The `float` type is a basic data type that can indicate a 32-bit floating-point decimal

number. A `float`-type variable or constant can have an absolute value from +3.40282347e+38 to +1.175494351e-38.

The keyword `float` allows you to declare and initialize a `float`-type variable or constant. You can use this keyword only in the DEFINE section. When you specify two or more variables after this keyword, use a comma (`,`) to delimit them.

You can declare a constant by using the keyword `const` in combination with the keyword `float`. The values of the constants declared in the DEFINE section can only be referenced, and cannot be changed in the MAIN or ERROR section.

(1) Format

```
DEFINE
{
    [const] float variable_name1 [= float_constant1] [,
variable_name2 [= float_constant2] ];
}
```

Floating-point values can include a decimal point or an exponential notation (using `E` or `e`). In the exponential part, you can specify `E` or `e` followed by an integer. A sign (`+` or `-`) can be added to the integer. A constant of type `float` must consist of one or more digits, and must have a decimal point or exponent.

(2) Example of coding

```
DEFINE
{
    float DefaultTimeOut = 0.01, DefaultSleep=5.0 ; // Valid
    float SleepMax;                                 // Valid
}
```

(3) Notes

- A constant of type `float` can be specified only with a decimal number.
- A `float`-type variable can be assigned only a value of type `bool`, `float`, or `integer`.
- A `float`-type variable is initialized at 0 by default. In the example shown in (2) above, `SleepMax` is initialized at 0.
- The number of significant digits for data of type `float` is 11 (including a decimal point).

3.3.3 bool

The `bool`-type data has the value `true` or `false`. There are the following relationships between the values `true` and `false`:

- The `!false` value has the same meaning as the `true` value.
- The `!true` value has the same meaning as the `false` value.

The keyword `bool` allows you to declare and initialize a `bool`-type variable or constant. You can use this keyword only in the DEFINE section. When you specify two or more variables after this keyword, use a comma (`,`) to delimit them.

You can declare a constant by using the keyword `const` in combination with the keyword `bool`. The values of the constants declared in the DEFINE section can only be referenced, and cannot be changed in the MAIN or ERROR section.

If a comparison expression evaluates to 0, the result is assumed to be `true`. If a comparison expression evaluates to a non-0 value, the result is assumed to be `false`.

(1) Format

```
DEFINE
{
   [const] bool variable_name1 [= true|false] [, variable_name2
[= true|false] ] ;
}
```

(2) Example of coding

```
DEFINE
{
    bool sInvalidPathFlag, sDirectorySetFlag = true;
    bool SMemoryInsuff = false;
    bool sEndGUI = false;
}
```

(3) Notes

- A `bool`-type variable can be assigned a value of type `integer` or `float`. However, if data of type `integer` or `float` is assigned, because the value may become inaccurate, a warning message appears in the output window upon a syntax check.
- All variables are initialized as `false` by default. In the example shown in (2) above, `sInvalidPathFlag` is initialized as `false`.

3.3.4 string

The `string`-type is a basic data type that indicates a variable-length character string.

The keyword `string` allows you to declare and initialize a `string`-type variable or constant. You can use this keyword only in the DEFINE section. When you specify two or more variables after this keyword, use a comma (`,`) to delimit them.

You can declare a constant by using the keyword `const` in combination with the keyword `string`. The values of the constants declared in the DEFINE section can only be referenced, and cannot be changed in the MAIN or ERROR section.

To specify a character-string constant, you have to enclose the character string with double quotation marks (`"`). If you want to use a double quotation mark (`"`) in a

character string, place a single quotation mark (') immediately before the double quotation mark.

(1) Format

```
DEFINE
{
    [const] string variable_name1 [= "StringValue"] [,
variable_name2 [= integer_constant2]] ;
}
```

(2) Example of coding

```
DEFINE
{
    string CaptionName="Setup";  // Valid
    string ErrorText;            // Valid
}
```

(3) Notes

- The length of a `string`-type variable may be increased as a result of combination.
- All variables that have not been assigned any value become empty. In the example shown in (2) above, `ErrorText` is empty.
- You can write the value of a `string`-type variable over two or more lines by placing an underscore (_) at the end of the preceding lines. In the following example, the value of `string`-type variable `ErrorText` is `Sample testing success`.

  Example:
  ```
  DEFINE
  {
     string ErrorText = "Sample_
   Testing_
   success";   // string is written over multiple lines.
  }
  ```

- The following characters are handled as special characters in a character string.

Character	Handled as:
`\n`	Line feed
`\r`	Return
`\t`	Tab character
`'\`	Backslash (\)
`'"`	Double quotation mark (")

Character	Handled as:
''	Single quotation mark (')

In the following example, the character-string variable `ErrorText` is assigned `"Sample Testing"`.

Example:

```
DEFINE
{
   string ErrorText = "'"Sample Testing'"";
          // Sample Testing is enclosed in '"...'".
}
```

In the following example, the character-string variable `Path` is assigned `C:\Windows\system32`.

Example:

```
DEFINE
{
   string Path = "C:'\Windows'\system32";
          // "C:\Windows\system32" is assigned to Path.
}
```

3.4 Operators

You specify operators to evaluate:

- One operand (unary operator)
- Two operands (binary operator)

The order in which to evaluate multiple expressions including operators is defined according to strict priority. The operator is linked with either the left or right operand. This is referred to as *linking order*. The operators in the same group have the same priority. They are evaluated from the left to the right unless you explicitly change the priority using parentheses (( )).

The following tables list the operators supported by the AIT language.

Table 3-1: Unary operators supported by the AIT language

Unary operator	Description
+	Unary plus sign
-	Unary minus sign
!	Unary not sign

Table 3-2: Binary operators supported by the AIT language

Binary operator	Description
+ -	Adding operators
* / %	Multiplying operators
< <= > >= != ==	Comparison operators
& |	Bitwise operators
&& ||	Logical operators

3.4.1 Assignment

The *assignment* operation in the AIT language assigns the value of the right operand to the left operand. In assignment, you cannot use any constant as the left operand.

(1) Format

Assignment statement

```
assignment_expression END_STMT
```

Assignment expression

```
identifier assign_operand expression
```

(2) Description

In assignment, the value of the right operand is stored in the left operand. The left operand must neither be a function nor be a constant.

If the data type of the right operand differs from that of the left operand, type conversion is performed when possible. This type conversion depends on the specified operator, and the type of operator or operand.

The following table lists the results of type conversion in the AIT language. Warning messages and errors for data conversion are displayed upon syntax checks.

Left operand	Right operand	Result	Description
`integer`	`integer`	`integer`	The value of one `integer`-type variable is assigned to the other `integer`-type variable.
`integer`	`float`	`integer`	The truncated value is assigned to the `integer`-type variable. A warning message is displayed because the data may become inaccurate.
`integer`	`bool`	`integer`	The value `true` is assigned to the `integer`-type variable as `1`. The value `false` is assigned to the `integer`-type variable as `0`.
`integer`	`string`	Error	Since the `string` type cannot be converted to the `integer` type, an error message is displayed.
`float`	`integer`	`float`	The value of the `integer`-type variable is assigned to the `float`-type variable.
`float`	`float`	`float`	The value of one `float`-type variable is assigned to the other `float`-type variable.
`float`	`bool`	`float`	The value `true` is assigned to the `float`-type variable as `1`. The value `false` is assigned to the `float`-type variable as `0`.
`float`	`string`	Error	Since the `string` type cannot be converted to the `float` type, an error message is displayed.
`bool`	`integer`	`bool`	A non-0 value is assigned to the `bool`-type variable as `true`. 0 is assigned to the `bool`-type variable as `false`. Since the original data is lost, a warning message is displayed.
`bool`	`float`	`bool`	
`bool`	`bool`	`bool`	The value of one `bool`-type variable is assigned to the other `bool`-type variable.
`bool`	`string`	Error	Since the `string` type cannot be converted to the `bool` type, an error message is displayed.

Left operand	Right operand	Result	Description
`string`	`integer`	Error	Since the `integer` type cannot be converted to the `string` type, an error message is displayed.
`string`	`float`	Error	Since the `float` type cannot be converted to the `string` type, an error message is displayed.
`string`	`bool`	Error	Since the `bool` type cannot be converted to the `string` type, an error message is displayed.
`string`	`string`	`string`	The value of one `string`-type variable is assigned to the other `string`-type variable

The AIT language supports multi-assignment. The multi-assignment here means assignment of a value to two variables.

(3) Example of coding

```
MAIN
    {
        a = 10;          // Value 10 is assigned to a.
        a = b = 20;      // Multi-assignment is coded.
                         // Value 20 is assigned to variable b,
                         // then to variable a.
    }
```

3.4.2 Unary plus

The *unary plus* is a unary operator that returns the positive value of the result of the expression.

(1) Format

`+[(]`*expression*`[)]`

(2) Description

The operand for the unary plus operator (+) must be arithmetic type. The unary operator is placed before an operand, linking from the right to the left.

Specifying the positive unary operator for a negative value will return a negative value. Specifying the negative operator for a negative value will return a positive value.

(3) Example of coding

```
const integer NG_END = +1;          // Statement in the DEFINE
section
sloopcnt = +(sloopmin - sloopmax); // Statement in the MAIN
section
```

3.4.3 Unary minus

The *unary minus* is a unary operator that returns the negative value of the result of the expression.

(1) Format

`-[(]`*expression*`[)]`

(2) Description

The operand for the unary minus operator (`-`) must be arithmetic type. The unary operator is placed before an operand, linking from the right to the left.

Specifying the negative unary operator for a negative value will return a positive value. Specifying the positive unary operator for a negative value will return a negative value.

(3) Example of coding

```
const integer NG_END = -1;          //Statement in the DEFINE
section
integer sloopcnt, sloopmax, sloopmin;
sloopcnt = -(sloopmax - sloopmin);   //Statement in the MAIN
section
```

3.4.4 Unary not

The *unary not* is a unary operator that logically negates the result of the expression.

(1) Format

`!(`*expression*`)`

(2) Description

If the negation of an operand (for example, when an operand is `false`) is `true`, the value `true` is returned. On the contrary, if the negation of an operand (for example, when an operand is `true`) is `false`, the value `false` is returned.

The unary operator is placed before an operand, linking from the right to the left.

(3) Example of coding

```
bool IsLastDialog;
IsLastDialog = false;
if (!IsLastDialog)
   AIT_LogMessage("Installable Software Extracting... is
opened.");
else
   AIT_LogMessage("Installable Software Extracting... is not
opened.");
endif;
```

3.4.5 Adding operators

Adding operators perform addition (+) or subtraction (-).

(1) Format

Additive expression
expression + *expression*

Subtractive expression
expression - *expression*

(2) Description

The adding operator performs normal arithmetic conversion for `integer`-type and `float`-type operands, according to the data type.

- The adding operator (+) adds two operands. If the two operands are of the `string` type, the two character strings are combined. If only one of the two operands is the `string` type, this results in an error.
- The subtractive operator (-) subtracts the second operand from the first operand. Both of the operands must have numeric values. If one or both of the operands are the `string` type, this results in an error.

(3) Example of coding

```
MAIN
{
          ...
          sloop_cnt = sloop_cnt+1;
          sloop_cnt = sloop_cnt-1;
          ...
          ...
}
```

(4) Notes

The conversion processing provided by adding operators does not handle overflow and underflow. If the result of conversion by an adding operator cannot be represented with the data type of the operand, information may be lost.

3.4.6 Multiplying operators

Multiplying operators perform multiplication (*), division (/), and remainder calculation (%).

(1) Format

Multiplicative expression
expression * *expression*

Divisional expression
expression `/` *expression*

Remainder calculation expression
expression `%` *expression*

(2) Description

Multiplying operators perform normal arithmetic conversion for operands, according to the data type. Both of the operands must have numeric values. If one or both of the operands are of the `string` type, this results in an error.

- The multiplication operator (`*`) multiplies two operands. The operands may be `integer` or `float` type.
- The division operator (`/`) divides the first operand by the second operand. The operands may be `integer` or `float` type. The data types of both operands may be different. While the result of division by 0 is undefined, the error message is displayed upon a syntax check or during runtime. If both of the operands are positive or unsigned, the result is rounded off to the nearest integer.
- The operands subject to the remainder calculation operator (%) must have integers. The result of the calculation provides the remainder with the first operand divided by the second operand. If the calculation is indivisible, the result is determined according to the following rules:
  - If the right operand is 0, the result is undefined.
  - If both of the operands are positive or unsigned, the result is positive.

(3) Example of coding

```
MAIN
{
if (sloop_cnt > 10)
   AIT_Sleep(SLEEP_TIME / 2);
else
   AIT_Sleep(SLEEP_TIME * 2);
endif;
}
```

(4) Notes

- The conversion provided by multiplying operators does not handle overflow and underflow. If the result of conversion by a multiplying operator cannot be represented with the data type of the operand, information may be lost.
- If the operand is divided by 0 during runtime, control moves to the ERROR section.

3.4.7 Comparison operators

The binary comparison operator compares the first operand with the second operand to verify that the specified relationship is valid. If the comparison expression evaluates to `true`, `1` is returned. If the comparison expression evaluates to `false`, `0` is returned. The result is the `bool` type.

(1) Format

Comparison expression

expression `<` *expression*
expression `>` *expression*
expression `<=` *expression*
expression `>=` *expression*

Equality

expression `==` *expression*

Inequality

expression `!=` *expression*

(2) Description

The following gives the relationships checked by the comparison operator.

Operator	Relation to be checked
`<`	The first operand is smaller than the second operand.
`>`	The first operand is greater than the second operand.
`<=`	The first operand is equal to or smaller than the second operand.
`>=`	The first operand is equal to or greater than the second operand.
`==`	The first operand is equal to the second operand.
`!=`	The first operand is not equal to the second operand.

You can specify an operand of type `integer`, `float`, or `string`. You may specify operands of different types. Comparison operators perform normal arithmetic conversion for operands of `integer` and `float` types. You can also use a combination of some operand types, and the comparison or equal operator.

The following table shows how the comparison operators evaluate the comparison results as `true` or `false`.

Operator	Meaning	True	False
`<`	Smaller than	*expression-1* < *expression-2*	*expression-1* >= *expression-2*

Operator	Meaning	True	False
>	Greater than	*expression-1* > *expression-2*	*expression-1* <= *expression-2*
<=	Equal to or smaller than	*expression-1* <= *expression-2*	*expression-1* > *expression-2*
>=	Equal to or greater than	*expression-1* >= *expression-2*	*expression-1* < *expression-2*
==	Equal to	*expression-1* == *expression-2*	*expression-1* != *expression-2*
!=	Not equal to	*expression-1* != *expression-2*	*expression-1* == *expression-2*

The following table gives the results of comparison expressions depending on the data types of expressions.

Data types of expressions	Operation
Both expressions are numeric.	Compares the numeric values.
Both expressions are `string` type.	Compares the character strings.
One expression is numeric and the other expression is `string` type.	Results in an error.

(3) Example of coding

```
if (sloop_cnt < (sloop_max - 25))  // <
    AIT_LogMessage("Searching for Active windows");  //Search
Active windows
    if (AIT_FocusWindow("Installable Software Extracting...",
"#32770",0.0) > 0)   // >
        AIT_LogMessage("Installable Software Extracting... is
opened");
        sloop_cnt= 0;
    endif;
endif;
```

3.4.8 Bitwise operators

Bitwise operators perform bitwise AND (&) and OR (|) operations.

(1) Format

Bitwise AND

expression & *expression*

Bitwise OR

expression | *expression*

(2) Description

Either or both of the operands for a bitwise operator must be the `integer` type.

Bitwise operators perform normal arithmetic conversion according to the data type.

The following explains the bitwise operators.

Operator	Description
&	The bitwise AND operator compares a bit of the first operand with the corresponding bit of the second operand. If both the bits are 1, the resulting bit is set at 1. If not, the resulting bit is set at 0.
|	The bitwise OR operator compares a bit of the first operand with the corresponding bit of the second operand. If one of both the bits is 1, the resulting bit is set at 1. If not, the resulting bit is set at 0.

- The bitwise AND operator compares two expressions bit by bit, and sets the resulting bit as follows:

Bits of expression 1	Bits of expression 2	Result
0	0	0
0	1	0
1	0	0
1	1	1

- The bitwise OR operator compares two expressions bit by bit, and sets the resulting bit as follows:

Bits of expression 1	Bits of expression 2	Result
0	0	0
0	1	1
1	0	1
1	1	1

3.4.9 Logical operators

The logical operator provides logical AND && and OR (||).

(1) Format

Logical AND operation

expression-1 && *expression-2*

Logical OR operation

expression-1 || *expression-2*

(2) Description

Logical operators do not perform normal arithmetic conversion. Instead, it evaluates the operands in terms of whether they match 0. The result of a logical operation is `true` or `false`. The result is the `bool` type.

The following explains the logical operators.

Operator	Description
`&&`	If both the operands are `true`, the result is `true`. If one of the operands is `false`, the result is `false`. If the first operand for logical AND is `false`, the second operand is not evaluated.
`||`	If both the operands are `false`, the result is `false`. If one of the operands is `true`, the result is `true`. If the first operand for logical OR is `true`, the second operand is not evaluated.

The operands for logical AND and OR expressions are evaluated from the left to the right. If the operational result can be identified only by the value for the first operand, the second operand is not evaluated. This is referred to as *quick evaluation*.

(a) Logical AND operator (&&)

If both the expressions evaluate to `true`, the result is `true`. If one of the expressions is evaluates to `false`, the result is `false`. The following table shows how the results are evaluated.

Evaluation of expression 1	Evaluation of expression 2	Resulting evaluation
true	true	true
true	false	false
false	true	false
false	false	false

(b) Logical OR operator (||)

If one or both of the expressions evaluate to `true`, the result is `true`. The following table shows how the results are evaluated.

Evaluation of expression 1	Evaluation of expression 2	Resulting evaluation
true	true	true
true	false	true
false	true	true
false	false	false

(3) Example of coding

```
DEFINE
{
   float varfloat1 = 1.567e-1;
   integer varint1 = 10;
   integer varfloat2 = 0;
   integer varint2 = 0;
   bool varbool;
   integer WINH;
}
MAIN
{
   varbool = varfloat1 && varint1;
   AIT_LogMessage("The expected value of varbool is: true");
   varbool =varfloat2 && varint1;
   AIT_LogMessage("The expected value of varbool is: false");
   varbool=varfloat1 && varint2;
   AIT_LogMessage("The expected value of varbool is: false");
   varbool=varfloat2 && varint2;
   AIT_LogMessage("The expected value of varbool is: false");
}
```

3.4.10 Priority of operators

The priority and linking order of operators affect operand grouping and evaluation for expressions. The priority of operators is meaningful only when there are operators with different priorities in the same expression. If the expression contains a higher-priority operator, the operator is evaluated first. If operators have the same priority, the order in which to evaluate them depends on their linking order.

The following table shows the priority and linking order of the operators. The following arrangement shows the operators in descending priority order.

Symbol	Operator	Linked:
`( )`	Parentheses	From the left to the right
`+ - !`	Unary operators	From the right to the left
`* / %`	Multiplying operators	From the left to the right
`+ -`	Adding operators	From the left to the right
`< > <= >=`	Comparison operators	From the left to the right
`== !=`	Equality	From the left to the right
`&`	Bitwise AND	From the left to the right
`|`	Bitwise OR	From the left to the right

Symbol	Operator	Linked:
&&	Logical AND	From the left to the right
||	Logical OR	From the left to the right
=	Assignment	From the left to the right

3.5 Variables and constants

Variables and constants are typical data. The values of variables may be changed, and the values of constants cannot be changed.

To use variables or constants, you have to declare them only once in the DEFINE section in the AIT file. The name of each variable or constant can have a maximum of 64 characters and must begin with any alphabetic character (A-Z). You cannot use special characters other than an underscore (_). The alphabetic characters are not case sensitive.

You should use easy-to-understand variable and constant names that indicate the roles or meanings of them. You must not use any keywords, label names, and macro names as variable names.

3.5.1 Example of variables

```
integer LoopCount;
string CountryName;
bool answer;
float r_nTimeOut;
integer ABC;
integer abc;     //Disabled as the variable is redefined.
```

In the above example, `integer` means that the listed variable is the `integer` type. In other words, `LoopCount` is an integer, while `CountryName` is a character string.

3.5.2 Example of constants

```
const integer OK_END = 0;       //DM_RTN:OK_END
const integer NG_END = -1;      //DM_RTN:NG_END
const integer SET_SLEEP_TIME = 2;
```

3.6 Program flow control

Normally, statements are executed in sequence. However, you can move control to another statement by using the statements explained in this section.

3.6.1 goto

The goto statement provides an unconditional jump to a valid label position declared in the same section.

(1) Format

Statements
```
    Jump statement
    Label statement
```

Jump statement
```
    goto identifier;
```

Label statement
```
    identifier:
```

(2) Description

The `goto` statement is used to exit deeply nested loops directly. Unlike this statement, the `break` statement can exit only one nesting level of repetitive statements.

(3) Example of coding

```
//MAIN section
MAIN
{
   ...
   ...
   AIT_LogMessage("LBL030: JUMP TO LABEL");
   goto label1;
       AIT_LogMessage("LBL030: NOT DISPLAYED 1");       // Not
executed

       label1:                       // The control shifts here.
          AIT_LogMessage("LBL030: JUMPING TO LABEL ");  //
Executed
}
```

(4) Notes

For better programming, you should use the `break`, `continue`, or `return` statement rather than the `goto` statement wherever possible.

3.6.2 Label

To use the `goto` statement to directly jump to any other statement, you have to add a label at the destination statement.

A label is specified in the *identifier*`:` format. You may declare a label at any position in the MAIN and ERROR sections. You must specify a unique label name according to the same rules as for variables and constants. You must not reuse any variables and constants as labels.

The label has no meaning in any sections other than MAIN and ERROR. It only has meaning if it is associated with a `goto` statement. If a label is not associated with any `goto` statement, the program does not interpret the label but outputs a warning message.

(1) Format

Label statement
identifier`:`

(2) Example of coding

```
AIT_MessageBox("ss","xx");
goto label18;
   AIT_LogMessage("LBL040: NOT DISPLAYED 29");
                  // Not executed
   label18:       // The control shifts here.
   AIT_LogMessage("LBL040: INSIDE LABELLED STATEMENT");
                  // Executed
   AIT_Exit();
```

3.6.3 if-else-endif

The `if` statement allows a conditional branch to be processed.

If the condition evaluates to a value other than 0, the main part of the `if` statement is executed. If it is 0, the sections other than the main part are executed.

In this statement, the sections other than the main part are options. You have to enclose a condition by parentheses.

(1) Format

```
if (condition)
    [expression-1;]
[else
    [expression-2;]]
endif;
```

(2) Description

Conditions are subject to evaluation. If the condition is not 0, *expression-1* in (1) is

executed and, if it is 0, *expression-2* is executed.

In the `if-else-endif` statement, the `else` clause is associated with the immediately preceding `if` statement which does not have the associated `else` statement.

(3) Example of coding

```
if(AIT_FocusWindow("Installable Software-Setup","#32770"))
                    // The following two lines are executed
                    // if the condition is true (other than 0).
   AIT_LogMessage("INSIDE Installable Software SETUP");
   AIT_PlayKey("ENTER");
else
                    // The following two lines are executed
                    // if the condition is false (0).
   AIT_LogMessage("PROBLEM IN SETUP");
   AIT_Exit();
endif;
```

3.6.4 while-loop

The `while-loop` statement executes the specified expressions repeatedly until the specified condition becomes false. You have to enclose a condition to be specified with parentheses.

(1) Format

```
while (condition)
     expression-1;
     expression-2;
loop;
```

(2) Description

1. Conditions are evaluated.
2. If the condition first becomes false, the main part of the `while` statement is never executed, and the control moves from the `while` statement to the next statement in the same program.
3. If the condition is true (not 0), the main part of the statement is executed, with all expressions executed repeatedly from *expression-1*.
4. When the system encounters a `break` statement specified in the main part of that statement, the loop ends.
5. When the system encounters a `continue` statement specified in the main part of that statement, the subsequent steps are skipped, with the condition evaluated. If the condition is true, the execution is repeated.

You should not nest more than 255 `while` loops.

(3) Example of coding

```
DEFINE
{
integer WINH,count,length;
float SLEEP_TIME=0.5;
string s1,s2;
integer i,sloop_cnt   = 0;
integer sloop_max = 30;
}
...
...
while ( sloop_cnt < sloop_max)
   AIT_LogMessage("The active window is being found");
   if (AIT_FocusWindow("Unpacking...", "#32770",0.0) > 0)
   if(AIT_FocusWindow("Unpacking software to be installed...",
"#32770", 0.0) > 0)
         AIT_LogMessage("Unpacking software to be installed...
is opened");
      sloop_cnt= 0;
      AIT_Sleep(SLEEP_TIME);
   endif;
   endif;
   AIT_Sleep(SLEEP_TIME);
   sloop_cnt = sloop_cnt + 1;
loop;
```

3.6.5 do-while

The `do-while` statement executes the specified expressions repeatedly until the specified end condition is evaluated as false. You have to enclose a condition to be specified with parentheses.

(1) Format

```
do
   expression-1
   expression-2
   ...
   ...
while (condition);
```

(2) Description

1. The main part of the `do-while` statement is executed.
2. Next, the condition is evaluated. If the condition is false, the `do-while` statement ends, and the control moves to the next statement in the same program. If the condition is true (not 0), the expressions are executed repeatedly from *expression-1*.

3. When the system encounters a `break` statement specified in the main part of that statement, the loop ends.
4. When the system encounters a `continue` statement specified in that statement, the subsequent steps are skipped, with the condition evaluated. If the condition is true, the execution is repeated.
5. In other words, the main part of the loop is executed at least once.

You should not specify more than 255 `break` or `continue` statements in a loop. Moreover, you should not nest more than 255 `do-while` statements.

(3) Example of coding

```
DEFINE
{
integer WINH,count,length;
float SLEEP_TIME=0.5;
string s1,s2;
integer i,sloop_cnt   = 0;
integer sloop_max = 30;
}
...
...
do
   AIT_LogMessage("Searching for Active windows");
   if (AIT_FocusWindow("Installable Software", "#32770",0.0) >
0)
        if(AIT_FocusWindow("Unpacking Installable Software...",
"#32770", 0.0) > 0)
           AIT_LogMessage("Unpacking Installable Software... is
opened");
             sloop_cnt= 0;
             AIT_Sleep(SLEEP_TIME);
         endif;
   endif;
   AIT_Sleep(SLEEP_TIME);
   sloop_cnt = sloop_cnt + 1;
while ( sloop_cnt < sloop_max);
...
...
```

3.6.6 for-next

The for-next statement executes the expression repeatedly until the condition becomes false. The optional expression supported in the for-next statement allows you to initialize or change a value during for-next statement execution.

Typically, the number of times a loop is repeated depends on the counter.

(1) Format

```
for ( [initialization-expression] ; [condition-expression] ; [loop-expression]
)
      expression-1;
      expression-2;
next;
```

(2) Description

1. When an initialization expression is specified, it is evaluated. It specifies loop initialization. The initialization expression may be any type.
2. A specified condition expression is evaluated before repetition, with three possible results:
   - If the condition expression is true (not 0), the statement is executed. A specified loop expression is evaluated next. According to the evaluation, execution is repeated.
   - With no condition expression specified, the condition expression is interpreted as true, and executed in the same way as above. With a condition expression not specified by a parameter, the `for` statement ends when the `goto` statement (associated with a labeled statement outside the main part of that statement) or the `break` statement is executed inside that statement.
   - If the condition expression is false (0), the `for-next` statement ends, and the control moves to the next statement in the same program.
3. When the system encounters a `break` statement specified in the main part of that statement, the loop stops.
4. When the system encounters a `continue` statement specified in the main part of that statement, the subsequent steps are skipped, with the condition expression evaluated. If the condition is true, the execution is repeated.

You should not nest more than 255 `for-next` structures.

(3) Example of coding

```
DEFINE
{
integer WINH,count,length;
float SLEEP_TIME=0.5;
string s1,s2;
integer i,sloop_cnt   = 0;
integer sloop_max = 30;
}
...
...
sloop_cnt=1;
AIT_LogMessage("Searching for Active windows - For");
```

```
for(; sloop_cnt < sloop_max ;sloop_cnt = sloop_cnt + 1)
    if (AIT_FocusWindow("Unpacking", "#32770",0.0) > 0)
       if(AIT_FocusWindow("Unpacking Installable Software...",
"#32770", 0.0) > 0)
          AIT_LogMessage("Unpacking Installable Software... is
opened");
           sloop_cnt= 0;
           AIT_Sleep(SLEEP_TIME);
        endif;
        endif;
   AIT_Sleep(SLEEP_TIME);
next;
```

3.6.7 continue

The `continue` statement moves the control to the starting position of the nearest `do`, `for`, or `while` loop that encloses this statement. Typically, the `continue` statement is used to return the control to the loop starting position from a deep nesting level.

(1) Format

Jump statement

```
continue;
```

(2) Description

The following shows the rules to determine where the next repetition starts for the `do`, `for`, or `while` statement when the `continue` statement is encountered.

- The next repetition is started by reevaluation of expressions in the `do` or `while` statement inside it.
- When the `continue` statement is specified in the `for` statement, the condition expression in the `for` statement is reevaluated. The result of reevaluation determines whether to exit or repeat the main part of the statement.

You should not specify more than 255 `continue` statements in a loop structure.

(3) Example of coding

```
for (count=1;count<=10;count=count+1)
    if (count < 5)
           continue;
    endif;
    ...
next;
```

In the above example, the codes following the `continue` statement are skipped until the value of `count` reaches 5.

3.6.8 break

The `break` statement ends execution of the nearest `do-while`, `for-next`, `switch-endswitch`, or `while-loop` statement that encloses it. The control moves to the statement following the ended statement. The `break` statement is used to exit a loop.

(1) Format

Jump statement

```
break;
```

(2) Description

The `break` statement is used to exit a loop before the end criterion is satisfied in the loop. Also, this statement may be used to exit a `switch` statement in a specific case. Unless the `break` statement is used within a repetitive statement or a `switch` statement, an error occurs.

You should not specify more than 255 `break` statements in a loop structure.

(3) Example of coding

```
i = 0;
for(;;)
   AIT_LogMessage("Inside Loop");
   i = i + 1;
   if(i>100)
      break;
   endif;
next;
```

3.6.9 switch-endswitch

The `switch` statement selects the processing according to the result of the expression. The expression is specified within parentheses.

The `switch` statement can contain `case` labels and a `default` label, which are options of processing to be executed. The `case` labels can have unique constants. For the `switch` statement to be meaningful, it must contain at least one `case` label.

You can specify only one `default` label. This label is not a required item. When you use this, do not specify any value.

(1) Format

```
switch (expression)

  [case constant-value:]+
       [expression;]*
  ...
  ...
```

```
 [default:]
       [expression;]*

endswitch;
```

(2) Description

1. The expression is evaluated.
2. The control moves to the block that has the same `case` label value as the result of the expression. Until the `break` statement is encountered, the subsequent statements are executed (dependent of the `case` label value).
3. If the `break` statement is encountered, the control leaves the `switch` statement.
4. If the result of the expression is not equal to the `case` label value, the control will move to the `default` label, when it is specified.

When specifying the `switch-endswitch` statement, you have to follow the rules below:

- The data type of the value returned by the expression in the `switch` statement must be the same as the constants specified in `case` labels.
- You must not nest more than 255 `switch` statements.
- `case` labels do not need to have associated executable statements (with the exception of last `case` statement, which must have at least one associated executable statement).
- The `switch-endswitch` statement can have up to 255 `case` labels.
- In `case` labels, you can specify a numeric constant, a string constant, or an AIT language macro. You cannot specify any expressions in them.

  Example
  ```
  case -5:        // Valid
  case +6:        // Valid
  case "String":  // Valid
  case intvar:    // Invalid
  case 3+2:       // Invalid
  ```

- The constants you specify in `case` labels must have the same data type as the result of the expression in the `switch` statement.

  Example
  ```
  switch (Stringvar1+Stringvar2) //Both variables are string
  type.
  case 1:                        // Invalid
  case "caption-1":              // Valid
    .
    .
  ```

```
endswitch;
```

- If no `case` statement is specified in a `switch` case statement, the statement is interpreted as a syntax error.
- You can specify a maximum of 255 `break` statements in one `switch` `case` statement.
- You do not need to specify any statements for all `case` labels except the last `case` label.

  Example

```
switch(i)
{
  case 1:
  case 2:
   a=b+c; // If you do not specify this statement, the script
          // analyzer will issue a syntax error.
}
```

(3) Example of coding

```
s1="abcdefghijk";
switch (!AIT_IsEmpty(s1 ))
case true:            // Executed if s1 is empty.
    s2 = AIT_StrUpper(s1);
    AIT_MessageBox("s2",s2);
       if ( ( length = AIT_StrLength(s2)) > 10)
           break;
    endif;
    break;
default:            // Executed if the expression evaluates to
false.
    break;            //
endswitch;
```

3.7 Function calls

The AIT language supports various types of API functions to carry out standard operations. The API functions can be categorized as follows:

- Window operations
- Check operations
- Resolution checks
- Date/time operations
- IME operations
- Character string operations
- Message operations
- Registry operations
- Directory operations
- Redirect operations
- File operations
- INI file operations
- Recorder operations
- Taskbar operations
- Utility operations
- Interfacing with JP1/IT Desktop Management 2

3.7.1 Format

- The above API functions can be called from the MAIN and ERROR sections.
- When you pass parameters to API functions, make sure that the data types of the parameters conform to the API specifications.
- You can specify an expression in a function call. If the data types of expressions and function calls conform to the API specifications, you can nest the function calls.
- If a runtime error occurs during execution of an API, the control moves to the ERROR section.
- If a function fails, you can use `AIT_GetLastError` to acquire the return code of the function. You can also use `AIT_GetErrorText` to acquire the text of the

error message.

3.7.2 Example of coding

```
integer intvar1;
string Caption;
string Stringvar1, Stringvar2;
...
...
AIT_LogMessage("SAMPLE FUNCTION CALL");  // Function call
if(AIT_FocusWindow("Installable-Setup","#32770"))
       // Return value of the function is used in the expression.
   AIT_LogMessage("INSIDE Installable Software SETUP");
        // Executed
   AIT_PlayKey("{Enter}");
endif;
        // Check for the abnormal end of the called function
Caption = "Installable Software";
intvar1    = AIT_GetSubStr(Stringvar1, Stringvar2, 50);
                                          // intvar1 is 0.
if(!intvar1)
          AIT_LogMessage(AIT_GetErrorText(AIT_GetLastError()));
endif;
```

3.8 Keywords

Keywords are predefined, reserved identifiers that have special meanings. You cannot use any keywords in a program as variable, constant, and label names. Variable and constant names must differ from keywords. Keywords are not case-sensitive. Therefore, you cannot use them as variable and constant names by changing the case of the letters used in them.

The following table lists the keywords defined in the AIT language.

Table 3-3: Keywords defined in the AIT language

No.	Keyword
1	`bool`
2	`break`
3	`case`
4	`const`
5	`continue`
6	`default`
7	`define`
8	`do`
9	`else`
10	`endif`
11	`endswitch`
12	`ERROR`
13	`float`
14	`for`
15	`goto`
16	`IconGroupName`
17	`if`
18	`InstallDirectory`
19	`InstallDrive`

No.	Keyword
20	InstallerName
21	integer
22	JCompany
23	JUser
24	loop
25	MAIN
26	next
27	PACKAGE_INFO
28	PackageID
29	Product
30	ScriptFileVersion
31	SerialNumber
32	string
33	switch
34	Version
35	while

Other keywords include API names, predefined AIT macro names, and several Win32 error codes.

3.9 Macros

The AIT script language has predefined constants you cannot assign a value to. These constants are macros classified as shown below.

You cannot declare any macros in the DEFINE section.

3.9.1 Macros for window and check operations

The table below lists the macros for window and check operations. The data of these macros is the `integer` type.

Table 3-4: Macros for window and checking operations

Macro name	
ALT_OFF	ALT_ON
BUTTON_CTRL	CALENDAR_CTRL
CAPSLOCK	CHECKBOX_CTRL
COMBO_CTRL	COMMANDBUTTON_CTRL
CONTROL_CAPTION_SIZE	CONTROL_CLASS_SIZE
CTRL_OFF	CTRL_ON
DTPICKER_CTRL	EDIT_CTRL
HSCROLL	IDABORT
IDCANCEL	IDIGNORE
IDNO	IDOK
IDRETRY	IDYES
INSERTLOCK	IPADDRESS_CTRL
KEYSTATE_OFF	KEYSTATE_ON
LBUTTON	LISTBOX_CTRL
LIST_CTRL	MBUTTON
MB_ABORTRETRYIGNORE	MB_ICONEXCLAMATION
MB_ICONINFORMATION	MB_ICONQUESTION
MB_ICONSTOP	MB_OK

Macro name	
MB_OKCANCEL	MB_RETRYCANCEL
MB_YESNO	MB_YESNOCANCEL
NOTIFICATION_CTRL	NUMLOCK
OPTIONBUTTON_CTRL	PROGRESS_CTRL
RBUTTON	REBAR_CTRL
SB_BOTTOM	SB_LEFT
SB_LINEDOWN	SB_LINELEFT
SB_LINERIGHT	SB_LINEUP
SB_PAGEDOWN	SB_PAGELEFT
SB_PAGERIGHT	SB_PAGEUP
SB_RIGHT	SB_THUMBPOSITION
SB_THUMBTRACK	SB_TOP
SCROLLBAR_CTRL	SCROLLLOCK
SHIFT_OFF	SHIFT_ON
SLIDER_CTRL	SPIN_CTRL
STARTBUTTON_CTRL	STATIC_CTRL
STATUSBAR_CTRL	STYLE_CHECK_BUTTON
STYLE_PUSH_BUTTON	STYLE_RADIO_BUTTON
SW_HIDE	SW_MAXIMIZE
SW_MINIMIZE	SW_NORMAL
SW_SHOW	SW_SHOWMAXIMIZED
SW_SHOWMINIMIZED	SW_SHOWMINNOACTIVE
SW_SHOWNOACTIVATE	SW_SHOWNORMAL
TAB_CTRL	TASKBARCLOCK_CTRL
TASKBARITEMS_CTRL	TASKBAR_CTRL
TOOLBAR_CTRL	TOOLTIPS_CTRL
TREE_CTRL	VSCROLL

3.9.2 Macros for message operations

The following macro is for message operations. The data of this macro is the `integer` type.

- `HWND_BROADCAST`

3.9.3 Macros for file operations

The table below lists the macros for file operations. The data of these macros is the `integer` type.

Table 3-5: Macros for file operations

Macro name	
AIT_ALLFILES	CREATE_ALWAYS
CREATE_NEW	FILE_ALL
FILE_ATTRIBUTE_ARCHIVE	FILE_ATTRIBUTE_DIRECTORY
FILE_ATTRIBUTE_HIDDEN	FILE_ATTRIBUTE_READONLY
FILE_ATTRIBUTE_SYSTEM	FILE_READ
FILE_WRITE	GENERIC_READ
GENERIC_WRITE	OPEN_ALWAYS
OPEN_EXISTING	TRUNCATE_EXISTING

3.9.4 Macros for IME operations

The table below lists the macros for IME operations. The data of these macros is the `integer` type.

Table 3-6: Macros for IME operations

Macro name	
IGP_CONVERSION	IGP_GETIMEVERSION
IGP_PROPERTY	IGP_SELECT
IGP_SENTENCE	IGP_SETCOMPSTR
IGP_UI	IME_CHOTKEY_IME_NONIME_TOGGLE
IME_CHOTKEY_SHAPE_TOGGLE	IME_CHOTKEY_SYMBOL_TOGGLE
IME_CMODE_CHARCODE	IME_CMODE_EUDC
IME_CMODE_FULLSHAPE	IME_CMODE_HANJACONVERT

Macro name	
IME_CMODE_KATAKANA	IME_CMODE_NATIVE
IME_CMODE_NOCONVERSION	IME_CMODE_ROMAN
IME_CMODE_SOFTKBD	IME_CMODE_SYMBOL
IME_JHOTKEY_CLOSE_OPEN	IME_KHOTKEY_ENGLISH
IME_KHOTKEY_HANJACONVERT	IME_KHOTKEY_SHAPE_TOGGLE
IME_PROP_AT_CARET	IME_PROP_CANDLIST_START_FROM_1
IME_PROP_SPECIAL_UI	IME_PROP_UNICODE
IME_SMODE_AUTOMATIC	IME_SMODE_CONVERSATION
IME_SMODE_NONE	IME_SMODE_PHRASEPREDICT
IME_SMODE_PLAURALCLAUSE	IME_SMODE_PLURALCLAUSE
IME_SMODE_SINGLECONVERT	IME_THOTKEY_IME_NONIME_TOGGLE
IME_THOTKEY_SHAPE_TOGGLE	IME_THOTKEY_SYMBOL_TOGGLE
SCS_CAP_COMPSTR	SCS_CAP_MAKEREAD
SELECT_CAP_CONVERSION	SELECT_CAP_SENTENCE
UI_CAP_2700	UI_CAP_ROT90
UI_CAP_ROTANY	-

Legend:

-: None

3.9.5 Macros for utility operations

The table below lists the macros for utility operations. The data of these macros is the `integer` type.

Table 3-7: Macros for utility operations

Macro name	
VER_PLATFORM_WIN32_NT	VER_PLATFORM_WIN32_WINDOWS

3.9.6 Macros for registry operations

The table below lists the macros for registry operations. The data of these macros is the `integer` type.

Table 3-8: Macros for registry operations

Macro name	
HKEY_CLASSES_ROOT	HKEY_CURRENT_CONFIG
HKEY_CURRENT_USER	HKEY_LOCAL_MACHINE
HKEY_USERS	-

Legend:

-: None

3.9.7 Macros for directory operations

The table below lists the macros for directory operations. The data of these macros is the `integer` type.

Table 3-9: Macros for directory operations

Macro name	
AIT_CURRENTDIRECTORY	AIT_PARENTDIRECTORY

3.9.8 Macros for error logging

The table below lists the macros for error logging. The data of these macros is the `integer` type.

Table 3-10: Macros for error logging

Macro name	
ERROR_ACCESS_DENIED	ERROR_ALREADY_EXISTS
ERROR_BADDB	ERROR_BADKEY
ERROR_BAD_COMMAND	ERROR_BAD_NETPATH
ERROR_BAD_NET_NAME	ERROR_BAD_PATHNAME
ERROR_BUFFER_OVERFLOW	ERROR_CANNOT_MAKE
ERROR_CANTOPEN	ERROR_CANTREAD
ERROR_CANTWRITE	ERROR_CHILD_MUST_BE_VOLATILE
ERROR_CONTROL_ID_NOT_FOUND	ERROR_CRC
ERROR_CURRENT_DIRECTORY	ERROR_DIRECTORY
ERROR_DISK_CORRUPT	ERROR_DISK_FULL

Macro name	
ERROR_FILENAME_EXCED_RANGE	ERROR_FILE_CORRUPT
ERROR_FILE_EXISTS	ERROR_FILE_NOT_FOUND
ERROR_HANDLE_DISK_FULL	ERROR_HANDLE_EOF
ERROR_INSUFFICIENT_BUFFER	ERROR_INVALID_COMPUTERNAME
ERROR_INVALID_DATA	ERROR_INVALID_DRIVE
ERROR_INVALID_FUNCTION	ERROR_INVALID_HANDLE
ERROR_INVALID_INDEX	ERROR_INVALID_MENU_HANDLE
ERROR_INVALID_NAME	ERROR_INVALID_NETNAME
ERROR_INVALID_PARAMETER	ERROR_INVALID_SCROLLBAR_RANGE
ERROR_INVALID_SHARENAME	ERROR_INVALID_WINDOW_HANDLE
ERROR_KEY_DELETED	ERROR_KEY_HAS_CHILDREN
ERROR_LOCK_VIOLATION	ERROR_MENU_ITEM_NOT_FOUND
ERROR_NETWORK_BUSY	ERROR_NETWORK_UNREACHABLE
ERROR_NOACCESS	ERROR_NOT_ENOUGH_MEMORY
ERROR_NOT_READY	ERROR_NOT_REGISTRY_FILE
ERROR_NO_LOG_SPACE	ERROR_NO_MORE_FILES
ERROR_NO_MORE_ITEMS	ERROR_NO_MORE_SEARCH_HANDLES
ERROR_NO_SCROLLBARS	ERROR_OUTOFMEMORY
ERROR_PATH_BUSY	ERROR_PATH_NOT_FOUND
ERROR_READ_FAULT	ERROR_REGISTRY_CORRUPT
ERROR_REGISTRY_IO_FAILED	ERROR_REGISTRY_RECOVERED
ERROR_SHARING_VIOLATION	ERROR_SUCCESS
ERROR_TIMEOUT	ERROR_TOO_MANY_OPEN_FILES
ERROR_WRITE_FAULT	ERROR_WRITE_PROTECT

Chapter

4. API Function Reference

This chapter describes the API functions that can be used in the AIT language.

4.1 API functions
4.2 Details about the API functions
4.3 Examples of using API functions

4.1 API functions

The API functions provided by the AIT language can be categorized as follows:

- Window operations
- Check operations
- Resolution checks
- Date/time operations
- IME operations
- Character string operations
- Message operations
- Registry operations
- Redirect operations
- Directory operations
- File operations
- INI file operations
- Recorder operations
- Taskbar operations
- Utility operations
- Interfacing with JP1/IT Desktop Management 2

The following subsections describe the API functions for each category shown above.

4.1.1 Window operations

The API functions below allow the application to perform operations on windows and controls. These operations include finding a window, setting the focus on a specific control, and setting the status of a check box or radio button.

API function name	Description
AIT_FocusWindow	Finds the window, and sets the focus on it.
AIT_ExistWindow	Checks whether the window exists.
AIT_MinWnd	Minimizes the window.
AIT_SetWndPos	Changes the position of the specified window.

API function name	Description
AIT_SetWndPosSize	Changes the position and size of the specified window.
AIT_GetWindowText	Acquires the title of the specified window.
AIT_SetActWnd	Activates the window.
AIT_GetCtrlText	Acquires text from the window control.
AIT_CtrlSetFocus	Sets the focus on the control.
AIT_SetSpinPos	Sets the position of the spin or slider control.
AIT_SetScrollPos	Sets the position of the scroll bar.
AIT_CtrlClick	Performs a mouse click on the control.
AIT_SelectMultipleListItem	Selects multiple items in the list.
AIT_SelectListItem	Selects an item in the list.
AIT_SelectIPAddressField	Selects the IP address field.
AIT_SelectText	Selects the text of the control.
AIT_DefaultButtonCount	Acquires the number of default buttons.
AIT_SetCheck	Sets the status of the radio button or check box.
AIT_CtrlItemCount	Acquires the number of control items.
AIT_GetIndexText	Acquires text from the index.
AIT_CtrlItemIndex	Acquires the index of the text.
AIT_GetIndexTextLen	Acquires the character length for a 0-based index character string.
AIT_SetKeyState	Sets the key status.
AIT_GetKeyState	Acquires the key status.
AIT_MouseClick	Performs a mouse click.
AIT_MouseUp	Performs a mouse release.
AIT_MouseDown	Holds down the mouse button.
AIT_MouseMoveTo	Moves the mouse pointer.
AIT_MouseDragDrop	Performs a mouse drag-and-drop operation.
AIT_MouseDblClk	Performs a mouse double-click.
AIT_SetComboEditSelText	Selects text in the combo box.

API function name	Description
AIT_ComboBoxCloseUp	Performs a close-up operation on the combo box.
AIT_ComboBoxDropDown	Performs a drop-down operation on the combo box.
AIT_GetEditFirstLineIndex	Acquires the index of the first line in the edit box.
AIT_GetEditCurrentLineIndex	Acquires the index of the current line in the edit box.
AIT_GetCtrlTextLen	Acquires the length of the control text.
AIT_GetEditTextLineLen	Acquires the length of the line in the edit box.
AIT_GetDtPickerTime	Acquires the time from the date/time picker.
AIT_GetDtPickerDate	Acquires the date from the date/time picker.
AIT_SetDtPickerTime	Sets a time with the date/time picker.
AIT_SetDtPickerDate	Sets a date with the date/time picker.
AIT_GetMenu	Acquires a menu handle.
AIT_GetSubMenu	Acquires a sub-menu handle.
AIT_GetMenuText	Acquires a menu text.
AIT_GetMenuIndex	Acquires the menu index.
AIT_MenuItemClick	Clicks on the specified menu item.

4.1.2 Check operations

The API functions below allow the application to carry out check operations such as checking the existence and status of a control, and checking whether the focus is set on the control.

API function name	Description
AIT_VerifyExistence	Checks whether the control exists.
AIT_VerifyEnabled	Checks whether the control can be used.
AIT_VerifyFocus	Checks whether the focus is set on the control.
AIT_VerifyState	Checks the status of the control.
AIT_VerifyCharPos	Checks the position of the character string of the control.
AIT_VerifyLine	Checks the index of the line.
AIT_VerifyText	Checks for the specified text.

API function name	Description
AIT_VerifySelected	Checks for the text selected in the edit box.
AIT_VerifyFirstVisible	Checks the first visible item of the control.
AIT_VerifyCount	Checks the number of items.
AIT_VerifyPos	Checks the position of the control.
AIT_VerifyIndex	Checks the index of the control.
AIT_VerifyLocation	Checks the position of the control.
AIT_VerifyDateTime	Checks date or time value for the control.
AIT_VerifyKeyState	Checks the status of the key.
AIT_VerifyDefaultButton	Checks the style of the default button.
AIT_VerifyNoOfCtrls	Checks the number of controls in the window.
AIT_VerifyMenuChecked	Checks whether the menu item is selected.
AIT_VerifyMenuEnabled	Checks whether the menu item can be used.

4.1.3 Resolution check

The API function below checks the resolution set in the system.

API function name	Description
AIT_CheckResolution	Checks the resolution of the display.

4.1.4 Date/time operations

The API functions below allow the application to process a date/time.

API function name	Description
AIT_GetDate	Acquires a system's date.
AIT_GetTime	Acquires a system's time.

4.1.5 IME operations

The API functions below allow the application to simulate enabling/disabling the IME, selecting the conversion mode, and other operations the user frequently performs.

API function name	Description
AIT_IMEGetOpenStatus	Acquires the open status of the IME.

API function name	Description
AIT_IMESetOpenStatus	Sets the open status of the IME.
AIT_IMEGetConversionStatus	Acquires the conversion status of the IME.
AIT_IMESetConversionStatus	Sets the conversion status of the IME.
AIT_IMEGetStatusWindowPos	Acquires the position of the status window of the IME.
AIT_IMESetStatusWindowPos	Sets the position of the status window of the IME.
AIT_IMEGetProperty	Acquires the properties of the IME.
AIT_IMESimulateHotKey	Simulates the hot key of the IME.

4.1.6 Character string operations

The API functions below allow the application to carry out character string processing including partial character string processing, cut processing and conversion to the ASCII code.

API function name	Description
AIT_GetSubStr	Returns a character string with the specified length from the character string.
AIT_FindSubStr	Finds a character sting, and returns the position of the first matching character string.
AIT_StrLength	Acquires the length of the character string.
AIT_IsEmpty	Checks whether the character string is empty.
AIT_StrLTrim	Truncates the character string from the left.
AIT_StrRTrim	Truncates the character string from the right.
AIT_StrTrim	Removes characters from a character string.
AIT_StrUpper	Converts the characters in the character string to uppercase.
AIT_StrLower	Converts the characters in the character string to lowercase.
AIT_StrLeft	Acquires the specified number of characters on the left in the character string.
AIT_StrRight	Acquires the specified number of characters on the right in the character string.
AIT_CharToASCII	Returns the ASCII code of the character.
AIT_ASCIIToChar	Converts an ASCII code into the corresponding character.

4.1.7 Message operations

The API functions below define new window messages, and post a recorded message to another window.

API function name	Description
AIT_RegisterWindowMessage	Defines a window message.
AIT_PostMessage	Posts the message.

4.1.8 Registry operations

The API functions below allow the application to perform registry operations such as creating a registry key, deleting a registry key, and referencing a key value.

API function name	Description
AIT_RegCreateKey	Creates a registry key.
AIT_RegDeleteKey	Deletes a registry key.
AIT_RegDeleteValue	Deletes a registry value.
AIT_RegOpenKey	Opens a registry key.
AIT_RegCloseKey	Closes a registry key.
AIT_RegGetStringValue	Acquires a string-type value for the registry.
AIT_RegGetDWORDValue	Acquires a DWORD-type value of the registry.
AIT_RegSetStringValue	Sets a string-type value for the registry.
AIT_RegSetDWORDValue	Sets a DWORD-type value of the registry.
AIT_RegKeyExists	Checks whether the registry key exists.
AIT_RegValueExists	Checks whether the registry value exists.

4.1.9 Redirect operations

The API functions below allow the application to switch the mode in which to access registry paths or file paths.

API function name	Description
AIT_Wow64DisableWow64Redirection	Switches to the mode in which the application accesses registry paths or file paths as a 64-bit application.
AIT_Wow64RevertWow64Redirection	Switches to the mode in which the application accesses registry paths or file paths as a 32-bit application.

4.1.10 Directory operations

The API functions below allow the application to create, delete, and copy a directory.

API function name	Description
AIT_DirCreate	Creates a directory.
AIT_DirRemove	Deletes the current directory.
AIT_DirCopy	Copies a directory.
AIT_SetCurrentDirectory	Sets the current directory.
AIT_GetCurrentDirectory	Acquires the current directory.

4.1.11 File operations

The API functions below allow the application to create, copy, delete, and rename a file.

API function name	Description
AIT_FileOpen	Opens or creates the file.
AIT_FileClose	Closes the file.
AIT_FileGetLine	Acquires data from the file.
AIT_FilePutLine	Writes data into the file.
AIT_FileGetPos	Acquires the position of the file pointer.
AIT_FileSetPos	Sets the file pointer at the specified position.
AIT_FileEOF	Checks whether the file pointer is at the end of the file.
AIT_FileSize	Acquires the file size.
AIT_FileCopy	Copies the file.
AIT_FileDelete	Deletes the file.
AIT_FileExists	Checks whether the file exists.
AIT_FileRename	Renames the file.
AIT_ChangeFileAttribute	Changes the file attribute.
AIT_FindFirstFile	Finds the first file.
AIT_FindNextFile	Finds the next file.
AIT_FindCloseFile	Closes the search handle.

4.1.12 INI file operations

The API functions below allow the application to perform operations on initialization files (`.INI`). These operations include acquiring a key value in a section and acquiring all the contents of a section.

API function name	Description
`AIT_GetProfileString`	Acquires a profile string.
`AIT_SetProfileString`	Sets a profile string.
`AIT_GetProfileFirstSection`	Acquires the first section.
`AIT_GetProfileNextSection`	Acquires the next profile section.
`AIT_GetProfileFirstSectionNames`	Acquires the first section name.
`AIT_GetProfileNextSectionNames`	Acquires the next section name.

4.1.13 Recorder operations

The API functions below are used to create an AIT file for automatic installation.

API function name	Description
`AIT_Sleep`	Stops execution of the AIT file for the specified period.
`AIT_SetDefaultWaitTimeout`	Sets the default time-out for a wait.
`AIT_Exec`	Performs the specified processing.
`AIT_ExecCommand`	Executes the specified MS-DOS or system command.
`AIT_PlayKey`	Simulates keyboard entries on the active window.

4.1.14 Taskbar operations

The API functions below are used to perform taskbar operations such as checking whether a taskbar item has been clicked, and checking whether the focus is set on the taskbar.

API function name	Description
`AIT_TaskbarClk`	Sets a hot spot on the taskbar where a single-click of the specified mouse button can be simulated.
`AIT_TaskbarHasFocus`	Check whether the input focus is set on the taskbar.
`AIT_TaskbarSetFocus`	Sets the input focus on the taskbar.

4.1.15 Utility operations

The utility API functions below allow the application to display message and status boxes, acquire the OS type and carry out other processing.

API function name	Description
AIT_MessageBox	Displays a message box.
AIT_StatusBox	Displays a status box.
AIT_StatusBoxClose	Closes the status box.
AIT_GetEnv	Acquires the contents of an environment variable.
AIT_GetLastError	Acquires the last occurring error.
AIT_GetErrorText	Acquires an error message.
AIT_InitLog	Initializes a log file.
AIT_LogMessage	Stores a message into a log file.
AIT_Exit	Exits AIT file processing.
AIT_GetOSType	Acquires the OS type.

4.1.16 Interfacing with JP1/IT Desktop Management 2

The API function below enables interfacing with JP1/IT Desktop Management 2.

API function name	Description
AIT_DMPSTRC	Sets a global variable specific to JP1/IT Desktop Management 2. Always specify this API function when you want to use an AIT file or perform remote installation with JP1/IT Desktop Management 2.

4.2 Details about the API functions

This section describes the API functions.

Format of API function explanations

The API functions are described in the following format. The API functions are sorted in alphabetical order.

Description

This describes the functionality of the API function.

Format

This provides the coding format of the API function.

The parameters enclosed with `[]` are optional. If you omit such a parameter, the default is used. Optional parameters are shown at the end of the parameter list. You can omit only the last parameter(s).

Example 1:

```
bool AIT_SelectListItem ( strCaption, nCtrlType, strItemText
[, fTimeOut] );
```

For the format of the above API function, you do not need to specify fTimeOut when calling the function.

Example 2:

```
AIT_SelectListItem ( "Countries" LISTBOX_CTRL, "Japan" );
integer AIT_MessageBox ( strMessage, strTitle [, nIconType]
[, nMsgBoxType] );
```

For the format of the above API function, you can call the following function with both `nIconType` and `nMsgBoxType` omitted:

```
AIT_MessageBox( "Hello World","Error");
```

If you specify `nMsgBoxType` but do not want to specify `nIconType`, the default of `nIconType` will be used.

```
AIT_MessageBox( "Hello World","Error", MB_ICONEXCLAMATION );
```

Parameters

This provides the parameters you can specify in the API function. There are input and output parameters. Some of the parameters are optional.

Return value

This describes the return values of the API function.

Each API has its error code returned if the API fails during processing. In an AIT file, if you need to code special error handling for special processing, you first check return codes to see whether API functions were successfully executed.

If a run-time error has occurred, the script coded in the ERROR section of the AIT file is executed automatically.

Note

This provides points you should note when executing the API function.

AIT_ASCIIToChar

Description

Converts the specified ASCII code into the corresponding character.

Format

```
string AIT_ASCIIToChar  (
  integer nASCIIValue     // ASCII code
);
```

Parameters

- `nASCIIValue` (input)

  Specify an ASCII code.

Return value

This API function returns the character corresponding to the ASCII code.

AIT_ChangeFileAttribute

Description

Changes the attribute of the specified file or directory.

Format

```
bool AIT_ChangeFileAttribute  (
  string strFileName,     // Filename
  integer nFileAttributes // File attribute
```

```
);
```

Parameters

- `strFileName` (input)

  Specify a filename. You can also specify a directory.

- `nFileAttributes` (input)

  Specify a new attribute. For the values you can use for input, see *AIT_FileExists*.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, use `AIT_GetLastError` to acquire an extended error code.

The following gives the error codes that `AIT_GetLastError` might return if the function has not been executed normally:

Extended error number	Error code
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
18	ERROR_NO_MORE_FILES
21	ERROR_NOT_READY
53	ERROR_BAD_NETPATH
87	ERROR_INVALID_PARAMETER
123	ERROR_INVALID_NAME
148	ERROR_PATH_BUSY
161	ERROR_BAD_PATHNAME
183	ERROR_ALREADY_EXISTS
206	ERROR_FILENAME_EXCED_RANGE
1005	ERROR_UNRECOGNIZED_VOLUME
1210	ERROR_INVALID_COMPUTERNAME
1214	ERROR_INVALID_NETNAME

Extended error number	Error code
1231	ERROR_NETWORK_UNREACHABLE
1392	ERROR_FILE_CORRUPT

AIT_CharToASCII

Description

Converts the specified character into the corresponding ASCII code.

Format

```
integer AIT_CharToASCII (
  string strStrName     // Character string consisting of at
least one character
);
```

Parameters

- `strStrName` (input)

  Specify a character string.

Return value

This API function returns the ASCII code of the first character in the specified character string.

AIT_CheckResolution

Description

Checks whether the specified resolution matches the resolution of the current screen.

Format

```
integer AIT_CheckResolution  (
  integer nWidth,     // Width of a screen to be checked
  integer nHeight     // Height of a screen to be checked
);
```

Parameters

- `nWidth` (input)

  Specify the width (in units of pixels) you want to check against the width of the primary display monitor screen.

- `nHeight` (input)

  Specify the height (in units of pixels) you want to check against the height of the primary display monitor screen.

Return values

This API function returns `1` if the specified resolution matches the current resolution. Otherwise, this API function returns `0`. This API function returns `-1` if the function fails. In this case, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error code that `AIT_GetLastError` might return:

Extended error number	Error code
87	ERROR_INVALID_PARAMETER

AIT_ComboBoxCloseUp

Description

Simulates a close-up operation on the combo box.

Format

```
bool AIT_ComboBoxCloseUp  (
  string strCaption    // Control's caption
  [,float fTimeOut]    // Time-out
);
bool AIT_ComboBoxCloseUp  (
  integer nCtrlID     // Control ID
  [,float fTimeOut]   // Time-out
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `fTimeOut` (input, optional)

  Specify a time-out value in units of seconds for a retry if no control has been returned. Also specify the maximum time this function can use to find the control, in units of seconds. The default is the value set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is `true` if the function was executed normally, and `false` if not.

If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_ComboBoxDropDown

Description

Simulates a drop-down operation on the combo box.

Format

```
bool AIT_ComboBoxDropDown  (
  string strCaption       // Control's caption
  [,float fTimeOut]       // Time-out
);
bool AIT_ComboBoxDropDown  (
  integer nCtrlID         // Control ID
  [,float fTimeOut]       // Time-out
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `fTimeOut` (input, optional)

  Specify the maximum time the function can use to find the control, in units of seconds. The default is the value set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is `true` if the function was executed normally, and `false` if not.

If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_CtrlClick

Description

Simulates a mouse click on a specific control in the active window.

Format

```
bool AIT_CtrlClick  (
  string strCaption,     // Control's caption
  integer nCtrlType,     // Control type
  integer nMouseButton   // Mouse button
  [,float fTimeOut]      // Time-out
);
bool AIT_CtrlClick  (
  integer nCtrlID,       // Control ID
  integer nCtrlType,     // Control type
  integer nMouseButton   // Mouse button
```

```
  [,float fTimeOut]      // Time-out
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nCtrlType` (input)

  Specify a control type, which must be one of the following values.

Value	Description
`BUTTON_CTRL`	The control type is a command button.
`CHECKBOX_CTRL`	The control type is a check box.
`OPTIONBUTTON_CTRL`	The control type is an option button.
`EDIT_CTRL`	The control type is an edit box.
`STATIC_CTRL`	The control type is a static text control.
`COMBO_CTRL`	The control type is a combo box.
`LISTBOX_CTRL`	The control type is a list box.
`SPIN_CTRL`	The control type is a spin control.
`TREE_CTRL`	The control type is a tree control.
`LIST_CTRL`	The control type is a list control.
`DTPICKER_CTRL`	The control type is a date/time picker.

- `nMouseButton` (input, optional)

  Specify the mouse button for which a click operation should be simulated. You have to set one of the following values.

Value	Description
`LBUTTON`	Left mouse button
`MBUTTON`	Center mouse button
`RBUTTON`	Right mouse button

The default is `LBUTTON`.

- `fTimeOut` (input, optional)

  Specify the maximum time the function can use to find a control, in units of seconds. The default is the value set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_CtrlItemCount

Description

Uses a specific control in the active window to acquire the number of items.

Format

```
bool AIT_CtrlItemCount  (
  string strCaption,     // Control's caption
  integer nCtrlType,     // Control type
  integer nItemCount     // Number of items
  [,float fTimeOut]      // Time-out
);
bool AIT_CtrlItemCount  (
  integer nCtrlID,       // Control ID
  integer nCtrlType,     // Control type
```

```
  integer nItemCount     // Item count
  [,float fTimeOut]      // Time-out
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nCtrlType` (input)

  Specify a control type, which must be one of the following values.

Value	Description
`COMBO_CTRL`	The control type is a combo box.
`LISTBOX_CTRL`	The control type is a list box.
`TREE_CTRL`	The control type is a tree control.
`LIST_CTRL`	The control type is a list control.

- `nItemCount` (output)

  Specify a variable for receiving the number of items in the control. When the control is returned from the function, the variable stores the number of items.

- `fTimeOut` (input, optional)

  Specify the maximum time the function can use to find the control, in units of seconds. The default is the value set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is `true` if the function was executed normally, and `false` if not.

If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
6	`ERROR_INVALID_HANDLE`
8	`ERROR_NOT_ENOUGH_MEMORY`
14	`ERROR_OUTOFMEMORY`

Extended error number	Error code
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_CtrlItemIndex

Description

Uses a specific control in the active window to acquire the index of the item text.

Format

```
bool AIT_CtrlItemIndex  (
  string strCaption,     // Control's caption
  integer nCtrlType,     // Control type
  string strItemText,    // Item text
  integer nIndex         // Item index
  [,float fTimeOut]      // Time-out
);
bool AIT_CtrlItemIndex  (
  integer nCtrlID,       // Control ID
  integer nCtrlType,     // Control type
  string strItemText,    // Item text
  integer nIndex         // Item index
  [,float fTimeOut]      // Time-out
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nCtrlType` (input)

  Specify a control type, which must be one of the following values.

Value	Description
`COMBO_CTRL`	The control type is a combo box.
`LISTBOX_CTRL`	The control type is a list box.
`TREE_CTRL`	The control type is a tree control.
`LIST_CTRL`	The control type is a list control.

- `strItemText` (input)

  Specify an item text for acquiring an index.

- `nIndex` (output)

  Specify a variable for receiving the text index. When the function returns, the variable stores the index for the item text. The default index value is 0.

- `fTimeOut` (input, optional)

  Specify the maximum time the function can use to find the control, in units of seconds. The default is the value set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is `true` if the function was executed normally, and `false` if not.

If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
6	`ERROR_INVALID_HANDLE`
8	`ERROR_NOT_ENOUGH_MEMORY`
14	`ERROR_OUTOFMEMORY`
87	`ERROR_INVALID_PARAMETER`
112	`ERROR_DISK_FULL`
1400	`ERROR_INVALID_WINDOW_HANDLE`
1413	`ERROR_INVALID_INDEX`
1460	`ERROR_TIMEOUT`

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_CtrlSetFocus

Description

Sets the focus on a specific control.

Format

```
bool AIT_CtrlSetFocus (
  string strCaption,     // Control's caption
  integer nCtrlType      // Control type
  [,float fTimeOut]      // Time-out
);
bool AIT_CtrlSetFocus (
  integer nCtrlID,       // Control ID
  integer nCtrlType      // Control type
  [,float fTimeOut]      // Time-out
);
bool AIT_CtrlSetFocus (
  integer nIndex         // Control's index
  [,float fTimeOut]      // Time-out
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nCtrlType` (input)

  Specify a control type, which must be one of the following values.

Value	Description
`BUTTON_CTRL`	The control type is a command button.
`CHECKBOX_CTRL`	The control type is a check box.
`OPTIONBUTTON_CTRL`	The control type is an option button.
`EDIT_CTRL`	The control type is an edit box.

Value	Description
STATIC_CTRL	The control type is a static text.
COMBO_CTRL	The control type is a combo box.
LISTBOX_CTRL	The control box is a list box.
SPIN_CTRL	The control type is a spin control.
TREE_CTRL	The control type is a tree control.
LIST_CTRL	The control type is a list control.
DTPICKER_CTRL	The control type is a date/time picker.

- `nIndex` (Can be entered)

  Specify the tab order of the control.

- `fTimeOut` (input, optional)

  Specify the maximum time the function can use to find the control, in units of seconds. The default is the value set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is `true` if the function was executed normally, and `false` if not.

If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_DefaultButtonCount

Description

Acquires the number of command buttons that have the default button style in the active window.

Format

```
integer AIT_DefaultButtonCount ();
```

Parameters

None

Return values

The return value is the number of command buttons that have the default button style if the function was executed normally, and is `false` if not.

If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
112	ERROR_DISK_FULL
1460	ERROR_TIMEOUT

AIT_DirCopy

Description

Copies a directory to another location. If the specified copy destination directory already exists, it is overwritten.

Format

```
bool AIT_DirCopy  (
  string strSourceDirName,   // Copy source directory name
  string strTargetDirName    // Copy destination directory name
);
```

Parameters

- `strSourceDirName` (input)

  Specify a copy source directory name.

- `strTargetDirName` (input)

  Specify a copy destination directory name.

Return values

The return value is `true` if the function was executed normally, and `false` if not.

If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
87	ERROR_INVALID_PARAMETER
266	ERROR_CANNOT_COPY

AIT_DirCreate

Description

Creates a new directory.

Format

```
bool AIT_DirCreate  (
  string strDirName    // Directory name
);
```

Parameters

- `strDirName` (input)

  Specify the name of a directory you want to create.

Return values

The return value is `true` if the function was executed normally, and `false` if not.

If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
3	ERROR_PATH_NOT_FOUND

Extended error number	Error code
5	ERROR_ACCESS_DENIED
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
15	ERROR_INVALID_DRIVE
19	ERROR_WRITE_PROTECT
21	ERROR_NOT_READY
23	ERROR_CRC
53	ERROR_BAD_NETPATH
64	ERROR_NETNAME_DELETED
82	ERROR_CANNOT_MAKE
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
123	ERROR_INVALID_NAME
148	ERROR_PATH_BUSY
183	ERROR_ALREADY_EXISTS
206	ERROR_FILENAME_EXCED_RANGE
267	ERROR_DIRECTORY
1005	ERROR_UNRECOGNIZED_VOLUME
1210	ERROR_INVALID_COMPUTERNAME
1214	ERROR_INVALID_NETNAME
1231	ERROR_NETWORK_UNREACHABLE

AIT_DirRemove

Description

Removes an existing directory.

Format

```
bool AIT_DirRemove  (
```

```
  string strDirName    // Directory name
);
```

Parameters

- `strDirName` (input)

  Specify a directory name.

Return values

The return value is `true` if the function was executed normally, and `false` if not.

If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
16	ERROR_CURRENT_DIRECTORY
87	ERROR_INVALID_PARAMETER

AIT_DMPSTRC

Description

Sets a global variable that is specific to JP1/Desktop Management 2. Always specify this API function when you want to use an AIT file or perform remote installation with JP1/Desktop Management 2.

This API function is called before the AIT file starts the installer.

Format

```
bool AIT_DMPSTRC ();
```

Parameters

None

Return values

The return value is `true` if the function was executed normally, and `false` if not.

If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
2	ERROR_FILE_NOT_FOUND

Extended error number	Error code
3	ERROR_PATH_NOT_FOUND
5	ERROR_ACCESS_DENIED
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
15	ERROR_INVALID_DRIVE
21	ERROR_NOT_READY
38	ERROR_HANDLE_EOF
53	ERROR_BAD_NETPATH
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
123	ERROR_INVALID_NAME
148	ERROR_PATH_BUSY
161	ERROR_BAD_PATHNAME
206	ERROR_FILENAME_EXCED_RANGE
1005	ERROR_UNRECOGNIZED_VOLUME
1210	ERROR_INVALID_COMPUTERNAME
1214	ERROR_INVALID_NETNAME
1231	ERROR_NETWORK_UNREACHABLE

AIT_Exec

Description

Executes a specified application file.

Format

```
bool AIT_Exec  (
  string strExeName,     // Application filename
  integer nShowState     // Displayed
);
```

Parameters

- `strExeName` (input)

  Specify an application filename.

- `nShowState` (input)

  Specify how to display the application. You have to specify one of the following values.

Value	Description
`SW_HIDE`	Does not display the application.

`SW_SHOWNORMAL`
or
`SW_NORMAL`
	Normally displays the application.

`SW_SHOWMINIMIZED`	Minimize the application.

`SW_SHOWMAXIMIZED`
or
`SW_MAXIMIZE`
	Maximizes the application.

`SW_SHOWNOACTIVATE`	Normally displays the application with no focus.
`SW_SHOWMINNOACTIVE`	Minimizes the application with no focus.

Return values

The return value is `true` if the function was executed normally, and `false` if not.

If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
2	`ERROR_FILE_NOT_FOUND`
3	`ERROR_PATH_NOT_FOUND`
5	`ERROR_ACCESS_DENIED`
8	`ERROR_NOT_ENOUGH_MEMORY`
14	`ERROR_OUTOFMEMORY`
21	`ERROR_NOT_READY`
87	`ERROR_INVALID_PARAMETER`

Extended error number	Error code
123	ERROR_INVALID_NAME

AIT_ExecCommand

Description

Executes an MS-DOS or system command.

Format

```
bool AIT_ExecCommand  (
  string strCommandName     // MS-DOS command
);
```

Parameters

- `strCommandName` (input)

  Specify an application filename or a system command.

Return values

The return value is `true` if the function was executed normally, and `false` if not.

If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
5	ERROR_ACCESS_DENIED
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
21	ERROR_NOT_READY
87	ERROR_INVALID_PARAMETER
123	ERROR_INVALID_NAME

AIT_ExistWindow

Description

Checks whether the window exists which matches a specified window name and class name.

Format

```
integer AIT_ExistWindow  (
  string strWndCaption,     // Window's caption
  string strClassName       // Class name
  [,float fTimeOut]         // Time-out
);
```

Parameters

- `strWndCaption` (input)

  Specify the caption of a window.

- `strClassName` (input)

  Specify a window's class name.

- `fTimeOut` (input, optional)

  Specify the maximum time the function can use to find the control, in units of seconds. The default is the value set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is `1` if the window exists, and `0` if not. If the function has not been executed successfully, the return value is `-1`. If `-1` has been returned, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

AIT_Exit

Description

Exits the AIT file.

Format

```
AIT_Exit ();
```

Parameters

None

Return values

None

AIT_FileClose

Description

Closes a file handle.

Format

```
bool AIT_FileClose  (
  integer nFileHandle     // File handle
);
```

Parameters

- `nFileHandle` (input)

  Specify a file handle you have opened using the `AIT_FileOpen` function.

Return values

The return value is `true` if the function was executed normally, and `false` if not.

If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
4	ERROR_TOO_MANY_OPEN_FILES

Extended error number	Error code
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY

AIT_FileCopy

Description

Copies a copy source file to a copy destination file. If the specified copy destination file already exists, it is overwritten.

Format

```
bool AIT_FileCopy  (
  string strSourceFileName,     // Copy source filename
  string strTargetFileName      // Copy destination filename
);
```

Parameters

- `strSourceFileName` (input)

  Specify a copy source filename. You can also use a wildcard (`*`).

- `strTargetFileName` (input)

  Specify a copy destination file or directory name. You cannot use the wild card. If the specified copy destination directory does not exist, it is created.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
5	ERROR_ACCESS_DENIED
87	ERROR_INVALID_PARAMETER

Extended error number	Error code
117	ERROR_INVALID_CATEGORY

AIT_FileDelete

Description

Deletes a specified file.

Format

```
bool AIT_FileDelete  (
  string strFileName     // Filename
);
```

Parameters

- `strFileName` (input)

  Specify the name of a file you want to remove. You can also use a wildcard (`*`).

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
5	ERROR_ACCESS_DENIED
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
15	ERROR_INVALID_DRIVE
19	ERROR_WRITE_PROTECT
21	ERROR_NOT_READY
53	ERROR_BAD_NETPATH
123	ERROR_INVALID_NAME
148	ERROR_PATH_BUSY

Extended error number	Error code
161	ERROR_BAD_PATHNAME
1005	ERROR_UNRECOGNIZED_VOLUME
1210	ERROR_INVALID_COMPUTERNAME
1214	ERROR_INVALID_NETNAME
1231	ERROR_NETWORK_UNREACHABLE

AIT_FileEOF

Description

Checks whether the file pointer has reached the EOF.

Format

```
integer AIT_FileEOF (
  integer nFileHandle     // File handle
);
```

Parameters

- `nFileHandle` (input)

  Specify a file handle.

Return values

The return value is `1` if the file pointer has reached the EOF, and `0` if not. If the function has not been executed, the return value is `-1`. If `-1` has been returned, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
21	ERROR_NOT_READY
32	ERROR_SHARING_VIOLATION
33	ERROR_LOCK_VIOLATION

Extended error number	Error code
112	ERROR_DISK_FULL
148	ERROR_PATH_BUSY
1231	ERROR_NETWORK_UNREACHABLE
1392	ERROR_FILE_CORRUPT

AIT_FileExists

Description

Checks whether the file with a specified attribute exists.

Format

```
integer AIT_FileExists  (
  string strFileName            // Filename
  [,integer nFileAttributes]    // File attribute
);
```

Parameters

- `strFileName` (input)

  Specify the name of a file to be found.

- `nFileAttributes` (input, optional)

  Specify a file attribute, which must be one of the following values.

Value	Description
FILE_ATTRIBUTE_DIRECTORY	The file is a directory.
FILE_ATTRIBUTE_HIDDEN	The file is a hidden one.
FILE_ATTRIBUTE_SYSTEM	The file is part of the OS, or is OS-specific.
FILE_ATTRIBUTE_ARCHIVE	The file is an archive one. The application uses this attribute as a mark for file backup or deletion.
FILE_ATTRIBUTE_READONLY	The file is a read-only one. The application can read the file, but cannot program and delete it.

By default, a file is detected independent of the file attribute.

Return values

The return value is `1` if the file exists, and `0` if not. If the function has not been executed

successfully, the return value is `-1`. If `-1` has been returned, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
15	ERROR_INVALID_DRIVE
21	ERROR_NOT_READY
53	ERROR_BAD_NETPATH
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
123	ERROR_INVALID_NAME
148	ERROR_PATH_BUSY
161	ERROR_BAD_PATHNAME
998	ERROR_NOACCESS
1005	ERROR_UNRECOGNIZED_VOLUME
1210	ERROR_INVALID_COMPUTERNAME
1214	ERROR_INVALID_NETNAME
1231	ERROR_NETWORK_UNREACHABLE

AIT_FileGetLine

Description

Reads data from a specified file.

Format

```
bool AIT_FileGetLine  (
```

```
  integer nFileHandle,    // File handle
  string strReadData      // Data to be read from the file
);
```

Parameters

- `nFileHandle` (input)

  Specify a file handle.

- `strReadData` (output)

  Specify a variable for receiving data to be read from the file. When the function returns, the variable stores data.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
21	ERROR_NOT_READY
32	ERROR_SHARING_VIOLATION
33	ERROR_LOCK_VIOLATION
38	ERROR_HANDLE_EOF
112	ERROR_DISK_FULL
148	ERROR_PATH_BUSY
1231	ERROR_NETWORK_UNREACHABLE
1392	ERROR_FILE_CORRUPT

AIT_FileGetPos

Description

Acquires the current position of the file pointer.

Format

```
bool AIT_FileGetPos  (
  integer nFileHandle,    // File handle
  integer nFilePos        // Current file pointer position
);
```

Parameters

- `nFileHandle` (input)

  Specify a file handle.

- `nFilePos` (output)

  Specify a variable for receiving the current position of the file pointer. When the function returns, the variable stores the pointer position.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
21	ERROR_NOT_READY
32	ERROR_SHARING_VIOLATION
33	ERROR_LOCK_VIOLATION
38	ERROR_HANDLE_EOF
112	ERROR_DISK_FULL
148	ERROR_PATH_BUSY

Extended error number	Error code
1231	ERROR_NETWORK_UNREACHABLE
1392	ERROR_FILE_CORRUPT

AIT_FileOpen

Description

Opens an existing file, or creates a new file in set access mode.

Format

```
bool AIT_FileOpen  (
  string strFileName,     // Filename
  integer nAccessMode,    // Access mode
  integer nOperation,     // How to create
  integer nFileHandle     // File handle
);
```

Parameters

- `strFileName` (input)

  Specify the name of a file you want to create or open.

- `nAccessMode` (input)

  Specify mode of access to a file, which must be one or combination of the following values.

Value	Description
GENERIC_READ	Specify access to a file to be read. You can read data from the file and move the file pointer. To set mode of reading/writing access, specify the mode in combination with GENERIC_WRITE.
GENERIC_WRITE	Specify access to a file to be programmed. You can program the file with data, and move the file pointer. To set mode of reading/writing access, specify the mode in combination with GENERIC_READ.

- `nOperation` (input)

  Specify how to handle a file when it exists or does not exist. You have to set one of the following values.

Value	Description
CREATE_NEW	Creates a file. If a specified file already exists, the function will not be executed successfully.

Value	Description
CREATE_ALWAYS	Creates a new file. If a specified file already exists, the function overwrites the file.
OPEN_EXISTING	Opens a file. If a specified file does not exist, the function will not be executed successfully.
OPEN_ALWAYS	Opens a specified file if it exists. If a specified file does not exist, it is created.
TRUNCATE_EXISTING	Opens a file. The opened file is arranged for the 0-byte size. You have to specify at least GENERIC_WRITE access in a calling process to open a file. If no file exists, the function will not be executed successfully.

- nFileHandle (output)

  Specify a variable for receiving a file handle. When the function returns, the variable stores the file handle.

Return values

The return value is true if the function was executed normally, and false if not. If the function has returned false, you can use AIT_GetLastError to acquire an extended error code. The following gives the error codes that AIT_GetLastError might return:

Extended error number	Error code
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
4	ERROR_TOO_MANY_OPEN_FILES
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
15	ERROR_INVALID_DRIVE
18	ERROR_NO_MORE_FILES
19	ERROR_WRITE_PROTECT
21	ERROR_NOT_READY
32	ERROR_SHARING_VIOLATION
33	ERROR_LOCK_VIOLATION

Extended error number	Error code
53	ERROR_BAD_NETPATH
80	ERROR_FILE_EXISTS
82	ERROR_CANNOT_MAKE
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
123	ERROR_INVALID_NAME
148	ERROR_PATH_BUSY
161	ERROR_BAD_PATHNAME
183	ERROR_ALREADY_EXISTS
206	ERROR_FILENAME_EXCED_RANGE
1005	ERROR_UNRECOGNIZED_VOLUME
1210	ERROR_INVALID_COMPUTERNAME
1214	ERROR_INVALID_NETNAME
1231	ERROR_NETWORK_UNREACHABLE
1392	ERROR_FILE_CORRUPT

Note

Use the `AIT_FileClose` function to close a file handle returned by `AIT_OpenFile`.

AIT_FilePutLine

Description

Writes data into a specified file.

Format

```
bool AIT_FilePutLine  (
  integer nFileHandle,     // File handle
  string strWriteData      // Data to be written into a file
);
```

Parameters

- `nFileHandle` (input)

  Specify a file handle.

- `strWriteData` (input)

  Specify data to be written into a file.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
19	ERROR_WRITE_PROTECT
21	ERROR_NOT_READY
32	ERROR_SHARING_VIOLATION
33	ERROR_LOCK_VIOLATION
38	ERROR_HANDLE_EOF
112	ERROR_DISK_FULL
148	ERROR_PATH_BUSY
1231	ERROR_NETWORK_UNREACHABLE
1392	ERROR_FILE_CORRUPT

Notes

If you specify that zero bytes be written, the system considers that null writing operations are specified.

The `AIT_FilePutLine` function writes data at the current file pointer position, which is updated after writing operations.

AIT_FileRename

Description

Change a file or directory name.

Format

```
bool AIT_FileRename (
  string strFileName,       // Current filename
  string strNewFileName     // New filename
);
```

Parameters

- `strFileName` (input)

  Specify a file or directory name you want to change.

- `strNewFileName` (input)

  Specify a new file or directory name.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
15	ERROR_INVALID_DRIVE
19	ERROR_WRITE_PROTECT
21	ERROR_NOT_READY
38	ERROR_HANDLE_EOF
53	ERROR_BAD_NETPATH

Extended error number	Error code
80	ERROR_FILE_EXISTS
82	ERROR_CANNOT_MAKE
87	ERROR_INVALID_PARAMETER
123	ERROR_INVALID_NAME
148	ERROR_PATH_BUSY
161	ERROR_BAD_PATHNAME
1005	ERROR_UNRECOGNIZED_VOLUME
1210	ERROR_INVALID_COMPUTERNAME
1214	ERROR_INVALID_NETNAME
1231	ERROR_NETWORK_UNREACHABLE

AIT_FileSetPos

Description

Sets the file pointer at a specified position.

Format

```
bool AIT_FileSetPos  (
  integer nFileHandle,     // File handle
  integer nSetPos          // New file pointer position
);
```

Parameters

- `nFileHandle` (input)

  Specify a file handle.

- `nSetPos` (input)

  Specify a new file pointer position.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
19	ERROR_WRITE_PROTECT
21	ERROR_NOT_READY
32	ERROR_SHARING_VIOLATION
33	ERROR_LOCK_VIOLATION
38	ERROR_HANDLE_EOF
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
148	ERROR_PATH_BUSY
1231	ERROR_NETWORK_UNREACHABLE
1392	ERROR_FILE_CORRUPT

Notes

You must not use the file pointer indicated by the `nFileHandle` parameter value for duplicate reading or writing operations.

If you execute this function with the new file pointer position set at 0, the current file pointer position will be held.

AIT_FileSize

Description

Acquires a file size.

Format

```
bool AIT_FileSize  (
  integer nFileHandle,     // File handle
  integer nFileSize        // File size
);
```

Parameters

- `nFileHandle` (input)

  Specify a file handle.

- `nFileSize` (output)

  Specify a variable for receiving a file size. When the function returns, the variable stores the file size.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
2	ERROR_FILE_NOT_FOUND
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
21	ERROR_NOT_READY
32	ERROR_SHARING_VIOLATION
33	ERROR_LOCK_VIOLATION
38	ERROR_HANDLE_EOF
112	ERROR_DISK_FULL
148	ERROR_PATH_BUSY
1231	ERROR_NETWORK_UNREACHABLE
1392	ERROR_FILE_CORRUPT

Note

The `AIT_FileSize` function acquires a file size not compressed.

AIT_FindCloseFile

Description

Closes a file search handle returned by the AIT_FindFirstFile function.

Format

```
bool AIT_FindCloseFile  (
  integer nSearchHandle     // File search handle
);
```

Parameters

- nSearchHandle (input)

  Specify a file search handle returned by the AIT_FindFirstFile function.

Return values

The return value is true if the function was executed normally, and false if not. If the function has returned false, you can use AIT_GetLastError to acquire an extended error code. The following gives the error codes that AIT_GetLastError might return:

Extended error number	Error code
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
112	ERROR_DISK_FULL

Note

After having called the AIT_FindClose function, you cannot use the handle set in the nSearchHandle parameter to subsequently call the AIT_FindNextFile or AIT_FindCloseFile function.

AIT_FindFirstFile

Description

Uses a file search handle to return the first filename that matches a specified filename.

Format

```
bool AIT_FindFirstFile  (
  string strFileNamePattern,    // Filename
```

```
  string strFileName,           // Found filename
  integer nSearchHandle         // File search handle
);
```

Parameters

- `strFileNamePattern` (input)

  Specify a valid directory name, path or filename. You can also use a wildcard (`*`). If the character string ends with the wild card, period or directory name, the user must be authorized to access all the subdirectories on the path.

- `strFileName` (output)

  Specify a variable for receiving the name of a found file that matches a set filename. When the function returns, the variable stores the found filename.

- `nSearchHandle` (output)

  Specify a variable for receiving a file search handle to be used to subsequently find `AIT_FindNextFile` and `AIT_FindCloseFile`. When the function returns, the variable stores the file search handle.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
18	ERROR_NO_MORE_FILES
21	ERROR_NOT_READY
112	ERROR_DISK_FULL
123	ERROR_INVALID_NAME
148	ERROR_PATH_BUSY

Extended error number	Error code
1005	ERROR_UNRECOGNIZED_VOLUME
1210	ERROR_INVALID_COMPUTERNAME
1214	ERROR_INVALID_NETNAME
1231	ERROR_NETWORK_UNREACHABLE

Notes

This function uses only a filename to find the file. It cannot use an attribute to find the file. Independent of the presence or absence of subsequent \, you cannot specify the root directory as a `strFileName` input character string for `AIT_FindFirstFile`.

Use `AIT_FindCloseFile` to close a file search handle returned by `AIT_FindFirstFile`.

AIT_FindNextFile

Description

Uses a search handle returned by the `AIT_FindFirstFile` function to find the next file.

Format

```
bool AIT_FindNextFile  (
  integer nSearchHandle,     // File search handle
  string strFileName         // Found filename
);
```

Parameters

- `nSearchHandle` (input)

  Specify a file search handle returned in response to previously called `AIT_FindFirstFile`.

- `strFileName` (output)

  Specify a variable for receiving a found filename. When the function returns, the variable stores the found filename.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
18	ERROR_NO_MORE_FILES
21	ERROR_NOT_READY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
148	ERROR_PATH_BUSY
1231	ERROR_NETWORK_UNREACHABLE

Note

This function uses only a name to find the file. It cannot use any attributes to find the file.

AIT_FindSubStr

Description

Finds a specified character string from the position specified in `nStartPos`, and returns the first matching character string position.

Format

```
integer AIT_FindSubStr  (
  string strStrName,          // Character string
  string strSearchStr         // Character string to be found
  [,integer nStartPos]       // Character position at which to
start finding
);
```

Parameters

- `strStrName` (input)

  Specify a character string.

- `strSearchStr` (input)

  Specify a character string to be found.

- `nStartPos` (input, optional)

  Specify the position at which to start finding a character string. The default `nStartPos` value is `0`, which corresponds to the first character of a character string. By default, this operation starts at the first character.

Return values

This API function returns the 0-based index for the first character of the target character string. This API function returns `-1` if no character string exists or the null character string is specified.

You can use `AIT_GetLastError` to acquire an extended error code. The following gives the error code that `AIT_GetLastError` might return:

Extended error number	Error code
87	ERROR_INVALID_PARAMETER

AIT_FocusWindow

Description

Acquires a window handle having specified window and class names, and sets the focus.

Format

```
integer AIT_FocusWindow  (
  string strWndCaption,     // Window's caption
  string strClassName       // Class name
  [,float fTimeOut]         // Time-out
);
```

Parameters

- `strWndCaption` (input)

  Specify the caption of a window.

- `strClassName` (input)

  Specify a window's class name.

- `fTimeOut` (input, optional)

  Specify the maximum time this function can use to find the control, in units of seconds. The default is the value set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is the window handle if the function was executed normally, and `0` if

not. If the function has returned 0, you can use AIT_GetLastError to acquire an extended error code. The following gives the error codes that AIT_GetLastError might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Note

- With an empty character string set in strWndCaption, the function finds the window with a blank caption. If multiple windows have a blank caption, the focus is set to the first found window.

AIT_GetCtrlText

Description

Acquires text from a specific control in the active window.

Format

```
bool AIT_GetCtrlText  (
  string strCaption,     // Control's caption
  integer nCtrlType,     // Control type
  string strCtrlText     // Control's text
  [,float fTimeOut]      // Time-out
);
bool AIT_GetCtrlText  (
  integer nCtrlID,       // Control ID
  integer nCtrlType,     // Control type
  string strCtrlText     // Control's text
  [,float fTimeOut]      // Time-out
);
```

Parameters

- strCaption (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nCtrlType` (input)

  Specify a control type, which must be one of the following values.

Value	Description
`BUTTON_CTRL`	The control type is a command button.
`CHECKBOX_CTRL`	The control type is a check box.
`OPTIONBUTTON_CTRL`	The control type is an option button.
`EDIT_CTRL`	The control type is an edit box.
`STATIC_CTRL`	The control type is a static text control.
`COMBO_CTRL`	The control type is a combo box.
`LISTBOX_CTRL`	The control type is a list box.
`SPIN_CTRL`	The control type is a spin control.
`TREE_CTRL`	The control type is a tree control.
`LIST_CTRL`	The control type is a list control.
`DTPICKER_CTRL`	The control type is a date/time picker.

- `strCtrlText` (output)

  Specify a variable to receive the text of a control. When the function returns, the variable stores text.

- `fTimeOut` (input, optional)

  Specify the maximum time this function can use to find the control, in units of seconds. The default is the value set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
6	`ERROR_INVALID_HANDLE`
8	`ERROR_NOT_ENOUGH_MEMORY`

Extended error number	Error code
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Notes

- For an edit box, the function acquires a text for its contents. For a static text and button, it acquires a control's caption. For the other controls, it acquires the currently selected item as the text.
- You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_GetCtrlTextLen

Description

Acquires the text length from a specific control in the active window.

Format

```
bool AIT_GetCtrlTextLen  (
  string strCaption,     // Control's caption
  integer nCtrlType,     // Control type
  integer nTextLen       // Control text length
  [,float fTimeOut]      // Time-out
);
bool AIT_GetCtrlTextLen  (
  integer nCtrlID,       // Control ID
  integer nCtrlType,     // Control type
  integer nTextLen       // Control text length
  [,float fTimeOut]      // Time-out
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nCtrlType` (input)

  Specify a control type, which must be one of the following values.

Value	Description
`BUTTON_CTRL`	The control type is a command button.
`CHECKBOX_CTRL`	The control type is a check box.
`OPTIONBUTTON_CTRL`	The control type is an option button.
`EDIT_CTRL`	The control type is an edit box.
`STATIC_CTRL`	The control type is a static text.
`COMBO_CTRL`	The control type is a combo box.
`LISTBOX_CTRL`	The control type is a list box.
`SPIN_CTRL`	The control type is a spin control.
`TREE_CTRL`	The control type is a tree control.
`LIST_CTRL`	The control type is a list control.
`DTPICKER_CTRL`	The control type is a date/time picker.

- `nTextLen` (output)

  Specify a variable to receive the text length of the control. When the function returns, the variable stores the text length.

- `fTimeOut` (input, optional)

  Specify the maximum time this function can use to find the control, in units of seconds. The default is the value set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
6	`ERROR_INVALID_HANDLE`
8	`ERROR_NOT_ENOUGH_MEMORY`

Extended error number	Error code
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_GetCurrentDirectory

Description

Acquires the current directory.

Format

```
bool AIT_GetCurrentDirectory  (
  string strDirName      // Directory name
);
```

Parameters

- `strDirName` (output)

  Specify a variable to receive a directory name. When the function returns, the variable stores the directory name.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY

AIT_GetDate

Description

Acquires the system date in the short format.

Format

```
string AIT_GetDate ();
```

Parameters

None

Return value

The system date is returned in the short format as a character string.

AIT_GetDtPickerDate

Description

Acquires a date from the date/time picker.

Format

```
bool AIT_GetDtPickerDate  (
  string strCaption,    // Control's caption
  string strOutDate     // Control's date
  [,float fTimeOut]     // Time-out
);
bool AIT_GetDtPickerDate  (
  string strCaption,    // Control's caption
  integer nYear,        // Year
  integer nMonth,       // Month
  integer nDay          // Day
  [,float fTimeOut]     // Time-out
);
bool AIT_GetDtPickerDate  (
  integer nCtrlID,      // Control ID
  string strOutDate     // Control's date
  [,float fTimeOut]     // Time-out
);
bool AIT_GetDtPickerDate  (
  integer nCtrlID,     // Control ID
  integer nYear,       // Year
  integer nMonth,      // Month
  integer nDay         // Day
  [,float fTimeOut]    // Time-out
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `strOutDate` (output)

  Specify a variable to receive the date value of a control. When the function returns, the variable stores the date in the *MM/DD/YYYY* format where *MM* indicates the month, *DD* indicates the day, and *YYYY* indicates the year.

- `nYear` (output)

  Specify a variable to receive a control's year value. When the function returns, the variable stores the year value.

- `nMonth` (output)

  Specify a variable to receive a control's month value. When the function returns, the variable stores the month value.

- `nDay` (output)

  Specify a variable to receive a control's day value. When the function returns, the variable stores the day value.

- `fTimeOut` (input, optional)

  Specify the maximum time this function can use to find the control, in units of seconds. The default is the time set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL

Extended error number	Error code
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_GetDtPickerTime

Description

Acquires time for the date/time picker.

Format

```
bool AIT_GetDtPickerTime  (
  string strCaption,    // Control's caption
  string strOutTime     // Control time
  [,float fTimeOut]     // Time-out
);
bool AIT_GetDtPickerTime  (
  string strCaption,    // Control ID
  integer nHour,        // Hours
  integer nMinute,      // Minutes
  integer nSecond       // Seconds
  [,float fTimeOut]     // Time-out
);
bool AIT_GetDtPickerTime  (
  integer nCtrlID,      // Control ID
  string strOutTime     // Control time
  [,float fTimeOut]     // Time-out
);
bool AIT_SetDtPickerTime  (
  integer nCtrlID,      // Control ID
  integer nHour,        // Hours
  integer nMinute,      // Minutes
  integer nSecond       // Seconds
  [,float fTimeOut]     // Time-out
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `strOutTime` (output)

  Sets a variable to receive a control time value. When the function returns, the variable stores the time value in the *hh*:*mm*:*ss* format where *hh* indicates the hour, *mm* indicates the minute, and *ss* indicates the second.

- `nHour` (output)

  Specify a variable to receive a control hour value. When the function returns, the variable stores the hour value.

- `nMinute` (output)

  Specify a variable to receive a control minute value. When the function returns, the variable stores the minute value.

- `nSecond` (output)

  Specify a variable to receive a control seconds value. When the function returns, the variable stores the seconds value.

- `fTimeOut` (input, optional)

  Specify the maximum time this function can use to find the control, in units of seconds. The default is the value set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL

Extended error number	Error code
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_GetEditCurrentLineIndex

Description

Acquires the index of the current line in an edit box containing multiple lines. Data is entered into the current line.

Format

```
bool AIT_GetEditCurrentLineIndex  (
  string strCaption,   // Control's caption
  integer nIndex       // Current line's index
  [,float fTimeOut]    // Time-out
);
bool AIT_GetEditCurrentLineIndex  (
  integer nCtrlID,     // Control ID
  integer nIndex       // Current line's index
  [,float fTimeOut]    // Time-out
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nIndex` (output)

  Specify a variable to receive the index of the current line. When the function returns, the variable stores the index. The default index value is 0.

- `fTimeOut` (input, optional)

  Specify the maximum time this function can use to find the control, in units of seconds. The default is the value set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_GetEditFirstLineIndex

Description

Acquires the index of the first line in the edit box containing multiple lines, or for the first character in the edit box containing a single line.

Format

```
bool AIT_GetEditFirstLineIndex  (
  string strCaption,      // Control's caption
  integer nFirstVisible   // Index for the first line or
character
  [,float fTimeOut]       // Time-out
);
bool AIT_GetEditFirstLineIndex  (
  integer nCtrlID,        // Control ID
  integer nFirstVisible   // Index for the first line or
character
  [,float fTimeOut]       // Time-out
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nFirstVisible` (output)

  Specify the variable for receiving the index value for the first line or character. When the function returns, the variable stores the index. The default index value is 0.

- `fTimeOut` (input, optional)

  Specify the maximum time this function can use to find the control, in units of seconds. The default is the value set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_GetEditTextLineLen

Description

Acquires the length of any line in the edit box containing multiple lines in the active window.

Format

```
bool AIT_GetEditTextLineLen  (
  string strCaption,    // Control's caption
  integer nLineIndex,   // Line's index
  integer nLineLength   // Line length
  [,float fTimeOut]     // Time-out
);
bool AIT_GetEditTextLineLen  (
  integer nCtrlID,      // Control ID
  integer nLineIndex,   // Line's index
  integer nLineLength   // Line length
  [,float fTimeOut]     // Time-out
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nLineIndex` (input)

  Specify the index of the edit box containing multiple lines. The default index value is 0.

- `nLineLength` (output)

  Specify a variable to receive the length of the index of a line set in `nLineIndex`. When the function returns, the variable stores the length.

- `fTimeOut` (input, optional)

  Specify the maximum time this function can use to find the control, in units of seconds. The default is the value set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError`

might return:

Extended error number	Error code
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_GetEnv

Description

Acquires the contents of a set environment variable.

Format

```
string AIT_GetEnv  (
  string strEnvVar     // Environment variable name
);
```

Parameters

- `strEnvVar` (input)

  Specify an environment variable name.

Return value

This API function returns the contents of the environment variable.

AIT_GetErrorText

Description

Acquires the system error text corresponding to a specified error code.

Format

```
string AIT_GetErrorText  (
  integer nErrorCode     // Error code
);
```

Parameters

- `nErrorCode` (input)

  Specify the error code returned by the `AIT_GetLastError` function.

Return value

This API function returns the error message corresponding to a specified error code.

AIT_GetIndexText

Description

Acquires the item text specified by an index from a specific control in the active window.

Format

```
bool AIT_GetIndexText  (
  string strCaption,    // Control's caption
  integer nCtrlType,    // Control type
  integer nIndex,       // Index
  string strItemText    // Item text
  [,float fTimeOut]     // Time-out
);
bool AIT_GetIndexText  (
  integer nCtrlID,      // Control ID
  integer nCtrlType,    // Control type
  integer nIndex,       // Index
  string strItemText    // Item text
  [,float fTimeOut]     // Time-out
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nCtrlType` (input)

  Specify a control type, which must be one of the following values.

Value	Description
COMBO_CTRL	The control type is a comb box.
LISTBOX_CTRL	The control type is a list box.
TREE_CTRL	The control type is a tree control.
LIST_CTRL	The control type is a list control.

- `nIndex` (input)

  Specify the index of the item text you want to acquire. The default index value is 0.

- `strItemText` (output)

  Specify a variable to receive the item text with an index set on the control. When the function returns, the variable stores the item text.

- `fTimeOut` (input, optional)

  Specify the maximum time this function can use to find the control, in units of seconds. The default is the value set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1413	ERROR_INVALID_INDEX
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name

or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_GetIndexTextLen

Description

Acquires the length of the item text specified by an index from a specific control in the active window.

Format

```
bool AIT_GetIndexTextLen  (
  string strCaption,    // Control's caption
  integer nCtrlType,    // Control type
  integer nIndex,       // Index
  integer nTextLen      // Text length
  [,float fTimeOut]     // Time-out
);
bool AIT_GetIndexTextLen  (
  integer nCtrlID,      // Control ID
  integer nCtrlType,    // Control type
  integer nIndex,       // Index
  integer nTextLen      // Text length
  [,float fTimeOut]     // Time-out
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nCtrlType` (input)

  Specify a control type, which must be one of the following values.

Value	Description
`COMBO_CTRL`	The control type is a combo box.
`LISTBOX_CTRL`	The control type is a list box.
`TREE_CTRL`	The control type is a tree control.
`LIST_CTRL`	The control type is a list control.

- `nIndex` (input)

  Specify the index of the item text you want to acquire. The default index value is 0.

- `nTextLen` (output)

  Specify a variable to receive the length of the item text with an index set on the control. When the function returns, the variable stores the length of the item text.

- `fTimeOut` (input, optional)

  Specify the maximum time this function can use to find the control, in units of seconds. The default is the value set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_GetKeyState

Description

Acquires key status.

Format

```
int AIT_GetKeyState  (
  integer nVirtualKey    // Virtual key
```

```
);
```

Parameters

- `nVirtualKey` (input)

  Specify a virtual key code whose key status you want to acquire.

  You have to set one of the following values.

Value	Description
`NUMLOCK`	**Num Lock** key
`SCROLLLOCK`	**Scroll Lock** key
`CAPSLOCK`	**Caps Lock** key

Return values

The return value is `1` if the key is on, and `0` if it is off. The return value is `-1` if the function has not been executed successfully.

If the function has returned `-1`, you can use `AIT_GetLastError` to acquire an extended code. The following gives the error code that `AIT_GetLastError` might return:

Extended error number	Error code
87	ERROR_INVALID_PARAMETER

AIT_GetLastError

Description

Acquires a detailed code on the previously executed function.

Format

```
integer AIT_GetLastError ();
```

Parameters

None

Return value

This API function returns a detailed code, which is the same as a run time error code.

AIT_GetMenu

Description

Used to acquire a menu handle for a window.

Format

```
bool AIT_GetMenu  (
  integer nWndHandle,     // Window handle
  integer nMenu           // Menu handle
  [,float fTimeOut]       // Time-out
);
```

Parameters

- `nWndHandle` (input)

  Specify a window handle.

- `nMenu` (output)

  Specify a variable to receive a menu handle. When the function returns, the variable stores the handle.

- `fTimeOut` (input, optional)

  Specify the maximum time this function can use to find the control, in units of seconds. The default is the value set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

AIT_GetMenuIndex

Description

Acquires the index of a specified menu item.

Format

```
bool AIT_GetMenuIndex  (
  integer nMenu,          // Menu handle
  string strMenuText,     // Menu item
  integer nIndex          // Index for a menu item
  [,float fTimeOut]       // Time-out
);
```

Parameters

- `nMenu` (input)

  Specify a menu handle.

- `strMenuText` (input)

  Specify a menu item.

- `nIndex` (output)

  Specify a variable to receive the index of a menu item. When the function returns, the variable stores the index. The default index value is 0.

- `fTimeOut` (input, optional)

  Specify the maximum time this function can use to find the control, in units of seconds. The default is the value set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
112	ERROR_DISK_FULL
1401	ERROR_INVALID_MENU_HANDLE
1460	ERROR_TIMEOUT

AIT_GetMenuText

Description

Acquires a specified menu item.

Format

```
bool AIT_GetMenuText  (
  integer nMenu,        // Menu handle
  integer nIndex,       // Index for a menu item
  string strMenuText    // Menu item
  [,float fTimeOut]     // Time-out
);
```

Parameters

- `nMenu` (input)

  Specify a menu handle.

- `nIndex` (input)

  Specify the index of a menu item. The default index value is 0.

- `strMenuText` (output)

  Specify a variable to receive a menu item. When the function returns, the variable stores the menu item.

- `fTimeOut` (input, optional)

  Specify the maximum time this function can use to find the control, in units of seconds. The default is the value set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1401	ERROR_INVALID_MENU_HANDLE

Extended error number	Error code
1460	ERROR_TIMEOUT

AIT_GetOSType

Description

Acquires major and minor OS versions, and a platform ID.

Format

```
bool AIT_GetOSType  (
  integer nMajorVersion,    // Major OS version
  integer nMinorVersion,    // Minor OS version
  integer nPlatformID       // Platform ID
);
```

Parameters

- nMajorVersion (output)

  Specify a variable to receive a major OS version. When the function returns, the variable stores one of the following values.

Value	Description
5	Windows XP
5	Windows Server 2003
6	Windows Vista
6	Windows Server 2008
6	Windows 7
6	Windows Server 2012
6	Windows Server 2012 R2
6	Windows 8
6	Windows 8.1

- nMinorVersion (output)

  Specify a variable to receive a minor OS version. When the function returns, the variable stores one of the following values.

Value	Description
1	Windows XP
2	Windows Server 2003
0	Windows Vista
0	Windows Server 2008
1	Windows Server 2008 R2
1	Windows 7
2	Windows Server 2012
3	Windows Server 2012 R2
2	Windows 8
3	Windows 8.1

- `nPlatformID` (output)

  Specify a variable to receive an OS platform ID. When the function returns, the variable stores one of the following values.

Value	Description
VER_PLATFORM_WIN32_NT	Windows 8.1, Windows 8, Windows Server 2012 R2, Windows Server 2012, Windows 7, Windows Server 2008, Windows Vista, Windows Server 2003, or Windows XP

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error code that `AIT_GetLastError` might return:

Extended error number	Error code
122	ERROR_INSUFFICIENT_BUFFER

Note

You have to use a desired version number or a greater number to identify what OS version the application is working on. This allows you to test even a new OS version in the same way.

AIT_GetProfileFirstSection

Description

Acquires the first key and its value from a specified INI file section.

Format

```
bool AIT_GetProfileFirstSection  (
  string strIniFileName,     // INI filename
  string strSectionName,     // INI file's section name
  string strValues           // Section data
);
```

Parameters

- `strIniFileName` (input)

  Specify an INI filename.

- `strSectionName` (input)

  Specify the name of a section in an INI file.

- `strValues` (output)

  Specify a variable to receive a key and its value from a set section. When the function returns, the variable stores a key and its value. A key and its value are combined in the *key=value* format.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
15	ERROR_INVALID_DRIVE

Extended error number	Error code
21	ERROR_NOT_READY
38	ERROR_HANDLE_EOF
53	ERROR_BAD_NETPATH
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
123	ERROR_INVALID_NAME
148	ERROR_PATH_BUSY
161	ERROR_BAD_PATHNAME
998	ERROR_NOACCESS
1005	ERROR_UNRECOGNIZED_VOLUME
1169	ERROR_NO_MATCH
1210	ERROR_INVALID_COMPUTERNAME
1214	ERROR_INVALID_NETNAME
1231	ERROR_NETWORK_UNREACHABLE

AIT_GetProfileFirstSectionNames

Description

Acquires the name of the first section in a specified INI file.

Format

```
bool AIT_GetProfileFirstSectionNames  (
  string strIniFileName,     // INI filename
  string strSectionName      // Section name
);
```

Parameters

- `strIniFileName` (input)

  Specify an INI filename.

- `strSectionName` (output)

  Specify a variable to receive the first section name from a specified INI file. When the function returns, the variable stores the section name.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
15	ERROR_INVALID_DRIVE
21	ERROR_NOT_READY
53	ERROR_BAD_NETPATH
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
123	ERROR_INVALID_NAME
148	ERROR_PATH_BUSY
161	ERROR_BAD_PATHNAME
998	ERROR_NOACCESS
1005	ERROR_UNRECOGNIZED_VOLUME
1169	ERROR_NO_MATCH
1210	ERROR_INVALID_COMPUTERNAME
1214	ERROR_INVALID_NETNAME
1231	ERROR_NETWORK_UNREACHABLE

AIT_GetProfileNextSection

Description

Acquires the next key and its value from a section in an INI file specified in the `AIT_GetProfileFirstSection` function.

Format

```
bool AIT_GetProfileNextSection  (
  string strValues       // Sectional data
);
```

Parameters

- `strValues` (output)

  Specify a variable to receive the next key in a section set in `AIT_GetProfileFirstSection`, and its value. When the function returns, the variable stores the key and its value. A key and its value are combined in the *key=value* format.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the key, the section or the specified INI file does not exist, the function returns `false`.

AIT_GetProfileNextSectionNames

Description

Acquires the next section name from an INI file specified in the `AIT_GetProfileFirstSectionNames` function.

Format

```
bool AIT_GetProfileNextSectionNames  (
  string strSectionName       // Section name
);
```

Parameters

- `strSectionName` (output)

  Specify a variable to receive the next section name in an INI file set in `AIT_GetProfileFirstSectionNames`. When the function returns, the variable stores the section name.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If

the key, the section or the specified INI file does not exist, the function returns `false`.

AIT_GetProfileString

Description

Acquires the value of a specified key from a section in a specified INI file.

Format

```
bool AIT_GetProfileString  (
  string strIniFileName,     // INI filename
  string strSectionName,     // Section name
  string strKeyName,         // Key name
  string strValue            // Key value
);
```

Parameters

- `strIniFileName` (input)

  Specify an INI filename.

- `strSectionName` (input)

  Specify a section name in an INI file.

- `strKeyName` (input)

  Specify a key name belonging to a section name.

- `strValue` (output)

  Specify a variable to receive a key value. When the function returns, the variable stores the key value.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
15	ERROR_INVALID_DRIVE
21	ERROR_NOT_READY
38	ERROR_HANDLE_EOF
53	ERROR_BAD_NETPATH
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
123	ERROR_INVALID_NAME
148	ERROR_PATH_BUSY
161	ERROR_BAD_PATHNAME
998	ERROR_NOACCESS
1005	ERROR_UNRECOGNIZED_VOLUME
1169	ERROR_NO_MATCH
1210	ERROR_INVALID_COMPUTERNAME
1214	ERROR_INVALID_NETNAME
1231	ERROR_NETWORK_UNREACHABLE

AIT_GetSubMenu

Description

Acquires a submenu handle in a menu.

Format

```
bool AIT_GetSubMenu  (
  integer nMenu,      // Menu handle
  integer nIndex,     // Index for a menu item
  integer nSubMenu    // Submenu handle
  [,float fTimeOut]   // Time-out
);
```

Parameters

- `nMenu` (input)

  Specify a menu handle you have got by calling the `AIT_GetMenu` function.

- `nIndex` (input)

  Specify the index of a menu item. The default index value is 0.

- `nSubMenu` (output)

  Specify a variable to receive a submenu handle. When the function returns, the variable stores the handle.

- `fTimeOut` (input, optional)

  Specify the maximum time this function can use to find the control, in units of seconds. The default is the value set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1401	ERROR_INVALID_MENU_HANDLE
1460	ERROR_TIMEOUT

AIT_GetSubStr

Description

This API function returns a character string of specified length from a character string.

Format

```
bool AIT_GetSubStr (
  string  strSubString,      // Extracted character string
  string  strStrName,        // Character string
  integer nStartPos          // Character position at which to
```

```
start extraction
  [,integer nLength]          // Number of characters
);
```

Parameters

- `strSubString` (output)

  Specify a variable to receive an extracted character string. When the function returns, the variable stores the character string.

- `strStrName` (input)

  Specify a character string name.

- `nStartPos` (input)

  Specify a character position at which to start extraction. The default is 0, corresponding to the first character of a character string.

- `nLength` (input, optional)

  Specify the number of characters you want to extract. So long as this parameter does not exceed the number of characters in a character string, the character string with a character count set from `nStartPos` is extracted. The default is the length to the end string character.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error code that `AIT_GetLastError` might return:

Extended error number	Error code
87	ERROR_INVALID_PARAMETER

AIT_GetTime

Description

Acquires system time.

Format

```
string AIT_GetTime ();
```

Parameters

None

Return value

Acquires system time.

AIT_GetWindowText

Description

Acquires the title of a specified window.

Format

```
bool AIT_GetWindowText  (
  integer nWndHandle,     // Window handle
  string strCaption       // Control's caption
);
```

Parameters

- `nWndHandle` (input)

  Specify a window handle. Acquires the title of the active window with 0 set.

- `strCaption` (output)

  Specify a variable to receive a control's caption. When the function returns, the variable stores the caption.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE

AIT_IMEGetConversionStatus

Description

Acquires the current status of IME conversion.

Format

```
bool AIT_IMEGetConversionStatus  (
  integer nWndHandle,     // Window handle
  integer nConvMode,      // Conversion mode
  integer nSentenceMode   // Statement mode
);
```

Parameters

- `nWndHandle` (input)

  Specify a window handle whose status you want to acquire.

  With 0 set, the window with an input focus is used.

- `nConvMode` (output)

  Specify a variable to receive conversion status. When the function returns, the variable stores a combination of the following values.

Value	Description
IME_CMODE_CHARCODE	With this value on, the IME is in character code input mode.
IME_CMODE_EUDC	With this value on, the IME is in EUDC conversion mode.
IME_CMODE_FULLSHAPE	With this value on, the IME is in two-byte mode. With this value off, it is in one-byte mode.
IME_CMODE_HANJACONVERT	With this value on, the IME is in HANJA conversion mode.
IME_CMODE_KATAKANA	With this value on, katakana mode is set. With this value off, kana mode is set.
IME_CMODE_NATIVE	With this value on, NATIVE mode is set. With this value off, ALPHANUMERIC mode is set.
IME_CMODE_NOCONVERSION	With this value on, the IME does not carry out conversion. This is the same as if IME were closed.
IME_CMODE_ROMAN	With this value on, the IME is in alphabetic input mode.
IME_CMODE_SOFTKBD	With this value on, the IME is in soft keyboard mode.
IME_CMODE_SYMBOL	With this value on, the IME is in SYMBOL conversion mode.

- nSentenceMode (output)

  Specify a variable to receive character mode. When the function returns, the variable stores a combination of the following values.

Value	Description
IME_SMODE_AUTOMATIC	The IME is set in automatic conversion mode.
IME_SMODE_NONE	No sentence information.
IME_SMODE_PHRASEPREDICT	The IME uses phrase information to predict the next character. (Continuous clause)
IME_SMODE_PLURALCLAUSE	The IME uses multiple-clause information for conversion processing. (Complex words are prioritized.)
IME_SMODE_SINGLECONVERT	The IME is set in single conversion mode.
IME_SMODE_CONVERSATION	The IME uses conversion mode.

Return values

The return value is true if the function was executed normally, and false if not. If the function has returned false, you can use AIT_GetLastError to acquire an extended error code. The following gives the error codes that AIT_GetLastError might return:

Extended error number	Error code
87	ERROR_INVALID_PARAMETER
1400	ERROR_INVALID_WINDOW_HANDLE

AIT_IMEGetOpenStatus

Description

Checks to see if the IME is open or closed.

Format

```
integer AIT_IMEGetOpenStatus  (
  [integer nWndHandle]     // Window handle
);
```

Parameters

- nWndHandle (input, optional)

  Specify a window handle whose status you want to acquire.

By default, the window with an input focus is used.

Return values

The return value is `1` if the IME is open, `0` if it is closed, and `-1` if the function has not been processed successfully.

If the function has returned `-1`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
87	ERROR_INVALID_PARAMETER
1400	ERROR_INVALID_WINDOW_HANDLE

AIT_IMEGetProperty

Description

Acquires IME properties or functionalities associated with an input focus window.

Format

```
integer AIT_IMEGetProperty  (
  integer nPropertyInfo     // Property information
);
```

Parameters

- `nPropertyInfo` (input)

  Specify property information you want to acquire. You have to set one of the following values.

Value	Description
IGP_PROPERTY	Property information
IGP_CONVERSION	Conversion functionality
IGP_SENTENCE	Sentence mode functionality
IGP_UI	User interface functionality
IGP_SETCOMPSTR	Composition character string functionality
IGP_SELECT	Selection inheritance functionality

Return values

The value returned if the function was executed normally is the property or functionality value corresponding to the `nPropertyInfo` value. In the other cases, the return value is `-1`.

The value returned if `nPropertyInfo` is `IGP_PROPERTY` is a combination of the following values.

Value	Description
`IME_PROP_AT_CARET`	With this value on, the conversion window is at the caret position. If not, it is near the caret.
`IME_PROP_SPECIAL_UI`	With this value on, the IME has no standard user interface. In this case, do not use the application to draw in the IME window.
`IME_PROP_CANDLIST_START_FROM_1`	With this value on, the character strings in a candidate list are numbered in sequence beginning with 1. If not, they are numbered in sequence beginning with 0.
`IME_PROP_UNICODE`	With this value on, the input context character string contains Unicode characters. If not, it contains one- and two-byte characters.

If `nPropertyInfo` is `IGP_UI`, the return value is a combination of the following values.

Value	Description
`UI_CAP_2700`	Supports a value of 0 or 2700 as the text printing direction.
`UI_CAP_ROT90`	Supports values of 0, 900, 1800, and 2700 as the text printing direction.
`UI_CAP_ROTANY`	Supports any printing direction.

If `nPropertyInfo` is `IGP_SETCOMPSTR`, the return value is a combination of the values below.

Value	Description
`SCS_CAP_COMPSTR`	You can use the `SCS_SETSTR` value of `IMESetCompositionString` to create a composition character string.
`SCS_CAP_MAKEREAD`	You can use the `SCS_SETSTR` value of `IMESetCompositionString` to create a read character string from the appropriate composition character string.

If `nPropertyInfo` is `IGP_SELECT`, the return value is a combination of the values below.

Value	Description
SELECT_CAP_CONVERSION	Inherits conversion mode if a new IME has been selected.
SELECT_CAP_SENTENCE	Inherits sentence mode if a new IME has been selected.

If the function has returned -1, you can use AIT_GetLastError to acquire an extended error code.

The following gives the error code that AIT_GetLastError might return:

Extended error number	Error code
87	ERROR_INVALID_PARAMETER

AIT_IMEGetStatusWindowPos

Description

Acquires the position of a status window.

Format

```
bool AIT_IMEGetStatusWindowPos  (
  integer nWndHandle,     // Window handle
  integer nX,             // X coordinate
  integer nY              // Y coordinate
);
```

Parameters

- nWndHandle (input)

  Specify a window handled to be used to acquire a status window position.

  With 0 set, the window handle with an input focus is used.

- nX (output)

  Specify a variable to receive the X coordinate of a status window. When the function returns, the variable stores this value.

- nY (output)

  Specify a variable to receive the Y coordinate of a status window. When the function returns, the variable stores this value.

Return values

The return value is true if the function was executed normally, and false if not. If the function has returned false, you can use AIT_GetLastError to acquire an

extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
87	ERROR_INVALID_PARAMETER
1400	ERROR_INVALID_WINDOW_HANDLE

AIT_IMESetConversionStatus

Description

Sets the current conversion status.

Format

```
bool AIT_IMESetConversionStatus  (
  integer nWndHandle,     // Window handle
  integer nConvMode,      // Conversion mode
  integer nSentenceMode   // Statement mode
);
```

Parameters

- `nWndHandle` (input)

  Specify a window handle to which to set status.

  With 0 set, the window handle with an input focus is used.

- `nConvMode` (input)

  Specify a combination of conversion modes.

  For details on bit values, see *AIT_IMEGetConversionStatus*.

- `nSentenceMode` (input)

  Specify statement mode.

  For details on bit values, see *AIT_IMEGetConversionStatus*.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
87	ERROR_INVALID_PARAMETER
1400	ERROR_INVALID_WINDOW_HANDLE

AIT_IMESetOpenStatus

Description

Opens and closes the IME.

Format

```
bool AIT_IMESetOpenStatus  (
  integer nWndHandle,     // Window handle
  bool bCondition         // Condition
);
```

Parameters

- `nWndHandle` (input)

  Specify a window handle to be assigned status.

  With 0 set, the window handle with an input focus is used.

- `bCondition` (input)

  Specify whether to open or close the IME. Setting `true` opens the IME, while setting `false` closes the IME.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error code that `AIT_GetLastError` might return:

Extended error number	Error code
1400	ERROR_INVALID_WINDOW_HANDLE

AIT_IMESetStatusWindowPos

Description

Sets the position of a status window.

Format

```
bool AIT_IMESetStatusWindowPos  (
  integer nWndHandle,     // Window handle
  integer nX,             // X coordinate
  integer nY              // Y coordinate
);
```

Parameters

- `nWndHandle` (input)

  Specify a window handle to be positioned.

  With 0 set, the window handle with an input focus is used.

- `nX` (input)

  Specify the X coordinate of a status window.

- `nY` (input)

  Specify the Y coordinate of a status window.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
87	ERROR_INVALID_PARAMETER
1400	ERROR_INVALID_WINDOW_HANDLE

AIT_IMESimulateHotKey

Description

Simulates a specified IME hot key.

Format

```
bool AIT_IMESimulateHotKey  (
  integer nWndHandle,     // Window handle
  integer nHotKeyId       // Hot key ID
);
```

Parameters

- `nWndHandle` (input)

  Specify a window handle.

- `nHotKeyId` (input)

  Specify the ID of an IME hot key, which must be one of the following values.

Value	Description
`IME_CHOTKEY_IME_NONIME_TOGGLE`	This hot key used in the simplified Chinese version switches between IME operation and non-IME operation.
`IME_CHOTKEY_SHAPE_TOGGLE`	This hot key used in the simplified Chinese version selects appropriate IME shape conversion mode.
`IME_CHOTKEY_SYMBOL_TOGGLE`	This hot key used in the simplified Chinese version selects appropriate IME shape conversion mode. You can enter Chinese segmentations and two-byte symbols if you assign the keyboard them.
`IME_JHOTKEY_CLOSE_OPEN`	This hot key used in the Japanese version opens and closes the IME.
`IME_KHOTKEY_ENGLISH`	This hot key used in the Korean version translates into English.
`IME_KHOTKEY_SHAPE_TOGGLE`	This hot key used in the Korean version selects appropriate IME shape conversion mode.
`IME_KHOTKEY_HANJACONVERT`	This hot key used in the Korean version selects conversion to Hanja mode.
`IME_THOTKEY_IME_NONIME_TOGGLE`	This hot key used in the traditional Chinese version switches between IME operation and non-IME operation.
`IME_THOTKEY_SHAPE_TOGGLE`	This hot key used in the traditional Chinese version selects appropriate IME shape conversion mode.
`IME_THOTKEY_SYMBOL_TOGGLE`	This hot key used in the traditional Chinese version selects appropriate IME symbol conversion mode.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
87	ERROR_INVALID_PARAMETER
1400	ERROR_INVALID_WINDOW_HANDLE

AIT_InitLog

Description

Initializes a RecDFile.log file to be used by the AIT_LogMessage function. Before executing the AIT_LogMessage function, be sure to execute this function.

The RecDFile.log file exists in the LOG directory path specified by either of the following registry key values.

- When the OS is a 32-bit version:

  HKEY_LOCAL_MACHINE\SOFTWARE\HITACHI\IT DESKTOP MANAGEMENT 2/P\PathName

- When the OS is a 64-bit version:

  HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Hitachi\IT DESKTOP MANAGEMENT 2/P\PathName

Format

```
bool AIT_InitLog  (
  string strMessage     // Message character string
);
```

Parameters

- strMessage (input)

  Specify a character string message you want to write into a log file.

Return values

The return value is true if the function was executed normally, and false if not. If the function has returned false, you can use AIT_GetLastError to acquire an extended error code. The following gives the error codes that AIT_GetLastError might return:

Extended error number	Error code
3	ERROR_PATH_NOT_FOUND
4	ERROR_TOO_MANY_OPEN_FILES

Extended error number	Error code
5	ERROR_ACCESS_DENIED
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
19	ERROR_WRITE_PROTECT
32	ERROR_SHARING_VIOLATION
33	ERROR_LOCK_VIOLATION
53	ERROR_BAD_NETPATH
82	ERROR_CANNOT_MAKE
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
123	ERROR_INVALID_NAME
148	ERROR_PATH_BUSY
161	ERROR_BAD_PATHNAME
183	ERROR_ALREADY_EXISTS
206	ERROR_FILENAME_EXCED_RANGE
1005	ERROR_UNRECOGNIZED_VOLUME
1210	ERROR_INVALID_COMPUTERNAME
1214	ERROR_INVALID_NETNAME
1231	ERROR_NETWORK_UNREACHABLE
1392	ERROR_FILE_CORRUPT

Notes

If you have run this function, the previous `RecDFile.log` is changed to `Rec1File.log`, with a maximum of five files saved as history.

This function records the current date and time as well as messages.

AIT_IsEmpty

Description

Checks to see if the entered character string is empty.

Format

```
bool AIT_IsEmpty (
  string strStrName     // Character string name
);
```

Parameters

- `strStrName` (input)

  Specify a character string name.

Return values

This API function returns `true` if the character string is empty, and `false` if not.

AIT_LogMessage

Description

Saves a message into a `RecDFile.log` file opened by the `AIT_InitLog` function.

Format

```
bool AIT_LogMessage  (
  string strMessage     // Message character string
);
```

Parameters

- `strMessage` (input)

  Specify a character string message you want to write into a log file.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
3	ERROR_PATH_NOT_FOUND
4	ERROR_TOO_MANY_OPEN_FILES

Extended error number	Error code
5	ERROR_ACCESS_DENIED
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
19	ERROR_WRITE_PROTECT
32	ERROR_SHARING_VIOLATION
33	ERROR_LOCK_VIOLATION
53	ERROR_BAD_NETPATH
82	ERROR_CANNOT_MAKE
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
123	ERROR_INVALID_NAME
148	ERROR_PATH_BUSY
161	ERROR_BAD_PATHNAME
183	ERROR_ALREADY_EXISTS
206	ERROR_FILENAME_EXCED_RANGE
1005	ERROR_UNRECOGNIZED_VOLUME
1210	ERROR_INVALID_COMPUTERNAME
1214	ERROR_INVALID_NETNAME
1231	ERROR_NETWORK_UNREACHABLE
1392	ERROR_FILE_CORRUPT

Note

This function records the current date and time as well as messages.

AIT_MenuItemClick

Description

Clicks a specified menu item.

Format

```
bool AIT_MenuItemClick  (
  integer nWndHandle,    // Window handle
  integer nMenu,         // Menu handle
  integer nIndex         // Index for a menu item
  [,float fTimeOut]      // Time-out
);
```

Parameters

- `nWndHandle` (input)

  Specify a window handle.

- `nMenu` (input)

  Specify a menu handle.

- `nIndex` (input)

  Specify the index of a menu item. The default index value is 0.

- `fTimeOut` (input, optional)

  Specify the maximum time this function can use to find the control, in units of seconds. The default is the value set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1401	ERROR_INVALID_MENU_HANDLE
1460	ERROR_TIMEOUT

AIT_MessageBox

Description

Displays a specified message in a dialog box, waits for the user to click a button, and returns the value indicating a button selected by the user.

Format

```
integer AIT_MessageBox  (
  string strMessage,       // Message
  string strTitle          // Title
  [,integer nIconType]     // Icon type
  [,integer nMsgBoxType]   // Message box type
);
```

Parameters

- `strMessage` (input)

  Specify a message to be displayed in a message box.

- `strTitle` (input)

  Specify the title of a message box.

- `nIconType` (input, optional)

  Specify the type of icon to be displayed. This type must be one of the following values:

Value	Description
`MB_ICONEXCLAMATION`	Displays an exclamation mark (`!`) icon in the message box.
`MB_ICONINFORMATION`	Displays an icon having circled **i**.
`MB_ICONQUESTION`	Displays a question mark (`?`) icon in the message box.
`MB_ICONSTOP`	Displays a stop mark icon in the message box.

  The default is the `MB_ICONEXCLAMATION` icon type.

- `nMsgBoxType` (input, optional)

  Specify a message box type, which must be one of the following values.

Value	Description
`MB_ABORTRETRYIGNORE`	Displays a message box that has the **Stop**, **Retry**, and **Ignore** buttons.
`MB_OK`	Displays a message box that has only the **OK** button.
`MB_OKCANCEL`	Displays a message box that has the **OK** and **Cancel** buttons.

Value	Description
MB_RETRYCANCEL	Displays a message box that has the **Retry** and **Cancel** buttons.
MB_YESNO	Displays a message box that has the **Yes** and **No** buttons.
MB_YESNOCANCEL	Displays a message box that has the **Yes**, **No**, and **Cancel** buttons.

The default is the MB_OK message box type.

Return values

This API function returns the value indicating a button the user has selected, which is one of the values below.

Value	Description
IDABORT	You have chosen the **Stop** button.
IDCANCEL	You have chosen the **Cancel** button.
IDIGNORE	You have chosen the **Ignore** button.
IDNO	You have chosen the **No** button.
IDOK	You have chosen the **OK** button.
IDRETRY	You have chosen the **Retry** button.
IDYES	You have chosen the **Yes** button.

If a value other than the above has been returned, you can use AIT_GetLastError to acquire an extended error code. The following gives the error codes that AIT_GetLastError might return:

Extended error number	Error code
87	ERROR_INVALID_PARAMETER

AIT_MinWnd

Description

Minimizes a specified window to activate the next highest-order window.

Format

```
bool AIT_MinWnd  (
  integer nWndHandle     // Window handle
);
bool AIT_MinWnd  (
```

```
  string strCaption,     // Control's caption
  string strClassName    // Class name
);
```

Parameters

- `nWndHandle` (input)

  Specify a window handle.

- `strCaption` (input)

  Specify the caption of a control.

- `strClassName` (input)

  Specify a window's class name.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
5	ERROR_ACCESS_DENIED
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE

AIT_MouseClick

Description

Clicks the mouse at specified coordinates.

Format

```
bool AIT_MouseClick  (
  integer nMouseButton,     // Mouse button
  integer nX,               // X coordinate
  integer nY                // Y coordinate
);
```

Parameters

- nMouseButton (input)

  Specify a mouse button you want to click. You have to set one of the following values.

Value	Description
LBUTTON	Left mouse button
MBUTTOM	Center mouse button
RBUTTON	Right mouse button

- nX (input)

  Specify the X coordinate of a position to be clicked.

- nY (input)

  Specify the Y coordinate of a position to be clicked.

Return values

The return value is true if the function was executed normally, and false if not. If the function has returned false, you can use AIT_GetLastError to acquire an extended error code. The following gives the error codes that AIT_GetLastError might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL

AIT_MouseDblClk

Description

Double-clicks the mouse at specified coordinates.

Format

```
bool AIT_MouseDblClk  (
  integer nX,             // X coordinate
  integer nY,             // Y coordinate
  integer nButton,        // Mouse button
  integer nKeyState       // Key status
```

```
);
```

Parameters

- `nX` (input)

  Specify the X coordinate of a position to be double-clicked.

- `nY` (input)

  Specify the Y coordinate of a position to be double-clicked.

- `nButton` (input)

  Specify a mouse button to be double-clicked. You have to set one of the following values.

Value	Description
`LBUTTON`	Left mouse button
`MBUTTON`	Center mouse button
`RBUTTON`	Right mouse button

- `nKeyState` (input)

  Specify key status, which must be one of the following values.

Value	Description
`SHIFT_ON`	The **Shift** key is on.
`ALT_ON`	The **Alt** key is on.
`CTRL_ON`	The **Ctrl** key is on.
`SHIFT_OFF`	The **Shift** key is off.
`ALT_OFF`	The **Alt** key is off.
`CTRL_OFF`	The **Ctrl** key is off.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	`ERROR_NOT_ENOUGH_MEMORY`

Extended error number	Error code
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL

AIT_MouseDown

Description

Presses a mouse button at specified coordinates.

Format

```
bool AIT_MouseDown  (
  integer nX,              // X coordinate
  integer nY,              // Y coordinate
  integer nButton,         // Mouse button
  integer nKeyState        // Key status
);
```

Parameters

- nX (input)

  Specify the X coordinate of a position at which to press a mouse button.

- nY (input)

  Specify the Y coordinate of a position at which to press a mouse button.

- nButton (input)

  Specify a button to be pressed by the mouse. You have to set one of the following values.

Value	Description
LBUTTON	Left mouse button
MBUTTON	Center mouse button
RBUTTON	Right mouse button

- nKeyState (input)

  Specify key status, which must be one of the following values.

Value	Description
SHIFT_ON	The **Shift** key is on.
ALT_ON	The **Alt** key is on.
CTRL_ON	The **Ctrl** key is on.
SHIFT_OFF	The **Shift** key is off.
ALT_OFF	The **Alt** key is off.
CTRL_OFF	The **Ctrl** key is off.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL

AIT_MouseDragDrop

Description

Drags and drops the section from a specified start position to a specified end position.

Format

```
bool AIT_MouseDragDrop  (
  integer nXStartPos,       // Start X coordinate
  integer nYStartPos,       // Start Y coordinate
  integer nXEndPos,         // End X coordinate
  integer nYEndPos,         // End Y coordinate
  integer nButton,          // Mouse button
  integer nKeyState         // Key status
);
```

Parameters

- `nXStartPos` (input)

  Specify the X coordinate of a position at which to start dragging.

- `nYStartPos` (input)

  Specify the Y coordinate of a position at which to start dragging.

- `nXEndPos` (input)

  Specify the X coordinate of a position at which to drop the section.

- `nYEndPos` (input)

  Specify the Y coordinate of a position at which to drop the section.

- `nButton` (input)

  Specify a mouse button to be used for dragging and dropping. You have to set one of the following values.

Value	Description
`LBUTTON`	Left mouse button
`MBUTTON`	Center mouse button
`RBUTTON`	Right mouse button

- `nKeyState` (input)

  Specify key status, which must be one of the following values.

Value	Description
`SHIFT_ON`	The **Shift** key is on.
`ALT_ON`	The **Alt** key is on.
`CTRL_ON`	The **Ctrl** key is on.
`SHIFT_OFF`	The **Shift** key is off.
`ALT_OFF`	The **Alt** key is off.
`CTRL_OFF`	The **Ctrl** key is off.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError`

might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL

AIT_MouseMoveTo

Description

Moves the mouse to specified coordinates.

Format

```
bool AIT_MouseMoveTo  (
  integer nX,              // X coordinate
  integer nY,              // Y coordinate
  integer nButton,         // Mouse button
  integer nKeyState        // Key status
);
```

Parameters

- nX (input)

  Specify the X coordinate of a position to which to move the mouse.

- nY (input)

  Specify the Y coordinate of a position to which to move the mouse.

- nButton (input)

  Specify a mouse button to be clicked during movement. You have to set one of the following values.

Value	Description
LBUTTON	Left mouse button
MBUTTON	Center mouse button
RBUTTON	Right mouse button

- `nKeyState` (input)

  Specify key status, which must be one of the following values.

Value	Description
`SHIFT_ON`	The **Shift** key is on.
`ALT_ON`	The **Alt** key is on.
`CTRL_ON`	The **Ctrl** key is on.
`SHIFT_OFF`	The **Shift** key is off.
`ALT_OFF`	The **Alt** key is off.
`CTRL_OFF`	The **Ctrl** key is off.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	`ERROR_NOT_ENOUGH_MEMORY`
14	`ERROR_OUTOFMEMORY`
87	`ERROR_INVALID_PARAMETER`
112	`ERROR_DISK_FULL`

AIT_MouseUp

Description

Releases the mouse button at specified coordinates.

Format

```
bool AIT_MouseUp  (
  integer nX,          // X coordinate
  integer nY,          // Y coordinate
  integer nButton,     // Mouse button
  integer nKeyState    // Key status
);
```

Parameters

- `nX` (input)

  Specify the X coordinate of a position at which to release the mouse button.

- `nY` (input)

  Specify the Y coordinate of a position at which to release the mouse button.

- `nButton` (input)

  Specify a mouse button to be released. You have to set one of the following values.

Value	Description
`LBUTTON`	Left mouse button
`MBUTTON`	Center mouse button
`RBUTTON`	Right mouse button

- `nKeyState` (input)

  Specify key status, which must be one of the following values.

Value	Description
`SHIFT_ON`	The **Shift** key is on.
`ALT_ON`	The **Alt** key is on.
`CTRL_ON`	The **Ctrl** key is on.
`SHIFT_OFF`	The **Shift** key is off.
`ALT_OFF`	The **Alt** key is off.
`CTRL_OFF`	The **Ctrl** key is off.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	`ERROR_NOT_ENOUGH_MEMORY`
14	`ERROR_OUTOFMEMORY`

Extended error number	Error code
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL

AIT_PlayKey

Description

Sends keyboard input information to the active window as if the user had typed that information.

Format

```
bool AIT_PlayKey  (
  string strKeys     // Character string
);
```

Parameters

- `strKeys` (input)

  Specify a key or a combination of key character strings, or a character string. You can specify a combination of the following characters.

  - All the uppercase and lowercase alphabetic characters from a to z
  - All the numeric characters from 0 to 9
  - Following special characters:

    ```
    ~ ! @ # $ % ^ & * ( ) _ + | ? > < " } { [ ] ' ; / . , ` - = \
    ```

  Example:

  `AIT_PlayKey("ABC")` simulates processing for typing a character string of ABC.

  The following gives the special keys you can specify:

Special keys that can be specified						
{ALT}	{ALT_LOCK}	{ALT_UNLOCK}	{ALT+SHIFT}	{APPS}	{BACKSPACE}	{CTRL}
{CTRL+ALT}	{CTRL+SHIFT}	{CTRL+ALT+SHIFT}	{CTRL_LOCK}	{CAPSLOCK}	{CAPSLOCK_ON}	{CAPSLOCK_OFF}
{CTRL_UNLOCK}	{DELETE}	{DOWN}	{END}	{ENTER}	{ESC}	{F1}

Special keys that can be specified						
{F2}	{F3}	{F4}	{F5}	{F6}	{F7}	{F8}
{F9}	{F10}	{F11}	{F12}	{HOME}	{INS}	{KANA_ON}
{KANA_OFF}	{KANJI_ON}	{KANJI_OFF}	{LEFT}	{LWIN}	{LWIN_LOCK}	{LWIN_UNLOCK}
{NUMLOCK_ON}	{NUMLOCK_OFF}	{NUMLOCK}	{NUMPAD0}	{NUMPAD1}	{NUMPAD2}	{NUMPAD3}
{NUMPAD4}	{NUMPAD5}	{NUMPAD6}	{NUMPAD7}	{NUMPAD8}	{NUMPAD9}	{PAUSE}
{PGDOWN}	{PGUP}	{PRNSCR}	{PROCESS}	{RIGHT}	{RWIN}	{RWIN_LOCK}
{RWIN_UNLOCK}	{SCROLLLOCK}	{SCROLLLOCK_ON}	{SCROLLLOCK_OFF}	{SHIFT}	{SHIFT_LOCK}	{SHIFT_UNLOCK}
{SPACE}	{TAB}	{UP}	-	-	-	-

Legend:

-: No value.

Example:

`AIT_PlayKey("{TAB}")` simulates processing for pressing the **Tab** key.

When specifying a key in combination with the **Shift**, **Ctrl**, or **Alt** key, add the following key operation codes before a regular key text.

Key	Key operation code
Shift	+
Ctrl	^
Alt	%

Example:

`AIT_PlayKey("%(N)")` indicates that **Alt**+**N** key is pressed.

To repeat keys in the same order, type:

{REPEAT *n*}*character-to-repeat*{END_REPEAT}

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an

extended error code. The following gives the error code that AIT_GetLastError might return:

Extended error number	Error code
87	ERROR_INVALID_PARAMETER

AIT_PostMessage

Description

Posts a message to a message queue associated with the thread used to create a specified window. This API function returns the control without waiting for that thread to process the message.

Format

```
bool AIT_PostMessage  (
  integer nWndHandle,     // Window handle
  integer nMessage,       // Message
  integer nWParam,        // First parameter for a message
  integer nLParam         // Second parameter for a message
);
```

Parameters

- nWndHandle (input)

  Specify a window handle to which to post a message.

  The following value has a special meaning.

Value	Description
HWND_BROADCAST	Posts a message to all the top-level windows in the system, including an invalid window not owned, an invisible window not owned, an overlapped window (hidden by another front window), and a pop-up window. Does not post a message to any child windows.

- nMessage (input)

  Specify a message you want to post.

- nWParam (input)

  Specify additional information particular to a message.

- nLParam (input)

  Specify additional information particular to a message.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER

AIT_RegCloseKey

Description

Closes a handle to a specified registry key.

Format

```
bool AIT_RegCloseKey(
  integer nHKeyHandle     // Key handle
);
```

Parameters

- `nHKeyHandle` (input)

  Specify an already opened registry key.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
6	ERROR_INVALID_HANDLE
87	ERROR_INVALID_PARAMETER
1009	ERROR_BADDB
1010	ERROR_BADKEY
1015	ERROR_REGISTRY_CORRUPT

Extended error number	Error code
1016	ERROR_REGISTRY_IO_FAILED
1019	ERROR_NO_LOG_SPACE

AIT_RegCreateKey

Description

Creates a specified registry key. Opens a specified registry key if it already exists.

Format

```
bool AIT_RegCreateKey(
  integer nHKeyHandle,       // Key handle
  string strRegKeyName,      // Name of a key to be created
  integer nOutputHkeyHandle  // Output key handle
);
```

Parameters

- `nHKeyHandle` (input)

  Specify a handle to an already opened registry key, or one of the following values.

  - `HKEY_CLASSES_ROOT`
  - `HKEY_CURRENT_CONFIG`
  - `HKEY_CURRENT_USER`
  - `HKEY_LOCAL_MACHINE`
  - `HKEY_USERS`

- `strRegKeyName` (input)

  Specify the name of a registry key to be created or opened.

- `nOutputHkeyHandle` (output)

  Specify a variable to receive a handle to a registry key. When the function returns, the variable stores the handle.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
6	ERROR_INVALID_HANDLE
87	ERROR_INVALID_PARAMETER
1009	ERROR_BADDB
1010	ERROR_BADKEY
1015	ERROR_REGISTRY_CORRUPT
1016	ERROR_REGISTRY_IO_FAILED
1019	ERROR_NO_LOG_SPACE

Note

- Before termination, call `AIT_RegCloseKey` to close `nOutputHkeyHandle`.
- To create a registry key, you must have permission to write at the location where you want to create the registry key.
- If you attempt to open an existing registry key on which you do not have write permission, the registry key opens in the read-only mode.

AIT_RegDeleteKey

Description

Deletes a sub-key for a specified registry.

Format

```
bool AIT_RegDeleteKey(
  integer nHKeyHandle,     // Key handle
  string strRegKeyName     // Name of a key to be deleted
);
```

Parameters

- `nHKeyHandle` (input)

  Specify a handle to an already opened registry key.

- `strRegKeyName` (input)

  Specify a sub-key name for a registry to be deleted.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If

the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
2	ERROR_FILE_NOT_FOUND
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
87	ERROR_INVALID_PARAMETER
1009	ERROR_BADDB
1010	ERROR_BADKEY
1015	ERROR_REGISTRY_CORRUPT
1016	ERROR_REGISTRY_IO_FAILED
1019	ERROR_NO_LOG_SPACE

Note

When the OS of the computer is Windows, if the specified key has a sub-key, you cannot delete the specified key. Before deleting a key, delete all of its sub-keys.

AIT_RegDeleteValue

Description

Deletes a specified registry value.

Format

```
bool AIT_RegDeleteValue(
  integer nHKeyHandle,     // Key handle
  string strRegValueName   // Registry value name
);
```

Parameters

- `nHKeyHandle` (input)

  Specify a handle to an already opened registry key.

- `strRegValueName` (input)

  Specify a registry key name you want to delete.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
2	ERROR_FILE_NOT_FOUND
6	ERROR_INVALID_HANDLE
87	ERROR_INVALID_PARAMETER
1009	ERROR_BADDB
1010	ERROR_BADKEY
1015	ERROR_REGISTRY_CORRUPT
1016	ERROR_REGISTRY_IO_FAILED
1019	ERROR_NO_LOG_SPACE

AIT_RegGetDWORDValue

Description

Acquires a registry value of the `DWORD`-type.

Format

```
bool AIT_RegGetDWORDValue(
  integer nHKeyHandle,      // Key handle
  string strRegKeyName,     // Registry sub-key name
  string strRegValueName,   // Registry value name
  integer nRegValueData     // Registry value data
);
```

Parameters

- `nHKeyHandle` (input)

  Specify a handle to an already opened registry key, or one of the following values.

  - `HKEY_CLASSES_ROOT`
  - `HKEY_CURRENT_CONFIG`
  - `HKEY_CURRENT_USER`

- HKEY_LOCAL_MACHINE
- HKEY_USERS

- strRegKeyName (input)

  Specify a sub-key name for a registry.

- strRegValueName (input)

  Specify a registry value name you want to acquire.

- nRegValueData (output)

  Specify a variable to receive registry value data of the DWORD-type. When the function returns, the variable stores the registry value data.

Return values

The return value is true if the function was executed normally, and false if not. If the function has returned false, you can use AIT_GetLastError to acquire an extended error code. The following gives the error codes that AIT_GetLastError might return:

Extended error number	Error code
2	ERROR_FILE_NOT_FOUND
6	ERROR_INVALID_HANDLE
87	ERROR_INVALID_PARAMETER
234	ERROR_MORE_DATA
1009	ERROR_BADDB
1010	ERROR_BADKEY
1011	ERROR_CANTOPEN
1012	ERROR_CANTREAD
1015	ERROR_REGISTRY_CORRUPT
1016	ERROR_REGISTRY_IO_FAILED
1018	ERROR_KEY_DELETED
1019	ERROR_NO_LOG_SPACE
1069	ERROR_NO_MATCH

AIT_RegGetStringValue

Description

Acquires a registry value of the character string data type (`REG_SZ`, `REG_EXPAND_SZ`, or `REG_MULTI_SZ`).

Format

```
bool AIT_RegGetStringValue(
  integer nHKeyHandle,      // Key handle
  string strRegKeyName,     // Registry sub-key name
  string strRegValueName,   // Registry value name
  string strRegValueData    // Registry value data
);
```

Parameters

- `nHKeyHandle` (input)

  Specify a handle to an already opened registry key, or one of the following values.

  - `HKEY_CLASSES_ROOT`
  - `HKEY_CURRENT_CONFIG`
  - `HKEY_CURRENT_USER`
  - `HKEY_LOCAL_MACHINE`
  - `HKEY_USERS`

- `strRegKeyName` (input)

  Specify a registry sub-key name.

- `strRegValueName` (input)

  Specify a registry value name you want to acquire.

- `strRegValueData` (output)

  Specify a variable to receive registry value data of the character string data type. When the function returns, the variable stores the registry data value.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
2	ERROR_FILE_NOT_FOUND
6	ERROR_INVALID_HANDLE
87	ERROR_INVALID_PARAMETER
234	ERROR_MORE_DATA
1009	ERROR_BADDB
1010	ERROR_BADKEY
1011	ERROR_CANTOPEN
1012	ERROR_CANTREAD
1015	ERROR_REGISTRY_CORRUPT
1016	ERROR_REGISTRY_IO_FAILED
1018	ERROR_KEY_DELETED
1019	ERROR_NO_LOG_SPACE
1169	ERROR_NO_MATCH

AIT_RegisterWindowMessage

Description

Defines a new window message particular to the system. You can use this message value to post or send a message.

Typically, this function is used to register a message to be applied to communication between two cooperating applications.

Format

```
integer AIT_RegisterWindowMessage  (
  string strMessageString     // Message character string
);
```

Parameters

- `strMessageString` (input)

  Specify a message you want to register.

Return values

The return value is a message value (message ID) ranging from 49152 to 65535 if the message has been registered normally.

The return value is `0` if the function has not been processed successfully. You can use `AIT_GetLastError` to acquire an extended error code.

The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER

AIT_RegKeyExists

Description

Checks whether the specified registry key exists.

Format

```
integer AIT_RegKeyExists(
  integer nHKeyHandle,       // Key handle
  string strRegKeyName       // Registry sub-key name
);
```

Parameters

- `nHKeyHandle` (input)

  Specify a handle to an already opened registry key.

- `strRegKeyName` (input)

  Specify the name of a registry sub-key to check if it exists.

Return values

The return value is `1` if the key exists, `0` if not, and `-1` if the function has not been processed successfully.

If the function has returned `-1`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
2	ERROR_FILE_NOT_FOUND
6	ERROR_INVALID_HANDLE
87	ERROR_INVALID_PARAMETER
1009	ERROR_BADDB
1010	ERROR_BADKEY
1011	ERROR_CANTOPEN
1015	ERROR_REGISTRY_CORRUPT
1016	ERROR_REGISTRY_IO_FAILED
1019	ERROR_NO_LOG_SPACE

AIT_RegOpenKey

Description

Opens a specified registry key.

Format

```
bool AIT_RegOpenKey(
  integer nHKeyHandle,       // Key handle
  string strRegKeyName,      // Name of a key to be opened
  integer nOutputHkeyHandle  // Output key handle
);
```

Parameters

- `nHKeyHandle` (input)

  Specify a handle to an already opened registry key, or one of the following values.

  - `HKEY_CLASSES_ROOT`
  - `HKEY_CURRENT_CONFIG`
  - `HKEY_CURRENT_USER`
  - `HKEY_LOCAL_MACHINE`
  - `HKEY_USERS`

- `strRegKeyName` (input)

  Specify a sub-key name for a registry to be opened.

- `nOutputHkeyHandle` (output)

  Specify a variable to receive a handle to a registry key. When the function returns, the variable stores the handle.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
2	ERROR_FILE_NOT_FOUND
6	ERROR_INVALID_HANDLE
87	ERROR_INVALID_PARAMETER
1009	ERROR_BADDB
1010	ERROR_BADKEY
1011	ERROR_CANTOPEN
1015	ERROR_REGISTRY_CORRUPT
1016	ERROR_REGISTRY_IO_FAILED
1018	ERROR_KEY_DELETED
1019	ERROR_NO_LOG_SPACE

Note

- Before termination, call `AIT_RegCloseKey` to close `nOutputHkeyHandle`.
- If you attempt to open a registry key for which you do not have write permission, the registry key opens in the read-only mode.

AIT_RegSetDWORDValue

Description

Sets data to a registry value of the `DWORD`-type.

Format

```
bool AIT_RegSetDWORDValue(
  integer nHKeyHandle,      // Key handle
  string strRegKeyName,     // Registry sub-key name
```

```
  string strRegValueName,   // Registry value name
  integer nRegValueData     // Registry value data
);
```

Parameters

- `nHKeyHandle` (input)

  Specify a handle to an already opened registry key, or one of the following values.

  - `HKEY_CLASSES_ROOT`
  - `HKEY_CURRENT_CONFIG`
  - `HKEY_CURRENT_USER`
  - `HKEY_LOCAL_MACHINE`
  - `HKEY_USERS`

- `strRegKeyName` (input)

  Specify a sub-key name for a registry.

- `strRegValueName` (input)

  Specify the name of a registry value you want to set.

- `nRegValueData` (input)

  Specify data you want to set to a registry value of the `DWORD`-type.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
6	ERROR_INVALID_HANDLE
87	ERROR_INVALID_PARAMETER
234	ERROR_MORE_DATA
1009	ERROR_BADDB
1010	ERROR_BADKEY
1011	ERROR_CANTOPEN
1013	ERROR_CANTWRITE

Extended error number	Error code
1015	ERROR_REGISTRY_CORRUPT
1016	ERROR_REGISTRY_IO_FAILED
1018	ERROR_KEY_DELETED
1019	ERROR_NO_LOG_SPACE
1169	ERROR_NO_MATCH

AIT_RegSetStringValue

Description

Sets data to a registry value of the character string data type (REG_SZ, REG_EXPAND_SZ, or REG_MULTI_SZ).

Format

```
bool AIT_RegSetStringValue(
  integer nHKeyHandle,      // Key handle
  string strRegKeyName,     // Registry sub-key name
  string strRegValueName,   // Registry value name
  string strRegValueData    // Registry value data
);
```

Parameters

- nHKeyHandle (input)

  Specify a handle to an already opened registry key, or one of the following values.

  - HKEY_CLASSES_ROOT
  - HKEY_CURRENT_CONFIG
  - HKEY_CURRENT_USER
  - HKEY_LOCAL_MACHINE
  - HKEY_USERS

- strRegKeyName (input)

  Specify a registry sub-key name.

- strRegValueName (input)

  Specify a registry value name you want to set.

- `strRegValueData` (input)

  Specify data you want to set to a registry value of the character string data type.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
6	ERROR_INVALID_HANDLE
87	ERROR_INVALID_PARAMETER
234	ERROR_MORE_DATA
1009	ERROR_BADDB
1010	ERROR_BADKEY
1011	ERROR_CANTOPEN
1013	ERROR_CANTWRITE
1015	ERROR_REGISTRY_CORRUPT
1016	ERROR_REGISTRY_IO_FAILED
1018	ERROR_KEY_DELETED
1019	ERROR_NO_LOG_SPACE
1169	ERROR_NO_MATCH

Notes

- When you set character string data to a new registry value, the data type is `REG_SZ`. You cannot change the data type.
- When you use this API function to update an existing registry value, you cannot change the data type.

AIT_RegValueExists

Description

Checks whether the specified registry value exists.

Format

```
integer AIT_RegValueExists(
  integer nHKeyHandle,     // Key handle
  string strRegValueName   // Registry value name
);
```

Parameters

- `nHKeyHandle` (input)

  Specify a handle to an already opened registry key.

- `strRegValueName` (input)

  Specify the name of a registry value to check if it exists.

Return values

The return value is `1` if this value exists, `0` if not, and `-1` if the function has not been processed successfully.

If the function has returned `-1`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
2	ERROR_FILE_NOT_FOUND
6	ERROR_INVALID_HANDLE
87	ERROR_INVALID_PARAMETER
236	ERROR_MORE_DATA
1009	ERROR_BADDB
1010	ERROR_BADKEY
1011	ERROR_CANTOPEN
1015	ERROR_REGISTRY_CORRUPT
1016	ERROR_REGISTRY_IO_FAILED
1019	ERROR_NO_LOG_SPACE

AIT_SelectIPAddressField

Description

Selects a text in the IP address control on the active window.

Format

```
bool AIT_SelectIPAddressField  (
  string strCaption,       // Control's caption
  integer nFieldIndex,     // Field index
  integer nStartSel,       // Text start position
  integer nEndSel          // Text count
  [,float fTimeOut]        // Time-out
);
bool AIT_SelectIPAddressField  (
  integer nCtrlID,         // Control ID
  integer nFieldIndex,     // Field index
  integer nStartSel,       // Text start position
  integer nEndSel          // Text count
  [,float fTimeOut]        // Time-out
);
```

Parameters

- `strCaption` (input)

  Specify the control's caption.

- `nCtrlID` (input)

  Specify the control ID.

- `nFieldIndex` (input)

  Specify the index of a field in the IP address control. The default index value is 0.

- `nStartSel` (input)

  Specify the start character position at which to select text. The default `nStartSel` value is 0, corresponding to the first text character.

- `nEndSel` (input)

  Specify the number of texts you want to select.

- `fTimeOut` (input, optional)

  Specify the maximum time this function can use to find the control, in units of seconds. The default is the value set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Notes

The start value may be greater than the end value. The smaller of the two values refers to the first character position of selected text. The greater value refers to the first character position beyond selected text.

The start value is a fixed selected text point, while the end value is a varying end point.

If the start point is 0 and the end point is -1, all the texts in the edit control are selected. If the start point is -1, active selection is released.

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_SelectListItem

Description

Selects an item specified in a specific control on the active window.

Format

```
bool AIT_SelectListItem  (
  string strCaption,    // Control's caption
  integer nCtrlType,    // Control type
  string strItemText    // Item text selected
  [,float fTimeOut]     // Time-out
);
bool AIT_SelectListItem  (
  integer nCtrlID,      // Control ID
  integer nCtrlType,    // Control type
  string strItemText    // Item text selected
  [,float fTimeOut]     // Time-out
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nCtrlType` (input)

  Specify a control type, which must be one of the following values.

Value	Description
`COMBO_CTRL`	The control type is a combo box.
`LISTBOX_CTRL`	The control type is a list box.
`LIST_CTRL`	The control type is a list control.

- `strItemText` (input)

  Specify text you want to select on a specific control.

- `fTimeOut` (input, optional)

  Specify the maximum time this function can use to find the control, in units of seconds. The default is the value set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
6	`ERROR_INVALID_HANDLE`
8	`ERROR_NOT_ENOUGH_MEMORY`
14	`ERROR_OUTOFMEMORY`
87	`ERROR_INVALID_PARAMETER`
112	`ERROR_DISK_FULL`
1400	`ERROR_INVALID_WINDOW_HANDLE`
1460	`ERROR_TIMEOUT`

Notes

Do not use the `AIT_SelectListItem` function for controls that allow selection of multiple values.

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_SelectMultipleListItem

Description

Selects the specified items in a specific multi-selection control on the active window.

Format

```
bool AIT_SelectMultipleListItem  (
  string strCaption,    // Control's caption
  integer nCtrlType,    // Control type
  string strItemText    // Item text selected
  [,float fTimeOut]     // Time-out
);
bool AIT_SelectMultipleListItem  (
  integer nCtrlID,      // Control ID
  integer nCtrlType,    // Control type
  string strItemText    // Item text selected
  [,float fTimeOut]     // Time-out
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nCtrlType` (input)

  Specify a control type, which must be one of the following values.

Value	Description
`LISTBOX_CTRL`	The control type is a list box.
`LIST_CTRL`	The control is a list control.

- `strItemText` (input)

  Specify the text items you want to select in the multi-selection control. You can specify multiple values delimited by a comma.

- `fTimeOut` (input, optional)

  Specify the maximum time this function can use to find the control, in units of seconds. The default is the value set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Notes

Use the `AIT_SelectMultipleListItem` function only for controls that allow selection of multiple values.

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_SelectText

Description

Selects text in the edit control on the active window.

Format

```
bool AIT_SelectText  (
  string strCaption,   // Control's caption
```

```
  integer nStartPos,   // Control start position
  integer nEndPos      // Control end position
  [,float fTimeOut]    // Time-out
);
bool AIT_SelectText  (
  integer nCtrlID,     // Control ID
  integer nStartPos,   // Control start position
  integer nEndPos      // Control end position
  [,float fTimeOut]    // Time-out
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nStartPos` (input)

  Specify a start character position at which to select text. The default `nStartPos` value is 0, corresponding to the first text character.

- `nEndPos` (input)

  Specify the end character position at which to select text.

- `fTimeOut` (input, optional)

  Specify the maximum time this function can use to find the control, in units of seconds. The default is the value set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE

Extended error number	Error code
1460	ERROR_TIMEOUT

Notes

The start value may be greater than the end value. The smaller of the two values refers to the first character position of selected text. The greater value refers to the first character position beyond selected text.

The start value is a fixed selected text point, while the end value is a varying end point.

If the start point is 0 and the end point is -1, all the texts in the edit control are selected. If the start point is -1, active selection is released.

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_SetActWnd

Description

Activates a specified window.

Format

```
bool AIT_SetActWnd  (
  integer nWndHandle     // Window handle
);
```

Parameters

- `nWndHandle` (input)

  Specify a window's handle.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER

Extended error number	Error code
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE

AIT_SetCheck

Description

Turns on and off, and deactivates a specific control on the active window.

Format

```
bool AIT_SetCheck  (
  string strCaption,     // Control's caption
  integer nCtrlType      // Control type
  [,integer nCondition]  // Check type
  [,float fTimeOut]      // Time-out
);
bool AIT_SetCheck  (
  integer nCtrlID,       // Control ID
  integer nCtrlType      // Control type
  [,integer nCondition]  // Check status type
  [,float fTimeOut]      // Time-out
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nCtrlType` (input)

  Specify a control type, which must be one of the following values.

Value	Description
CHECKBOX_CTRL	The control type is a check box.
OPTIONBUTTON_CTRL	The control type is an option button.

- `nCondition` (input, optional)

  Specify a check status type, which must be one of the following values.

Value	Description
0	Turned off
1	Turned on
2	Deactivated or grayed

- fTimeOut (input, optional)

  Specify the maximum time this function can use to find the control, in units of seconds. The default is the value set in the AIT_SetDefaultWaitTimeout function.

Return values

The return value is true if the function was executed normally, and false if not. If the function has returned false, you can use AIT_GetLastError to acquire an extended error code. The following gives the error codes that AIT_GetLastError might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_SetComboEditSelText

Description

Sets text selection in a combo box on the active window.

Format

```
bool AIT_SetComboEditSelText  (
  string strCaption,    // Control's caption
  integer nStartSel,    // Control start position
```

```
  integer nEndSel       // Control end position
  [,float fTimeOut]     // Time-out
);
bool AIT_SetComboEditSelText  (
  integer nCtrlID,      // Control's caption
  integer nStartSel,    // Control start position
  integer nEndSel       // Control end position
  [,float fTimeOut]     // Time-out
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nStartSel` (input)

  Specify the start position of text selection in an edit control. The default `nStartSel` value is 0.

- `nEndSel` (input)

  Specify the end position of text selection in an edit control.

- `fTimeOut` (input, optional)

  Specify the maximum time this function can use to find the control, in units of seconds. The default is the value set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Notes

The start value may be greater than the end value. The smaller of the two values refers to the first character position of selected text. The greater value refers to the first character position beyond selected text.

The start value is a fixed selected text point, while the end value is a varying end point. The user can use the **Shift** key to adjust the size of a selected text. If the start point is 0 and the end point is -1, all the texts in the edit control are selected. If the start point is -1, active selection is released.

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_SetCurrentDirectory

Description

Changes a selected directory to the current directory.

Format

```
bool AIT_SetCurrentDirectory  (
  string strDirName    // Directory name
);
```

Parameters

- `strDirName` (input)

  Specify a directory name.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
5	ERROR_ACCESS_DENIED
21	ERROR_NOT_READY
53	ERROR_BAD_NETPATH

Extended error number	Error code
123	ERROR_INVALID_NAME
161	ERROR_BAD_PATHNAME
1005	ERROR_UNRECOGNIZED_VOLUME
1210	ERROR_INVALID_COMPUTERNAME
1214	ERROR_INVALID_NETNAME

AIT_SetDefaultWaitTimeout

Description

Sets the default time-out to be used in an API function associated with a control.

Format

```
AIT_SetDefaultWaitTimeout  (
  float fTimeOut      // Time-out value (seconds)
);
```

Parameters

- `fTimeOut` (input)

  Specify the default time-out in units of seconds. If you have omitted this function, the time-out is five seconds.

Return values

None

AIT_SetDtPickerDate

Description

Sets a date to the date/time picker.

Format

```
bool AIT_SetDtPickerDate  (
  string strCaption,     // Control's caption
  integer nYear,         // Year
  integer nMonth,        // Month
  integer nDay           // Day
  [,float fTimeOut]      // Time-out
);
bool AIT_SetDtPickerDate  (
```

```
  string strCaption,     // Control's caption
  string strInputDate    // Date
  [,float fTimeOut]      // Time-out
);
bool AIT_SetDtPickerDate  (
  integer nCtrlID,       // Control ID
  integer nYear,         // Year
  integer nMonth,        // Month
  integer nDay           // Day
  [,float fTimeOut]      // Time-out
);
bool AIT_SetDtPickerDate  (
  integer nCtrlID,        // Control ID
  string  strInputDate    // Date
  [,float fTimeOut]       // Time-out
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nYear` (input)

  Specify a year to be set to a control.

- `nMonth` (input)

  Specify a month to be set to a control.

- `nDay` (input)

  Specify a day to be set to a control.

- `strInputDate`

  Specify a date to be set to **in** control in the *MM/DD/YYYY* format, where *MM* indicates the month, *DD* indicates the day, and *YYYY* indicates the year.

- `fTimeOut` (input, optional)

  Specify the maximum time this function can use to find the control, in units of seconds. The default is the value set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an

extended error code. The following gives the error codes that AIT_GetLastError might return:

Extended error number	Error code
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_SetDtPickerTime

Description

Sets time to the date/time picker.

Format

```
bool AIT_SetDtPickerTime  (
  string strCaption,     // Control's caption
  integer nHour,         // Hours
  integer nMinute,       // Minutes
  integer nSecond        // Seconds
  [,float fTimeOut]      // Time-out
);
bool AIT_SetDtPickerTime  (
  string strCaption,     // Control's caption
  string strInputTime    // Time
  [,float fTimeOut]      // Time-out
);
bool AIT_SetDtPickerTime  (
  integer nCtrlID,       // Control ID
  integer nHour,         // Hours
  integer nMinute,       // Minutes
  integer nSecond        // Seconds
```

```
  integer [nTimeOut]     // Time-out
);
bool AIT_SetDtPickerTime  (
  integer nCtrlID,       // Control ID
  string strInputTime    // Time
  [,float fTimeOut]      // Time-out
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nHour` (input)

  Specify hours to be set to a control.

- `nMinute` (input)

  Specify minutes to be set to a control.

- `nSecond` (input)

  Specify seconds to be set to a control.

- `strInputTime` (input)

  Specify time to be set to a control in the *hh*:*mm*:*ss* format, where *hh* indicates the hour, *mm* indicates the minute, and *ss* indicates the second.

- `fTimeOut` (input, optional)

  Specify the maximum time this function can use to find the control, in units of seconds. The default is the value set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY

Extended error number	Error code
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_SetKeyState

Description

Sets specified key status.

Format

```
bool AIT_SetKeyState  (
  integer nVirtualKey,     // Virtual key
  integer nKeyState        // Key status
);
```

Parameters

- `nVirtualKey` (input)

  Specify a virtual key you want to set. You have to set one of the following values.

Value	Description
NUMLOCK	**Num Lock** key
SCROLLLOCK	**Scroll Lock** key
CAPSLOCK	**Caps Lock** key

- `nKeyState` (input)

  Specify key status you want to set. You have to set one of the following values.

Value	Description
KEYSTATE_OFF	The key is off.

Value	Description
KEYSTATE_ON	The key is on.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL

AIT_SetProfileString

Description

Creates or changes a key value in an INI file section.

Format

```
bool AIT_SetProfileString  (
  string strIniFileName,     // INI filename
  string strSectionName,     // Section name
  string strKeyName,         // Key name
  string strValue            // Value
);
```

Parameters

- `strIniFileName` (input)

  Specify an INI filename.

- `strSectionName` (input)

  Specify an INI file section name.

- `strKeyName` (input)

  Specify a key name belonging to a section name.

- `strValue` (input)

  Specify a key value you want to set.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
5	ERROR_ACCESS_DENIED
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
15	ERROR_INVALID_DRIVE
21	ERROR_NOT_READY
23	ERROR_CRC
53	ERROR_BAD_NETPATH
67	ERROR_BAD_NET_NAME
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
123	ERROR_INVALID_NAME
148	ERROR_PATH_BUSY
1005	ERROR_UNRECOGNIZED_VOLUME

AIT_SetScrollPos

Description

Moves the scroll bar.

Format

```
bool AIT_SetScrollPos (
  integer nCtrlID,          // Control ID
```

```
  integer nPosition        // Position
  [,float fTimeOut]        // Time-out
);
bool AIT_SetScrollPos (
  string strCaption,       // Control's caption
  integer nCtrlType,       // Control type
  integer nScrollType,     // Scroll type
  integer nPosition,       // Set position
  integer nScrollMovement  // Movement type
  [,float fTimeOut]        // Time-out
);
bool AIT_SetScrollPos (
  integer nCtrlID,         // Control ID
  integer nCtrlType,       // Control type
  integer nScrollType,     // Scroll type
  integer nPosition,       // Set position
  integer nScrollMovement  // Movement type
  [,float fTimeOut]        // Time-out
);
```

Parameters

- `nCtrlID` (input)

  Specify a control ID.

- `nPosition` (input)

  Specify a position to be set.

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlType` (input)

  Specify a control type for the scroll bar, which must be one of the following values.

Value	Description
`EDIT_CTRL`	The control type is an edit control.
`LISTBOX_CTRL`	The control type is a list box.
`LIST_CTRL`	The control type is a list control.
`TREE_CTRL`	The control type is a tree control.

- `nScrollType` (input)

  Specify a scroll bar type, which must be one of the following values.

Value	Description
VSCROLL	Vertical scroll bar
HSCROLL	Horizontal scroll bar

- nScrollMovement (input)

  Specify the type of scroll bar control movement. If nScrollType is VSCROLL, you have to set one of the following values.

Value	Description
SB_BOTTOM	Moves the scroll bar to the bottom right.
SB_LINEDOWN	Moves the scroll bar to the next line.
SB_LINEUP	Moves the scroll bar to the above line.
SB_PAGEDOWN	Moves the scroll bar to the next page.
SB_PAGEUP	Moves the scroll bar to the previous page.
SB_THUMBPOSITION	The user has dragged the scroll box (thumb), and released the mouse button.
SB_THUMBTRACK	The user is dragging the scroll box.
SB_TOP	Moves the scroll bar to the top left.

If nScrollType is HSCROLL, you have to set one of the following values.

Value	Description
SB_LEFT	Moves the scroll bar to the top left.
SB_RIGHT	Moves the scroll bar to the bottom right.
SB_LINELEFT	Moves the scroll bar to the left by one unit.
SB_LINERIGHT	Moves the scroll bar to the right by one unit.
SB_PAGELEFT	Moves the scroll bar to the left by the window width.
SB_PAGERIGHT	Moves the scroll bar to the right by the window width.
SB_THUMBPOSITION	The user has dragged the scroll box (thumb), and released the mouse button.
SB_THUMBTRACK	The user is dragging the scroll box.

- `nTimeOut` (input, optional)

  Specify the maximum time this function can use to find the control, in units of seconds. The default is the value set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_SetSpinPos

Description

Sets a position in a specific control on the active window.

Format

```
bool AIT_SetSpinPos (
  string strCaption,     // Control's caption
  integer nCtrlType,     // Control type
  integer nPosition      // Set position
  [,float fTimeOut]      // Time-out
);
bool AIT_SetSpinPos (
  integer nCtrlID,       // Control ID
  integer nCtrlType,     // Control type
  integer nPosition      // Set position
  [,float fTimeOut]      // Time-out
```

```
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nCtrlType` (input)

  Specify a control type, which must be one of the following values.

Value	Description
`SPIN_CTRL`	The control type is a spin control.
`SLIDER_CTRL`	The control type is a slider control.

- `nPosition` (input)

  Specify a position you want to set.

- `fTimeOut` (input, optional)

  Specify the maximum time this function can use to find the control, in units of seconds. The default is the value set in the `AIT_SetDefaultWaitTimeout` function.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	`ERROR_NOT_ENOUGH_MEMORY`
14	`ERROR_OUTOFMEMORY`
87	`ERROR_INVALID_PARAMETER`
112	`ERROR_DISK_FULL`
1400	`ERROR_INVALID_WINDOW_HANDLE`
1460	`ERROR_TIMEOUT`

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_SetWndPos

Description

Changes a specified window position.

Format

```
bool AIT_SetWndPos  (
  integer nWndHandle,     // Window handle
  integer nLeft,          // Horizontal position
  integer nTop            // Vertical position
);
```

Parameters

- `nWndHandle` (input)

  Specify a window handle. If you set 0, the active window position will be set.

- `nLeft` (input)

  Specify the X coordinate of the top left corner of a new window (horizontal position).

- `nTop` (input)

  Specify the Y coordinate of the top left corner of a new window (vertical position).

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE

AIT_SetWndPosSize

Description

Changes the position and size of a specified window.

Format

```
bool AIT_SetWndPosSize  (
  integer nHandle,      // Window handle
  integer nLeft,        // Horizontal position
  integer nTop,         // Vertical position
  integer nWidth,       // Window width
  integer nHeight       // Window height
);
```

Parameters

- `nHandle` (input)

  Specify a window handle. If you specify 0, the active window position will be set.

- `nLeft` (input)

  Specify the X coordinate of the top left corner of a new window (horizontal position).

- `nTop` (input)

  Specify the Y coordinate of the top left corner of a new window (vertical position).

- `nWidth` (input)

  Specify the width of a new window.

- `nHeight` (input)

  Specify the height of a new window.

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER

Extended error number	Error code
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE

AIT_Sleep

Description

Interrupts execution of the AIT file for a certain period.

Format

```
AIT_Sleep  (
  float fSeconds        // Seconds
);
```

Parameters

- `fSeconds` (input)

  Specify the period of interruption in units of seconds.

Return values

None

AIT_StatusBox

Description

Displays the dialog box containing an entered message.

Format

```
bool AIT_StatusBox(
  string    strMessage,        // Message character string
  [[[[,integer nXCord]         // X coordinate
  [,integer    nYCord]         // Y coordinate
  [,integer    nWidth]         // Message box width
  [,integer    nHeight]]       // Message box height
  [,bool       bIsTop]         // Most front message box
  [,bool       bIsMovable]]    // Movable message box
  [,string     strFontName]]   // Message font name
  [,integer    nFontSize]      // Message font size
  [,integer    nFontWeight]    // Message font width
);
```

Parameters

- `strMessage` (input)

  Specify a message character string you want to display in the dialog box.

- `nXCord` (input, optional)

  Specify the X coordinate of the top left corner of the dialog box. If you specify -1, the dialog box is displayed at the center of the X axis.

- `nYCord` (input, optional)

  Specify the Y coordinate of the top left corner of the dialog box. If you specify -1, the dialog box is displayed at the center of the Y axis.

  If you omit `nXCord` and `nYCord`, the dialog box will be positioned at the center.

- `nWidth` (input, optional)

  Specify the width of the dialog box.

- `nHeight` (input, optional)

  Specify the height of the dialog box.

  If you have omitted `nWidth` and `nHeight`, the dialog box size becomes equal to the `strMessage` size.

- `bIsTop` (input, optional)

  If you set `true`, the dialog box will always be positioned on the most front. If you set `false`, it will be displayed on the front at first. If you move or create another window, however, the dialog box will be moved behind it. For arbitrary operation, set `false`.

  Note that an operation on another window may move the focus from the dialog box even when `true` is set. If the dialog box is displayed beneath another window, you can re-execute this API function to display the dialog box on the front again.

  When specifying this parameter, you must also specify `nXCord`, `nYCord`, `nWidth`, and `nHeight`.

- `bIsMovable` (input, optional)

  With `true` set, you can move the dialog box. With `false` set, you cannot move the dialog box.

  When specifying this parameter, you must also specify `nXCord`, `nYCord`, `nWidth`, and `nHeight`.

- `strFontName` (input, optional)

  Specify a character string describing a message display font. The usable fonts may vary with systems.

For arbitrary operation, specify "". In this case, the default font is determined by the system.

When specifying this parameter, you must also specify nXCord, nYCord, nWidth, nHeight, bIsTop, and bIsMovable.

- nFontSize (input, optional)

  Specify the message font size using an integer in units of points.

  For arbitrary operation, set 0. In this case, the default font size is determined by the system.

- nFontWeight (input, optional)

  Specify the message font size by using an integer. The range of valid values is from 0 to 900. As the value increases, the font becomes thicker.

  For arbitrary operation, set 0. In this case, the default font thickness is determined by the system.

Return values

The return value is true if the function was executed normally, and false if not. If the function has returned false, you can use AIT_GetLastError to acquire an extended error code. The following gives the error code that AIT_GetLastError might return:

Extended error code	Error code
87	ERROR_INVALID_PARAMETER

Note

The parameter you can omit according to the above description is not independent. You have to set the default of the preceding omissible parameter. For example, when specifying only nXCord and nYCord, you can omit the subsequent omissible parameters. When specifying only nFontSize and nFontWeight, however, you cannot omit the subsequent omissible parameters, and you have to set the preceding omissible parameters at defaults.

AIT_StatusBoxClose

Description

Closes a displayed status box.

Format

```
AIT_StatusBoxClose ();
```

Parameters

None

Return values

None

AIT_StrLeft

Description

Returns the specified number of characters from the left of the character string.

The number of characters you have specified using `nNumChars` are extracted from the start (left end) of `strStrName`. If the number of characters set in `nNumChars` exceeds the length of the character string, the entire character string is extracted.

Format

```
string AIT_StrLeft (
  string strStrName,    // Character string
  integer nNumChars     // Character count
);
```

Parameters

- `strStrName` (input)

  Specify a character string.

- `nNumChars` (input)

  Specify the number of characters to be extracted.

Return value

The return value is the extracted character string.

Note

For a multi-byte character set (MBCS), eight bits are assumed to be a single character. That is, the first and last bytes of a multi-byte character are counted as two characters.

AIT_StrLength

Description

Returns the length of a character string.

Format

```
integer AIT_StrLength (
```

```
  string strStrName    // Character string
);
```

Parameters

- `strStrName` (input)

  Specify a character string.

Return value

This API function returns the length of a character string.

AIT_StrLower

Description

Converts the characters in a character string to the lowercase.

Format

```
string AIT_StrLower (
  string strStrName    // Character string
);
```

Parameters

- `strStrName` (input)

  Specify a character string.

Return value

This API function returns the character string with its characters converted to lowercase.

AIT_StrLTrim

Description

Returns a character string where all blanks or specified characters are deleted from its start (left end).

Format

```
string AIT_StrLTrim (
  string strStrName        // Character string
  [,string strCharValue]   // Character value
);
```

Parameters

- `strStrName` (input)

  Specify a character string.

- `strCharValue` (input, optional)

  Specify characters you want to delete. By default, blanks (such as breaks, spaces, and tabs) are deleted from the start of a character string.

Return value

This API function returns a character string where all blanks or specified characters are deleted from its start.

AIT_StrRight

Description

Returns the specified number of characters from the right of the character string.

The number of characters you have specified using `nNumChars` are extracted from the start (left end) of `strStrName`. If the number of characters set in `nNumChars` exceeds the length of the character string, the entire character string is returned.

Format

```
string AIT_StrRight (
  string strStrName,    // Character string
  integer nNumChars     // Character count
);
```

Parameters

- `strStrName` (input)

  Specify a character string.

- `nNumChars` (input)

  Specify the number of characters to be extracted.

Return value

The return value is the extracted character string.

Note

For a multi-byte character set (MBCS), eight bits are assumed to be a single character. That is, the first and last bytes of a multi-byte character are counted as two characters.

AIT_StrRTrim

Description

Returns a character string where all blanks or specified characters are deleted from its right end.

Format

```
string AIT_StrRTrim (
  string strStrName        // Character string
  [,string strCharValue]   // Character value
);
```

Parameters

- `strStrName` (input)

  Specify a character string.

- `strCharValue` (input, optional)

  Specify a character you want to delete. By default, blanks (such as breaks, spaces, and tabs) are deleted from the end of a character string.

Return value

This API function returns a character string where all blanks or specified characters are deleted from its end.

AIT_StrTrim

Description

Returns a character string where all blanks or specified characters are deleted from its left and right ends.

Format

```
string AIT_StrTrim (
  string strStrName        // Character string
  [,string strCharValue]   // Character value
);
```

Parameters

- `strStrName` (input)

  Specify a character string.

- `strCharValue` (input, optional)

  Specify a character to be deleted. By default, blanks (such as breaks, spaces, and tabs) are deleted from the start and end.

Return value

This API function returns a character string where all specified characters are deleted from its start and end.

AIT_StrUpper

Description

Converts the characters in a character string to uppercase.

Format

```
string AIT_StrUpper (
  string strStrName    // Character string
);
```

Parameters

- `strStrName` (input)

  Specify a character string.

Return value

This API function returns a character string with its characters converted to uppercase.

AIT_TaskbarClk

Description

Clicks a free area on the taskbar.

Format

```
bool AIT_TaskbarClk  (
  [integer nMouseButton]     // Mouse button
);
```

Parameters

- `nMouseButton` (input, optional)

  Specify a button to be clicked by the mouse. You have to set one of the following values.

Value	Description
LBUTTON	Left mouse button
MBUTTON	Center mouse button
RBUTTON	Right mouse button

The default is RBUTTON.

Return values

The return value is true if the function was executed normally, and false if not. If the function has returned false, you can use AIT_GetLastError to acquire an extended error code. The following gives the error codes that AIT_GetLastError might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
1400	ERROR_INVALID_WINDOW_HANDLE

AIT_TaskbarHasFocus

Description

Checks whether the taskbar has an input focus.

Format

```
integer AIT_TaskbarHasFocus ();
```

Parameters

None

Return values

The return value is 1 if the taskbar has a focus, 0 if not, and -1 if the function has not been processed successfully. If -1 has been returned, you can use AIT_GetLastError to acquire an extended error code. The following gives the error codes that AIT_GetLastError might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
1400	ERROR_INVALID_WINDOW_HANDLE

AIT_TaskbarSetFocus

Description

Sets an input focus to the taskbar.

Format

```
bool AIT_TaskbarSetFocus ();
```

Parameters

None

Return values

The return value is `true` if the function was executed normally, and `false` if not. If the function has returned `false`, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
1400	ERROR_INVALID_WINDOW_HANDLE

AIT_VerifyCharPos

Description

Checks a character position in a specific control on the active window.

Format

```
integer AIT_VerifyCharPos  (
  string strCaption,     // Control's caption
  integer nCtrlType,     // Control type
  integer nVerifyPos     // Character position
);
```

```
integer AIT_VerifyCharPos  (
  integer nCtrlID,       // Control ID
  integer nCtrlType,     // Control type
  integer nVerifyPos     // Character position
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nCtrlType` (input)

  Specify a control type for which only EDIT_CTRL is valid.

- `nVerifyPos` (input)

  Specify the positional value of a character to be checked. Because 0-based positioning is used, the position of the first character in the edit box is 0.

Return values

The return value is `1` if the found position is the same as a specified one, `0` if not, and `-1` if the function has not been processed normally. If `-1` has been returned, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_VerifyCount

Description

Checks the number of items in a specific control on the active window.

Format

```
integer AIT_VerifyCount  (
  string strCaption,     // Control's caption
  integer nCtrlType,     // Control type
  integer nItemCount     // Item count
);
integer AIT_VerifyCount  (
  integer nCtrlID,       // Control ID
  integer nCtrlType,     // Control type
  integer nItemCount     // Item count
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nCtrlType` (input)

  Specify a control type, which must be one of the following values.

Value	Description
`COMBO_CTRL`	The control type is a combo box.
`LISTBOX_CTRL`	The control type is a list box.
`TREE_CTRL`	The control type is a tree control.
`LIST_CTRL`	The control type is a list control.

- `nItemCount` (input)

  Specify the number of items to be checked.

Return values

The return value is `1` if the specified item count is matched, `0` if not, and `-1` if the function has not been processed successfully. If `-1` has been returned, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error

codes that AIT_GetLastError might return:

Extended error number	Error code
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_VerifyDateTime

Description

Checks a date or time in a specific control on the active window.

Format

```
integer AIT_VerifyDateTime  (
  string strCaption,     // Control's caption
  string strDateTime     // Date or time
);
integer AIT_VerifyDateTime  (
  integer nCtrlID,       // Control ID
  string strDateTime     // Date or time
);
```

Parameters

- strCaption (input)

  Specify the caption of a control.

- nCtrlID (input)

  Specify a control ID.

- `strDateTime` (input)

  Specify a control's date or time.

Return values

The return value is `1` if the specified date or time is matched, `0` if not, and `-1` if the function has not been processed successfully. If `-1` has been returned, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_VerifyDefaultButton

Description

Checks whether a specific command button in the active window is the default button.

Format

```
integer AIT_VerifyDefaultButton  (
  string strCaption    // Control's caption
);
integer AIT_VerifyDefaultButton  (
  integer nCtrlID      // Control ID
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

Return values

The return value is `1` if the button is the default one, `0` if not, and `-1` if the function has not been processed successfully. If `-1` has been returned, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_VerifyEnabled

Description

Checks whether a specific control in the active window is usable.

Format

```
integer AIT_VerifyEnabled  (
  string strCaption,     // Control's caption
  integer nCtrlType      // Control type
  [,integer nCtrlPos]    // Tab order
);
integer AIT_VerifyEnabled  (
  integer nCtrlID,       // Control ID
  integer nCtrlType      // Control type
  [,integer nCtrlPos]    // Tab order
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nCtrlType` (input)

  Specify a control type, which must be one of the following values.

Value	Description
`BUTTON_CTRL`	The control type is a command button.
`CHECKBOX_CTRL`	The control type is a check box.
`OPTIONBUTTON_CTRL`	The control type is an option button.
`EDIT_CTRL`	The control type is an edit box.
`STATIC_CTRL`	The control type is a static text.
`COMBO_CTRL`	The control type is a combo box.
`LISTBOX_CTRL`	The control type is a list box.
`SPIN_CTRL`	The control type is a spin control.
`TREE_CTRL`	The control type is a tree control.
`LIST_CTRL`	The control type is a list control.
`DTPICKER_CTRL`	The control type is a date/time picker.

- `nCtrlPos` (input, optional)

  Specify a control's tab order.

Return values

The return value is `1` if the control is usable, `0` if not, and `-1` if the function has not been processed successfully. If `-1` has been returned, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	`ERROR_NOT_ENOUGH_MEMORY`
14	`ERROR_OUTOFMEMORY`

Extended error number	Error code
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_VerifyExistence

Description

Checks whether the active window contains a specific control.

Format

```
integer AIT_VerifyExistence  (
  string strCaption,    // Control's caption
  integer nCtrlType     // Control type
);
integer AIT_VerifyExistence  (
  integer nCtrlID,      // Control ID
  integer nCtrlType     // Control type
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nCtrlType` (input)

  Specify a control type, which must be one of the following values.

Value	Description
BUTTON_CTRL	The control type is a command button.
CHECKBOX_CTRL	The control type is a check box.

Value	Description
OPTIONBUTTON_CTRL	The control type is an option button.
EDIT_CTRL	The control type is an edit box.
STATIC_CTRL	The control type is a static text.
COMBO_CTRL	The control type is a combo box.
LISTBOX_CTRL	The control type is a list box.
SPIN_CTRL	The control type is a spin control.
TREE_CTRL	The control type is a tree control.
LIST_CTRL	The control type is a list control.
DTPICKER_CTRL	The control type is a date/time picker.
IPADDRESS_CTRL	The control type is an IP address control.
SLIDER_CTRL	The control type is a slider control.
SCROLLBAR_CTRL	The control type is a scroll bar control.

Return values

The return value is `1` if the control exists, `0` if not, and `-1` if the function has not been processed successfully. If `-1` has been returned, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_VerifyFirstVisible

Description

Checks the index of the first visible item in a list box.

Format

```
integer AIT_VerifyFirstVisible  (
  string strCaption,     // Control's caption
  integer nCtrlType,     // Control type
  integer nIndex         // Index
);
integer AIT_VerifyFirstVisible  (
  integer nCtrlID,       // Control ID
  integer nCtrlType,     // Control type
  integer nIndex         // Index
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nCtrlType` (input)

  Specify a control type for which only LISTBOX_CTRL is valid.

- `nIndex` (input)

  Specify a control's index. As the index is based on 0, the index for the first item in a control is 0.

Return values

The return value is `1` if the specified index matches the index for the first visible item, `0` if not, and `-1` if the function has not been processed successfully. If `-1` has been returned, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER

Extended error number	Error code
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_VerifyFocus

Description

Checks whether the specific control in the active window has a focus.

Format

```
integer AIT_VerifyFocus  (
  string strCaption,     // Control's caption
  integer nCtrlType      // Control type
);
integer AIT_VerifyFocus  (
  integer nCtrlID,       // Control ID
  integer nCtrlType      // Control type
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nCtrlType` (input)

  Specify a control type, which must be one of the following values.

Value	Description
BUTTON_CTRL	The control type is a command button.
CHECKBOX_CTRL	The control type is a check box.
OPTIONBUTTON_CTRL	The control type is an option button.

Value	Description
EDIT_CTRL	The control type is an edit box.
STATIC_CTRL	The control type is a static text.
COMBO_CTRL	The control type is a combo box.
LISTBOX_CTRL	The control type is a list box.
SPIN_CTRL	The control type is a spin control.
TREE_CTRL	The control type is a tree control.
LIST_CTRL	The control type is a list control.
DTPICKER_CTRL	The control type is a date/time picker.

Return values

The return value is `1` if the control has a focus, `0` if not, and `-1` if the function has not been processed successfully. If `-1` has been returned, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_VerifyIndex

Description

Checks whether the text matches the index in a specific control on the active window.

Format

```
integer AIT_VerifyIndex  (
  string strCaption,     // Control's caption
  integer nCtrlType,     // Control type
  string strCtrlText,    // Control text
  integer nIndex         // Index
);
integer AIT_VerifyIndex  (
  integer nCtrlID,       // Control ID
  integer nCtrlType,     // Control type
  string strCtrlText,    // Control text
  integer nIndex         // Index
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nCtrlType` (input)

  Specify a control type, which must be one of the following values.

Value	Description
`COMBO_CTRL`	The control type is a combo box.
`LISTBOX_CTRL`	The control type is a list box.
`TREE_CTRL`	The control type is a tree control.
`LIST_CTRL`	The control type is a list control.

- `strCtrlText` (input)

  Specify the text of a control.

- `nIndex` (input)

  Specify a control's index based on 0.

Return values

The return value is `1` if the specified control's index matches the text, `0` if not, and `-1` if the function has not been processed successfully. If `-1` has been returned, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_VerifyKeyState

Description

Checks whether the key status is the same as that on the keyboard.

Format

```
integer AIT_VerifyKeyState  (
  integer nVirtualKey,     // Virtual key
  integer nKeyState        // Key status
);
```

Parameters

- `nVirtualKey` (input)

  Specify a virtual key having key status to be checked.

  You have to set one of the following values.

Value	Description
NUMLOCK	**Num Lock** key
SCROLLLOCK	**Scroll Lock** key
CAPSLOCK	**Caps Lock** key

- `nKeyState` (input)

  Specify key status to be checked, which is on or off. You have to set one of the following values.

Value	Description
KEYSTATE_OFF	Specify this value to check whether the key is off.
KEYSTATE_ON	Specify this value to check whether the key is on.

Return values

The return value is `1` if the key status matches specified status, `0` if not, and `-1` if the function has not been processed successfully. If `-1` has been returned, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

AIT_VerifyLine

Description

Verifies on the active window that the index matches the current line on an edit box containing multiple lines.

Format

```
integer AIT_VerifyLine  (
  string strCaption,     // Control's caption
  integer nCtrlType,     // Control type
  integer nIndex         // Index
);
integer AIT_VerifyLine  (
  integer nCtrlID,       // Control ID
  integer nCtrlType,     // Control type
  integer nIndex         // Index
```

```
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nCtrlType` (input)

  Specify a control type for which only `EDIT_CTRL` is valid.

- `nIndex` (input)

  Specify the index of the current line to be checked. As the index value is based on 0, the index for the first line in an edit box is 0.

Return values

The return value is `1` if the current line in an edit box matches a specified index, `0` if not, and `-1` if the function has not been processed successfully. If `-1` has been returned, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_VerifyLocation

Description

Checks whether a specific control in the active window matches specified coordinates.

Format

```
integer AIT_VerifyLocation  (
  string  strCaption,   // Control's caption
  integer nCtrlType,    // Control type
  integer nLeft,        // Coordinates of the top left control
corner
  integer nTop,         // Coordinates of the control top
  integer nRight,       // Coordinates the top right control
corner
  integer nBottom       // Coordinates of the control bottom
);
integer AIT_VerifyLocation  (
  integer nCtrlID,      // Control ID
  integer nCtrlType,    // Control type
  integer nLeft,        // Coordinates of the top left control
corner
  integer nTop,         // Coordinates of the control top
  integer nRight,      // Coordinates of the top right control
corner
  integer nBottom       // Coordinates of the control bottom
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nCtrlType` (input)

  Specify a control type, which must be one of the following values.

Value	Description
`BUTTON_CTRL`	The control type is a command button.
`CHECKBOX_CTRL`	The control type is a check box.
`OPTIONBUTTON_CTRL`	The control type is an option button.
`EDIT_CTRL`	The control type is an edit box.

Value	Description
STATIC_CTRL	The control type is a static text.
COMBO_CTRL	The control type is a combo box.
LISTBOX_CTRL	The control type is a list box.
SPIN_CTRL	The control type is a spin control.
TREE_CTRL	The control type is a tree control.
LIST_CTRL	The control type is a list control.
DTPICKER_CTRL	The control type is a date/time picker.

- `nLeft` (input)

  Specify the X coordinate of the top left control corner.

- `nTop` (input)

  Specify the Y coordinate of the top left control corner.

- `nRight` (input)

  Specify the X coordinate of the bottom right control corner.

- `nBottom` (input)

  Specify the Y coordinate of the bottom right control corner.

Return values

The return value is `1` if coordinates of a control match specified ones, `0` if not, and `-1` if the function has not been processed successfully. If `-1` has been returned, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_VerifyMenuChecked

Description

Checks whether a menu item has been checked.

Format

```
integer AIT_VerifyMenuChecked  (
  integer nMenu,      // Menu handle
  integer nIndex      // Index for a menu item
);
```

Parameters

- `nMenu` (input)

  Specify a menu handle returned by the `AIT_GetMenu` or `AIT_GetSubMenu` API function.

- `nIndex` (input)

  Specify the index of a menu item. As the menu index is based on 0, the index for the first menu item is 0.

  The indexes also include a menu separator index. If a menu separator index has been given as an input value, this function is not processed successfully, with `ERROR_INVALID_INDEX` returned as the extended error code.

Return values

The return value is `1` if the menu item has been checked, `0` if not, and `-1` if the function has not been processed successfully. If `-1` has been returned, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error code	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL

Extended error code	Error code
1401	ERROR_INVALID_MENU_HANDLE
1413	ERROR_INVALID_INDEX
1460	ERROR_TIMEOUT

AIT_VerifyMenuEnabled

Description

Checks whether the menu item is enabled.

Format

```
integer AIT_VerifyMenuEnabled  (
  integer nMenu,      // Menu handle
  integer nIndex      // Index for a menu item
);
```

Parameters

- `nMenu` (input)

  Specify a menu handle returned by the `AIT_GetMenu` or `AIT_GetSubMenu` API function.

- `nIndex` (input)

  Specify the index of a menu item. As the menu index is based on 0, the index for the first menu item is 0.

  The indexes also include a menu separator index. If a menu separator index has been given as an input value, this function is not processed successfully, with `ERROR_INVALID_INDEX` returned as the extended error code.

Return values

The return value is `1` if the menu item is enabled, `0` if not, and `-1` if the function has not been processed successfully.

If `-1` has been returned, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY

Extended error number	Error code
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1401	ERROR_INVALID_MENU_HANDLE
1413	ERROR_INVALID_INDEX
1460	ERROR_TIMEOUT

AIT_VerifyNoOfCtrls

Description

Checks whether the number of controls in an active control matches a specified value.

Format

```
integer AIT_VerifyNoOfCtrls  (
  integer nNumControls      // Control count
);
```

Parameters

- `nNumControls` (input)

  Specify the number of controls to be checked.

Return values

The return value is `1` if the number of controls matches a specified value, `0` if not, and `-1` if the function has not been processed successfully. If `-1` has been returned, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

AIT_VerifyPos

Description

Checks whether the specific control in the active window matches specified tab order.

Format

```
integer AIT_VerifyPos  (
  string strCaption,     // Control's caption
  integer nCtrlType,     // Control type
  integer nCtrlPos       // Tab order
);
integer AIT_VerifyPos  (
  integer nCtrlID,       // Control ID
  integer nCtrlType,     //Control type
  integer nCtrlPos       // Tab order
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nCtrlType` (input)

  Specify a control type, which must be one of the following values.

Value	Description
`BUTTON_CTRL`	The control type is a command button.
`CHECKBOX_CTRL`	The control type is a check box.
`OPTIONBUTTON_CTRL`	The control type is an option button.
`EDIT_CTRL`	The control type is an edit box.
`STATIC_CTRL`	The control type is a static text.
`COMBO_CTRL`	The control type is a combo box.
`LISTBOX_CTRL`	The control type is a list box.
`SPIN_CTRL`	The control type is a spin control.
`TREE_CTRL`	The control type is a tree control.

Value	Description
LIST_CTRL	The control type is a list control.
DTPICKER_CTRL	The control type is a date/time picker.

- nCtrlPos (input)

  Specify a control's tab order.

Return values

The return value is 1 if the control's tab order matches specified tab order, 0 if not, and -1 if the function has not been processed successfully. If -1 has been returned, you can use AIT_GetLastError to acquire an extended error code. The following gives the error codes that AIT_GetLastError might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_VerifySelected

Description

Checks whether the specified text is selected in a specific control on the active window.

Format

```
integer AIT_VerifySelected  (
  string strCaption,      // Control's caption
  integer nCtrlType,      // Control type
  string strSelectedText  // Selected text
);
```

```
integer AIT_VerifySelected  (
  integer nCtrlID,        // Control ID
  integer nCtrlType,      // Control type
  string strSelectedText  // Selected text
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nCtrlType` (input)

  Specify a control type for which only `EDIT_CTRL` is valid.

- `strSelectedText` (input)

  Specify text to be checked.

Return values

The return value is `1` if the text selected in a control matches specified text, `0` if not, and `-1` if the function has not been processed successfully. If `-1` has been returned, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_VerifyState

Description

Checks whether the specific control has been checked in the active window.

Format

```
integer AIT_VerifyState  (
  string strCaption,    // Control's caption
  integer nCtrlType     // Control type
);
integer AIT_VerifyState  (
  integer nCtrlID,      // Control ID
  integer nCtrlType     // Control type
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nCtrlType` (input)

  Specify a control type, which must be one of the following values.

Value	Description
`CHECKBOX_CTRL`	The control type is a check box.
`OPTIONBUTTON_CTRL`	The control type is an option button.

Return values

The return value is `1` if the control is checked, `0` if not, 2 if unknown, and `-1` if the function has not been processed successfully. The unknown status is applied only to `CHECKBOX_CTRL`. If `-1` has been returned, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	`ERROR_NOT_ENOUGH_MEMORY`
14	`ERROR_OUTOFMEMORY`
87	`ERROR_INVALID_PARAMETER`

Extended error number	Error code
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

Note

You can identify the control by using a complete caption or an associated label name or specifying the first part of such a caption or label name. When specifying the first part, prefix a swung dash (~) to the character string that you specify.

AIT_VerifyText

Description

Checks whether the control text matches a specified text in the active window.

Format

```
integer AIT_VerifyText  (
  string strCaption,    // Control's caption
  integer nCtrlType,    // Control type
  string strText        // Text
);
integer AIT_VerifyText  (
  integer nCtrlID,      // Control ID
  integer nCtrlType,    // Control type
  string strText        // Text
);
```

Parameters

- `strCaption` (input)

  Specify the caption of a control.

- `nCtrlID` (input)

  Specify a control ID.

- `nCtrlType` (input)

  Specify a control type for which only `EDIT_CTRL` is valid.

- `strText` (input)

  Specify text to be checked.

Return values

The return value is `1` if the control text matches specified text, `0` if not, and `-1` if the function has not been processed successfully. If `-1` has been returned, you can use `AIT_GetLastError` to acquire an extended error code. The following gives the error codes that `AIT_GetLastError` might return:

Extended error number	Error code
8	ERROR_NOT_ENOUGH_MEMORY
14	ERROR_OUTOFMEMORY
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL
1400	ERROR_INVALID_WINDOW_HANDLE
1460	ERROR_TIMEOUT

AIT_Wow64DisableWow64Redirection

Description

Switches to the mode in which the application accesses registry paths or file paths as a 64-bit application.

Format

```
integer AIT_Wow64DisableWow64Redirection ();
```

Parameters

None

Return values

The return value is `1` if the mode in which the application accesses registry or file paths as a 64-bit application has already been set before the function is executed. The return value is `0` if the mode in which the application accesses registry or file paths as a 32-bit application has been set before the function is executed.

AIT_Wow64RevertWow64Redirection

Description

Switches to the mode in which the application accesses registry paths or file paths as a 32-bit application.

Format

```
integer AIT_Wow64RevertWow64Redirection ();
```

Parameters

None

Return values

The return value is `1` if the mode in which the application accesses registry or file paths as a 64-bit application has been set before the function is executed. The return value is `0` if the mode in which the application accesses registry or file paths as a 32-bit application has already been set before the function is executed.

4.3 Examples of using API functions

This section gives examples of using the API functions provided by the AIT language.

4.3.1 Deleting carriage return and linefeed characters

The text read from a file may contain carriage return and linefeed characters (`\r\n`). This subsection gives an example of reading the file that contains the text `Deleted CR and LF.\r\n`, deleting carriage return and linefeed characters from the text, and outputting the result to `RecDFile.log`.

(1) Coding example

```
strFileName = "C:'\Sample.txt";
strCharValue = "\r\n";               // Deletes CR and LF.
if (AIT_FileOpen(strFileName, GENERIC_READ, OPEN_EXISTING,
nFileHandle))
  // Reads data from the file.
  if (!AIT_FileGetLine(nFileHandle, strReadData))
    AIT_LogMessage("AIT_FileGetLine failed");
  else
    // Deletes CR and LF from the data read from the file.
    strStrName = AIT_StrRTrim(strReadData, strCharValue);
    AIT_LogMessage("strStrName = " + strStrName);
  endif;
  AIT_FileClose(nFileHandle);
else
  AIT_LogMessage("AIT_FileOpen failed");
endif;
```

(2) Results

The following shows the results output to `RecDFile.log`.

```
strStrName = Deleted CR and LF.
```

4.3.2 Extracting characters

This subsection gives an example of extracting only alphanumeric characters from the character string `0123-4567-89AB-CDEF` and outputs the results to `RecDFile.log`.

(1) Coding example

```
strStrName = "0123-4567-89AB-CDEF";    // Original string
strSearchStr = "-";                    // Search string
nStartPos = 0;
while(TRUE)
  // Gets the length of string to be extracted.
  nLength = AIT_FindSubStr(strStrName, strSearchStr, nStartPos);
  if (nLength == -1)
    // Sets the string to be extracted last.
```

```
    strSubString = strStrName;
    AIT_LogMessage("strSubString = " + strSubString);
    // Ends extraction of strings.
    break;
  else
    // Extracts a string.
    if (!AIT_GetSubStr(strSubString, strStrName, nStartPos,
nLength))
      AIT_LogMessage("AIT_GetSubStr failed");
      break;
    else
      AIT_LogMessage("strSubString = " + strSubString);
    endif;
    // Deletes the extracted string from the original string.
    strStrName = AIT_StrLTrim(strStrName, strSubString);
    strStrName = AIT_StrLTrim(strStrName, strSearchStr);
    strSubString = "";
  endif;
loop;
// Value of strStrName changes from that before processing.
AIT_LogMessage("strStrName = " + strStrName);
```

(2) Results

The following shows the results output to `RecDFile.log`.

```
strSubString = 0123
strSubString = 4567
strSubString = 89AB
strSubString = CDEF
strStrName = CDEF
```

4.3.3 Manipulating the HKEY_CURRENT_USER registry key during remote installation

Be careful when you want to perform remote installation by using an AIT file in which an API function that manipulates the `HKEY_CURRENT_USER` hive is specified. If the logged-on user of a remote installation target computer is not a member of the Administrators group, the function manipulates the `HKEY_USERS\.DEFAULT` key, instead of `HKEY_CURRENT_USER`. In this case, to manipulate the `HKEY_CURRENT_USER` hive, you need to use **Run** from the **Start** menu to import the relevant registry file.

To manipulate the `HKEY_CURRENT_USER` hive during remote installation using an AIT file:

1. Create the registry file.

   Create the registry file for the registry branch that you want to add or edit.

Note:

It is recommended that you back up the registry and understand how to restore the system before manipulating the registry.

2. Create an AIT file.

   The following gives an example of coding that chooses **Run** from the **Start** menu, executes `regedit.exe`, and then imports the desired registry file. In this example, `Sample.reg` is the registry file to be imported.

```
while(iLoopCount < iLoopMax)
  if((AITEVENTFLAG1==0) && (AITIGNORE == 0))
    // Displays the Start menu.
    AIT_PlayKey("{LWIN}");
    AIT_Sleep(SLEEP_TIME_EVENTS);
    // Chooses Run.
    AIT_PlayKey("r");
    AIT_Sleep(SLEEP_TIME_EVENTS);
    AITEVENTFLAG1 = 1;
    AITIGNORE = 1;
    iLoopCount = 0;
  endif;
  if((AITEVENTFLAG1==1) && (AITIGNORE == 0) &&
(AIT_FocusWindow("Run", "#32770") != 0))
    // Enters the path of the registry file.
    AIT_GetCurrentDirectory(strPath);
    AIT_PlayKey("regedit.exe /s '"" + strPath +
"'\Sample.reg'"");
    AIT_Sleep(SLEEP_TIME_EVENTS);
    // Imports the registry file.
    AIT_PlayKey("{ENTER}");
    AIT_Sleep(SLEEP_TIME_EVENTS);
    iLoopCount = iLoopMax;
    DM_RTN = OK_END;
    continue;
  endif;
  AITIGNORE = 0;
  iLoopCount = iLoopCount + 1;
  AIT_Sleep(SLEEP_TIME);
loop;
```

3. Store the created registry file in the same directory as for the software to be distributed.
4. Perform packaging, specifying the created AIT file.
5. Perform remote installation.

The registry file will be copied to the same directory as for `setup.exe`, and the AIT file will be executed at the remote installation target computers.

Chapter

5. Troubleshooting

This chapter describes messages that may be output by the Automatic Installation Tool. This chapter describes the actions to be taken when a problem occurs in the Automatic Installation Tool.

5.1 Checking messages
5.2 Messages that may be displayed during editing
5.3 Messages that may be displayed during execution and parsing of AIT files

5.1 Checking messages

While the Automatic Installation Tool is executing an AIT file or parsing the codes in the file, incorrect simulations, insufficient disk space, memory damage, and other problems may occur. If such problems occur, first, check whether error messages have been output to the standard output and log files. When error messages have been output, you can find the causes of the problems from the message IDs.

5.1.1 Message output destination

Messages are displayed in a dialog box or in the output window. Some messages are also output to *log files*. The log files are stored in the following directory:

JP1/IT-Desktop-Management-2-installation-directory\LOG

The following table explains the log files.

Filename	Maximum number of lines	Description
ait.log	2000	This file stores error messages concerning GUI operations.
aitapi.log	2000	This file stores API-related error messages.
aitexec.log	2000	This file stores error messages concerning execution and parsing.

5.1.2 Format of message explanations

The message explanations in this chapter use the following format:

Message ID

Message text

Cause

aa...aa

Action

bb...bb

Example

cc...cc

(1) Message ID

Message format: `AIT`*`X`*`nnnn`*`-Z message-text`*

`AIT`

This indicates that the Automatic Installation Tool program displayed the message.

X

This identifies the component which output the message.

`G`: GUI

`CE` or `CW`: This indicates that the error occurred during execution or parsing of AIT file. (`E` stands for an error and `W` stands for warning.)

nnn or *nnnn*

This indicates a message number assigned by the component.

Z

This indicates a message type. For the messages that begin with `AITCE` or `AITCW` do not have this part. `E` or `W` in `AITCE` or `AITCW` substitutes for this part.

`E` (error):

This indicates that a fatal error occurred. Normally, processing stops if a message of this type is output.

`W` (warning):

This displays a warning message that indicates expected information was not detected, and the default was used instead.

`Q` (question):

This indicates a message to which the user has to respond.

`I` (information):

This displays an informational message that indicates important actions are being performed.

(2) Message text

Message text is the contents of a message. In message text, the portions indicated in the *xxx* style are replaced with appropriate strings.

(3) Cause

This describes why the message was output.

(4) Action

This describes actions to be taken if the message is output.

(5) Example

This shows invalid and valid API function usage examples.

5.2 Messages that may be displayed during editing

This section covers the messages that may be displayed while you are using the edit window. These messages are displayed in a dialog box or in the output window.

AITG100-E

```
The registry is either corrupt or invalid.
```

Cause

Possible causes are as follows:

- The registry database is damaged.
- A registry key is invalid.
- The registry is damaged. The file cannot be recovered because some file structures including registry data are damaged, the memory image for a system file is damaged, or an alternative backup or log does not exist or is damaged.
- Although the system attempted to load or restore the file into the registry, the specified file did not have the registry file format.

Action

Carry out required processing according to appropriate additional information. If the registry is damaged, use a recovery disk to restore settings.

If you cannot solve the problem, contact the system administrator.

AITG101-E

```
The registry operation failed.
```

Cause

Possible causes are as follows:

- A registry key could not be opened.
- A registry key could not be loaded.
- A registry key could not be updated.
- Because an invalid I/O operation on the registry caused a non-recoverable error, a read, write, or flush operation failed for one of the files including system registry images.
- An invalid operation was performed on a registry key that has already been indicated as deleted.
- A necessary area could not be allocated in the registry log.

- No symbolic link can be created for a registry key already having a sub-key or value.
- An attempt has been made to create a stable sub-key under a volatile parent key.
- The target multi-byte code page contains no Unicode mapping.
- No index matches the specified key.

Action

Carry out required processing according to appropriate additional information.

If you cannot solve the problem, contact the system administrator.

AITG102-E

`An internal error occurred.`

Cause

The application attempted to access resources by using a handle such as the following:

- an invalid handle;
- an invalid window handle; or
- an invalid hook handle.

Action

Re-execute the application.

If you cannot solve the problem, contact the system administrator.

AITG103-E

`Memory or disk space is insufficient.`

Cause

Possible causes are as follows:

- Memory is insufficient to process the command.
- Memory is insufficient to complete the processing.
- The disk is full.
- The disk contains no sufficient space.
- System resources are insufficient to complete the required service.

Action

Terminate all the unnecessary applications, and retry the processing.

Delete unnecessary files on the disk, and re-execute the application.

If you cannot solve the problem, contact the system administrator.

AITG104-E

`The file or directory operation failed.`

Cause

Possible causes are as follows:

- The system cannot find the specified file.
- The system cannot find the specified path.
- The system cannot open the file.
- The system cannot find the specified drive.
- No directory can be deleted.
- The system cannot move the file to another disk drive.
- The file does not exist.
- No directory or file can be created.
- The filename is too long.
- The directory is not a subdirectory of the root directory.
- The directory is not empty.
- During access to the hard disk, disk operations failed, with a retry failing.

Action

Carry out required processing according to appropriate additional information.

For example, use the valid filename, path name or drive name to retry the processing.

If you cannot solve the problem, contact the system administrator.

AITG105-E

`An attempt to initiate the Recorder failed.`

Cause

Any of the following items has been initialized in the edit window:

- Syntax check
- Run
- Recorder
- Window properties

Action

Re-execute the application.

If you cannot solve the problem, contact the system administrator.

AITG106-E

`The debugging operation failed.`

Cause

During debugging, the application cannot communicate with the execution engine.

Action

Re-execute the application.

If you cannot solve the problem, contact the system administrator.

AITG107-W

`A syntax file was not found. Syntax highlighting will not be enabled.`

Cause

The syntax file (`AIT.syn`) could not be found from the edit window.

Action

The path of the syntax file set in either of the following registry keys does not indicate the correct location.

- When the OS is a 32-bit version:

  `HKEY_LOCAL_MACHINE\SOFTWARE\HITACHI\JP1/IT Desktop Management - AIT\ConfPath`

- When the OS is a 64-bit version:

  `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Hitachi\JP1/IT Desktop Management - AIT\ConfPath`

Verify the path in the registry, and re-execute the application.

AITG108-E

`The line number is invalid.`

Cause

An invalid line number has been specified in the line number edit box of the **Go To** or **Breakpoints Setup** dialog box.

Action

Specify a valid line number, and re-execute the processing.

AITG109-E

`The string` *string*`.could not be found.`

Cause

The application could not find the character string specified in the **Find** or **Replace** dialog box.

Action

Unnecessary

AITG110-E

`The format of the PACKAGE_INFO section is invalid.`

Cause

When the Package Information dialog box was opened, a format error was detected in the PACKAGE_INFO section of the AIT file.

Action

Check the syntax of the AIT file, then open the Package Information dialog box.

AITG111-E

`Enter the required item` *item-name*`.`

Cause

Any of the following required package items has not been specified in the Package Information dialog box:

- Package ID
- Product
- Version
- Installer name
- Install drive
- Install directory

Action

Specify the required items, then generate package information.

AITG112-Q

`Do you want to update the Package Information?`

Cause

This message is displayed if you click the **Stop** button in the Recorder dialog box. Select whether to generate the PACKAGE_INFO section of the AIT file.

Action

Click the **Yes** button to display the Package Information dialog box to generate the PACKAGE_INFO section. Click the **No** button to create the AIT file without the PACKAGE_INFO section.

AITG113-E

`Specify a value between 1 and 64.`

Cause

An integer value that is not in the range from 1 to 64 is specified in the **Tab size** text box of the **Tabs page** in the Options dialog box.

Action

Specify an integer value from 1 to 64, and then retry the operation.

AITG114-E

`Invalid characters have been entered. Specify` *specifiable-character-string*`.`

Cause

An invalid character is specified for one or more of the following package items in the Package Information dialog box:

- Package ID
- Product
- Version
- Installer name
- Install directory

Action

Enter a specifiable character string according to the contents displayed in *specifiable-character-string*, then generate package information.

AITG115-W

`The registry key could not updated.`

Cause

Possible causes are as follows:

- The registry database is damaged.
- A registry key cannot be opened.
- A registry key cannot be read.
- A registry key cannot be updated.
- A registry key is invalid.

- The registry is damaged. The file cannot be recovered because part of the file structures including registry data are damaged, a memory image on a system file is damaged, or an alternative backup or log does not exist or is damaged.
- Although the system attempted to update a file in the registry, the specified file did not have the registry file format.

Action

Carry out required processing according to appropriate additional information.

AITG116-W

`The registry key could not be read.`

Cause

Possible causes are as follows:

- The registry database is damaged.
- A registry key cannot be opened.
- A registry key cannot be read.
- A registry key is invalid.
- The registry is damaged. The file cannot be recovered because part of the file structures including registry data are damaged, a memory image on a system file is damaged, or an alternative backup or log does not exist or is damaged.
- Although the system attempted to read a file from the registry, the specified file did not have the registry file format.

Action

Carry out required processing according to appropriate additional information.

AITG117-W

`The registry key could not be deleted.`

Cause

Possible causes are as follows:

- The registry database is damaged.
- A registry key cannot be deleted.
- A registry key is invalid.
- The registry is damaged. The file cannot be recovered because part of the file structures including registry data are damaged, a memory image on a system file is damaged, or an alternative backup or log does not exist or is damaged.
- Although the system attempted to delete the file from the registry, the

specified file did not have the registry file format.

Action

Carry out required processing according to appropriate additional information.

AITG118-W

```
The file cannot be closed. The file is being processed.
```

Cause

An attempt was made to close the AIT file during a syntax check on the file.

Action

Unnecessary

AITG119-W

```
The application cannot be closed. The file is being processed.
```

Cause

Possible causes are as follows:

- An attempt was made to close the AIT file during a syntax check on the file.
- An attempt was made to close the application during execution of the AIT file.

Action

Unnecessary

AITG120-E

```
The symbol was not found.
```

Cause

Possible causes are as follows:

- In the debug mode, a variable not defined in the AIT file was entered in the Watch window.
- In the debug mode, a symbol not defined in the AIT file was entered.

Action

Verify that the symbol you entered in the Watch window is a valid variable in the AIT file being debugged.

AITG121-W

```
The user permission is insufficient. Saved options will be
applied only to the current session.
```

Cause

You attempted to save a registry value although you were logged on as a guest.

Action

Unnecessary

AITG122-E

`The maximum of` *maximum-number-of-characters-you-can-use-to-specify-the-package-item* `characters is exceeded.`

Cause

The generated PACKAGE_INFO section includes a package item specified with too many characters.

Action

Make sure that all the items in the PACKAGE_INFO section are specified with a correct character length.

AITG123-E

`The maximum of` *maximum-number-of-characters-you-can-use-to-specify-the-package-item* `characters is exceeded.`

Cause

The generated program product ID file contains a package item longer than the maximum.

Action

After checking the maximum number of characters that can be used to specify the package item, specify the package item with a correct length.

AITG124-E

`An attempt to operate the Program product ID file has failed.`

Cause

Another application might be using the program product ID file.

Action

Close the program product ID files that are being used by other applications.

AITG125-E

`Enter the item` *item-name*`.`

Cause

One or the other of the following items was not specified in the Package Information dialog box:

- **AIT file path**
- **Package file ID**

Action

After specifying the necessary item, generate the package information.

AITG200-E

`The recording operation failed.`

Cause

The application attempted to access resources by using a handle such as the following:

- an invalid handle;
- an invalid window handle; or
- an invalid hook handle.

Action

Re-execute the application.

If you cannot solve the problem, contact the system administrator.

AITG201-E

`An attempt to acquire window details failed.`

Cause

Window information could not be acquired during recording.

Action

If you cannot solve the problem, contact the system administrator.

AITG202-E

`An attempt to acquire control details failed.`

Cause

Window control information could not be acquired during recording.

Action

If you cannot solve the problem, contact the system administrator.

AITG203-E

`An attempt to acquire event details failed.`

Cause

Event information could not be acquired during recording.

Action

If you cannot solve the problem, contact the system administrator.

AITG204-E

`An attempt to generate the PACKAGE_INFO section in the AIT file failed.`

Cause

The PACKAGE_INFO section could not be generated after the end of recording.

Action

Delete the temporary folder before re-execution.

If you cannot solve the problem, contact the system administrator.

AITG205-E

`An attempt to generate the DEFINE section in the AIT file failed.`

Cause

The DEFINE section could not be generated after the end of recording.

Action

Delete a temporary folder before re-execution.

If you cannot solve the problem, contact the system administrator.

AITG206-E

`An attempt to generate the MAIN section in the AIT file failed.`

Cause

The MAIN section could not be generated after the end of recording.

Action

Delete a temporary folder before re-execution.

If you cannot solve the problem, contact the system administrator.

AITG208-E

`An attempt to generate the AIT file failed.`

Cause

The AIT file could not be generated after the end of recording possibly because:

- disk operations and retries have not performed successfully during access to the hard disk;
- memory is insufficient to process the command;
- memory is insufficient to complete the processing; or
- the handle is invalid.

Action

Carry out required processing according to appropriate additional information.

If you cannot solve the problem, contact the system administrator.

AITG209-I

`The Installer application` *installer-name* `executed successfully.`

Cause

The AIT file has been executed normally.

Action

Unnecessary

AITG211-W

`The AIT File Log option is not set in the registry.`

Cause

The log option is not set in the registry.

Action

Set the log option in the registry.

AITG212-I

`The recording process completed.`

Cause

Recording has been completed normally.

Action

Unnecessary

AITG213-I

`Creation of the AIT file was successful.`

Cause

The AIT file has been generated successfully after the end of recording.

Action

Unnecessary

AITG214-W

`The user information file update was skipped because an event is unknown.`

Cause

The Recorder cannot resume recording after a temporary stop.

Action

Unnecessary

AITG215-E

`An attempt to continue recording failed.`

Cause

The Recorder cannot resume recording after a temporary stop.

Action

Carry out required processing according to appropriate additional information.

AITG216-Q

`An attempt to start the Installer application failed:`
installer-name`,` *additional-information*
`Do you want to continue recording?`

Cause

An invalid installer name is specified in the Recorder dialog box.

For *additional-information*, one of the following strings is displayed:

- `The specified file could not be found.`
- `The specified path could not be found.`
- `The access was rejected.`
- `Memory is insufficient to carry out the processing.`

Action

Click the **Yes** button to start recording, and click the **No** button to carry out required processing according to additional information.

5.3 Messages that may be displayed during execution and parsing of AIT files

This section covers the messages that may be displayed during execution and parsing of AIT files. These messages appear in the output window.

AITCE-0001

`Unexpected` *token-1*`, missing` *token-2*`.`

Cause

An invalid keyword or symbol *token-1* was detected. The script parsing expects the keyword or symbol *token-2*.

Action

Add *token-2* if necessary.

Example

Example of invalid specification:

```
DEFINE
{
   const integer OK_END = 0;
   const integer NG_END = -1     // ";" is not specified.
   const integer sloop_max = 30;
}
```

Example of valid specification:

```
DEFINE
{
   const integer OK_END = 0;
   const integer NG_END = -1;     // ";" has been added.
   const integer sloop_max = 30;
}
```

AITCE-0002

`Unexpected` *token*

Cause

An invalid keyword or symbol *token* was detected.

Action

Delete the unexpected keyword or symbol.

Example

Example of invalid specification:

```
DEFINE
```

```
{
   integer OK_END = 0;
   integer sloop_max = 0;;      // Two semicolons (;;)
are specified.
}
```

The `sloop_max` variable contains two semicolons (`;;`).

Example of valid specification:

```
DEFINE
{
   integer OK_END = 0;
   integer sloop_max = 0;      // The semicolon (;) has
been deleted.
}
```

One semicolon (`;`) has been deleted from the `sloop_max` variable.

AITCE-0003

`A function name is used as an identifier.`

Cause

A function name cannot be used as an identifier.

Action

Change the identifier name.

Example

Example of invalid specification:

```
DEFINE
{
   integer AIT_LogMessage = 10;
}
```

A function name is used as a variable name.

Example of valid specification:

```
DEFINE
{
   integer AIT_LogMessageNumber = 10;
}
```

The variable name has been changed to `AIT_LogMessageNumber.`

AITCE-0005

`The number of nested levels is greater than 255.`

Cause

The number of nesting levels in the AIT file exceeds 255, which is the maximum.

Action

If the number of nested `if`, `if-else`, `do-while`, `while`, or `switch` structure levels exceeds the maximum of 255, the AIT file cannot be parsed. Re-edit the AIT file to decrease the number of nested structure levels.

AITCE-0006

`The identifier name exceeds 64 characters.`

Cause

Each identifier name can have up to 64 characters. The AIT file contains an identifier name that has more than 64 characters.

Action

Specify an identifier name that has up to 64 characters.

Example

Example of invalid specification:

```
DEFINE
{
   integer sloop_max = 0;
   integer
sample123456sample123456sample123456sample123456sample1
23456sample = 0;     // The specified variable name has
66 characters.
}
```

The second variable name has more than 64 characters.

Example of valid specification:

```
DEFINE
{
   integer sloop_max = 0;
  integer sample123456 = 0;   // The specified variable
name
                              // has 64 or fewer characters.
}
```

The second variable name has 64 or fewer characters.

AITCE-0007

`The string constant should not span multiple lines.`

Cause

As the first line ends with `\n`, the string constant is written over two lines.

Action

Specify a string constant in one line.

Example

Example of invalid specification:

```
DEFINE
{
   integer sloop_max = 0;
   string SoftwareName = " My
                      Setup";  // The string constant is
                               // written over two lines.
}
```

A string constant in the `SoftwareName` variable is written over two lines.

Example of valid specification:

```
DEFINE
{
   integer sloop_max = 0;
   string SoftwareName = " My Setup";   // The specified
string constant is in one line.
}
```

The string constant for variable `SoftwareName` is specified in one line.

AITCE-0008

`Use a valid escape sequence.`

Cause

The string constant for the AIT file specified during execution contains an invalid escape sequence.

Action

Use a valid escape sequence for a string constant.

Example

Example of invalid specification:

```
DEFINE
{
   integer sloop_max = 0;
   string str1 = "sample'
                  testing";
}
```

The string variable `strl` is assigned a string written over two lines but an invalid escape sequence is used.

Instead of an apostrophe (`'`), an underscore (_) must be used to indicate that the string continues to the next line.

Example of valid specification:

```
DEFINE
{
   integer sloop_max=10;
   string str1 = "sample_
                  testing";
}
```

An apostrophe (') is replaced with an underscore (_), which is the correct escape sequence.

AITCE-0009

`Identifiers cannot be used in case statements.`

Cause

An identifier is specified in the `case` clause.

Action

Use only constants or macros in the `case` clause.

Example

Example of invalid specification:

```
DEFINE
{
   const string FileVersion = "7.1";
   string stMsgText;
}
MAIN
{
   switch (FileVersion)
      case stMsgText:      // An identifier is specified
                            // in the case clause.
    :
         break;
     default:
    :
         break;
   endswitch;
}
```

The identifier `stMsgText` is specified in the `case` clause.

Example of valid specification:

```
DEFINE
{
   const string FileVersion = "7.1";
   string stMsgText;
}
MAIN
```

```
{
   switch (FileVersion)
      case "7.1":      // A string constant which is not
                        // an identifier is specified.
    :
         break;
     default:
    :
         break;
   endswitch;
}
```

Instead of the identifier `stMsgText`, a string constant is specified in the `case` clause.

AITCE-0010

`Usage of the "+" or "-" signs is invalid.`

Cause

The `++`, `--`, `+-`, or `-+` operator is used in the AIT file. These operators specified in an AIT file cannot be parsed.

Action

Delete the `++`, `--`, `+-`, or `-+` operator from the AIT file.

Example

Example of invalid specification:

```
DEFINE
{
   integer sloop_count = 0;
}
MAIN
{
   if (AIT_FileExists("#setup.exe") == 0)
      :
      :
     sloop_count++;      // The invalid operator"++" is
used to
                          // increment the value of the
variable.
   endif;
}
```

You cannot use the increment and decrement operators.

Since `++` is specified for the `sloop_count` variable, an error message is displayed.

Example of valid specification:

```
DEFINE
{
   integer sloop_count = 0;
}
MAIN
{
   if (AIT_FileExists("#setup.exe") == 0)
      :
      :
     sloop_count = sloop_count + 1;   // "++" is not used.
   endif;
}
```

++ is replaced with another equivalent code.

AITCE-0011

`The AIT file contains more than 65535 lines.`

Cause

The AIT file can contain up to 65,535 lines. If the AIT file contains more than 65,535 lines, parsing of the AIT file stops.

Action

Reduce the number of lines in the AIT file.

AITCE-0012

identifier `: An identifier is undeclared.`

Cause

The variable of an undeclared variable type is specified.

Action

Correctly declare the variable in the DEFINE section.

Example

Example of invalid specification:

```
DEFINE
{
   string stMsgText;
}
MAIN
{
   if (AIT_FileExists("#setup.exe") == 0)
      stMsgText = "Setup(English) " + InstallerName + "
Not Found";
      AIT_LogMessage(stMsgText);
     sloop_max = 0;     // This variable is not declared
```

```
                                    // in the DEFINE section.
   endif;
}
```

The `sloop_max` variable is specified in the MAIN section although it is not declared in the DEFINE section.

Example of valid specification:

```
DEFINE
{
   string  stMsgText;
  integer sloop_max;     // Variable sloop_max has been
declared.

}
MAIN
{
   if (AIT_FileExists("#setup.exe") == 0)
      stMsgText = "Setup(English) " + InstallerName + "
Not Found";
      AIT_LogMessage(stMsgText);
      sloop_max = 0;     // A declared variable is used.
   endif;
}
```

The `sloop_max` variable has been declared in the DEFINE section, and is specified in the MAIN section.

AITCE-0013

identifier`: Redeclared.`

Cause

The already declared identifier is redeclared.

Action

Delete the redeclared variable.

Example

Example of invalid specification:

```
DEFINE
{
   integer sloop_max = 10;
  string sloop_max = "sample";    // Variable sloop_max
is
                                   // redeclared as the
string type.
}
```

The `sloop_max` variable that was declared as the `integer` type is redeclared as the `string` type.

Example of valid specification:

```
DEFINE
{
   integer sloop_max = 10;
   string stMsgText = "sample";    // Another variable
name is used.
}
```

The redeclared variable `sloop_max` has been deleted, and a new variable name is declared.

AITCE-0014

`The package information field` *package-item* `contains invalid data` *value*`.`

Cause

Any of the package items in the PACKAGE_INFO section contains an invalid value.

Action

Specify a value adapted to the package information field.

Example

Example of invalid specification:

```
PACKAGE_INFO
{
   PackageID = "#$@#ADOBEACROBATREADER";   // The package
ID contains
                                    // invalid string #$@#.
   Product = "Adobe Reader 6.0";
   Version = "0060";
   InstallerName = "AdbeRdr60_enu_full.exe";
   InstallDrive = "C:";
   InstallDirectory = "'Program Files'\Adobe'\Acrobat
6.0";
}
```

The package ID in the PACKAGE_INFO section contains an invalid string #$@.

Example of valid specification:

```
PACKAGE_INFO
{
   PackageID = "ACROBAT-READER";      // Invalid string
#$@#
```

```
                                                    // has been deleted.
   Product = "Adobe Reader 6.0";
   Version = "0060";
   InstallerName = "AdbeRdr60_enu_full.exe";
   InstallDrive = "C:";
   InstallDirectory = "'Program Files'\Adobe'\Acrobat
6.0";
}
```

The invalid string #$@ has been deleted from the package ID in the PACKAGE_INFO section.

AITCE-0015

`A 'const' cannot be re-assigned a value.`

Cause

The variable defined as the `constant` type is assigned a value in the MAIN section.

Action

Change the definition of `constant` or delete the right side of the constant.

Example

Example of invalid specification:

```
DEFINE
{
   const float SLEEP_TIME = 2.0;
   integer sloop_count;
}
MAIN
{
   if (AIT_FocusWindow("Setup", "#32770",0.0) > 0)
      AIT_PlayKey("{Enter}");
      AIT_LogMessage("Setup: Enter");
      sloop_count = 0;
      SLEEP_TIME = 3.0;      // The constant value for
variable
                             // SLEEP_TIME is being changed.
      AIT_Sleep(SLEEP_TIME);
   endif;
}
```

The `SLEEP_TIME` variable defined as the `constant float` type has been assigned a different value in the MAIN section.

Example of valid specification:

```
DEFINE
{
```

```
   const float SLEEP_TIME = 3.0;
   integer sloop_cnt;
}

MAIN
{
   if (AIT_FocusWindow("Setup", "#32770",0.0) > 0)
      AIT_PlayKey("{Enter}");
      AIT_LogMessage("Setup : Enter");
      sloop_cnt = 0;
      AIT_Sleep(SLEEP_TIME);
   endif;
}
```

In the MAIN section, assignment to the `SLEEP_TIME` variable of type `constant float` has been deleted.

AITCE-0018

`The AIT file analysis has been abnormally terminated because of a syntax error in the AIT file.`

Cause

The syntax check has encountered a fatal error (including invalid string end processing or invalid character count specification in a variable name).

Action

Correct the error, then re-check the AIT file.

Example

Example of invalid specification:

```
DEFINE
{
   string ErrorTxt = "ABC
def";
}
```

Although a string constant cannot be specified over multiple lines, the string is specified over multiple lines.

Example of valid specification:

```
DEFINE
{
   string ErrorTxt = "ABC_
def";      // "_" indicates the continuation of the
string.
}
```

Add an underscore (_) that indicates that the string continues to the next line.

AITCE-0019

`Division by zero`

Cause

Because the second operand for division is 0, the division is considered as being undefined for an AIT file parsing.

Action

Specify a value other than 0 as the divisor.

Example

Example of invalid specification:

```
DEFINE
{
   integer sloop_count = 0;
   const integer  sloop_max = 30;
}
MAIN
{
   sloop_count = sloop_max / 0;      // As the divisor is
0, this
                                     // results in and error.
}
```

Although division by 0 is impossible, the `sloop_max` variable is divided by 0.

Example of valid specification:

```
DEFINE
{
   integer sloop_count = 0;
   const integer sloop_max = 30;
}
MAIN
{
   sloop_count = sloop_max / 1;      // The divisor has
been changed
                                  // to a value other than 0.
}
```

The division of the `sloop_max` variable by 0 has been deleted.

AITCE-0020

data type-1 `and` *data type-2* `are incompatible for` *processing-name*`.`

Cause

The left-side and right-side values for an operator are assigned incompatible data

types. For example, if the `string` and `integer` types have been used for comparison, this message is displayed.

Action

Specify data types compatible between the left-side and right-side values for an operator.

Example

Example of invalid specification:

```
DEFINE
{
   const integer ExeVersion = 7;
   const string FileVersion = "7";
}
MAIN
{
   if (ExeVersion == FileVersion)   // The string and
integer types
                                    // are used for comparison.
      ...
      ...
   endif;
}
```

The `ExeVersion` variable of type `integer` and the `FileVersion` variable of type `string` are compared.

The `string` and `integer` types are incompatible for a comparison operation.

Example of valid specification:

```
DEFINE
{
   const integer ExeVersion = 7;
   const integer FileVersion = 7;
}
MAIN
{
   if (ExeVersion == FileVersion)      // Variables of
the same type
                                       // are compared.
      ...
      ...
   endif;
}
```

The data type of the `FileVersion` variable has been changed from `string` to `integer` to enable comparison.

AITCE-0021

operator-name `incompatible for operation` *data-type-1*`.`

Cause

For the indicated operation, an invalid data type is specified. For example, if the `float` type has been used for a remainder operation (`%`) or if the `string` type has been used for an operation, this message is displayed.

Action

Specify a data type that can be used for the operation.

Example

Example of invalid specification:

```
DEFINE
{
   float SLEEP_TIME = 7.1;
}
MAIN
{
   SLEEP_TIME = SLEEP_TIME % 2;     // float type cannot
be used for
                                 // remainder operation. This
                                      // results in an error.
}
```

Although the `float` type cannot be used for a remainder operation, `float`-type variable `SLEEP_TIME` is used for the remainder operation.

Example of valid specification:

```
DEFINE
{
   integer SLEEP_TIME = 7;
}
MAIN
{
   SLEEP_TIME = SLEEP_TIME % 2;      // Integer-type
variable is used
                                   // for remainder operation.
}
```

The `integer`-type value is used for the remainder operation.

AITCE-0022

`Specify an integer value between '-2147483648' and '2147483647'.`

Cause

An integer value out of the allowable range is specified.

Action

Specify an integer value between -2,147,483,648 and 2,147,483,647.

Example

Example of invalid specification:

```
DEFINE
{
   const integer OK_END = 21474836476;   // Value exceeds
the maximum.
}
```

The `OK_END` variable is assigned an `integer`-type value greater than the maximum.

Example of valid specification:

```
DEFINE
{
  const integer OK_END =214748364;     // Value is within
the
                                       // allowable range.
}
```

The `OK_END` variable has been assigned an `integer`-type value within the allowable range.

AITCE-0023

```
Specify a float value between '3.402823466e+38' and
'1.175494351e-38'.
```

Cause

A `float` value out of the allowable range is specified.

Action

Specify a `float` value between 3.402823466e+38 and 1.175494351e-38.

Example

Example of invalid specification:

```
DEFINE
{
  const float NG_END = 3.402823466e+40;  // Value exceeds
the maximum.
}
```

The `NG_END` variable is assigned a `float`-type value greater than the maximum.

Example of valid specification:

```
DEFINE
{
   const float NG_END = 3.402823466e+10;    // Value is
within the
                                            // allowable range.
}
```

The NG_END variable has been assigned a float-type value within the allowable range.

AITCE-0024

The data type for a switch expression is invalid.

Cause

The data type of a case label value does not match the data type of the switch statement. For example, if a switch statement of type integer contains a case label whose value is the float type, this message is displayed.

Action

Use the same data type for the switch statement and the case label values.

Example

Example of invalid specification:

```
DEFINE
{
   const string FileVersion = "7.1";
   integer sloop_max = 0;
}
MAIN
{
   switch (FileVersion)   // The switch statement is the
string type.
       case 7.1:      // The case label is assigned an
integer value.
          ...
          ...
          break;
      default:
          ...
          ...
          break;
   endswitch;
}
```

Although the switch statement is the string type, the case value is the integer type.

Example of valid specification:

```
DEFINE
{
   const string FileVersion = "7.1";
   integer sloop_max = 0;
}
MAIN
{
   switch (FileVersion)
      case "7.1":    // A value of the string type is
specified.
         ...
         ...
         break;
      default:
         ...
         ...
         break;
   endswitch;
}
```

The `case` label has been assigned a `string`-type value to match the data type with that of the `switch` statement.

AITCE-0025

`Unexpected EOF found in comment.`

Cause

There is a comment that begins with `/*` but the comment end symbol (`*/`) is not found until the end of the file.

Action

Specify `*/` to close the comment before the end of the file.

Example

Example of invalid specification:

```
DEFINE
{
   /* Data type used in  the AIT file   // The comment is
not closed.
   const integer NG_END = -1
   const integer sloop_max = 30;
}
```

All comments must be closed. In this example, the comment is started with `/*`, but is not ended with `*/`.

Example of valid specification:

```
DEFINE
```

```
{
   /* Data type used in the AIT file */   // The comment
is ended.
   const integer NG_END = -1;
   const integer sloop_max = 30;
}
```

The comment is ended with `*/`.

AITCE-0026

`A character is unknown:` *hexadecimal-character-code*`.`

Cause

A character not defined in AIT language specifications exists. The hexadecimal character code is displayed.

Action

Specify only valid characters covered in AIT language specifications.

Example

Example of invalid specification:

```
DEFINE
{
   const integer OK_END = 0;
   const integer NG_END = -1;
  const integer sloop_max = #30;     // Invalid character
"#" exists.
}
```

Although an invalid string like #$@ cannot be used in a string constant, variable `sloop_max` contains #.

Example of valid specification:

```
DEFINE
{
   const integer OK_END =0;
   const integer NG_END = -1;
  const integer sloop_max = 30;    // Invalid character
"#" is deleted.
}
```

has been deleted from the `sloop_max` variable.

AITCE-0027

`The use of the void data type in the expression is invalid. Use a valid datatype.`

Cause

An expression includes a function that returns a `void`-type value.

Action

In expressions, do not use any functions that return a `void`-type value.

AITCE-0029

`Case value` *value* `already used.`

Cause

The same `case` value is used twice or more times in a `switch` statement.

Action

Specify a unique `case` label value.

Example

Example of invalid specification:

```
DEFINE
{
   const string FileVersion = "7.1";
   string stMsgText;
}
MAIN
{
   switch(FileVersion )
      case "7.1":
          ...
          ...
          break;
     case "7.1":     // Value "7.1" has already been used.
          ...
          ...
          break;
      default:
          ...
          ...
          break;
   endswitch;
}
```

The `case` labels of `switch` statements must have unique values.

In this example, `7.1` is used as the value of the second `case` label but is also used as the value of the first `case` label.

Example of valid specification:

```
DEFINE
{
   const string FileVersion = "7.1";
```

```
   string stMsgText;
}
MAIN
{
   switch(FileVersion )
     case "7.1":      // Duplicate case value "7.1" has
been deleted.
         ...
         ...
         break;
      default:
         ...
         ...
         break;
   endswitch;
}
```

The second case label has been deleted.

AITCE-0030

function name `: A function name is invalid.`

Cause

An invalid function name (not found in the list of available API functions) is specified.

Action

Specify a valid function name in the AIT file. For details about valid functions (API names), see *4.1 API functions*.

Example

Example of invalid specification:

```
DEFINE
{
   const string ExeVersion = "7.1";
   const string FileVersion = "7.1";
   string stMsgText;
}
MAIN
{
   if (ExeVersion == FileVersion)
      if (AIT_FileExists1("#setup.exe") == 0) //
AIT_FileExists1 is
                                              // not a
valid API name.
        stMsgText = "Setup(English) " + InstallerName +
" Not Found";
         AIT_LogMessage(stMsgText);
```

```
      endif;
   endif;
}
```

`AIT_FileExists1()` is not a valid API.

Example of valid specification:

```
DEFINE
{
   const string ExeVersion = "7.1";
   const string FileVersion = "7.1";
   string stMsgText;
}
MAIN
{
   if (ExeVersion == FileVersion)
      if (AIT_FileExists("setup.exe") == 0)   // Valid
API name
                                                // has been
specified.
        stMsgText = "Setup(English) " + InstallerName +
" Not Found";
         AIT_LogMessage(stMsgText);
      endif;
   endif;
}
```

`AIT_FileExists()` has been specified as an API name.

AITCE-0031

function name `: The function does not take the` *number-of-specified-parameters* `parameter.`

Cause

There is a function that has an invalid number of parameters.

Each function uses a specific set of parameters. This message is displayed if the number of parameters specified for a function does not match the valid number of parameters for the function.

Action

Specify a valid number of parameters for the function. For a list of actual parameters, see *4.2 Details about the API functions*.

Example

Example of invalid specification:

```
DEFINE
{
```

```
   const string ExeVersion = "7.1";
   const string FileVersion = "7.1";
   string stMsgText;
}
MAIN
{
   if (ExeVersion == FileVersion)
      if (AIT_FileExists() == 0)
      // Argument was specified for AIT_FileExists.
        stMsgText = "Setup(English) " + InstallerName +
" Not Found";
         AIT_LogMessage(stMsgText);
      endif;
   endif;
}
```

Specify at least one parameter for `AIT_FileExists()`. In this example, no parameter is specified for `AIT_FileExists()`.

Example of valid specification:

```
DEFINE
{
   const string ExeVersion = "7.1";
   const string FileVersion = "7.1";
   string stMsgText;
}
MAIN
{
   if (ExeVersion == FileVersion)
      if (AIT_FileExists("#setup.exe") == 0)  // A
parameter has been
                                          // specified for
                                        // AIT_FileExists.
        stMsgText = "Setup(English) " + InstallerName +
" Not Found";
         AIT_LogMessage(stMsgText);
      endif;
   endif;
}
```

A parameter has been specified for `AIT_FileExists()`.

AITCE-0032

`One or more arguments specified for the function are of invalid data type.`

Cause

At least one parameter for the function has a data type that the function does not expect.

Each function has a specific set of parameters. This message is displayed if the data type of a parameter specified for a function does not match the data type of the function.

Action

Specify a valid data type for the function argument. For a list of actual arguments, see *4.2 Details about the API functions*.

Example

Example of invalid specification:

```
DEFINE
{
   const string ExeVersion = "7.1";
   const string FileVersion = "7.1";
   string stMsgText;
}
MAIN
{
   if (ExeVersion == FileVersion )
      if (AIT_FileExists("#setup.exe") == 0)
        stMsgText = "Setup(English) " + InstallerName +
" Not Found";
         AIT_LogMessage(stMsgText);
       AIT_Sleep(3);     // Integer value is specified in
                            // AIT_Sleep.
      endif;
   endif;
}
```

Although only a `float`-type parameter can be specified for `AIT_Sleep()`, an `integer`-type parameter is specified.

Example of valid specification:

```
DEFINE
{
   const string ExeVersion = "7.1";
   const string  FileVersion = "7.1";
   string stMsgText;
}
MAIN
{
   if (ExeVersion == Version)
      if (AIT_FileExists("#setup.exe") == 0)
        stMsgText = "Setup(English) " + InstallerName +
" Not Found";
         AIT_LogMessage(stMsgText);
       AIT_Sleep(3.1);     // Data type has been changed
from
```

```
                                                  // integer to float.
         endif;
      endif;
   }
```

The data type of the parameter for `AIT_Sleep()` has been changed from `integer` to `float`.

AITCE-0034

`The label` *label-name* `is undefined.`

Cause

The `goto` statement is used with no label defined.

In the AIT file, the associated label must be defined for each `goto` statement.

Action

Specify associated label names for all `goto` statements.

Example

Example of invalid specification:

```
DEFINE
{
   integer sloop_max = 0;
}
MAIN
{
   goto ErrorLabel; // Label ErrorLabel is undefined.
   sloop_max = 0;
}
```

An associated label statement must be specified for the `goto` statement.

In this example, only `goto ErrorLabel` is specified. The label `ErrorLabel` is not defined in the MAIN section.

Example of valid specification:

```
DEFINE
{
   integer sloop_max = 0;
}
MAIN
{
   goto ErrorLabel;
   sloop_max = 0;
ErrorLabel:      // The label has been defined.
}
```

The label `ErrorLabel` has been specified for the `goto` statement.

AITCE-0037

`The label` *label-name* `for the goto statement is in a different block.`

Cause

The specified label name and the associated `goto` statement exist in different sections.

A `goto` statement and the associated label must exist in the same section.

Action

Specify a label name and the associated `goto` statement in the same section.

Example

Example of invalid specification:

```
DEFINE
{
   string stMsgText;
}
MAIN
{
   goto ErrorLabel;      // goto statement is defined in
MAIN section.
}
ERROR
{
ErrorLabel:      // Label is defined in ERROR section.
      stMsgText = "Setup(English) " + InstallerName + "
Not Found";
}
```

A label and the associated `goto` statement must be defined in the same section. In this example, the `goto` statement is in the MAIN section, while the label `ErrorLabel` is in the ERROR section.

Example of valid specification:

```
DEFINE
{
   string stMsgText;
}
MAIN
{
      goto ErrorLabel;
ErrorLabel:      // The label is defined in the MAIN
section.
      stMsgText = "Setup(English) " + InstallerName + "
Not Found";
}
```

The label `ErrorLabel` and the associated `goto` statement have been defined in the MAIN section.

AITCE-0038

`The value assigned to the variable is of an invalid data type.`

Cause

The value assigned to a variable has an invalid data type. For example, this message is displayed if a `string` value has been assigned to an `integer`-type variable.

Action

Specify a value of a valid data type.

Example

Example of invalid specification:

```
DEFINE
{
   const integer OK_END =0;
   const integer NG_END = -1;
  const integer sloop_max = "30";   // A string value is
assigned
                                    // to an integer-type
variable.
}
```

In this example, a `string`-type constant is assigned to the `sloop_max` variable of type `integer constant`.

Example of valid specification:

```
DEFINE
{
   const integer OK_END =0;
   const integer NG_END = -1;
   const integer sloop_max = 30;      // integer value,
not a string
                                 // value has been assigned.
}
```

The `string` constant has been deleted, and the `sloop_max` variable of type `integer` has been assigned.

AITCE-0039

`An invalid 'break' statement was found.`

Cause

A `break` statement is used in a statement other than the `do-while`, `while-loop`, `for-next`, or `switch` statement. The `break` statement is valid

only in the loop structure.

Action

Use the `break` statement in the `do-while`, `while-loop`, `for-next`, or `switch` statement.

Example

Example of invalid specification:

```
DEFINE
{
   const string ExeVersion = "7.1";
   const string FileVersion = "7.1";
   string stMsgText;
}
MAIN
{
   if (ExeVersion == FileVersion )
      if (AIT_FileExists("#setup.exe") == 0)
        stMsgText = "Setup(English) " + InstallerName +
" Not Found";
         AIT_LogMessage(stMsgText);
        break;      // The break statement is in the if
statement.
      endif;
   endif;
}
```

In this example, the `break` statement is in the `if` structure.

Example of valid specification:

```
DEFINE
{
   const string ExeVersion = "7.1";
   const string  FileVersion = "7.1";
   string stMsgText;
}
MAIN
{
   if (ExeVersion == FileVersion)
      if (AIT_FileExists("#setup.exe") == 0)
        stMsgText = "Setup(English) " + InstallerName +
" Not Found";
         AIT_LogMessage(stMsgText);
      endif;
   endif;
}
```

The `break` statement has been deleted from the `if` structure.

AITCE-0040

`An invalid 'continue' statement was found.`

Cause

A `continue` statement is used in a statement other than `do-while`, `while-loop`, or `for-next` statement. The `continue` statement is valid only in the loop structure.

Action

Use the `continue` statement only in the `do-while`, `while-loop`, or `for-next` statement.

Example

Example of invalid specification:

```
DEFINE
{
   const string ExeVersion = "7.1";
   const string FileVersion = "7.1";
   string stMsgText;
}
MAIN
{
   if (ExeVersion == FileVersion)
      if (AIT_FileExists("#setup.exe") == 0)
        stMsgText = "Setup(English) " + InstallerName +
" Not Found";
         AIT_LogMessage(stMsgText);
         continue;      // The continue statement is in
the if statement.
      endif;
   endif;
}
```

In this example, the `continue` statement is in the `if` structure.

Example of valid specification:

```
DEFINE
{
   const string ExeVersion = "7.1";
   const string  FileVersion = "7.1";
   string stMsgText;
}
MAIN
{
   if (ExeVersion == FileVersion)
      if (AIT_FileExists("#setup.exe") == 0)
        stMsgText = "Setup(English) " + InstallerName +
```

```
" Not Found";
        AIT_LogMessage(stMsgText);
      endif;
   endif;
}
```

The `continue` statement has been deleted from the `if` structure.

AITCE-0041

loop-structure `cannot have more than 255 break statements.`

Cause

The loop structure contains more than 255 `break` statements. The loop structure (`do-while`, `while`, `for-next`, or `switch` structure) can contain up to 255 `break` statements.

Action

Reduce the number of `break` statements in the loop structure to 255 or fewer.

AITCE-0042

loop-structure `cannot have more than 255 continue statements.`

Cause

The loop structure contains more than 255 `continue` statements. The loop structure (`do-while`, `while`, or `for-next`) can use up to 255 `continue` statements.

Action

Reduce the number of `continue` statements in the loop structure to 255 or fewer.

AITCE-0043

`The 'switch' statement cannot have more than 255 case labels.`

Cause

The `switch` statement contains more than 255 `case` labels. The `switch` statement can use up to 255 `case` labels.

Action

Reduce the number of `case` labels in the `switch` statement to 255 or fewer.

AITCE-0044

`The 'switch' statement cannot have more than one default label.`

Cause

The `switch` statement can use only one `default` statement.

Action

Specify only one `default` statement in the `switch` statement.

Example

Example of invalid specification:

```
DEFINE
{
   const string FileVersion = "7.1";
   string stMsgText;
}
MAIN
{
   switch(FileVersion )
      case "7.1":
         ...
         ...
         break;
      default:      // First default statement
         ...
         ...
         break;
      default:      // Two default statements are
specified.
         ...
         ...
         break;
   endswitch;
}
```

Only one `default` statement can be specified in the `switch` statement. In this example, the `switch` statement contains two specified `default` statements.

Example of valid specification:

```
DEFINE
{
   const string FileVersion = "7.1";
   string stMsgText;
}
MAIN
{
   switch(FileVersion)
      case "7.1":
         ...
         ...
         break;
      default:      // Only one default statement is
specified.
         ...
```

```
            ...
            break;
      endswitch;
   }
```

One `default` statement has been deleted from the `switch` statement.

AITCE-0045

`A required field` *field-name* `is missing in the Package Info block.`

Cause

One of the fields which must be specified in the PACKAGE_INFO section is missing.

Action

Specify all the fields required in the PACKAGE_INFO section.

Example

Example of invalid specification:

```
PACKAGE_INFO
{
   // The package ID that must be specified is missing.
   Product = "Adobe Reader 6.0";
   Version = "0060";
   InstallerName = "AdbeRdr60_enu_full.exe";
   InstallDrive = "C:";
   InstallDirectory = "'Program Files'\Adobe'\Acrobat
6.0";
}
```

The package ID that must be specified is missing.

Example of valid specification:

```
PACKAGE_INFO
{
   PackageID = "ADOBEACROBATREADER";   // The required
package item
                                       // has been specified.
   Product = "Adobe Reader 6.0";
   Version = "0060";
   InstallerName = "AdbeRdr60_enu_full.exe";
   InstallDrive = "C:";
   InstallDirectory = "'Program Files'\Adobe'\Acrobat
6.0";
}
```

The package ID has added in the PACKAGE_INFO section.

AITCE-0046

```
Unary operators '+', '-' and '!' cannot be used with a string
constant.
```

Cause

A unary operator is specified together with a `string` constant.

Action

Do not specify a unary operator together with a `string` constant.

Example

Example of invalid specification:

```
DEFINE
{
   const integer OK_END =0;
   const integer NG_END = -1;
   const string szMsgText = !"30";  // "!" is used
together with a
                                    // string constant.
}
```

A unary operator (`+`, `-`, or `!`) cannot be used to initialize a `string`-type constant in the DEFINE section.

In this example, `!` is used together with a `string`-type constant in the `szMsgText` variable.

Example of valid specification:

```
DEFINE
{
   const integer OK_END =0;
   const integer NG_END = -1;
   const string szMsgText = "30";      // "!" has been
deleted.
}
```

`!` has been deleted from the `szMsgText` variable.

AITCE-0047

```
The use of labels is invalid in an expression.
```

Cause

A label is used in the expression.

Action

Do not use a label in the expression.

Example

Example of invalid specification:

```
DEFINE
{
   integer sloop_max = 0;
}
MAIN
{
ErrorLabel:
   if (ErrorLabel)      // The label is used in the if
structure.
      AIT_LogMessage("Setup(English)For Windows-Start");
      if (AIT_FileExists("#setup.exe") == 0)
         goto ErrorLabel;
         sloop_max = 0;
      endif;
   endif;
}
```

A label cannot be used in the expression. In this example, the label `ErrorLabel` is used in the `if` structure.

Example of valid specification:

```
DEFINE
{
   integer sloop_max = 0;
   string stMsgText;
}
MAIN
{
   if(1)      // The label has been deleted from the
expression.
      if (AIT_FileExists("#setup.exe") == 0)
         goto ErrorLabel;
         sloop_max = 0;
      endif;
ErrorLabel:
      stMsgText = "Setup(English) " + InstallerName + "
Not Found";
      AIT_LogMessage(stMsgText);
   endif;
}
```

The label `ErrorLabel` has been deleted from the `if` structure to validate the value in the expression.

AITCE-0048

`The "!" operator is not allowed in case statements.`

Cause

An expression is specified in a `case` label. Only constant values can be specified in the `case` labels of `switch` statements.

Action

Do not specify an expression in a `case` label.

Example

Example of invalid specification:

```
DEFINE
{
   const integer FileVersion = 7;
   string stMsgText;
}
MAIN
{
   switch (FileVersion)
      case 7:
         ...
         ...
         break;
      case 5 + 1:      // Expression cannot be used in a
case label.
         ...
         ...
         break;
      default:
         ...
         ...
         break;
   endswitch;
}
```

Only a constant can be used in the `switch` statement. In this example, the second `case` label is assigned an additive expression.

Example of valid specification:

```
DEFINE
{
   const integer FileVersion = 7;
   string stMsgText;
}
MAIN
{
   switch (FileVersion)
      case 7:
         ...
         ...
         break;
```

```
      case 6:      // The expression has been deleted,
and a constant
                   // is specified, instead.
         :
         :
         break;
      default:
         :
         :
         break;
   endswitch;
}
```

The additive expression has been deleted from the second `case` statement, and a constant is specified, instead.

AITCE-0050

`The AIT file version is higher than the DLL version of the execution engine.`

Cause

The AIT file version may be higher than the DLL version of the execution engine.

Action

Before executing an AIT file, check whether the DLL version of the script engine is higher than ScriptFileVersion indicated in the PACKAGE_INFO section. `ScriptFileVersion` must be lower than the DLL version.

AITCW-0016

`Conversion from` *higher-ranking-data-type* `to` *lower-ranking-data-type*`: Possible loss of accuracy`

Cause

If you attempt to assign a lower-ranking data type a higher-ranking data type in the AIT file, the value is rounded down, being assigned to the lower-ranking data type. For example, this message is displayed if a `float`-type variable has been assigned to an `integer`-type variable.

Action

Use a higher-ranking data type.

Example

Example of invalid specification:

```
DEFINE
{
   const integer OK_END = 0;
   const integer SLEEP_TIME = 3.8;     // An attempt has
```

```
been made to assign the integer variable the float-type
variable. The value is rounded down, with only 3 saves in
the variable.
}
```

In this example, a `float`-type value is assigned to the `SLEEP_TIME` variable of type `integer`. The assigned value is rounded down, with only the `integer`-type value saved in the `SLEEP_TIME` variable.

Example of valid specification:

```
DEFINE
{
   const integer OK_END =0;
  const float SLEEP_TIME = 3.8;     // The variable type
has been changed to the float type.
}
```

The data type of the `SLEEP_TIME` variable has been changed from `integer` to `float` to hold a `float`-type value.

AITCW-0017

`Unsafe use of boolean variable for` *processing*`.`

Cause

If you have specified a `bool`-type variable in an unexpected way, this warning message is displayed. For example, if you have specified a `bool`-type variable for a divisional or remainder operation (`/` or `%`), this leads to *division by 0* or *0/0 contradiction*, and this warning message will be displayed.

Action

In such a case, do not specify any variables of type `bool`.

Example

Example of invalid specification:

```
DEFINE
{
   const integer sloop_max = 30;
   integer sloop_count;
   bool IsPathSet = false;
}
MAIN
{
  sloop_count = sloop_max / IsPathSet;     // The Boolean
variable is specified as the divisor.
}
```

If a `bool`-type variable has been used, this leads to a division by 0, and this warning message will be displayed. In this example, the `IsPathSet`

variable of type `bool` is specified as the divisor, leading to a *division by 0*.

Example of valid specification:

```
DEFINE
{
   integer NG_END = 1;
   const integer sloop_max = 30;
   integer sloop_count;
}
MAIN
{
   sloop_count = sloop_max / NG_END;     // The Boolean
variable has been deleted, and the integer-type variable
has been specified as the divisor.
}
```

The `IsPathSet` variable of type `bool` has been deleted, and the `NG_END` variable of type `integer` has been specified as the divisor.

AITCW-0028

`Unsafe mix of type` *data-type-1* `and type` *data-type-2* `in operation operator.`

Cause

A mixture of `integer` and `bool` types is specified in such as bitwise logical AND and OR operations.

Action

For the indicated operand, make sure that the data types of the expressions are consistent.

Example

Example of invalid specification:

```
DEFINE
{
   integer sloop_max = 0;
   bool IsPathSet;
}
MAIN
{
  sloop_max = sloop_max & IsPathSet;     // The integer
and bool types are specified for bit logical AND.
}
```

In this example, the `integer` and `bool` types are specified for the AND operation, with data types mixed.

Example of valid specification:

```
DEFINE
{
   integer sloop_max = 0;
   integer sloop_count;
}
MAIN
{
   sloop_max = sloop_max & sloop_count;      // Integer
types have been specified for bit logical AND.
}
```

The operands of the & operator have the same data type `integer`.

AITCW-0033

`Switch statement contains only default label.`

Cause

The `switch` statement contains only a `default case` label, and contains no `case` labels. This is equivalent to a statement sequence error.

Action

Specify at least one `case` label in the `switch` statement.

Example

Example of invalid specification:

```
DEFINE
{
   const string FileVersion = "7.1";
   string stMsgText;
}
MAIN
{
   switch(FileVersion)
      default:      // Only the default label is used in
the switch statement.
            :
            :
           break;
   endswitch;
}
```

All `switch` statements require at least one `case` label. In this example, however, the `case` statement does not exist, with only the `default` statement specified.

Example of valid specification:

```
DEFINE
{
```

```
   const string FileVersion = "7.1";
   string stMsgText;
}
MAIN
{
   switch(Version )
      case "7.1":      // The switch statement contains
the case statement.
         ...
         ...
         break;
      default:
         ...
         ...
         break;
   endswitch;
}
```

A `case` statement has been added to the `switch` statement.

AITCW-0035

label-name`: Unreferenced label.`

Cause

The indicated label has been defined but has not been referenced. It is ignored.

Action

Specify a `goto` statement that references the label name.

Example

Example of invalid specification:

```
DEFINE
{
   const string ExeVersion = "7.1";
   const string FileVersion = "7.1";
   integer sloop_max = 0;
}
MAIN
{
   if (ExeVersion == FileVersion )
ErrorLabel:   // The goto statement associated with the
label does not exist.
      AIT_LogMessage("Setup(English)For Windows-Start");
      if (AIT_FileExists("#setup.exe") == 0)
         sloop_max = 0;
      endif;
   endif;
}
```

The label requires the associated `goto` statement. If you have specified a label statement with no `goto` statement, the label statement is ignored. In this example, label `ErrorLabel` is specified with no `goto` statement.

Example of valid specification:

```
DEFINE
{
   const string ExeVersion = "7.1";
   const string FileVersion = "7.1";
   integer sloop_max = 0;
}
MAIN
{
   if (ExeVersion == FileVersion)
      AIT_LogMessage("Setup(English)For Windows-Start");
       if (AIT_FileExists("#setup.exe") == 0)
         goto ErrorLabel;      // The goto statement has
been specified.
          sloop_max = 0;
       endif;
ErrorLabel:      // The label is defined.
       stMsgText = "Setup(English) " + InstallerName + "
Not Found";
       AIT_LogMessage(stMsgText);
   endif;
}
```

The `goto` statement associated with the label `ErrorLabel` has been specified.

Another means is to delete unnecessary label statements.

AITCW-0036

variable name`: Unreferenced variable.`

Cause

The variable defined in the DEFINE section is not referenced.

Action

Delete defines variables that are unnecessary or not used.

Example

Example of invalid specification:

```
DEFINE
{
   const string ExeVersion = "7.1";
   const string FileVersion = "7.1";
   integer sloop_max;      // This variable is not
```

```
referenced from anywhere in the program.
   string stMsgText;
}
MAIN
{
   if (ExeVersion == FileVersion)
      AIT_LogMessage("Setup(English)For Windows-Start");
       if (AIT_FileExists("#setup.exe") == 0)
         stMsgText = "Setup(English) " + InstallerName +
" Not Found";
       endif;
   endif;
}
```

In this example, the `sloop_max` variable is defined in the DEFINE section, but is not used in the MAIN section.

Example of valid specification:

```
DEFINE
{
   const string ExeVersion = "7.1";
   const string FileVersion = "7.1";
   integer sloop_max;
   string stMsgText;
}
MAIN
{
   if (ExeVersion == FileVersion)
      AIT_LogMessage("Setup(English)For Windows-Start");
       if (AIT_FileExists("#setup.exe") == 0)
          sloop_max = 0;// Variable sloop_max is used.
         stMsgText = "Setup(English) " + InstallerName +
" Not Found";
          AIT_LogMessage(stMsgText);
       endif;
   endif;
}
```

The `sloop_max` variable is used in the MAIN section.

Another means is to delete variables unnecessary in the MAIN section from the DEFINE section.

Appendixes

A. Menus
B. Editing a Program Product ID File
C. Reference Material for This Manual
D. Glossary

A. Menus

Table A-1 lists the menus available in the Automatic Installation Tool window.

Table A-1: Menus available in the Automatic Installation Tool window

Menu		Description	Shortcut
File	**New**	Creates a new AIT file.	**Ctrl+N**
	Open	Opens an existing AIT file.	**Ctrl+O**
	Close	Closes the active AIT file.	-
	Save	Saves the active AIT file.	**Ctrl+S**
	Save As	Saves the active AIT file under a new filename.	-
	Save All	Saves all the changed AIT files on the window.	-
	Print	Prints the active AIT file.	**Ctrl+P**
	Print Preview	Displays the print image of the active AIT file.	-
	Print Setup	Displays the Printer Setup dialog box. You can change printer setup.	-
	Recent File	Displays a recently used file selected from the list.	-
	Exit	Exits the Automatic Installation Tool.	**Alt+F4**
Edit	**Undo**	Cancels the immediately preceding edit operation.	**Ctrl+Z**
	Redo	Redoes the immediately preceding edit operation that was canceled.	**Ctrl+Y**
	Cut	Cuts and pastes a string to the clipboard.	**Ctrl+X**
	Copy	Copies and pastes a string to the clipboard.	**Ctrl+C**
	Paste	Pastes a string to the clipboard.	**Ctrl+V**
	Delete	Deletes a selected string.	**Delete**
	Select All	Selects all AIT files.	**Ctrl+A**
	Find	Displays the Find dialog box to find a specified string.	**Ctrl+F**
	Find Next	Continues to find a string in the AIT file following the cursor position.	**F3**
	Find Previous	Continues to find a string in the AIT file preceding the cursor position.	**Shift+F3**

Menu		Description	Shortcut
	Replace	Displays the Replace dialog box to replace the found string with a specified string. You can specify the normal expression for a found string.	**Ctrl+H**
	Go To	Displays the Go To dialog box to move the cursor to a specified line.	**Ctrl+G**
View	**Toolbar**	Selects the display or non-display of the tool bars below. • **Standard** tool bar • **Build** tool bar • **Utility** tool bar Displays the Customize dialog box, which allows tool bars and commands to be arranged.	-
	Status Bar	Selects the display or non-display of the status bar.	-
	Output	Displays the output window.	**Alt+F2**
	Watch	Displays the Watch window during debugging.	**Alt+F3**
	Workbook	• WorkBook Mode Selects the display or non-display of workbook mode during an active file view. • Toggle Icons Selects the display or non-display of the file workbook tab icon during a workbook mode view.	-
Build	**Syntax Check**	Checks the syntax of the active AIT file, displaying warning and error messages in the output window.	**Ctrl+F7**
	Execute	Checks the syntax of the active AIT file before the start of execution.	**Ctrl+F5**
Debug	**Go**	Executes the AIT file from the current statement to the breakpoint.	**F5**
	Stop Debugging	Terminates debugging to return to normal edit mode.	**Shift+F5**
	Step by Step	Executes only the current statement in the AIT file.	**F10**
	Run to Cursor	Stops execution of the AIT file before the cursor line as if a temporary breakpoint were set in the cursor line.	**Ctrl+F10**
	Add/Remove Breakpoint	Allows you to add or remove a breakpoint at a specified position.	**F9**
	Breakpoints Setup	Displays the Breakpoints Setup dialog box that allows you to add or remove a breakpoint.	**Alt+F9 or Ctrl+B**

Menu		Description	Shortcut
Tools	**Recorder**	Displays the Recorder dialog box used to record user operations for automatic AIT file generation.	**Ctrl+R**
	Window Properties	Displays the Window Properties dialog box that allows you to attributes of a specified window or control.	**Ctrl+W**
	Package Information	Displays the Package Information dialog box used to generate the PACKAGE_INFO section.	**Ctrl+P**
	Options	Displays the Options dialog box that allows you to set an AIT file format, the number of messages to be displayed during a syntax check, the number of generations to be covered by recorder log output, and other items.	**Ctrl+O**
	Customize	Displays the Customize dialog box that allows you to set tool bar and command positions.	**Ctrl+C**
Window	**New Window**	Opens a new active AIT file.	**Ctrl+N**
	Cascade	Cascades all opened windows.	**Ctrl+C**
	Tile	Tiles all opened windows.	**Ctrl+T**
	Arrange Icons	Arranges the window icons in the lower line.	**Ctrl+A**
	Close	Closes the active AIT file.	**Ctrl+O**
	Close All	Closes all AIT files.	**Ctrl+L**
Help	**Table Of Contents**	Displays the contents of the Automatic Installation Tool guide.	**Ctrl+C**
	About Automatic Installation Tool	Displays the version of the Automatic Installation Tool.	**Ctrl+A**

Legend:

-: Not applicable

B. Editing a Program Product ID File

The program product ID file generated when an AIT file is created is assigned the name `PPDEFAIT.DMP` and stored in *JP1/IT-Desktop-Management-2-installation-folder*`\DMPRM`. When you edit the generated file, make sure that the file has the format explained below.

The *program product ID file* contains the information to associate the software you want to distribute with the AIT file you created. You must place the program product ID file in a specified location under the name of `PPDEFAIT.DMP`.

The format of the program product ID file is:

information-map;*package-ID*;*version*;*product*;*AIT-file-full-path*;*file-name-1*;*file-name-2*;...;*file-name-n*;

In the program product ID file, the definition of a single software package is specified on one line, which must not exceed 499 bytes. If you want to remotely install multiple programs, define them in multiple lines. The items in a line are delimited by a semicolon (`;`). The following describes the items. When you perform packaging, *package-ID*, *version*, and *product* defined in the program product ID file are displayed in the JP1/ITDM2 Packaging dialog box. You cannot change them with the dialog box.

- *information-map* (Required)

  Specify `000001`.

- *package-ID* (Required)

  This is the package ID of software you want to distribute. Specify the package ID defined in the AIT file you want to package together with the program.

- *version* (Required)

  This is the version of software you want to distribute. Specify the version defined in the AIT file.

- *product* (Required)

  This is the package name of software you want to distribute. Specify the package name defined in the AIT file.

- *AIT-file-full-path* (Required)

  Specify the full path name of the AIT file including the drive name using a character string consisting of up to 256 bytes.

- *file-name-1 ... file-name-n* (Required)

  You can specify one or more file names for identifying the software to be distributed. When all the specified files are found during packaging, JP1/Software

Distribution considers the software as an item to be distributed.

The following gives an example of the *program product ID file* when AIT file `AR600.ais` is located in `C:\ADOBE`.

```
000001;ADOBEREADER;0600;Adobe Reader 6.0;C:\ADOBE\AR600.ais;AdbeRdr60_enu_full.exe;
```

C. Reference Material for This Manual

C.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):

- *Job Management Partner 1 Version 10 Job Management Partner 1/IT Desktop Management 2 Getting Started* (3021-3-367(E))
- *Job Management Partner 1 Version 10 Job Management Partner 1/IT Desktop Management 2 Overview and System Design Guide* (3021-3-368(E))
- *Job Management Partner 1 Version 10 Job Management Partner 1/IT Desktop Management 2 Configuration Guide* (3021-3-369(E))
- *Job Management Partner 1 Version 10 Job Management Partner 1/IT Desktop Management 2 Administration Guide* (3021-3-370(E))
- *Job Management Partner 1 Version 10 Job Management Partner 1/IT Desktop Management 2 Distribution Function Administration Guide* (3021-3-373(E))
- *Job Management Partner 1 Version 10 Job Management Partner 1/IT Desktop Management 2 - Asset Console Description* (3021-3-375(E))
- *Job Management Partner 1 Version 10 Job Management Partner 1/IT Desktop Management 2 - Asset Console Planning and Configuration Guide* (3021-3-376(E))
- *Job Management Partner 1 Version 10 Job Management Partner 1/IT Desktop Management 2 - Asset Console Administration Guide* (3021-3-377(E))
- *Job Management Partner 1 Version 10 Job Management Partner 1/IT Desktop Management 2 Messages* (3021-3-378(E))

C.2 Conventions: Abbreviations for product names

This manual uses the following abbreviations for the names of products related to JP1/IT Desktop Management 2:

Abbreviation	Full name or meaning
JP1/IT Desktop Management 2	Job Management Partner 1/IT Desktop Management 2 - Agent
	Job Management Partner 1/IT Desktop Management 2 - Manager

This manual also uses the following abbreviations:

Abbreviation	Full name or meaning
ActiveX	ActiveX(R)
InstallShield	InstallShield(R)
Itanium 2	Intel(R) Itanium(R) 2 processor

This manual uses the following abbreviations for Microsoft product names:

Abbreviation	Full name or meaning
Microsoft Internet Explorer	Microsoft(R) Internet Explorer
	Windows(R) Internet Explorer(R)
MS-DOS	Microsoft(R) MS-DOS(R)

Abbreviation				Full name or meaning
Windows	Windows 7	Windows 7 Enterprise		Microsoft(R) Windows(R) 7 Enterprise
		Windows 7 Home Basic		Microsoft(R) Windows(R) 7 Home Basic
		Windows 7 Home Premium		Microsoft(R) Windows(R) 7 Home Premium
		Windows 7 Professional		Microsoft(R) Windows(R) 7 Professional
		Windows 7 Starter		Microsoft(R) Windows(R) 7 Starter
		Windows 7 Ultimate		Microsoft(R) Windows(R) 7 Ultimate
	Windows 8	Windows 8		Windows(R) 8
		Windows 8 Enterprise		Windows(R) 8 Enterprise
		Windows 8 Pro		Windows(R) 8 Pro
	Windows 8.1	Windows 8.1		Windows(R) 8.1
		Windows 8.1 Enterprise		Windows(R) 8.1 Enterprise
		Windows 8.1 Pro		Windows(R) 8.1 Pro
	Windows Server 2003[#1]	Windows Server 2003 [#1]	Windows Server 2003 Datacenter	Microsoft(R) Windows Server(R) 2003 R2, Datacenter Edition
				Microsoft(R) Windows Server(R) 2003, Datacenter Edition
			Windows Server 2003 Enterprise	Microsoft(R) Windows Server(R) 2003 R2, Enterprise Edition
				Microsoft(R) Windows Server(R) 2003, Enterprise Edition
			Windows Server 2003 Standard	Microsoft(R) Windows Server(R) 2003 R2, Standard Edition
				Microsoft(R) Windows Server(R) 2003, Standard Edition
		Windows Server 2003 (x64)	Windows Server 2003 Datacenter (x64)	Microsoft(R) Windows Server(R) 2003 R2, Datacenter x64 Edition
				Microsoft(R) Windows Server(R) 2003, Datacenter x64 Edition
			Windows Server 2003 Enterprise (x64)	Microsoft(R) Windows Server(R) 2003 R2, Enterprise x64 Edition
				Microsoft(R) Windows Server(R) 2003, Enterprise x64 Edition

#1

If Windows Server 2003 is noted alongside Windows Server 2003 (x64), the description for Windows Server 2003 does not apply to Windows Server 2003 (x64).

#2

If Windows Server 2008 is noted alongside Windows Server 2008 R2, the description for Windows Server 2008 does not apply to Windows Server 2008 R2.

#3

If Windows Server 2012 R2 is noted alongside Windows Server 2012, the description for Windows Server 2012 does not apply to Windows Server 2012 R2.

C.3 Conventions: Acronyms

This manual also uses the following acronyms:

Acronym	Full name or meaning
API	application programming interface
ASCII	American Standard Code for Information Interchange
CD	Compact Disc
CD-ROM	Compact Disc Read Only Memory
DLL	dynamic linking library
FD	floppy disk
GUI	graphical user interface
HD	hard disk
HTML	Hyper Text Markup Language
I/O	input/output
ID	identifier
IME	input method editor
IP	Internet protocol
MBCS	multi-byte character set
MS-DOS	Microsoft Disk Operating System
ODBC	Open Database Connectivity
OS	operating system

Acronym	Full name or meaning
PC	personal computer
PDF	Portable Document Format
PP	program product

C.4 Conventions: Fonts and symbols

The Windows menu names used in this manual assume the operating systems shown below:

For managing servers, database servers, computers with Network Monitor enabled, and computers on which controllers are installed:

Windows Server 2008

For computers with agent software installed:

Windows XP

The **Start** menu is not displayed in Windows 8.1, Windows 8, and Windows Server 2012. Select the menu by opening the Start window from the bottom left corner of the desktop.

General conventions used in this manual

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.

Text formatting	Convention
`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.`

Syntactical conventions used in this manual

The following table explains the symbols used in this manual:

Symbol	Convention
`|`	In syntax explanations, a vertical bar separates multiple items, and has the meaning of OR. For example: `A|B|C` means A, or B, or C.
`{ }`	In syntax explanations, 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.
`[ ]`	In syntax explanations, square brackets indicate that the enclosed item or items are optional. For example: `[A]` means that you can specify `A` or nothing. `[B|C]` means that you can specify `B`, or `C`, or nothing.
`...`	In coding, an ellipsis (...) indicates that one or more lines of coding have been omitted. In syntax explanations, 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.

C.5 Conventions: Version numbers

The version numbers of Hitachi program products are usually written as two sets of two digits each, separated by a hyphen. For example:

- Version 1.00 (or 1.0) is written as 01-00.
- Version 2.05 is written as 02-05.
- Version 2.50 (or 2.5) is written as 02-50.
- Version 12.25 is written as 12-25.

The version number might be shown on the spine of a manual as *Ver. 2.00*, but the same version number would be written in the program as *02-00*.

C.6 About online help

JP1/IT Desktop Management 2 provides online help in relation to the following subjects:

How to use products

This help provides operation examples for products, instructions on how to use the product's functions, and troubleshooting procedures. You can view these help topics by selecting **IT Desktop Management 2 Help** from the **Help** menu in the JP1/IT Desktop Management 2 user interface.

Window descriptions

This help describes how to use the screen that is currently displayed. You can view these help topics by clicking the **Help** button in the user interface.

Distribution function

Help topics that relate to the distribution function are compiled from the following manuals:

Job Management Partner 1 Version 10 Job Management Partner 1/IT Desktop Management 2 Distribution Function Administration Guide

Job Management Partner 1 Version 10 Job Management Partner 1/IT Desktop Management 2 Automatic Installation Tool Administration Guide

Distribution function (agent side)

Help topics that relate to the distribution function on the agent side are excerpted from the description of the agent in the *Job Management Partner 1 Version 10 Job Management Partner 1/IT Desktop Management 2 Distribution Function Administration Guide*.

C.7 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.

D. Glossary

associated label

A type of control. The user cannot operate or change the text in the associated label. Typically, associated labels are used to explain other controls such as list boxes without captions. The text immediately before a control in tab order is the control's associated label.

caption

The text displayed at the top of a control or in the title bar of a window.

child window

A window included in another window (parent window). For example, typically, an application uses child windows to divide one parent window into several work areas. In a dialog box, check boxes, text boxes, and other controls are equivalent to child windows of the dialog box.

control

Text boxes, command buttons, and other graphical objects placed on windows are called controls. Controls displayed on a window are equivalent to child windows of the window.

date/time picker

A control that allows you to select a date or time.

event

Any user action or program status change that can be detected by a computer. Events generated by user operations include mouse clicks, menu selections, and key presses. Other events, which are generated inside the operating system or applications, include exceptions, window creations, and window closures.

focus

A temporary attribute of the user interface objects (windows, views, dialog boxes, buttons, etc.). The object on which the focus is placed can accept user entry. For example, when a text box is *focused*, if you enter a character string, the character string is displayed in the text box. Normally, the focused object is differentiated from other objects in its appearance (with highlighting, for example).

handle

A unique four-byte integer value used to identify each window or control to be accessed. The operating system assigns handles.

tab order

The order in which the focus is moved among the controls placed on a window by pressing the **Tab** key.

Index

A

abbreviations for products 371
acronyms 374
Add logging (check box) 23
Add/Remove Breakpoint 63
adding operators 84
AIT file 2
    checking and modifying automatically generated AIT file 42
    correctly identifying end of installation operations 44
    creating 4
    debugging 58
    editing 34
    example of completed AIT file 51
    format 70
    note on using text to identify window 46
    notes about creating and using 8
    procedure for creating 12
    structure 12, 13
AIT language
    binary operators supported by 80
    unary operators supported by 80
ait.log 308
AIT_ASCIIToChar 124
AIT_ChangeFileAttribute 124
AIT_CharToASCII 126
AIT_CheckResolution 126
AIT_ComboBoxCloseUp 127
AIT_ComboBoxDropDown 128
AIT_CtrlClick 129
AIT_CtrlItemCount 131
AIT_CtrlItemIndex 133
AIT_CtrlSetFocus 36, 135
    example of using 36
AIT_DefaultButtonCount 137
AIT_DirCopy 137
AIT_DirCreate 138
AIT_DirRemove 139
AIT_DMPSTRC 140
AIT_Exec 141
AIT_ExecCommand 143
AIT_ExistWindow 144
AIT_Exit 145
AIT_FileClose 145
AIT_FileCopy 146
AIT_FileDelete 147
AIT_FileEOF 148
AIT_FileExists 149
AIT_FileGetLine 150
AIT_FileGetPos 152
AIT_FileOpen 153
AIT_FilePutLine 155
AIT_FileRename 157
AIT_FileSetPos 158
AIT_FileSize 159
AIT_FindCloseFile 161
AIT_FindFirstFile 161
AIT_FindNextFile 163
AIT_FindSubStr 164
AIT_FocusWindow 36, 165
    example of using 36
AIT_GetCtrlText 166
AIT_GetCtrlTextLen 168
AIT_GetCurrentDirectory 170
AIT_GetDate 171
AIT_GetDtPickerDate 171
AIT_GetDtPickerTime 173
AIT_GetEditCurrentLineIndex 175
AIT_GetEditFirstLineIndex 176
AIT_GetEditTextLineLen 178
AIT_GetEnv 179
AIT_GetErrorText 179
AIT_GetIndexText 180
AIT_GetIndexTextLen 182
AIT_GetKeyState 183
AIT_GetLastError 184
AIT_GetMenu 185

AIT_GetMenuIndex 186
AIT_GetMenuText 187
AIT_GetOSType 188
AIT_GetProfileFirstSection 190
AIT_GetProfileFirstSectionNames 191
AIT_GetProfileNextSection 193
AIT_GetProfileNextSectionNames 193
AIT_GetProfileString 194
AIT_GetSubMenu 195
AIT_GetSubStr 196
AIT_GetTime 197
AIT_GetWindowText 198
AIT_IMEGetConversionStatus 199
AIT_IMEGetOpenStatus 200
AIT_IMEGetProperty 201
AIT_IMEGetStatusWindowPos 203
AIT_IMESetConversionStatus 204
AIT_IMESetOpenStatus 205
AIT_IMESetStatusWindowPos 205
AIT_IMESimulateHotKey 206
AIT_InitLog 208
AIT_IsEmpty 210
AIT_LogMessage 210
AIT_MenuItemClick 211
AIT_MessageBox 213
AIT_MinWnd 214
AIT_MouseClick 215
AIT_MouseDblClk 216
AIT_MouseDown 218
AIT_MouseDragDrop 219
AIT_MouseMoveTo 221
AIT_MouseUp 222
AIT_PlayKey 38, 224
    example of using 39
AIT_PostMessage 226
AIT_RegCloseKey 227
AIT_RegCreateKey 228
AIT_RegDeleteKey 229
AIT_RegDeleteValue 230
AIT_RegGetDWORDValue 231
AIT_RegGetStringValue 233
AIT_RegisterWindowMessage 234
AIT_RegKeyExists 235
AIT_RegOpenKey 236
AIT_RegSetDWORDValue 237
AIT_RegSetStringValue 239
AIT_RegValueExists 240
AIT_SelectIPAddressField 241
AIT_SelectListItem 243
AIT_SelectMultipleListItem 245
AIT_SelectText 246
AIT_SetActWnd 248
AIT_SetCheck 249
AIT_SetComboEditSelText 250
AIT_SetCurrentDirectory 252
AIT_SetDefaultWaitTimeout 253
AIT_SetDtPickerDate 253
AIT_SetDtPickerTime 255
AIT_SetKeyState 257
AIT_SetProfileString 258
AIT_SetScrollPos 259
AIT_SetSpinPos 262
AIT_SetWndPos 264
AIT_SetWndPosSize 265
AIT_Sleep 266
AIT_StatusBox 266
AIT_StatusBoxClose 268
AIT_StrLeft 269
AIT_StrLength 269
AIT_StrLower 270
AIT_StrLTrim 270
AIT_StrRight 271
AIT_StrRTrim 272
AIT_StrTrim 272
AIT_StrUpper 273
AIT_TaskbarClk 273
AIT_TaskbarHasFocus 274
AIT_TaskbarSetFocus 275
AIT_VerifyCharPos 275
AIT_VerifyCount 277
AIT_VerifyDateTime 278
AIT_VerifyDefaultButton 279
AIT_VerifyEnabled 37, 280
    example of using 37
AIT_VerifyExistence 37, 282
    example of using 37
AIT_VerifyFirstVisible 284
AIT_VerifyFocus 285

AIT_VerifyIndex 286
AIT_VerifyKeyState 288
AIT_VerifyLine 289
AIT_VerifyLocation 291
AIT_VerifyMenuChecked 293
AIT_VerifyMenuEnabled 294
AIT_VerifyNoOfCtrls 295
AIT_VerifyPos 38, 296
    example of using 38
AIT_VerifySelected 297
AIT_VerifyState 299
AIT_VerifyText 300
AIT_Wow64DisableWow64Redirection 301
AIT_Wow64RevertWow64Redirection 301
aitapi.log 308
aitexec.log 308
API for finding window 35
API for operating window 35
API function reference 113
API functions 114
    details 123
    for character string operations 118
    for check operations 116
    for checking resolution 117
    for date/time operations 117
    for directory operations 120
    for file operations 120
    for INI file operations 121
    for interfacing with JP1/IT Desktop Management 2 122
    for message operations 119
    for Recorder operations 121
    for redirect operations 119
    for registry operations 119
    for taskbar operations 121
    for utility operations 122
    for window operations 114
API functions, example of using
    deleting carriage return and linefeed characters 303
    extracting characters 303
    Manipulating HKEY_CURRENT_USER registry key during remote installation 304
assignment 80
Automatic Installation Tool
    functionalities 10

B

binary operator 80
bitwise AND operator 88
bitwise operators 87
bitwise OR operator 88
break (statement) 100
breakpoint 61
Breakpoints Setup dialog box 61
Build (menu) 59

C

comparison operators 86
compile 58
constants 92
continue (statement) 99
conventions
    abbreviations for products 371
    acronyms 374
    fonts and symbols 375
    KB, MB, GB, and TB 377
    version numbers 376
creating
    AIT file 4
    program product ID file 4

D

data types 75
    bool 76
    float 75
    integer 75
    string 77
debugging 60
    AIT file 58
    stopping execution of AIT file at breakpoints 64
    stopping execution of AIT file at specific point 64
    stopping execution of AIT file before cursor line 65

stopping execution of script in units of statements 65
values of variables, changing 65
values of variables, monitoring 65
DEFINE 72
do-while (statement) 96

E

editing
program product ID file 369
ERROR 74
error handling 48
example of coding 50
event flag 41
Execute (Build menu) 59

F

files
identifying software to be distributed 31
storing created files 4
Finder 17
flag, automatically generated 40
font conventions 375
for-next (statement) 97
format of API function explanations 123
format of message explanations 308
function calls 103

G

GB meaning 377
generating
PACKAGE_INFO section 28
program product ID file 28
global variables 46
example of using 48
glossary 378
goto (statement) 93

I

if-else-endif (statement) 94
IME operations 117
installation
conditions 42
operations, recording 20
installer
specifying 31
installer windows
acquiring properties 17
checking properties of 15
checking sequence of 15
invalid flag 41
items you should check 15

J

Jump to First Child Window 18
Jump to Next Window 19
Jump to Parent Window 18
Jump to Previous Window 18

K

KB meaning 377
keywords 105

L

label (statement) 94
linkage
with Packager 46
with Remote Installation Manager 46
linking order 90
log files 308
logical AND operator 89
logical operators 88
logical OR operator 89

M

macros 107
for check operations 107
for directory operations 111
for error logging 111
for file operations 109
for IME operations 109
for message operations 109
for registry operations 110
for utility operations 110
for window operations 107
MAIN 73

MB meaning 377
menus 366
messages
checking 308
displayed during editing 311
displayed during execution of AIT files 324
displayed during parsing of AIT files 324
multi-assignment 82
multiplying operators 84

O

operators 80
adding operators 84
bitwise operators 87
comparison operators 86
logical AND operator 89
logical operators 88
logical OR operator 89
multiplying operators 84
priority 90
unary minus 83
unary not 83
unary plus 82
output window 59, 60

P

Package Information dialog box 28
Package Information tool 28
PACKAGE_INFO 71
PACKAGE_INFO section
generating 28
packaging directory 31
pausing and resuming recording 26
PPDEFAIT.DMP 369
PPDEFAIT.DMP (program product ID file) 4
priority 90
priority of operators 90
program flow control 93
program product ID file 4, 370
creating 4
editing 369
format 369
generating 28
properties, acquiring
control 17
window 17

R

Recorder 20
procedure for recording your installation operations 23
Recorder dialog box 23
recording installation operations 20
request OS to be restarted 26
remote installation using AIT file 3
procedure for 4
return code
example of coding for setting 50
setting 48

S

sections 71
overview 13
software to be distributed
files for identifying 31
special characters 78
specifying
files for identifying software to be distributed 31
installer 31
starting Automatic Installation Tool 10
Step by Step (Debug menu) 65
Stop Debugging (Debug menu) 64
storing created files 4
switch-endswitch (statement) 100
symbol conventions 375
Syntax Check 59

T

TB meaning 377
terminating Automatic Installation Tool 10
time-out
AIT file 6
toolbar buttons
Jump to First Child Window 18
Jump to Next Window 19
Jump to Parent Window 18

Jump to Previous Window 18
troubleshooting 307
type conversion 81

U

unary minus 83
unary not 83
unary operator 80
unary plus 82

V

variables 92
version number conventions 376

W

Watch window 65
while-loop (statement) 95
window
conditions for identifying 46
window flag 40
window processing 34
example of automatically generated codes 39
Window Properties dialog box 17
Window Properties tool 15