

uCosminexus Service Platform

Overview

3020-3-Y42-40(E)

■ Relevant program products

For the relevant program products, see the preface section in the manual *uCosminexus Application Server Overview*.

■ Export restrictions

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

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

■ Trademarks

HITACHI and uCosminexus are trademarks or registered trademarks of Hitachi, Ltd.

IBM, AIX are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide.

All Borland brand and product names are trademarks or registered trademarks of Borland Software Corporation in the United States and other countries.

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

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

Netscape is a trademark of AOL Inc. in the U.S. and other countries.

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

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

SAP, mySAP, R/3, and other SAP products and services mentioned herein are trademarks or registered trademarks of SAP AG in Germany and other countries.

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

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

W3C is a trademark (registered in numerous countries) of the World Wide Web Consortium.

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

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

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

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

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

Eclipse is an open development platform for tools integration provided by Eclipse Foundation, Inc., an open source community for development tool providers.

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

■ Microsoft product name abbreviations

This manual uses the following abbreviations for Microsoft product names.

Abbreviation			Full name or meaning
Windows	Windows Server 2008	Windows Server 2008 x86	Microsoft(R) Windows Server(R) 2008 Standard 32-bit
			Microsoft(R) Windows Server(R) 2008 Enterprise 32-bit
		Windows Server 2008 x64	Microsoft(R) Windows Server(R) 2008 Standard
			Microsoft(R) Windows Server(R) 2008 Enterprise
		Windows Server 2008 R2	Microsoft(R) Windows Server(R) 2008 R2 Standard
			Microsoft(R) Windows Server(R) 2008 R2 Enterprise
			Microsoft(R) Windows Server(R) 2008 R2 Datacenter
	Windows Server 2012	Windows Server 2012 Standard	Microsoft(R) Windows Server(R) 2012 Standard
		Windows Server 2012 R2 Standard	Microsoft(R) Windows Server(R) 2012 R2 Standard

Abbreviation			Full name or meaning
Windows	Windows Server 2012	Windows Server 2012 Datacenter	Microsoft(R) Windows Server(R) 2012 Datacenter
		Windows Server 2012 R2 Datacenter	Microsoft(R) Windows Server(R) 2012 R2 Datacenter
	Windows XP		Microsoft(R) Windows(R) XP Professional Operating System
	Windows Vista	Windows Vista Business	Microsoft(R) Windows Vista(R) Business (32-bit Edition)
		Windows Vista Enterprise	Microsoft(R) Windows Vista(R) Enterprise (32-bit Edition)
		Windows Vista Ultimate	Microsoft(R) Windows Vista(R) Ultimate (32-bit Edition)
	Windows 7	Windows 7 x86	Microsoft(R) Windows(R) 7 Professional (32-bit Edition)
			Microsoft(R) Windows(R) 7 Enterprise (32-bit Edition)
			Microsoft(R) Windows(R) 7 Ultimate (32-bit Edition)
		Windows 7 x64	Microsoft(R) Windows(R) 7 Professional (64-bit Edition)
			Microsoft(R) Windows(R) 7 Enterprise (64-bit Edition)
			Microsoft(R) Windows(R) 7 Ultimate (64-bit Edition)
	Windows 8	Windows 8 x86	Windows(R) 8 Pro (32-bit Edition)
			Windows(R) 8 Enterprise (32-bit Edition)
		Windows 8 x64	Windows(R) 8 Pro (64-bit Edition)
			Windows(R) 8 Enterprise (64-bit Edition)
	Windows 8.1	Windows 8.1 x86	Windows(R) 8.1 Pro (32-bit Edition)
			Windows(R) 8.1 Enterprise (32-bit Edition)
		Windows 8.1 x64	Windows(R) 8.1 Pro (64-bit Edition)
			Windows(R) 8.1 Enterprise (64-bit Edition)

Note that a 32-bit edition of Windows might be called *Windows x86*. Also note that a 64-bit edition of Windows might be called *Windows x64*.

■ 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.

Printed in Japan.

■ Issued

Oct. 2015: 3020-3-Y42-40(E)

■ Copyright

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

Preface

For the basics required to understand this manual, see the preface section in the manual *uCosminexus Application Server Overview*.

Contents

1	Functional Overview of Cosminexus Service Platform	1
1.1	Functional overview of the development environment	2
1.1.1	Functionality for creating message formats	2
1.1.2	Functionality for creating user-defined reception	2
1.1.3	Functionality for creating service adapters	3
1.1.4	Functionality for creating business processes	4
1.1.5	Functionality for creating data transformation definitions	6
1.1.6	Functionality for defining deployment	7
1.1.7	Packaging	8
1.1.8	Functionality for deploying to starting or stopping to deleting HCSC components, in a batch	9
1.1.9	Functionality for creating service requesters	10
1.1.10	Functionality for debugging business processes	10
1.2	Functional overview of the execution environment	12
1.2.1	Functionality for distributing services	12
1.2.2	Functionality for invoking various services	12
1.2.3	Functionality for executing business processes	13
1.3	Functional overview of the operating environment	14
1.3.1	Functionality for setting up the execution environment	14
1.3.2	Functionality for deploying HCSC components	15
1.3.3	Functionality for managing operations in the execution environment	15
1.3.4	Functionality for managing execution logs	17
1.4	Configuring clusters of the HCSC server	18
1.4.1	Redundant configuration of the HCSC server using the load-balancing functionality	18
1.4.2	Redundant configuration of the HCSC server using cluster software	18
1.4.3	Redundant configuration of the HCSC server using N-to-1 cluster configuration	21
1.5	Configuration pattern of an operating environment in the cluster configuration	22
1.6	Importing repositories at the time of distributed development	24
1.6.1	Flow for distributed development using the import component function	24
1.6.2	Points to be considered for distributed development	26

2	Functionality for Connecting to Various Types of Systems	27
2.1	Basic procedure of invoking service components	28
2.1.1	Basic structure	28
2.1.2	Procedure for invoking service components	31
2.1.3	Flow of invoking a service component with a message format different from the service component side	36
2.1.4	Identification information provided in the service component invocation process	38
2.2	Connecting to services by using Web Services (SOAP communication)	41
2.2.1	Relationship with the Communication Infrastructure for Web Services	41

2.2.2 Relationship between user messages and WSDL when a service component is invoked	42
2.2.3 Transactions when using Web Services (SOAP communication)	48
2.2.4 Cookie information in Web service (SOAP communication)	48
2.2.5 Inheriting HTTP header in Web service (SOAP communication)	51
2.2.6 SOAP message configuration in Web service (SOAP communication)	54
2.2.7 Dynamic change in connection destination in Web service (SOAP communication)	57
2.3 Connecting to services using Session Beans	63
2.3.1 Relationship with Application Server (EJB containers)	63
2.3.2 Relationship between user messages and stubs when a service component is invoked	63
2.3.3 Transactions when using Session Bean	65
2.3.4 Points to be considered when using SessionBean adapter	66
2.4 Connecting to services Using WS-R (WS-Reliability)	67
2.4.1 Relationship with Reliable Messaging	67
2.4.2 Flow of messages when invoking service components	67
2.4.3 Handling response messages when a synchronous service component is invoked	70
2.4.4 Transactions when using an MDB (WS-R)	72
2.4.5 Dead message queue	76
2.4.6 Names and number of queues used in an MDB (WS-R)	76
2.5 Connecting to services by using a database queue	77
2.5.1 Relationship with Reliable Messaging	77
2.5.2 Flow of messages when invoking service components	78
2.5.3 Handling response messages when a synchronous service component is invoked	81
2.5.4 Transactions when using an MDB (DB queue)	83
2.5.5 Dead message queue	83
2.5.6 Names and number of queues used in an MDB (DB queue)	83
2.6 Connecting to a database	85
2.6.1 Accessing a database using DB adapters	85
2.6.2 Setting up a DB connector to be used by DB adapters	87
2.6.3 Transactions while using DB adapter	89
2.6.4 Dynamic change in connection destination information for DB adapters	90
2.7 Connecting to OpenTP1	95
2.8 Connecting to a system handling a file	96
2.8.1 Accessing files using file adapter	96
2.8.2 Reading and writing files using adapter	97
2.8.3 File formats that can be used in adapters	98
2.8.4 Modes for file operations	98
2.8.5 Exclusive control while reading and writing files	99
2.9 Connection with Object Wrapper system	100
2.9.1 Access to Object Wrapper system using Object Access adapter	100
2.9.2 Message format transformation at the time of connection	100
2.10 Connecting to Message Queues	101

2.10.1 Invoking service components using Message Queue reception	101
2.10.2 Access to message queues using Message Queue adapter	112
2.11 Connecting to FTP Client and FTP Server	117
2.11.1 File transfer integrated with FTP	117
2.12 Connecting to the mail server	119
2.12.1 Preconditions	119
2.12.2 System using the mail adapter	119
2.12.3 Transactions when using the mail adapter	124
2.12.4 Mail information that can be set on the mail adapter	125
2.12.5 Managing the mail adapter timer	126
2.12.6 Authentication when sending email messages from the mail adapter	127
2.13 Connecting to client using HTTP communication	128
2.13.1 Invoking a service component using HTTP reception	128
2.13.2 Example of online system using HTTP reception	129
2.13.3 Example of HTTP reception configuration	134
2.13.4 HTTP reception request process	138
2.13.5 HTTP reception response process	143
2.13.6 Managing HTTP request files	150
2.13.7 Encoding and decoding HTTP requests and HTTP responses	152
2.13.8 Transactions to be performed while using HTTP reception	153
2.13.9 Managing HTTP reception timer	153
2.13.10 Points to be considered when using HTTP reception	154
2.14 Connecting to services using HTTP communication	156
2.14.1 Preconditions	156
2.14.2 Invoking service components using HTTP adapter	156
2.14.3 Relation between HTTP requests and request messages	157
2.14.4 Relation between HTTP responses and response messages	169
2.14.5 Inheriting Cookie information using HTTP adapter	175
2.14.6 Communication through proxy servers	178
2.14.7 Communication using connection continuity (Keep-Alive)	178
2.14.8 Communication using chunk transmission	179
2.14.9 Compressing and extracting HTTP body	180
2.14.10 Secure connection using WWW authentication	181
2.14.11 Managing HTTP adapter timer	182
2.14.12 Examples of a system using the HTTP adapter	183
2.14.13 Corresponding relation between request message (header) and HTTP-adapter runtime-environment (common) property file	186
2.15 Connecting to a system by using a custom reception	190
2.15.1 Configuring a custom reception	190
2.15.2 Overview of the custom reception framework	190
2.16 Connecting to a system by using a General custom adapter	192
2.16.1 Overview of General custom adapters	192

2.16.2 Overview of the custom adapter development framework	192
-------------------------------------------------------------	-----

3

Business Process Functionality	195
3.1 Overview of business processes	196
3.1.1 Reception and response	196
3.1.2 Service invocation	197
3.1.3 Java invocation	197
3.1.4 Data transformation	198
3.1.5 ASSIGN	200
3.1.6 Repetition	201
3.1.7 SWITCH	202
3.1.8 FLOW	204
3.1.9 Standby processing	205
3.2 Mechanism of business processes	207
3.2.1 Business process flow	207
3.2.2 Communication models of business processes	207
3.2.3 Process instances	208
3.2.4 Activity	211
3.2.5 Types and roles of variables	213
3.2.6 Role of a correlation set	214
3.3 Business processes that are to be made persistent and that are not to be made persistent	215
3.3.1 Business processes that are to be made persistent	215
3.3.2 Business processes that are not to be made persistent	215
3.4 Transaction of a business process	216
3.4.1 Transactions when ON is specified in compatibility with the business process status	217
3.4.2 Transactions when OFF is specified for compatibility with the business process status and ON is specified for compatibility with the invoke service activity status	225
3.4.3 Transactions when OFF is specified in both compatibility with the business process status and compatibility with the invoke service activity status	233
3.4.4 Transactionsof the invoke service activity	237
3.4.5 Transactionswhen you select the option to commit at start time and end time of scope	238
3.4.6 Transactionsof the scope activity	244
3.4.7 Transactions of the reply activity	245
3.5 Re-executing a business process	251
3.5.1 Re-executing business processes from windows in an operating environment	251
3.5.2 Re-executing business processes with commands	251
3.5.3 Re-executing business processes from service requesters (SOAP communication)	251
3.5.4 Re-executing business processes from service requesters (Session Bean)	252
3.6 Standby processing using business processes	254
3.7 Asynchronous execution of an activity that is defined after reply activity	257

4	Functionality for Smooth System Operations	259
4.1	Setting up a timer for an HCSC server	260
4.1.1	Timers that can be set up on an HCSC server	260
4.1.2	Changing the timeout value of service components	265
4.1.3	Precautions to be taken when setting up a timer	265
4.2	Changing the connection destination of service components	267
4.3	Catching a fault by using the SOAP fault operation definition file	268
4.4	Checking the operation status from an application	271
4.4.1	Overview of checking the operation status	271
4.4.2	Flow of checking the operation status	271
4.4.3	Precautions on checking the operation status	272
4.5	Complementing name space prefixes specified in attribute values of XML messages	274
4.6	Defining message format of any format	275
4.6.1	Flow of process when any format is set	275
4.6.2	Combination of variables (Message type) when data is not transformed	276
4.7	General faults for converting system exceptions to faults	280
4.7.1	Overview of general faults for converting system exceptions to faults	280
4.7.2	Flow of identifying the location of a system exception by using a general fault	281
4.7.3	Mapping between general fault messages and message log	282
4.8	Changing value of user message wherein UOC is used	283
4.8.1	Overview of component-common UOC	283
4.8.2	Usage example of component-common UOC	286
5	Performance Upgrade Function	289
5.1	Multiplicity of the service component invocation process	290
5.1.1	Multiplicity of the HCSC server	290
5.1.2	Multiplicity related to database access	293
5.1.3	Multiplicity related to Web Services (SOAP communication)	295
5.1.4	Multiplicity related to XML analysis process	298
5.2	Cache functionality for XML messages	300
5.3	Pre-cache function of a business process	301
5.4	XML parser pool function of the data transformation process	302
6	Functionality for Failure Checking	305
6.1	Functionality for acquiring failure information, failure recovery, and failure analysis	306
6.1.1	Functionality for acquiring failure information	306
6.1.2	Troubleshooting functionality	306
6.1.3	Failure analysis functionality	306
6.2	Data validation functionality	307
6.2.1	Overview of the data validation functionality	307

6.2.2	Setting up the data validation functionality	308
6.2.3	Scoping of data validation function	308
6.3	User message trace functionality	310
6.3.1	Overview of the user message trace functionality	310
6.3.2	Setting up the user message trace functionality	310
6.3.3	Precautions on using the user message trace functionality	310

7

Functionality Used by Linking with Other Products	311
7.1 Operation management through linkage with JP1	312
7.1.1 Detecting failures by linking with JP1/IM	312
7.1.2 Monitoring processes through JP1	313
7.2 Databases that can be connected	316
7.3 Management of execution logs by linking with HiRDB	317
7.3.1 Partitioning storage of execution logs	317

8

Functionality to Transfer Files by Integrating with FTP (FTP Integration)	321
8.1 Overview of FTP integration	322
8.2 System configuration for using FTP integration	324
8.3 Examples of file transfer patterns using FTP integration	325
8.3.1 Transfer of files from an FTP client to an FTP server	325
8.3.2 Transfer of files from an FTP server to an FTP client	328
8.3.3 Transfer of files from an FTP client to another FTP client	329
8.3.4 Obtaining a file list from an FTP server	331
8.4 FTP reception functionality	334
8.4.1 Operations and FTP commands supported in an FTP reception	334
8.4.2 Communication model and business processes that can be used with FTP receptions	335
8.4.3 Transactions when FTP reception is used	336
8.4.4 Generating the request ID	337
8.4.5 Creating the work folder and intermediate files	338
8.4.6 Business process access control	339
8.4.7 Managing FTP commands executed before and after transfer and list	340
8.4.8 Executing the LIST and NLST commands by specifying list command options	341
8.4.9 Monitoring a timeout	341
8.5 FTP inbound adapter functionality	342
8.5.1 Checking the file size after sending and receiving files	342
8.5.2 Monitoring a timeout	342
8.6 FTP adapter functionality	350
8.6.1 Flow of FTP adapter processing	351
8.6.2 Executing the FTP commands	351
8.6.3 Folder specification used for sending and receiving files	353
8.6.4 Checking the file size after sending and receiving files	353

8.6.5 Timer management	354
8.7 File operations adapter functionality	357
8.7.1 Operations supported by the file operations adapter	357
8.7.2 Request and response messages for each operation	359
8.7.3 File conversion operation	359
8.7.4 File replication operation	372
8.7.5 File or folder deletion operation	373
8.7.6 File compression operation	374
8.7.7 File expansion operation	376
8.7.8 Managing files and folders used for each operation	377
8.8 File access using the work folder and common folder	379
8.8.1 File access using the work folder	379
8.8.2 File access using the common folder	379
8.9 Scope of FTP integration	381
8.9.1 Commands supported in FTP integration	381
8.9.2 Specification method for transfer commands	383
8.9.3 Points to be considered on using FTP integration	384
8.10 Flow of development in the FTP integration system	386
8.11 Design for troubleshooting	388
8.11.1 Overview of error design	388
8.11.2 Timer design	388
8.11.3 Transaction design	390
8.11.4 Troubleshooting design	396

Appendixes 399

A. TP1/Client/J Function	400
A.1 Remote procedure call	400
A.2 Transaction control	400
A.3 TCP/IP communication function	400
A.4 Function for receiving one-way communication from server	400
A.5 TP1/Web connection function	400
A.6 Function for changing dynamic definition	400
A.7 TCP/IP connection establishment monitoring function	400
A.8 Function for connecting to DCCM3	400
A.9 XA resource services function	400
A.10 Troubleshooting function	400
A.11 Data compression function	400
A.12 Source host specification function	401
A.13 Function for fixing reception port	401
A.14 Host switching function	401

B. TP1/Client/J definition	402
B.1 Overview of TP1/Client/J definition	402
B.2 TP1/Client/J environment definition details	402
C. Troubleshooting of TP1/Client/J	403
C.1 Output contents of trace file	403
C.2 UAP trace	403
C.3 Data trace	403
C.4 Error trace	403
C.5 Method trace	403
C.6 Debug trace	403
C.7 Performance analysis trace	403
D. Glossary	404

Index	405
-------	-----

1

Functional Overview of Cosminexus Service Platform

This chapter gives an overview of the functionality for implementing systems with SOA applied by using Cosminexus Service Platform.

1.1 Functional overview of the development environment

The development environment of the service platform is used to define service adapters and business processes to be operated in the execution environment. The development environment provides the following main functionality:

- Creating message formats
- Creating user-defined reception
- Creating service adapters
- Creating business processes
- Creating data transformation definitions
- Debugging business processes

You can define the above functionality by using windows of the development environment.

The functionality for validating the consistency of definitions created by using windows of the development environment is also provided. *HCSC component validation* functionality can be used at any stage, such as during or after creation of HCSC components.

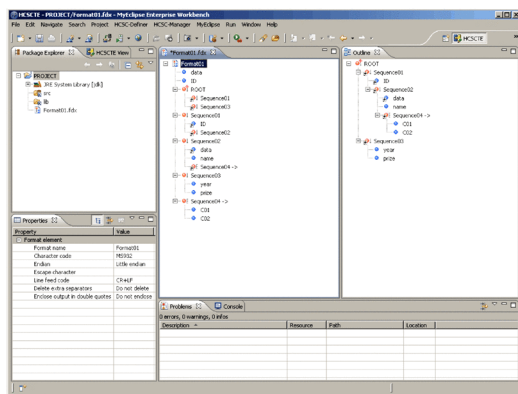
The following subsections give an overview of the functionality.

1.1.1 Functionality for creating message formats

Message formats are formats of messages exchanged between service requesters, service adapters, and service components. To use XML format data, create an XML message format. To use binary format data (data other than XML), create a binary message format. You can create the binary message format by using window operations.

The following figure shows an example of the window used for creating a binary message format.

Figure 1–1: Example of the window for creating a binary message format



After adding a new binary message format in the wizard, you can define detailed contents of elements configuring the message format in the window for creating a binary message format.

Note that if you create a new binary message format by diverting an already created binary message format, you can also duplicate and use the existing one.

1.1.2 Functionality for creating user-defined reception

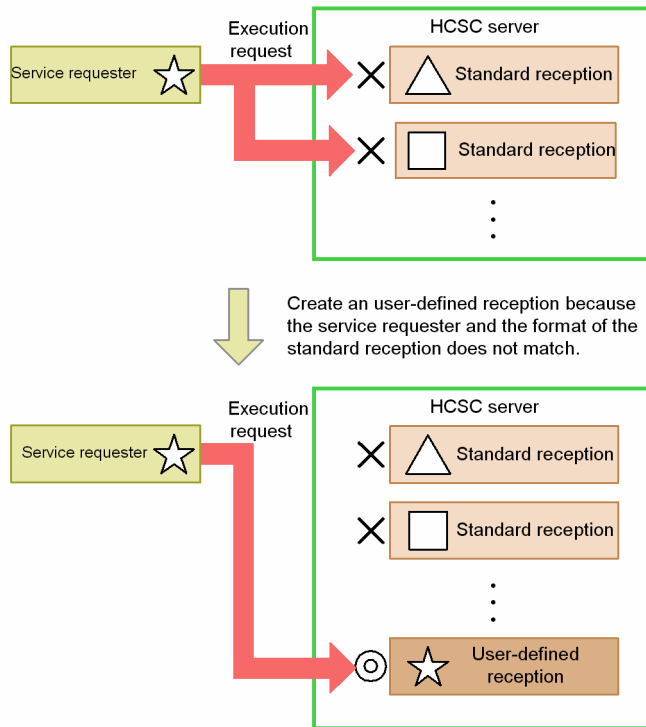
Request receptions used to receive execution requests from service requesters include standard reception and user-defined reception.

Standard reception is a fixed interface that receives execution requests for service components from service requesters. User-defined reception is the interface that allows users to define any format according to execution requests for service components from service requesters.

To use standard reception, you must create a service requester according to the format of each standard reception. Therefore, if you re-use an existing service requester that does not match the interface format of standard reception, you need to re-create a service requester or make major modifications.

In such cases, you can create a *user-defined reception* that enables reception of execution requests from existing service requesters. The following figure shows how to create a user-defined reception.

Figure 1–2: Creating a user-defined reception



The service platform provides a wizard for effectively creating user-defined receptions.

1.1.3 Functionality for creating service adapters

Service adapter is one of the HCSC components used for invoking service components.

The service platform provides various service adapters according to the usage, such as other systems, databases, and files. The service platform also provides a customer adapter developed by using a custom adapter development framework, to support systems having protocols not directly supported by the system adapters that are provided.

The following table lists the types of service adapters that can be used in the service platform:

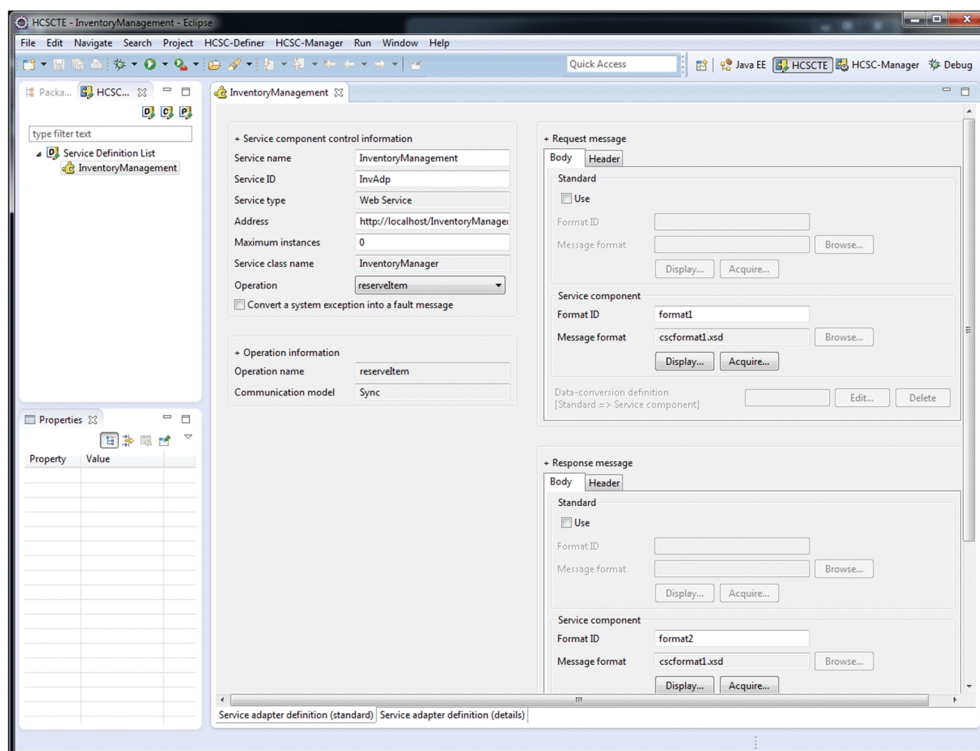
Table 1–1: Types of service adapters

Type	Usage
SOAP adapter	To invoke service components by using Web service (SOAP communication)
SessionBean adapter	To invoke service components by using SessionBean
MDB (WS-R) adapter	To invoke service components created with MDB (Message Driven Bean) by using WS-R (WS-Reliability)
MDB (DB queue) adapter	To invoke asynchronous service components by using DB queue
DB adapter	To use database operations as service components
TP1 adapter	To invoke service components available in the existing OpenTP1 system
File adapter	To input and output files directly to the local disk on the HCSC server

Type	Usage
Object Access adapter	To invoke service components of existing TPBroker system (Object Wrapper system)
Message Queue adapter	To send and receive messages to existing message queue (IBM WebSphere MQ system)
FTP adapter	To send and receive files between the service platform and FTP server
File operations adapter	To perform layout conversion, character code conversion, duplication, deletion, compression, and expansion of files
Mail adapter	To invoke the mail server that supports SMTP protocol as a service component on the service platform
HTTP adapter	To invoke resources published on a Web server and Web services (RESTful Web service) published in the REST style.
Custom adapter	To invoke service components of a system with protocols not directly supported by the system adapters provided in the service platform

The following figure shows an example of the screen used for creating a service adapter:

Figure 1–3: Example of the screen used for creating a service adapter



Define the details on these screens after you newly add a service adapter by using the wizard. Furthermore, if you create a service adapter by diverting an already created service adapter, you can duplicate and use the existing service adapter.

For details on service adapter definitions, see "3. Defining Adapters" in "Service Platform Reception and Adapter Definition Guide".

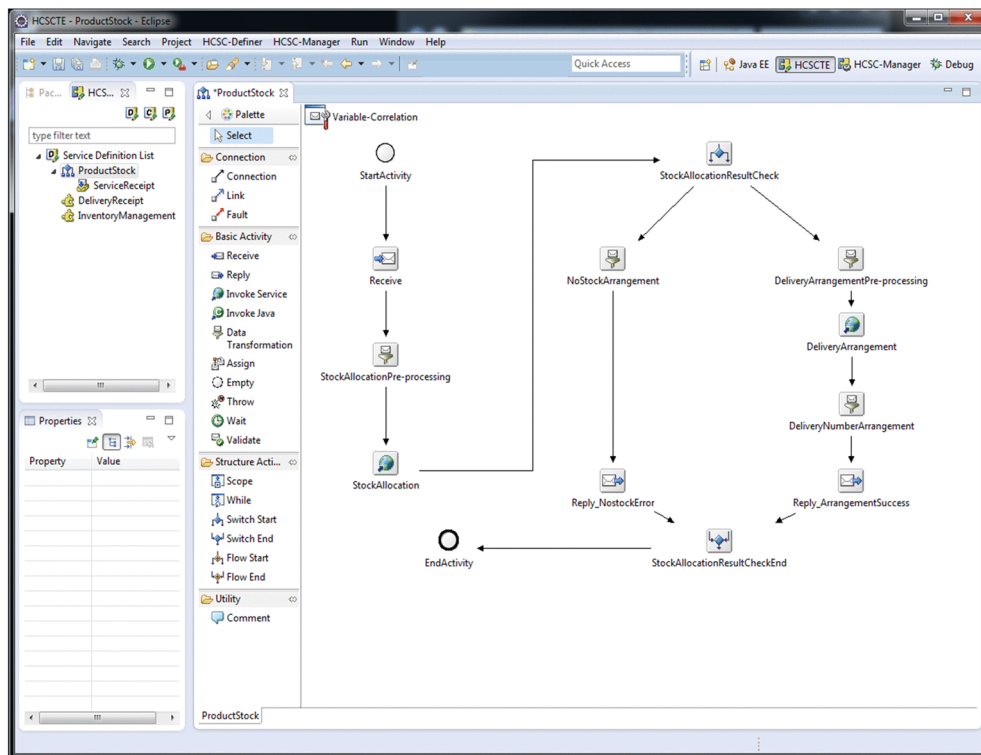
1.1.4 Functionality for creating business processes

A *business process* defines information such as the processing order and conditions of services as a sequence of tasks.

A business process is defined in a window by using BPEL.

The following figure shows an example of the window used for creating a business process.

Figure 1–4: Example of a window for creating a business process



In the window for creating a business process, you can define a business process by using activities, connecting lines defining the relationship between two activities, variables, and correlation sets.

An *activity* is a configuration component of a business process, and represents the structure of processing. The flow of a business process is defined by connecting multiple activities.

You can also define variables and correlation sets for business processes and for each activity.

A *variable* is declared to use a variable value in a condition expression in processing of a business process.

A *correlation set* is a character string used to uniquely identify a request message sent from a service requester.

Note that if you create a new business process by diverting an already created business process, you can also duplicate and use the existing one.

Moreover, when you create a business process, the service platform allows you to manage the execution status of the business process. For more advanced business processes, you can create a business process in which multiple business processes are arranged in a hierarchy.

(1) Managing the execution status of a business process

When you add a new business process, you can use the functionality for specifying whether to manage the execution status of the business process in the execution environment.

If you specify management of the execution status of business processes, you can understand the stage at which an error occurred in the business process, and to which extent the process was executed. By acquiring the status information for the terminated business process, you can re-execute from the specific process within the business process.

For details about managing the execution status of business processes, see *1.3.4 Functionality for managing execution logs*.

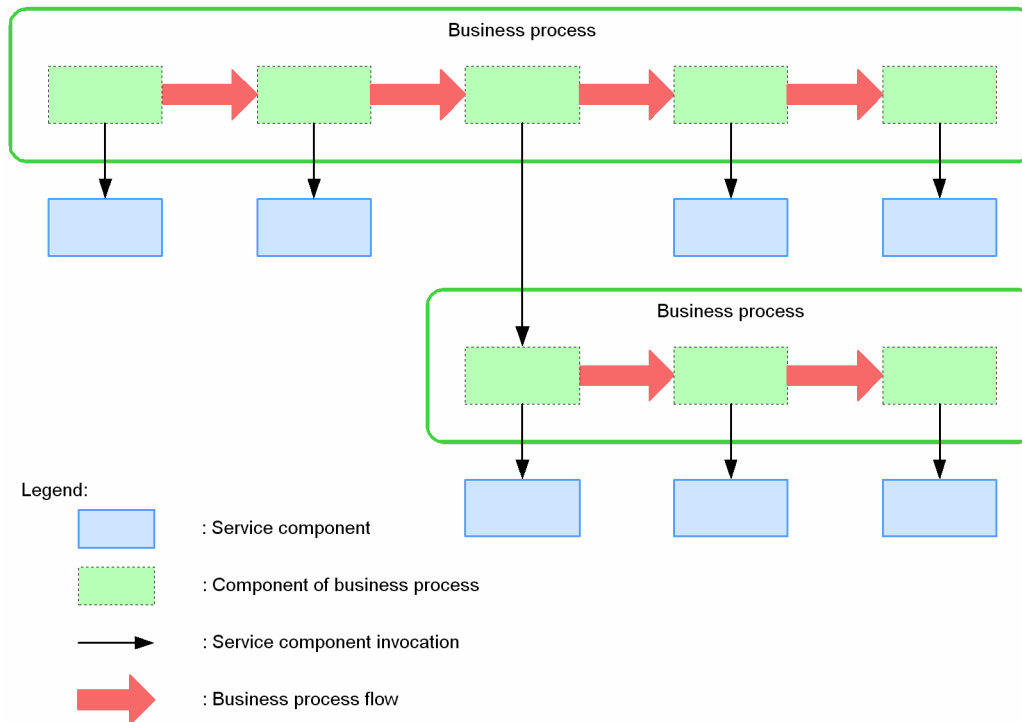
Because you can select whether to manage the execution status, you can also specify not to manage the execution status of business processes in which performance such as the processing speed is given importance.

(2) Arranging business processes in a hierarchy

You can arrange business processes in a hierarchy by defining business processes such that a business process is invoked from another business process.

The following figure shows an overview of arranging business processes in a hierarchy:

Figure 1–5: Arranging business processes in a hierarchy



As shown in *Figure 1-5 Arranging business processes in a hierarchy*, you can define the process for invoking a business process within the series of processes for invoking each service from the business process.

Because a business process can be modular components as the process invoked in the business process, user can define more complicated business processes.

(3) Importing the BPEL created in BPMN tool

When designing a business process, you can import a BPEL definition created by using a design tool (high-level design tool using BPMN) to the development environment (Service Architect) of the service platform. The imported BPEL definition is converted as a business process definition, and is displayed as an activity in the window. You can edit details of messages or processes.

1.1.5 Functionality for creating data transformation definitions

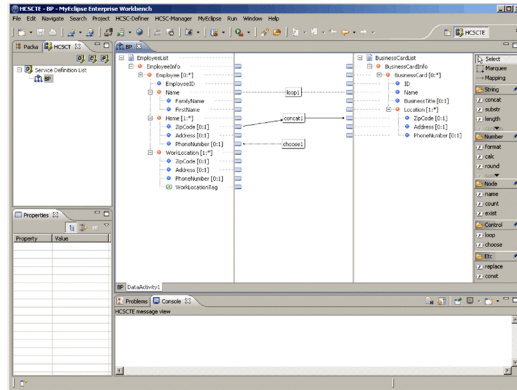
A *data transformation definition* defines how to transform message data sent from a service requester in the execution environment and how to transform the format of the data transferred between service components.

After creating the transformation source and destination schema files, define transformation processing based on the created schema files. For example, you can determine the transformation-destination elements and attributes to which the elements and attributes of the transformation source will be converted (mapped).

If you create in advance an Excel file defining the combination of transformation sources and destinations, you can set mapping information as a batch in the window. When defining many connections or applying the same mapping to multiple business processes, using this file improves development efficiency.

The following figure shows an example of the window used for creating data transformation definitions.

Figure 1–6: Example of a window for creating data transformation definitions



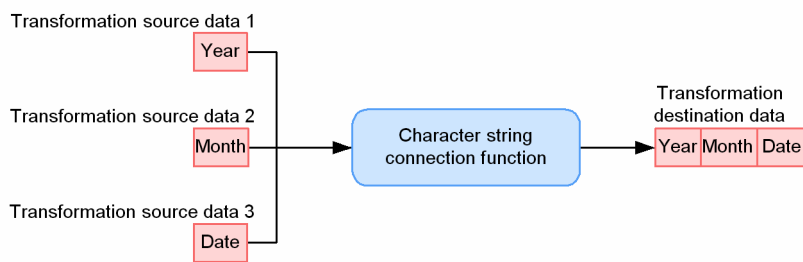
In the window for defining data transformation, you can connect (map) the transformation source and destination, and visually define the assigned value from the transformation source to the transformation destination.

If you want to process transformation source data and then assign it to the transformation destination, you can specify functions. The functions include the functionality for connecting character strings and computing numeric values, etc.

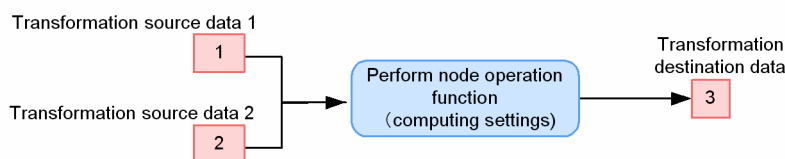
The following figure shows an example of using functions to process data.

Figure 1–7: Example of using functions to process data

- When connecting multiple character string data



- When computing multiple numerical data



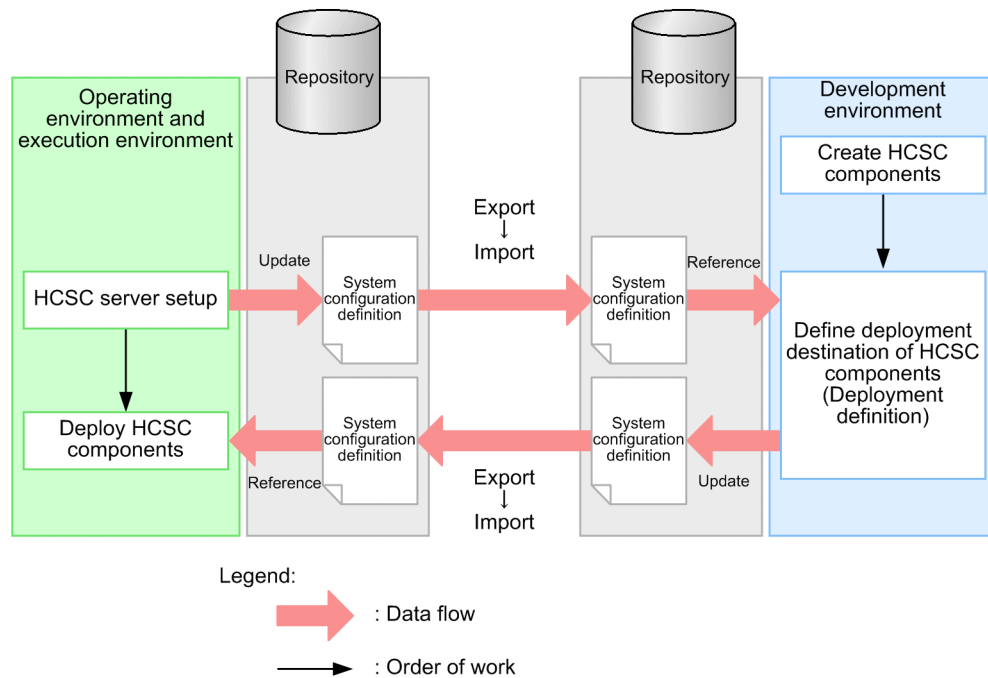
You can also specify the settings for iteration processing when multiple transformation source data is available and for assigning data only when certain conditions are met.

1.1.6 Functionality for defining deployment

You can deploy HCSC components while mutually exchanging information between the development environment and the operating environment via a repository.

The following figure shows an overview of deploying HCSC components.

Figure 1–8: Deploying HCSC components



In the operating environment, when you set up an HCSC server and a cluster, the configuration information of the execution environment (system configuration definition) available in the repository is updated. The updated system configuration definition is exported by using the repository management functionality.

In the development environment, import the system configuration definition including the setup information of the HCSC server and clusters that are exported in the operating environment into the repository. Then, update the system configuration definition by adding a definition of the HCSC server to which HCSC components are deployed (deployment definition), and then export the updated system configuration definition.

Tip

To specify a deployment definition in the development environment, deploy service components in clusters or by using a single HCSC server.

Then, import the system configuration definition updated in the development environment into the repository of the operating environment. Deploy HCSC components in the execution environment according to the system configuration definition.

For details about exporting repositories, see *4.2 Exporting repositories* in the *Service Platform System Setup and Operation Guide*. For details about importing repositories, see *4.3 Importing repositories* in the *Service Platform System Setup and Operation Guide*.

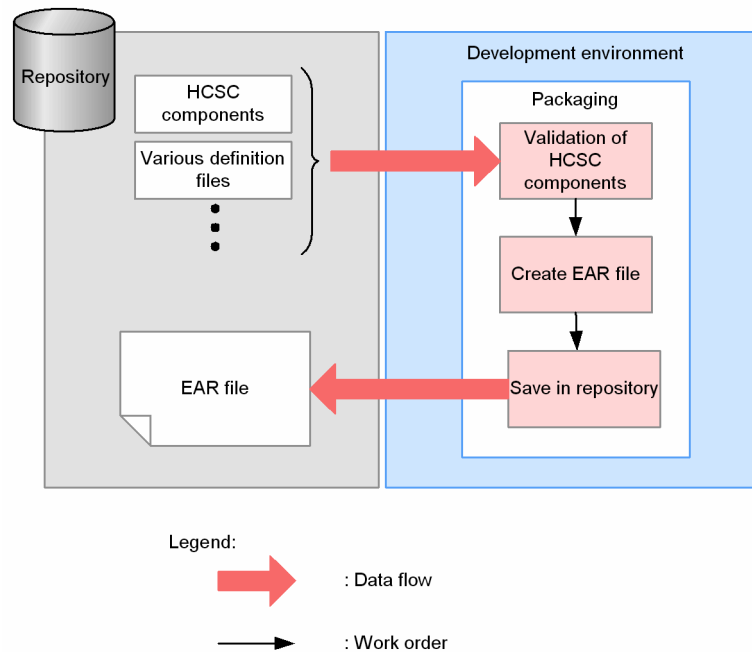
1.1.7 Packaging

Packaging implies assembling HCSC components created in the development environment and the required information in the EAR file format, and then saving in the repository.

In the development environment, you can execute packaging using a window.

The following figure shows an overview of packaging:

Figure 1–9: Packaging



When executing the packaging functionality from the development environment, the information required for packaging will be acquired from the repository and the HCSC components will be validated.

As a result of validation, if all the definitions required for HCSC components are defined, the EAR file will be created.

If the HCSC components have any flaws, packaging will be suspended.

The created EAR file will be saved in the repository. In the execution environment, you use the operating environment to deploy HCSC components of the EAR file format that are saved in the repository.

1.1.8 Functionality for deploying to starting or stopping to deleting HCSC components, in a batch

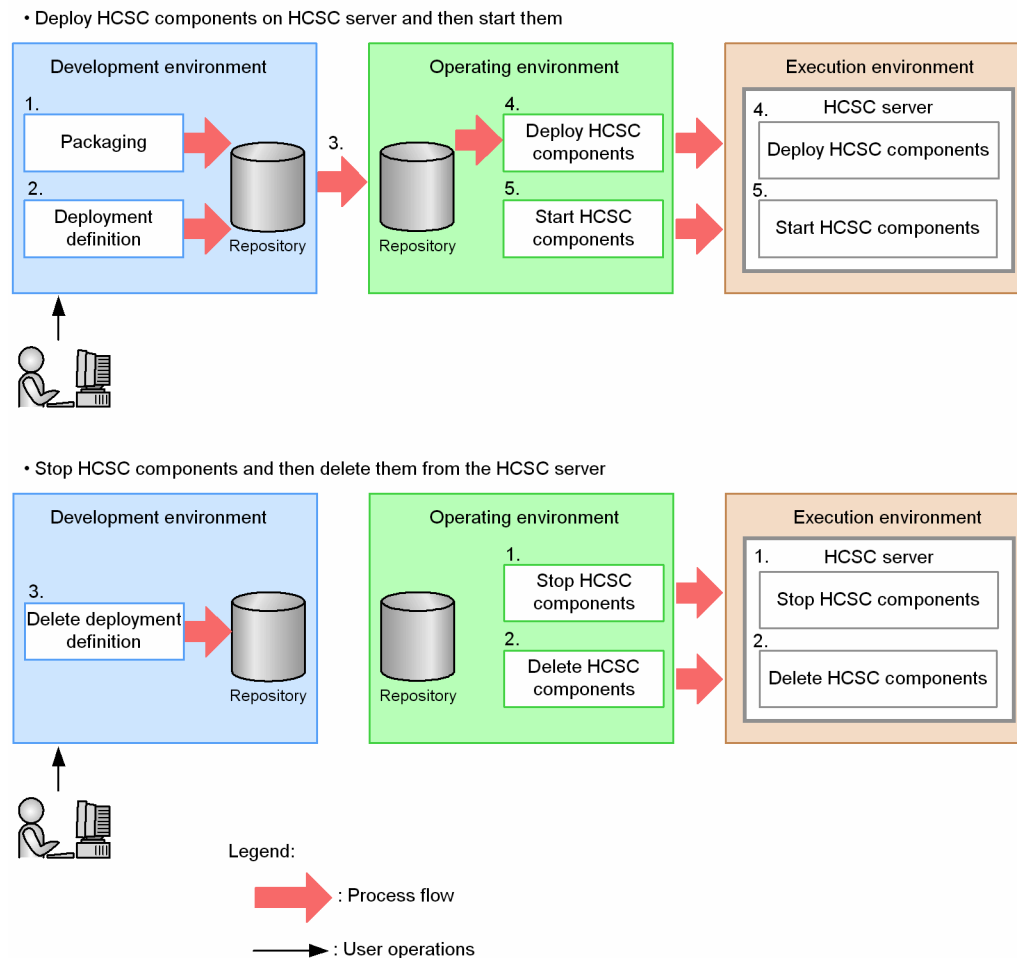
In the development environment, you can execute the following series of processes, in a batch:

- Deploy HCSC components on an HCSC server, and then start
- Stop HCSC components, and then delete from the HCSC server

In the development environment, the businesses executed individually in the development environment or operating environment will be executed in a batch. Therefore, the operation load of the user can be reduced. Note that you perform the batch execution during the system development or from the time of performing unit test to integration test.

The figure below shows the batch execution procedures. The numbers in the figure indicate the order of the processes executed automatically after the user performs operations in the development environment:

Figure 1–10: Flow of processes executed in a batch



1.1.9 Functionality for creating service requesters

A *service requester* is the application that receives execution requests for service components from the business person in-charge, and sends execution requests for service components to each HCSC component.

You create service requesters in the development environment; incorporate in the execution environment and then use.

The development environment window of Cosminexus Service Platform is equipped with the functionality for supporting the creation of service requesters, such as acquisition of the WSDL and the stubs required for creating service requesters.

1.1.10 Functionality for debugging business processes

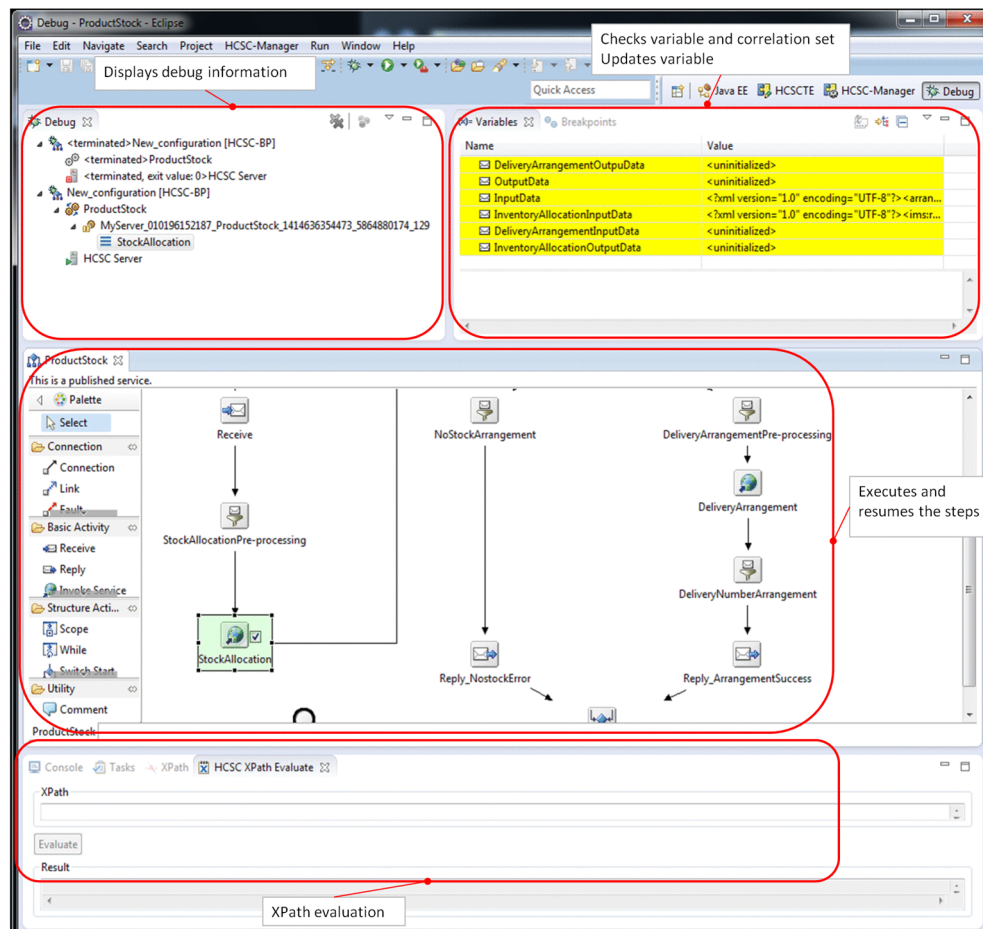
The *functionality for debugging business processes* is capable of conducting testing and debugging focused on business processes.

You can use this functionality to debug business processes in the development environment without deploying HCSC components in the execution environment. Development is more efficient because business processes can be defined and debugged in the same environment.

By debugging a business process, you can check how processing of the business process is proceeding between activities, and also check and update the contents of variables used in the activities.

The following figure shows an example of a window used for debugging business processes and the usable functionality.

Figure 1–11: Example of a window used for debugging business processes and the usable functionality



You can use the following functionality for debugging business processes:

- **Step execution and resume**
You can specify a breakpoint in an activity to interrupt the processing of a business process at any time. In the interrupted business process, you can continue sequential processing while interrupting each activity.
- **Displaying the debug information**
You can display the name of the business process that you are debugging, and also names of activities.
- **Confirming variables and correlation sets**
You can confirm the values of variables and correlation sets used in business processes.
- **Updating variables**
You can change the values of variables used in business processes, and then apply them to business processes.
- **XPath evaluation**
You can evaluate the validity of the conditional expression specified in the switch activity and the assign activity.

The functionality for debugging business processes also allows you to emulate services.

1.2 Functional overview of the execution environment

The execution environment of the service platform is used to implement SOA. The execution environment provides the following main functionality:

- Distributing services
- Invoking various types of services
- Executing business processes

The following subsections give an overview of the functionality.

1.2.1 Functionality for distributing services

For the service platform, the system of a requesting user is called a *service requester*, and a provider of functions is called a *service component*. An SOA environment used for managing the enterprise service bus (ESB) that implements service linkage and BPEL-compliant business processes is called an *HCSC server*. When a user issues a request to execute a service component, SOA invokes and then executes an appropriate service component from business processes and service adapters deployed on the HCSC server. In addition, business and functions are managed as services, and services are distributed in response to requests.

For details about the functionality for distributing services, see *Chapter 2. Functionality for Connecting to Various Types of Systems*.

1.2.2 Functionality for invoking various services

The service platform provides service adapters to invoke different types of services.

By using the following service adapters, you can flexibly build a system, and effectively utilize data of the existing system as a service component:

- SOAP adapter for using Web services (SOAP communication)
- SessionBean adapter for using SessionBean
- MDB (WS-R) adapter to invoke service components created with MDB by using WS-R
- MDB (DBqueue) adapter to invoke asynchronous service components by using DB queue
- DB adapter for using database as a service component
- TP1 adapter for using the system on OpenTP1 and XDM/DCCM3
- File adapter for using the business processing system that uses files
- Object Access adapter for using mission critical systems such as Mainframe
- Message Queue adapter to send and receive messages for the existing message queue (IBM WebSphere MQ system)
- FTP adapter to transfer files by connecting to an FTP server
- File operations adapter for converting, duplicating, and deleting file formats
- Mail adapter for using the mail server that supports SMTP protocol on the service platform
- HTTP adapter for using Web services (RESTful Web Services) published in resources and REST style on the Web server
- Custom adapters to invoke services by connecting to systems with protocols not directly supported by the above mentioned service adapters provided in the service platform

For details on service adapters, see "*2. Functionality for Connecting to Various Types of Systems*".

1.2.3 Functionality for executing business processes

A business process is a series of tasks in which information such as the processing order of multiple service components and processing conditions are defined.

The service platform can sequentially invoke and execute service components in the order defined in a business process.

The functionality for executing business processes executes a business process for controlling the flow of service invocation.

For details about business processes, see *Chapter 3. Business Process Functionality*.

1.3 Functional overview of the operating environment

The operating environment of the Cosminexus Service Platform is used for setting up the execution environment, deploying the required information, performing the operation management, and managing execution logs. The operating environment mainly provides the following functionality:

- Setting up the execution environment
- Deploying HCSC components
- Managing operations in the execution environment
- Managing execution logs

Use commands or windows of the operating environment, and executing the functionality.

The following subsections give an overview of the functionality.

1.3.1 Functionality for setting up the execution environment

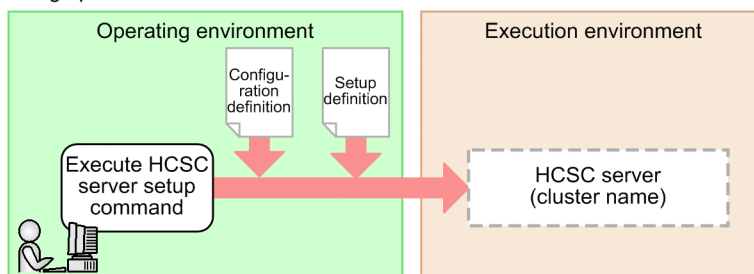
In the operating environment, you can use commands to set up an HCSC server in the execution environment. When you set up the HCSC server, you can also set up a cluster.

(1) Functionality for setting up an HCSC server and clusters

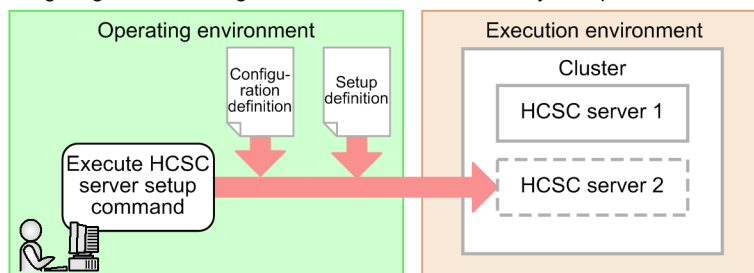
The following figure shows an overview of setting up an HCSC server.

Figure 1–12: Overview of setting up an HCSC server

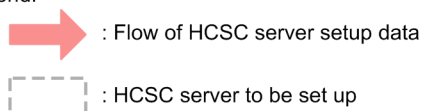
- Setting up an individual HCSC server



- Configuring a cluster using an HCSC server that is already set up



Legend:



If you execute a command in the operating environment, the HCSC server is set up by acquiring necessary information required from the HCSC server configuration definition file and setup definition file already created.

Setting up an individual HCSC server

Although the cluster name is specified when the command is executed, a cluster is not configured because only one HCSC server is set up at this stage.

When you set up another HCSC server, you can configure a cluster by specifying the cluster name.

Configuring a cluster by using an HCSC server that is already set up

If you execute the command by specifying a cluster name that was specified for setting up another HCSC server, you can set up the HCSC server that configures a cluster with the already set up HCSC server.

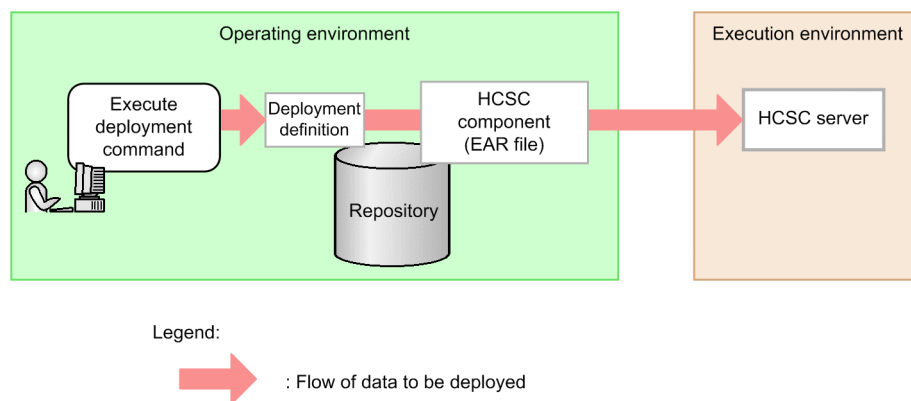
1.3.2 Functionality for deploying HCSC components

Deploying means arranging the required information in the execution environment in such a way so that HCSC components can be used in the execution environment.

You can deploy HCSC components in an execution environment by executing commands in an operating environment.

The following figure shows an overview of deployment:

Figure 1–13: Overview of deployment



Deploy HCSC components (EAR files) in the execution environment (HCSC server), according to the information of deployment definition that is created in the development environment and saved in the repository.

1.3.3 Functionality for managing operations in the execution environment

In the operating environment, you can manage business operations in the execution environment.

When starting a business in the execution environment, stopping the business to change the configuration in the execution environment, or stopping the business because of the occurrence of an error, you can check the operating status of the business in the execution environment using the operating environment, as and when required.

The following table describes the operations in the execution environment that you can execute from the operating environment:

Table 1–2: Operations in the execution environment that can be executed from the operating environment

Operation	Execution interface	
	Window	Command
Setting up HCSC servers	--	Y
Canceling HCSC server setup	--	Y
Starting HCSC servers	Y	Y
Starting HCSC servers (batch startup in the cluster)	--	Y
Terminating HCSC servers	Y	Y

Operation	Execution interface	
	Window	Command
Terminating HCSC servers (batch termination in the cluster)	--	Y
Referencing HCSC server information [#]	Y	Y
Referencing HCSC server setup information	--	Y
Defining HCSC servers	--	Y
Checking the HCSC server definition [#]	Y	Y
Changing the HCSC server definition	--	Y
Upgrading the version of HCSC servers	--	Y
Starting the standard reception	Y	Y
Stopping standard receptions	Y	Y
Deploying a service adapter	--	Y
Deleting a deployed service adapter	--	Y
Starting a service adapter	Y	Y
Terminating a service adapter	Y	Y
Referencing service adapter information [#]	Y	Y
Deploying a business process	--	Y
Deleting a deployed business process	--	Y
Starting a business process	Y	Y
Terminating a business process	Y	Y
Referencing business process information [#]	Y	Y
Deploying a user-defined reception	--	Y
Deleting a deployed user-defined reception	--	Y
Starting a user-defined reception	Y	Y
Terminating a user-defined reception	Y	Y
Referencing user-defined reception information [#]	Y	Y
Checking the definitions of user-defined reception	Y	Y
Changing the definitions of user-defined reception	--	Y
Checking the status for the HCSC servers, standard receptions, and HCSC components	Y	Y
Displaying the operating status of HCSC server resources	Y	Y
Running the applications	--	Y
Backing up the HCSC-Manager environment	--	Y
Restoring the HCSC-Manager environment	--	Y
Changing service information	--	Y
Checking service information	Y	Y

Legend:

Y: Available

--: Unavailable

#

The information that you can reference using the window and the command is different.

1.3.4 Functionality for managing execution logs

In the operating environment, you can manage execution logs of process instances of business processes.

(1) Managing execution logs of process instances

Manage execution logs of process instances by recording (making persistent) the execution status of process instances of a business process, as the log in the database.

(a) Searching execution logs of process instances

You can specify a condition to search the execution log of process instances. For a search key, you can use a correlation set, start date and time, or the status.

(b) Referencing detailed information of execution logs of process instances

You can reference detailed information of the execution log of process instances. You can reference the following information:

- Correlation set definition name and correlation set value
- Variable definition names and variable values for variables of each process instance
- Status of each activity defined within the business process

(c) Deleting execution logs of process instances

You can delete execution logs of unnecessary process instances from the database used for checking business processes. You can delete execution logs (one-by-one or as a batch) by specifying the status.

(d) Re-executing a process instance

You can check process instances of business processes that could not be processed (due to a fault or for any other reason), and then re-execute the process instances in which execution was interrupted. You can re-execute process instances (one-by-one or as a batch).

1.4 Configuring clusters of the HCSC server

You can combine multiple HCSC servers to set up an environment with a redundant cluster configuration.

1.4.1 Redundant configuration of the HCSC server using the load-balancing functionality

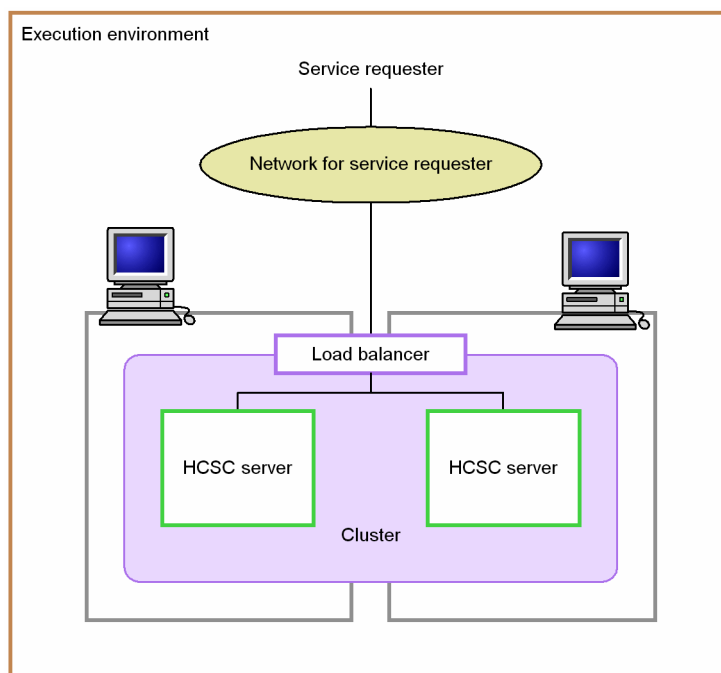
The *load-balancing functionality* is used for reducing the load of the J2EE server by concentrating the service execution requests from a service requester at one point through a parallel arrangement of multiple HCSC servers. The service platform allows you to distribute the load by using the following load balancers:

- Load balancer
Used for synchronous (Web Services) standard reception.
- CTM
Used for synchronous (Session Bean) standard reception.

In the execution environment of the service platform, you can combine multiple HCSC servers by using a load balancer to set up an environment of a more redundant cluster configuration.

The following figure shows the load balance cluster configuration of the HCSC server.

Figure 1–14: Load balance cluster configuration of the HCSC server



You can distribute and send execution requests for service components from the service requester to multiple HCSC servers by using a load balancer, only when the synchronous (Web Services or Session Bean) standard reception is used.

Depending on the load balancer you are using, you can add or delete HCSC servers configuring the load balance cluster, without stopping operation. You can also add a new service adapter or business process without stopping service adapters and business processes deployed on the HCSC server.

1.4.2 Redundant configuration of the HCSC server using cluster software

Cluster software is a program that implements switching of a system including server programs with the aim of improving the system's reliability and availability. The service platform can link with the following cluster software:

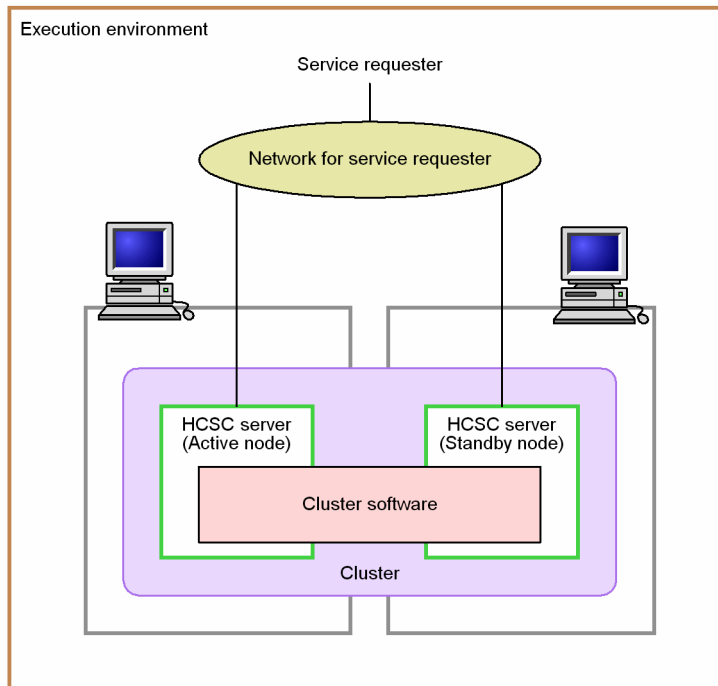
- Windows Server Failover Cluster (in Windows)
- HA monitor (in UNIX)

In the execution environment of the service platform, you can combine two HCSC servers that have the same contents by integrating with the cluster software to set up an environment of a more redundant cluster configuration (cold standby configuration).

The setup information of two HCSC servers configuring the cluster must be the same. The HCSC components to be deployed on these servers must have the same contents.

The following figure shows the cluster configuration of HCSC servers:

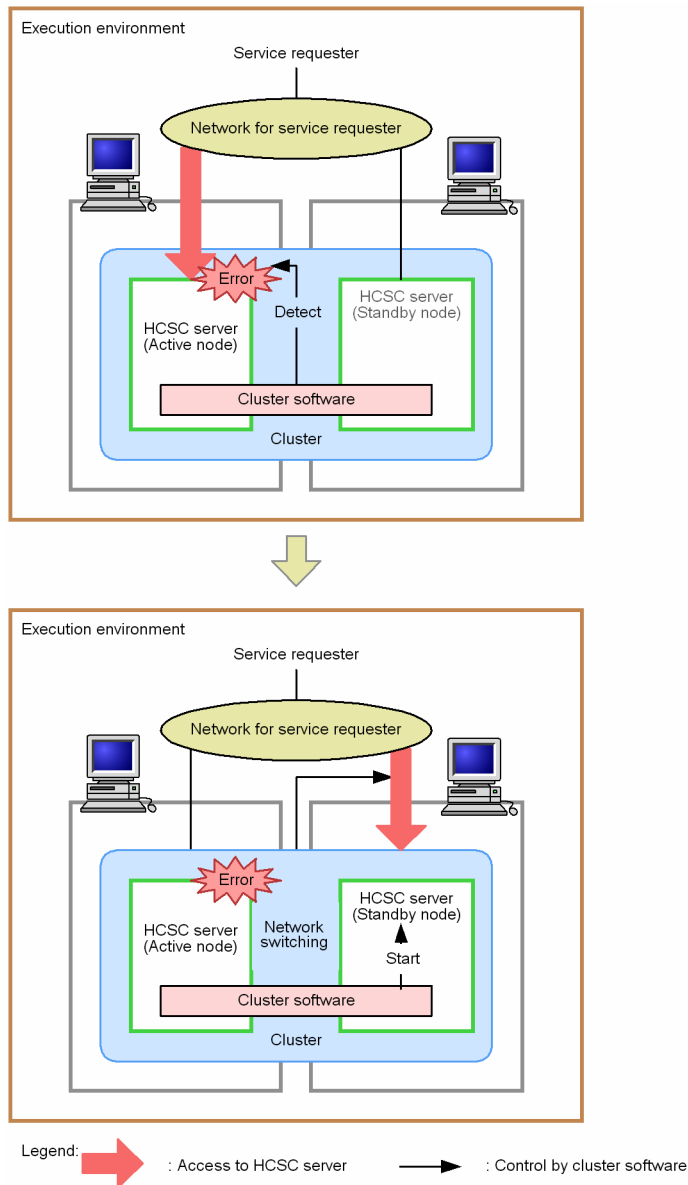
Figure 1–15: Cluster configuration of HCSC servers



In the execution environment of the service platform, you can set up a system in which the executing and standby nodes operate on a 1-to-1 basis (*1-to-1 node switching system*), by integrating with the cluster software.

If an error occurs on the executing node of a 1-to-1 node switching system, the cluster software detects the error and then automatically switches over to the standby node to continue applications. The following figure shows the switching of nodes when an error occurs.

Figure 1–16: Node switching when an error occurs



The following is the flow of operations from the occurrence of an error until the switching of nodes.

1. An error occurs on the HCSC server on the executing node.
2. The cluster software detects the error on the HCSC server on the executing node.
3. The network to one of the machines on the HCSC server on the executing node is disconnected.
4. The cluster software switches the network to one of the machines on the HCSC server on the standby node.
5. The cluster software starts the HCSC server on the standby node.

You must specify the settings in the cluster software to enable the HCSC server at the standby node to start automatically in the event of a failure.

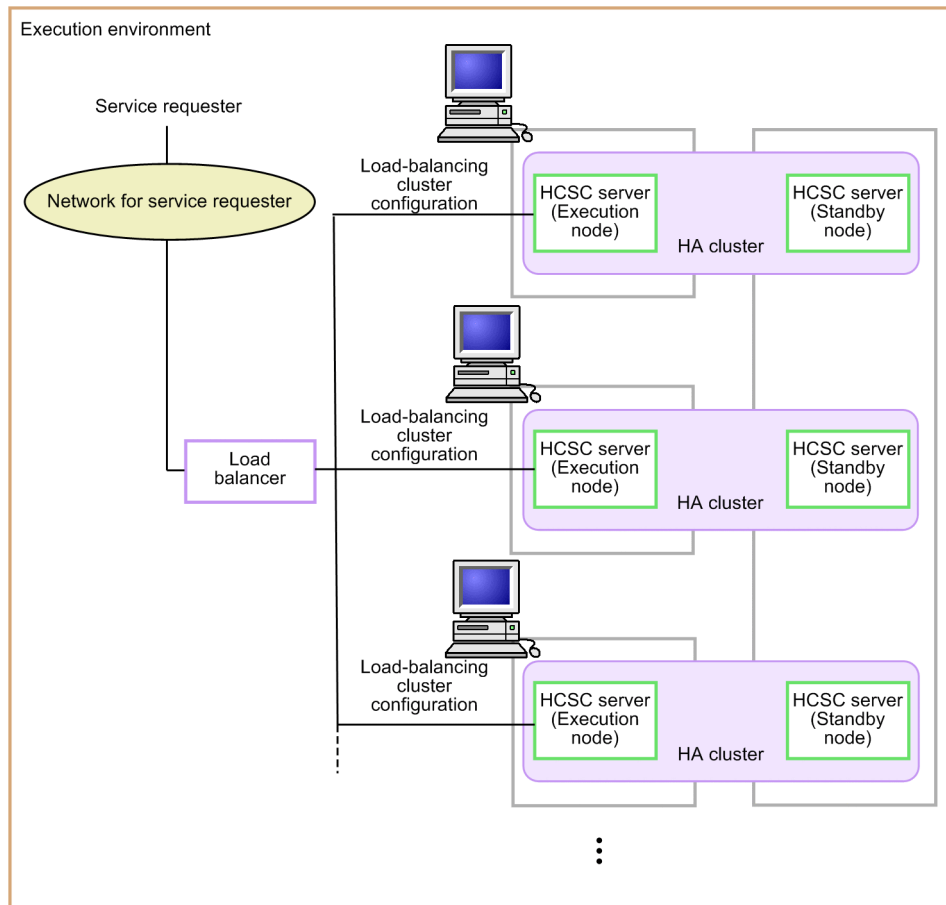
1.4.3 Redundant configuration of the HCSC server using N-to-1 cluster configuration

The N-to-1 cluster configuration is the configuration (cold standby configuration) equivalent to the high availability cluster for each HCSC server configuring the load-balancing cluster.

By operating all standby servers on a single machine, you can set up the configuration on a single machine in the standby node for N units of a machine in the executing node. Therefore, you can arrange sets of HCSC servers with the high availability cluster configuration in parallel, and perform load balancing.

The following figure shows the N-to-1 cluster configuration:

Figure 1–17: N-to-1 cluster configuration



1.5 Configuration pattern of an operating environment in the cluster configuration

The configuration pattern of an operating environment and an execution environment that can be implemented to set up a cluster configuration includes the configuration to deploy an operating environment and an execution environment on a 1-to-1 basis, in addition to the configuration in which a single operating environment is set up, irrespective of the number of execution environments.

The following are the merits and demerits of the configuration that deploys an operating environment and an execution environment on a 1-to-1 basis:

Merits

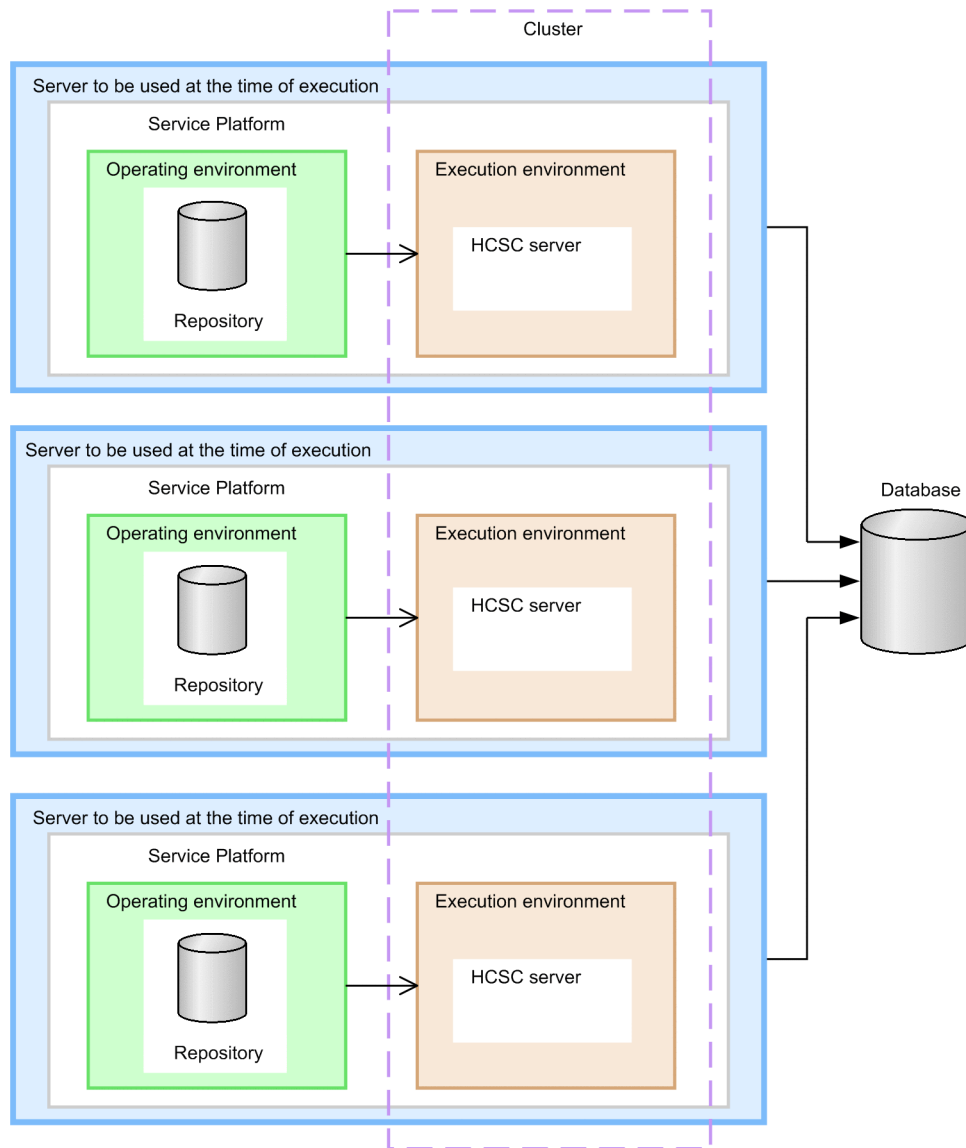
- The execution environment and operating environment are compiled on a single machine.
- Even if one operating environment is down, you can re-execute erroneous requests from other operating environments.
- You can reduce the number of licenses of the management software since machines for operating environment are no longer required.
- You no longer require a spare operating environment.

Demerits

- You cannot use the GUI of an operating environment when you set up an execution environment on UNIX.
- You cannot perform batch operations for each cluster.

The following figure shows the example 1-to-1 configuration of an operating environment and an execution environment:

Figure 1-18: Example of 1-to-1 configuration of an operating environment and an execution environment



Legend:

→ : Flow of start and stop command → : Flow of data

! Important note

Points to be considered when migrating to 1-to-1 configuration of an operating environment and execution environment from any other configuration of an operating environment and an execution environment.

For the operating environment and the execution environment configuration, see "1.2 Relationship of the operating environment and the execution environment with the entire system" in "Service Platform System Setup and Operation Guide".

- If the pre-migration environment is set up in version 08-70 or earlier, you must unset up all HCSC servers included in the cluster once. Therefore, you cannot migrate the information shared in the cluster, such as process instances and execution logs stored in the database.
- You cannot migrate from 1-to-1 configuration of an operating environment and an execution environment which is not using the database to 1-to-1 configuration of an operating environment and an execution environment using the database.

1.6 Importing repositories at the time of distributed development

When you perform distributed development in multiple development environments, you must consolidate repositories to use the developed repository in the operating environment. Implement any of the following import operations to consolidate repositories:

- Import services in the development environment.
- Import components in the operating environment.

When you consolidate repositories in a development environment, you can set more detailed conditions as compared to those set at the time of consolidating repositories in an operating environment such as selecting items to be imported and avoiding conflict of service IDs.

When you consolidate repositories in the operating environment, the development environment used to consolidate repositories is no longer required. Therefore, you can reduce the number of machines.

For details on importing services, see "3.2.3 Importing a repository" in "Service Platform Basic Development Guide". For details on importing components, see "4.6 Adding, modifying, and deleting repositories for each component" in "Service Platform System Setup and Operation Guide".

1.6.1 Flow for distributed development using the import component function

If you use the import component function, you can consolidate repositories by executing only the operation commands without preparing a development environment. This enables you to consolidate the development environment (master) and the test environment on a single machine, thereby reducing the number of machines.

For the flow of consolidating repositories in the development environment, see "*Figure 1-19 Flow for consolidating repositories in the development environment*", and for the flow of consolidating repositories in the operating environment, see "*Figure 1-20 Flow for consolidating repositories in the operating environment*".

Figure 1–19: Flow for consolidating repositories in the development environment

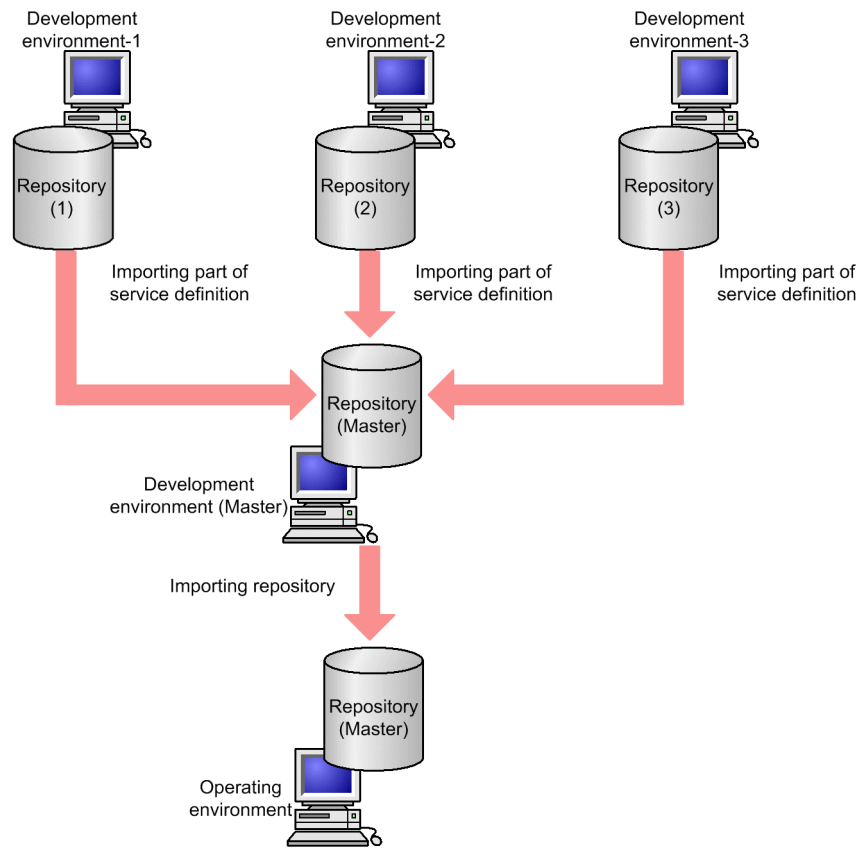
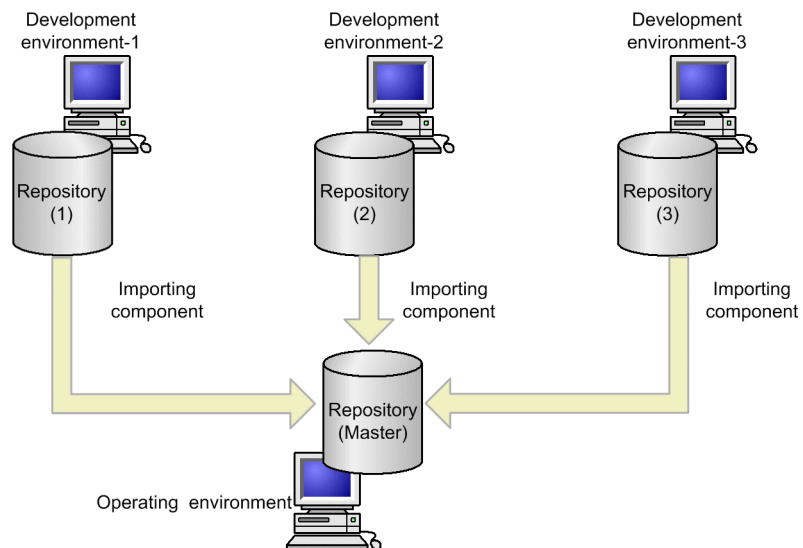
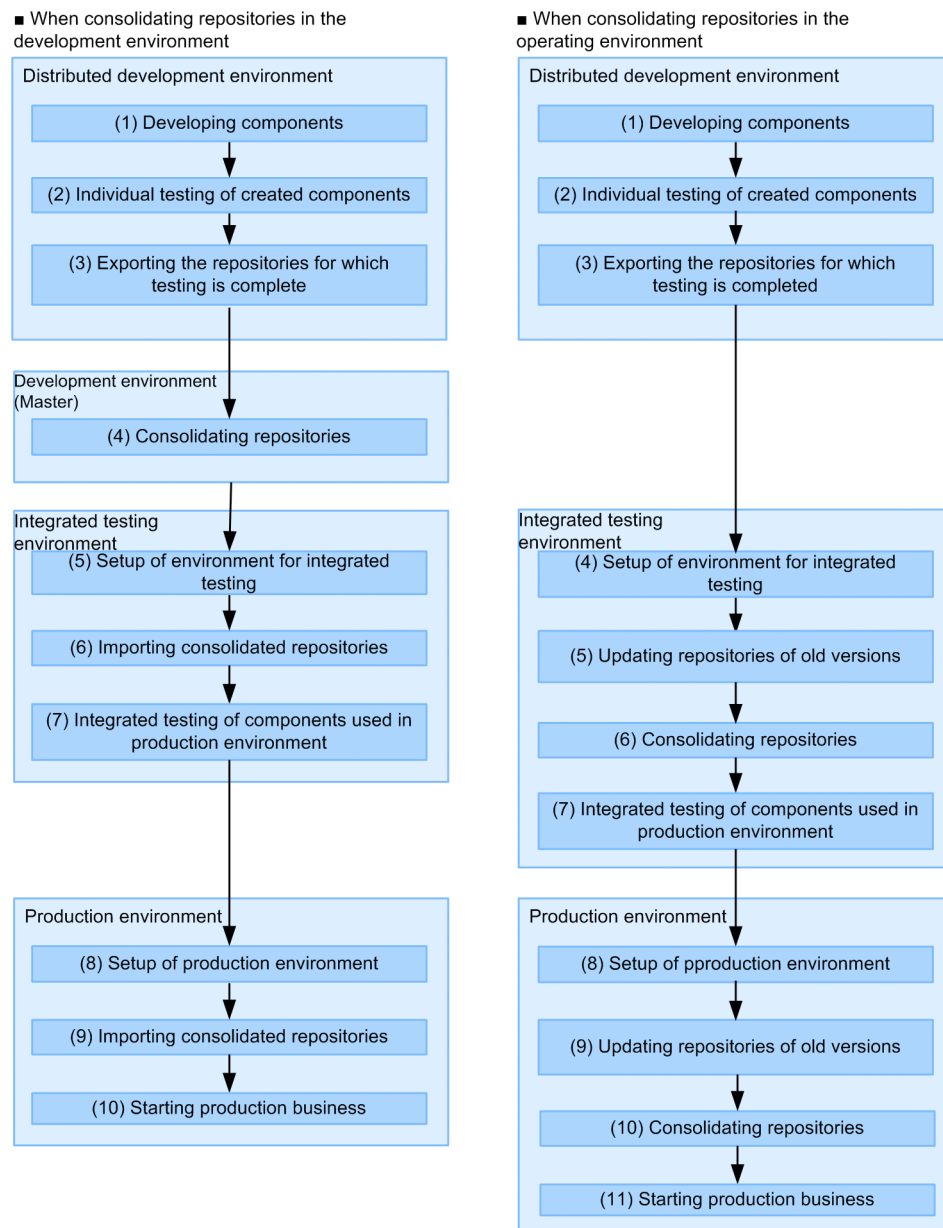


Figure 1–20: Flow for consolidating repositories in the operating environment



The following figure shows the procedure for migrating an environment when you consolidate repositories in the development environment and the operating environment:

Figure 1–21: Procedure for migrating an environment



1.6.2 Points to be considered for distributed development

The following are points to be considered for distributed development:

- Assign a unique name for the service ID, service name, context root, and UOC file in the entire distributed development environment such that there is no overlap when consolidating repositories.
- Synchronize a common package of UOC files.

2

Functionality for Connecting to Various Types of Systems

This chapter describes the basic procedure for invoking service components, and the mechanism for the execution environment of Service Platform. This chapter also describes the invocation of service components when various types of communication services and other systems are connected. Note that the definitions of the development environment are operated in the execution environment through the operating environment, but the description of the operating environment is omitted in this chapter.

2.1 Basic procedure of invoking service components

When you set up and operate a system using Cosminexus Service Platform, you can implement a better system by understanding the flow of invoking service components from the service requester via the HCSC server, and you can achieve the desired results by applying SOA.

Also, when you develop a system by applying SOA, you must design the process for invoking service components and the process when an error occurs after understanding how the operation is to be performed.

The flow of invoking service components includes the parts that are common in all the protocols and the parts that are different in each protocol. This section describes the basic procedure for invoking service components common to protocols.

2.1.1 Basic structure

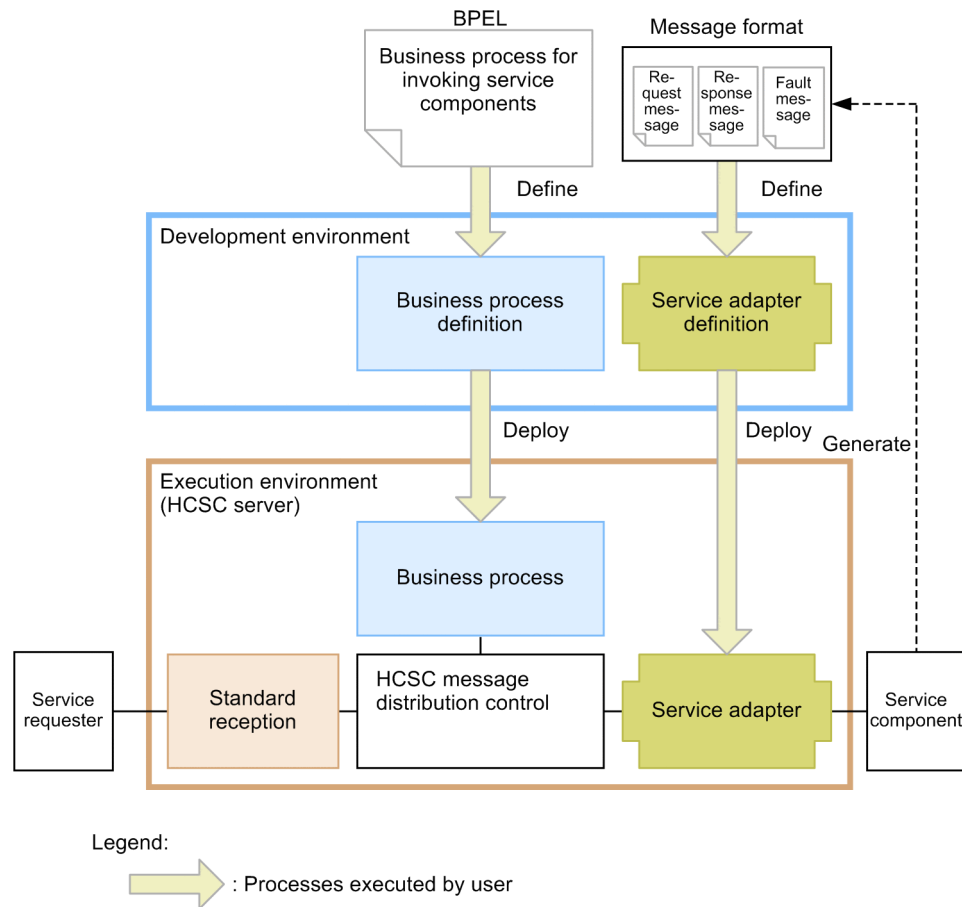
To invoke a service component, you basically require service requesters, reception, HCSC message delivery control, service adapters, and service components. A reception is used for receiving requests to invoke service components, and includes standard reception and user-defined reception.

Standard reception is the functionality (interface) used for receiving request messages from service requesters, and provided as standard by the HCSC server. *User-defined reception* is the functionality (interface) used for receiving execution requests from service requesters, and is deployed on the HCSC server after being defined by the user in the development environment. User-defined reception is able to communicate with interface which is defined by the user.

(1) For standard reception

The following figure shows the relationship between the structure and the development procedure when standard reception is used.

Figure 2-1: Relationship between the structure and the development procedure when standard reception is used



The relationship between the service requester that invokes service components, the HCSC server, and service components is as described below. This relationship is described according to the development procedure.

- Development procedure at the service component side

First, create the service component to be invoked. Next, create a message format and business process definition (BPEL) matching the type of service component to be invoked. Then define the service adapter based on the message format, and define the business process in the development environment. (Also define the message format for the business process.) The method of creating a message format matching the type of service component to be invoked differs depending on the protocol to be used (SOAP, RMI-IIOP, WS-R, or JMS). Deploy the defined service adapter and business process in the execution environment (HCSC server). For details about message formats, see the *Service Platform Basic Development Guide, Chapter 4. Creating a Message Format*. For details about the business process definitions (BPEL), see the *Service Platform Basic Development Guide, Chapter 5. Defining a Business Process*.

- Development procedure at the service requester side

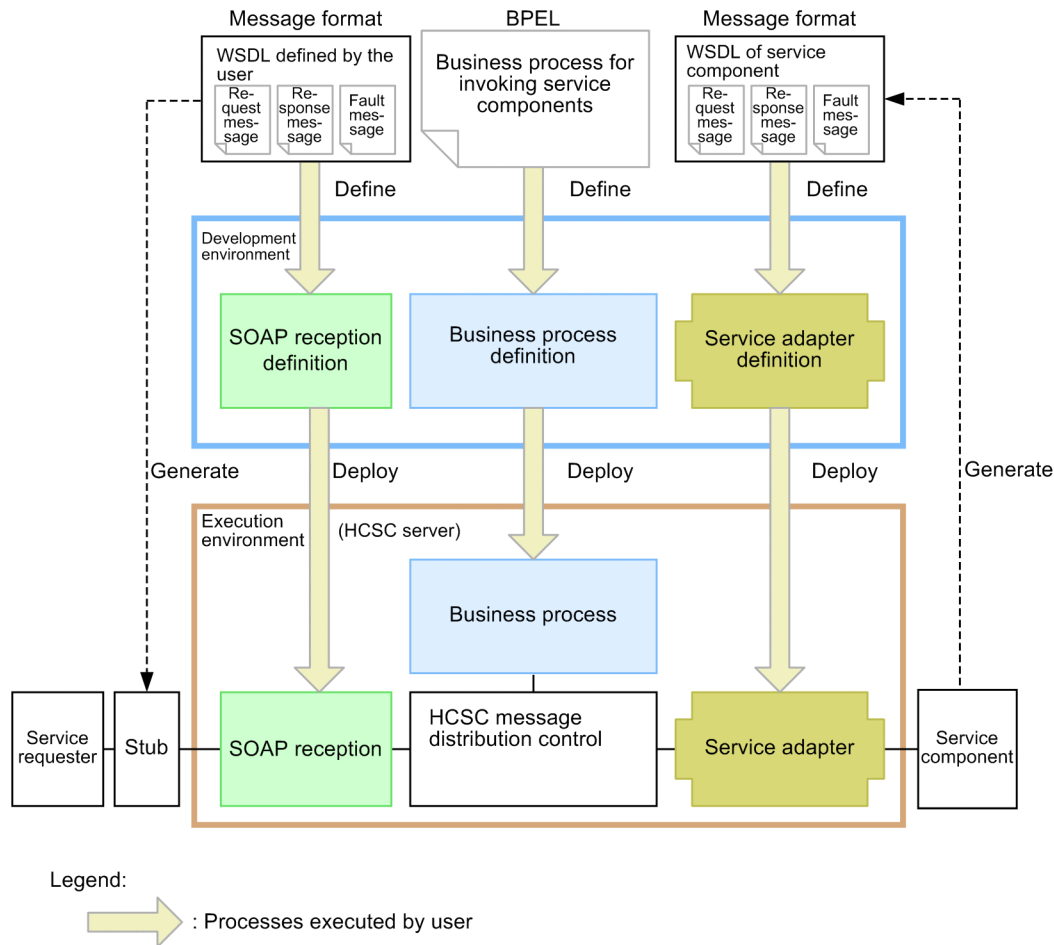
After creating the service component, create a service requester for invoking the defined service adapters and business processes. The method of creating service requesters differs depending on the type of standard reception to be used. For details about creating service requesters, see the *Service Platform Basic Development Guide, Chapter 8. Creating a Service Requester*.

(2) For user-defined reception

The following figure shows the relationship between the structure and the development procedure when user-defined reception is used.

Note that the following uses an example of using Web Services (SOAP communication).

Figure 2–2: Relationship between the structure and the development procedure when user-defined reception is used



When you use Web Services (SOAP communication), the relationship between the service requester that invokes service components, the HCSC server, and service components is as described below. This relationship is described according to the development procedure.

- **Development procedure at the service component side**
First, create the service component to be invoked. Next, create a message format and business process definition (BPEL) matching the type of service component to be invoked. Then define the service adapter based on the message format, and define the business process in the development environment (also define the message format for the business process). Deploy the defined service adapter, business process, and user-defined reception in the execution environment (HCSC server). For details about message formats, see *Chapter 4. Creating a Message Format* in the *Service Platform Basic Development Guide*. For details about the business process definitions (BPEL), see *Chapter 5. Defining a Business Process* in the *Service Platform Basic Development Guide*.
- **Development procedure at the reception side**
After creating the service component, create a WSDL matching the message format of the business process to be invoked, and then define the user-defined reception in the development environment. For details about how to define a user-defined reception, see *Chapter 2. Defining a User-defined Reception* in the *Service Platform Reception and Adapter Definition Guide*.
- **Development procedure at the service requester side**
After creating a user-defined reception, create a service requester for invoking the defined user-defined reception. Generate a stub from the WSDL that is used when defining the user-defined reception, and then create a service requester for invoking the stub. For details about creating a service requester, see *Chapter 8. Creating a Service Requester* in the *Service Platform Basic Development Guide*.

2.1.2 Procedure for invoking service components

The operation procedure for invoking service components from the service requester differs depending on the combination of reception type and HCSC component to be invoked. The possible combinations of reception types and HCSC components to be invoked are as follows:

- A service adapter is directly invoked from standard reception to execute service components
- A business process is invoked from standard reception to execute service components
- A business process is invoked from user-defined reception to execute service components

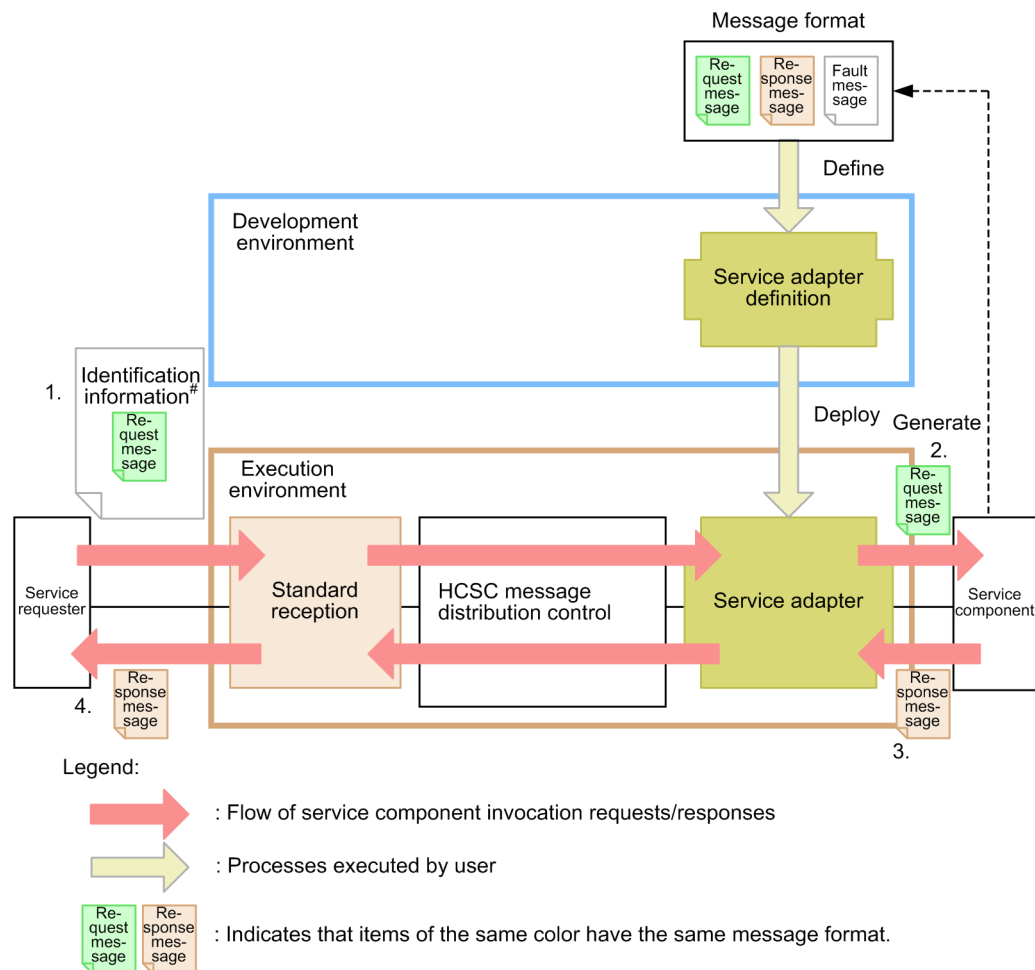
The flow of messages for each case is described below. For details about the flow of processing if an error occurs, see *Chapter 7. Troubleshooting in the Service Platform Setup and Operation Guide*.

(1) A service adapter is directly invoked from standard reception to execute service components

If a service adapter is directly invoked from standard reception (to execute a service component), the service name defined in the service adapter is specified in the service requester, and then a request message is sent. When the request message is sent, the service component defined in the service adapter is invoked.

The following figure shows the flow of messages when a service adapter is invoked directly.

Figure 2-3: Flow of messages when a service adapter is invoked directly



#: Identification information includes details such as service names and client correlation IDs.

2. Functionality for Connecting to Various Types of Systems

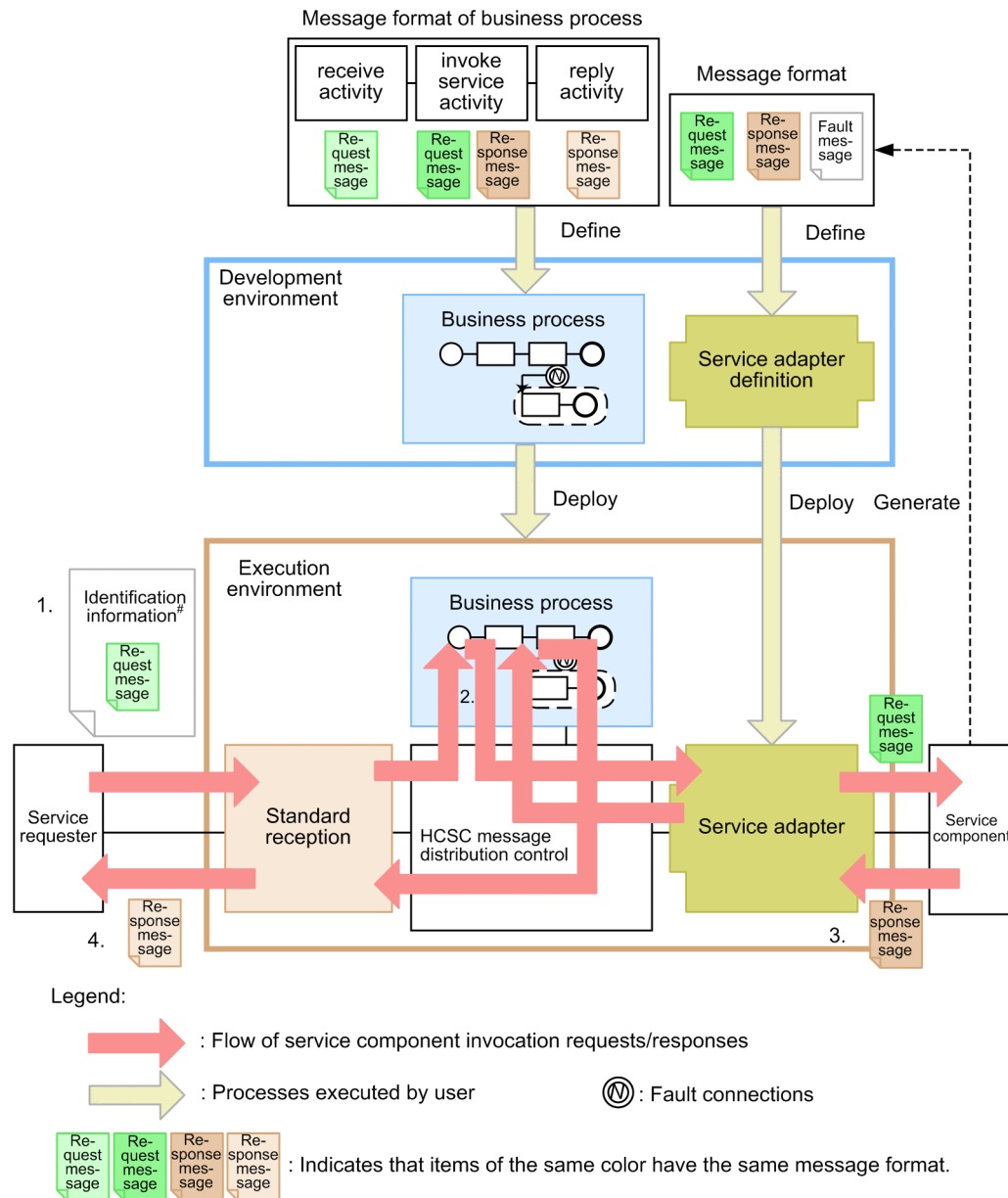
1. For request messages sent from the service requester to standard reception, the message format specified when the service adapter was defined in the development environment (matching the type of the service component) is used. Therefore, in the service requester, create a request message matching the message format defined in the service adapter. Then, specify the created message in the parameters of standard reception (parameters of the user message), and then request execution of a service component.
2. Invoke the service component from the service adapter. When invoking the service component, use the request message defined in the service adapter. In this way, the request message sent from the service requester and the request message received by the service component will be the same. Even if you use a message format other than the message format of the service component, you can send a request from the service requester if you define data transformation in the service adapter.
3. Send a response message from the service component to the service adapter. For the response message, use the same message format as the response message defined in the service adapter.
4. Send a response to the service requester. For the response, use the message format of the response message defined in the service adapter. In this way, the response message returned from the service component and the response message received by the service requester will be the same.

For details about data transformation definitions, see 6.3. *Defining data transformation* in the *Service Platform Basic Development Guide*.

(2) A business process is invoked from standard reception to execute service components

If a business process is invoked from standard reception to execute a service component, the service name (business process name) is specified in the service requester, and then a request message is sent. Then, processing is performed according to the process defined in the business process. A process consists of activities. Among those activities, an invoke service activity invokes the defined service adapter and then executes the service component defined in the service adapter. The following figure shows the flow of messages when a business process is invoked.

Figure 2-4: Flow of messages when a business process is invoked



#: Identification information includes details such as service names and client correlation IDs.

1. In the service requester, create a request message matching the message format of the request message defined in the receive activity of the business process. Then specify the created message in the parameters of standard reception (parameters of the user message), and then request execution of a service component. For the request message to be specified in the service requester, use the message format defined in the receive activity of the business process.
2. Invoke the service component from the business process. When invoking the service component, use the request message defined in the invoke service activity. The message format of this request message must be the same as the request message format defined in the service adapter of the service component to be invoked. Even if you use a message format other than the message format of the service component, you can send a request from the invoke service activity if you define data transformation in the service adapter.
3. Send a response message from the service component to the service adapter. For the response message, define the same message format as the response message defined in the invoke service activity of the business process.

4. Send a response to the service requester. For the response, use the message format of the response message defined in the reply activity of the business process.

For details about data transformation definitions, see 6.3. *Defining data transformation* in the *Service Platform Basic Development Guide*.

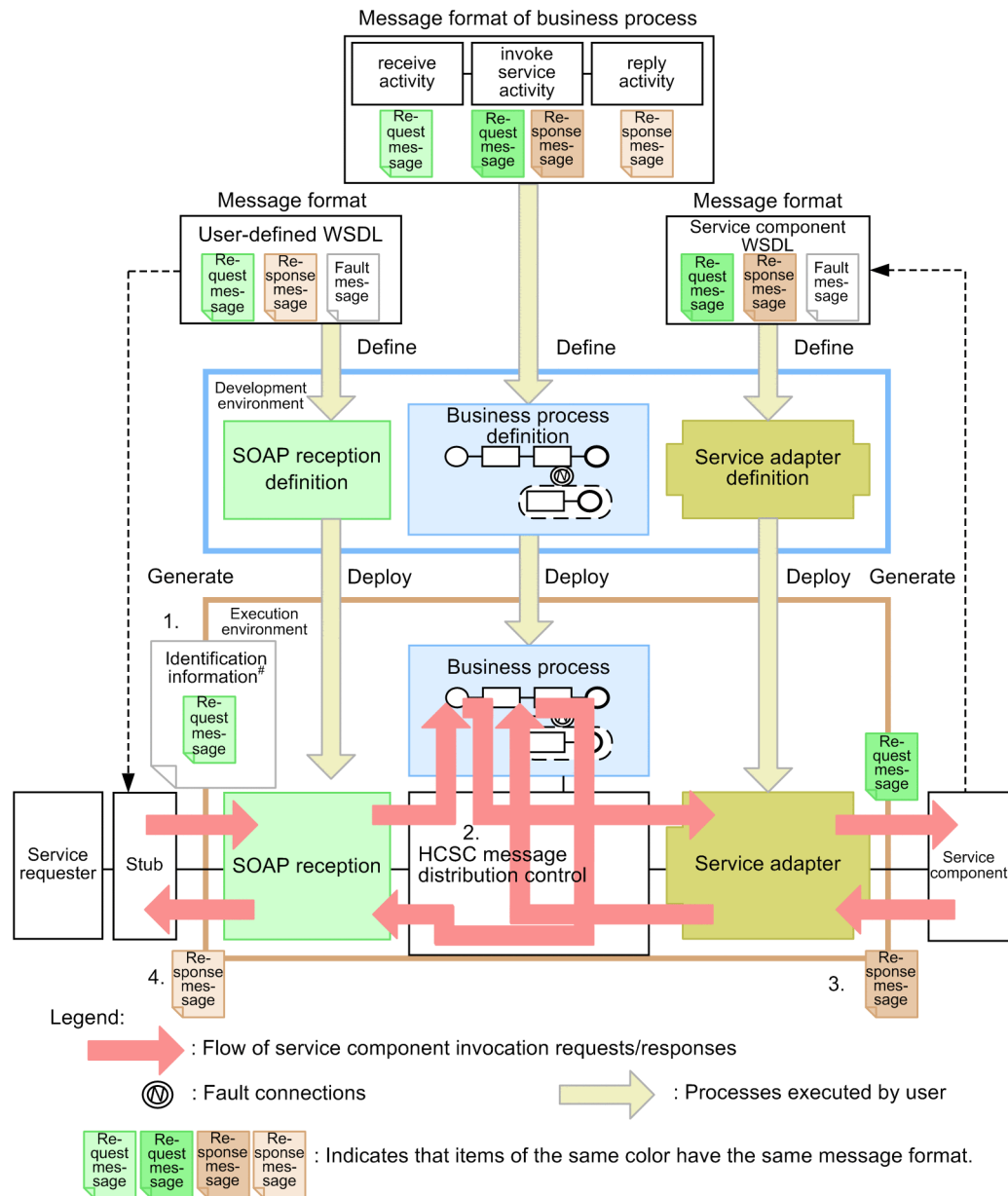
(3) A business process is invoked from user-defined reception to execute service components

User-defined reception is the interface that receives a service component execution request in any format defined by the user, and then returns a response.

A business process is invoked from the user-defined reception according to the definitions specified in the development environment, and then processing is performed according to the process defined in the business process. The invoke service activity invokes the defined service adapter, and then executes the service component defined in the service adapter. The following figure shows the flow of messages when the user-defined reception is used.

Note that the following is an example of using the SOAP Communication Infrastructure.

Figure 2-5: Flow of messages when the user-defined reception is used (example of SOAP Communication Infrastructure)



1. The request message sent from the service requester is the same format as the one defined in the receive activity of the business process. Therefore, a WSDL of the type matching the message format defined in the receive activity of the business process must be specified in the user-defined reception definition of the development environment. When invoking a SOAP reception from the service requester, use the WSDL defined in the SOAP reception in the development environment. To use a service requester that supports SOAP1.1 (SOAP Communication Infrastructure or JAX-WS engine), generate a stub by using the `WSDL2Java` command from the WSDL, and then implement the service requester to invoke the stub.

To use a service requester that supports SOAP1.2 (JAX-WS engine), generate a stub by using the `cjwsimport` command from the WSDL, and then implement the service requester to invoke the service class.

2. When invoking the service component from the business process, use the request message defined in the invoke service activity. The message format of this request message must be the same as the request message format defined in the service adapter of the service component to be invoked.

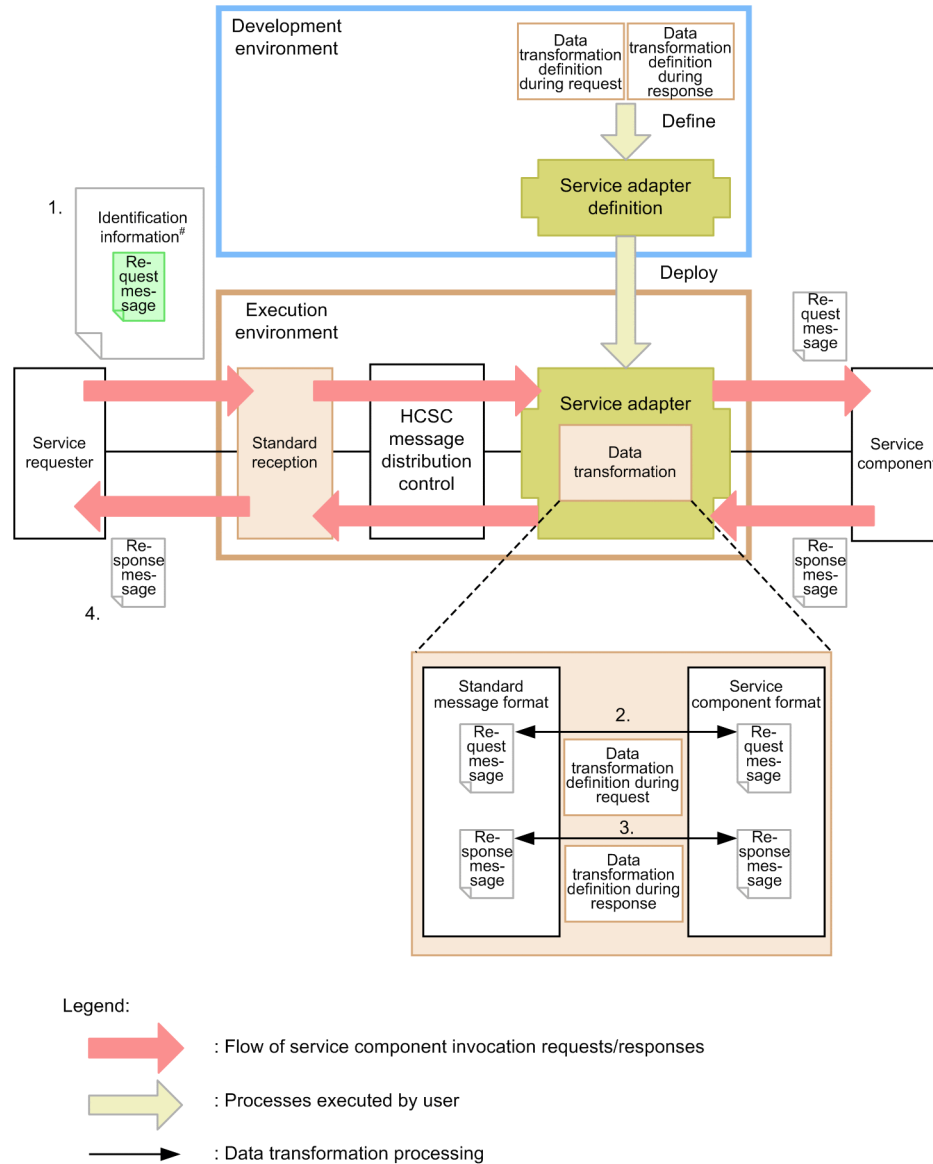
3. For the response message from the service component, also define the same message format as the response message defined in the invoke service activity of the business process. Even if you use a message format other than the message format of the service component, you can send a request from the invoke service activity if you define data transformation in the service adapter.
4. For the response to the service requester, use the message format of the response message defined in the reply activity of the business process. This message format must be the same as the format of the response message of the WSDL specified in the user-defined reception of the development environment.

For details about data transformation definitions, see 6.3. *Defining data transformation* in the *Service Platform Basic Development Guide*.

2.1.3 Flow of invoking a service component with a message format different from the service component side

You can invoke a service component by using a message different from the message format of the service component side, or receive a message different from the message format of the service component side as a response. To do this, you need to define standard message format in the development environment. In the development environment, you also need to define a data transformation definition that defines the transformation rule for standard message format and service component message format. You can invoke a service component by transforming messages in standard message format to messages in service component message format. The following figure shows the relationship between standard message format and a data transformation definition.

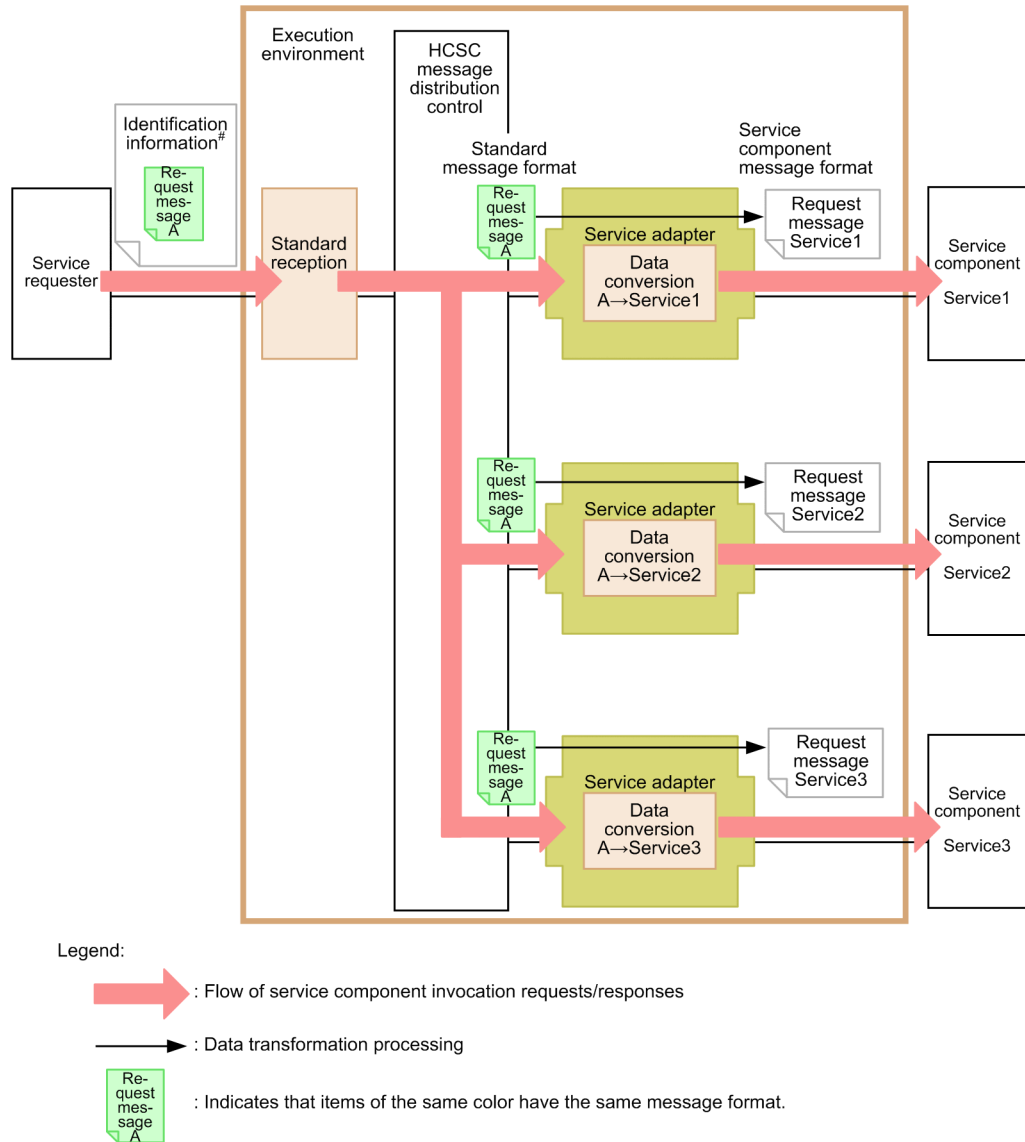
Figure 2-6: Relationship between standard message format and a data transformation definition



1. The service requester sends a request for invoking the service component by specifying a message format matching the type of standard message format in the request message.
2. The service adapter transforms data from standard message format to the service component message format, and then invokes the service component using the transformed message.
3. For the response sent from the service component, the service adapter also transforms the data from the service component message format to standard message format.
4. A response in standard message format is sent to the service requester.

Even for multiple adapters, you can define the same standard message format A and individual data transformation definition for each service adapter for *Service1*, *Service2*, and *Service3* (see the figure below). Thus, when sending a request from the service requester, you can use the same request message by simply changing the service name (such as *Service1*).

Figure 2–7: Process flow when the same standard message format is defined in multiple service adapters



#. Identification information includes details such as service names and client correlation IDs.

For details about standard message format and service component message format, see *Chapter 4. Creating a Message Format* in the *Service Platform Basic Development Guide*. For details about data transformation definitions, see 6.3. *Defining data transformation* in the *Service Platform Basic Development Guide*.

2.1.4 Identification information provided in the service component invocation process

To differentiate each request through on an HCSC, a message common ID and a service request ID is allocated to each request message. These IDs are output in the message log, request trace, and the performance analysis trace.

By following the message common ID and service request ID, you can trace the flow of the service component invocation process. For information about tracing messages, see 7.7 *Troubleshooting the requests to invoke a service component* in the *Service Platform Setup and Operation Guide*.

In addition to the identification information allocated on an HCSC server, a client correlation ID is also set up in which the user can specify any value.

(1) Types of identification information

- Message common ID

A message common ID is allocated automatically when a service component invocation request is received from a service requester. The same message common ID is used until a response is sent.

Even when a service requester invokes a business process, individual service component invocation processes within the business process can be identified as the same process. The format of the message common ID is as follows:

```
CSC_[server-name]_[reception-time (milliseconds)]_[serial-number]
```

#: The serial number added at the end is a number allocated in a sequence starting from 1.

Example: CSC_CSCServerName_2008-04-20_11:32:18.360_1

- Service request ID

The service request ID is identification information allocated to each individual service component invocation process within the HCSC server.

The service request ID is allocated not only when a service component invocation request from the service requester is received, but also when a service component invocation process from the business process is executed. The format of the service request ID is as follows:

```
MSG_[server-name]_[reception-time (milliseconds)]_[serial-number]
```

#: The serial number added at the end is a number allocated in a sequence starting from 1.

Example: MSG_CSCServerName_MDBWSR_2008-04-20_11:32:18.360_1

- Business process instance ID

The business process instance ID is identification information (process instance identifier) for identifying individual process instances of a business process. The business process instance ID is used when re-execution of a business process from a service requester is requested, and when the execution log of process instances is to be deleted.

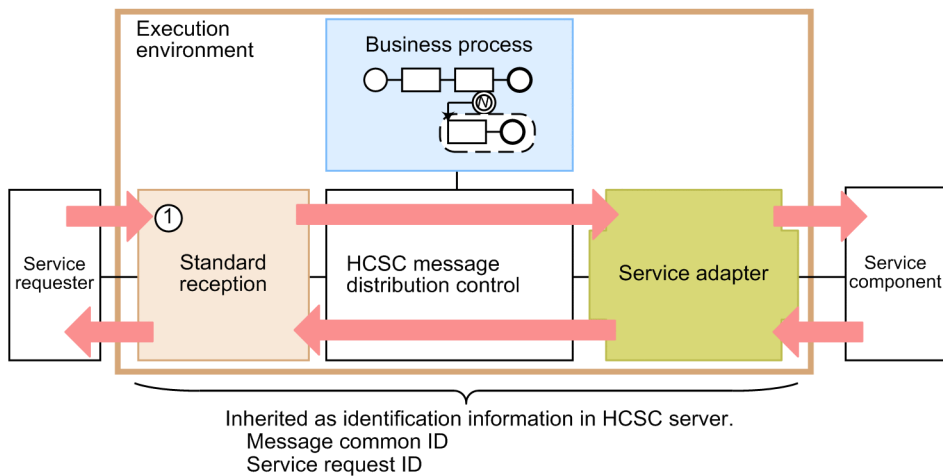
(2) Timing of allocation of identification information

When the service adapter is invoked directly from standard reception, the message common ID and service request ID are inherited as the identification information on the HCSC server. When the service adapter is invoked from standard reception and user-defined reception via the business process, the message common ID is inherited as the identification information on the HCSC server. However, the service request ID is inherited as the identification information for each service component invocation process. Furthermore, the business process instance ID is allocated when the business process first receives a request.

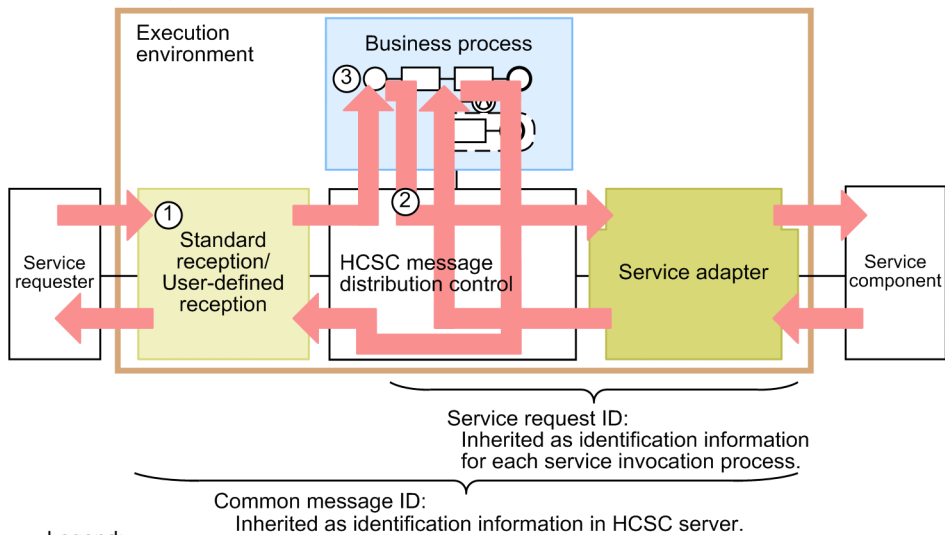
The following figure shows the timing of allocation of message common IDs and service request IDs.

Figure 2–8: Timing of allocation of identification information

- When invoking service adapters directly from standard reception:



- When invoking business processes:



Legend:

- ➡ : Flow of service component invocation requests/responses
- ① : Allocation of common message IDs and service request IDs
- ② : Allocation of service request IDs
- ③ : Allocation of business process instance IDs
- Ⓜ : Fault connections

2.2 Connecting to services by using Web Services (SOAP communication)

2.2.1 Relationship with the Communication Infrastructure for Web Services

The HCSC server of the execution environment operates on the Communication Infrastructure for Web Services and Web server (HTTP server). Therefore, when you want to use Web Services (SOAP communication), you must set up the Communication Infrastructure for executing Web Services and the Web server.

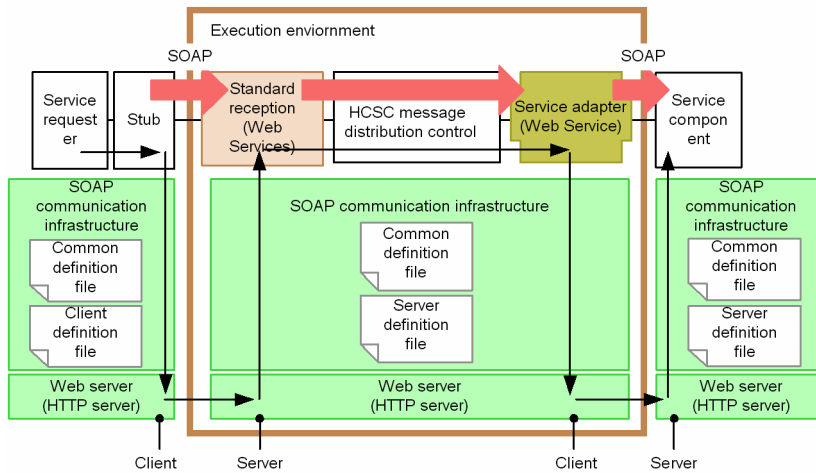
The relationship between a service requester and a request reception in an HCSC server is that of a client and a server, and the relationship between a service adapter of an HCSC server and a service component is again that of a client and a server.

The SOAP Communication Infrastructure and JAX-WS engine might be used as the Communication Infrastructure at the service requester side and the service component side. When using a service requester and service components compliant with SOAP1.1, use either the SOAP Communication Infrastructure or the JAX-WS engine. Also, when using a service requester and service components compliant with SOAP1.2, use the JAX-WS engine.

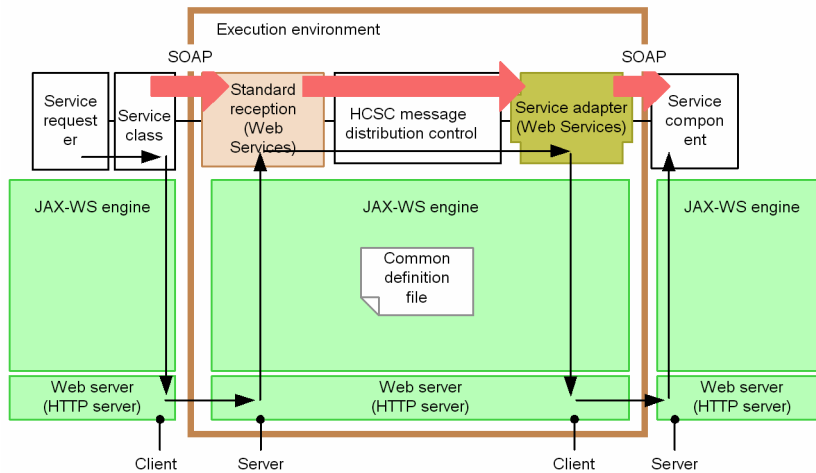
The following figure shows the relationship of the service requester and service components with the SOAP Communication Infrastructure. Note that for service requesters and service components, you might also use the SOAP Communication Infrastructure for one and JAX-WS engine for the other.



Figure 2–9: Relationship with the SOAP Communication Infrastructure

- When using SOAP communication infrastructure



- When using JAX-WS engine



Legend:
 : Flow of service component invocation request
 : Message flow

2.2.2 Relationship between user messages and WSDL when a service component is invoked

The relationship between user messages and the WSDL when a service component is invoked is different for standard reception and user-defined reception.

(1) For standard reception

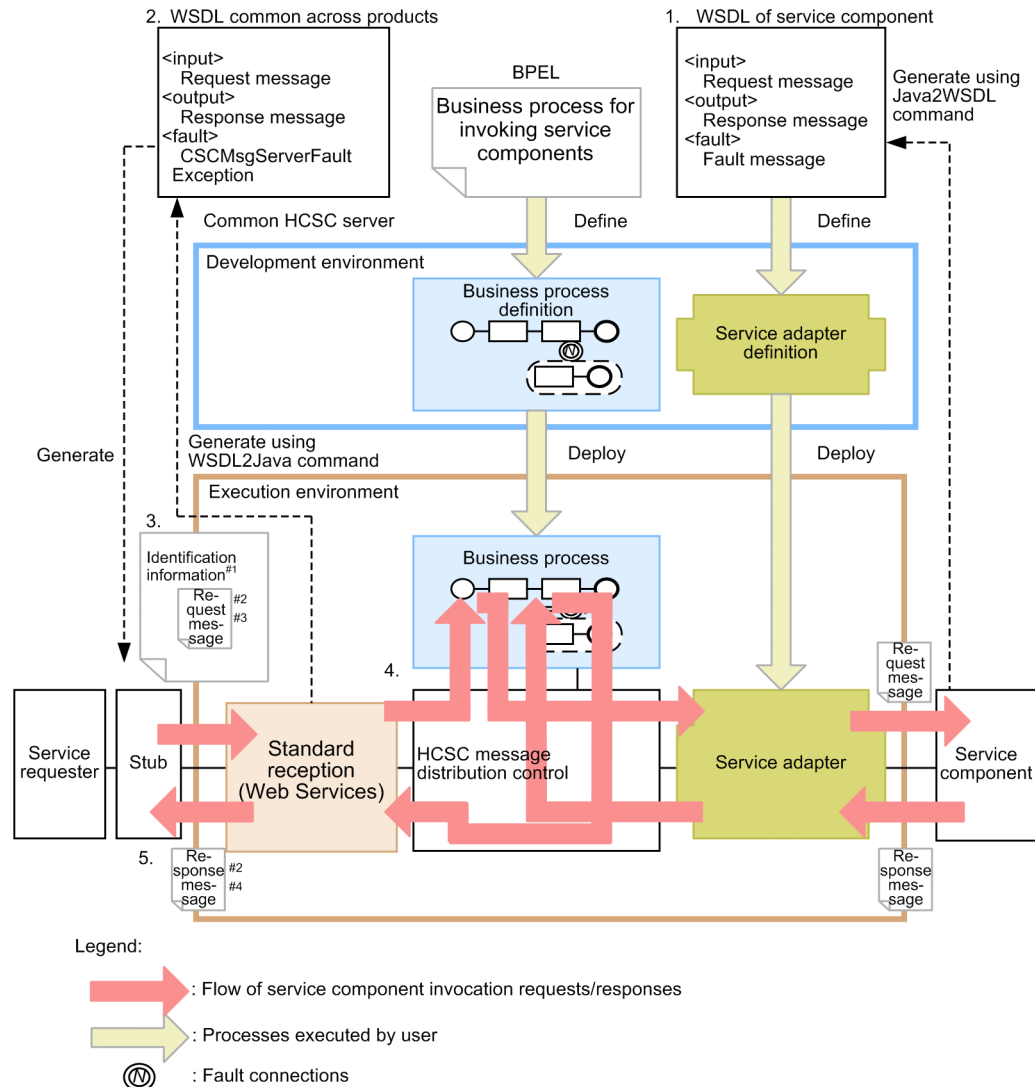
The following describes the relationship between user messages and the WSDL during SOAP communication (standard reception (Web Services)).

(a) When the SOAP Communication Infrastructure is used

The figure below shows the relationship between user messages and the WSDL during SOAP communication (standard reception (Web Services)) when the SOAP Communication Infrastructure is used as the communication infrastructure. You can also use a configuration in which the SOAP Communication Infrastructure is used at the

service requester side or the service component side and the JAX-WS engine is used at the other side. For details about the side on which JAX-WS engine is used, see (b) *When the JAX-WS engine is used*.

Figure 2–10: Relationship between user messages and the WSDL during SOAP communication (standard reception (Web Services)) when the SOAP Communication Infrastructure is used



- #1: Identification information includes details such as service names and client correlation IDs.
 #2: Request message or response message defined in the WSDL of a service component.
 #3: Request message defined in the reception activity of a business process when invoking the business process.
 #4: Response message defined in the response activity of a business process when invoking the business process.

1. When defining a service adapter, define the WSDL of the service component as the message format of the service component side (formats of the request message, response message, and fault message are defined in this WSDL). To create a WSDL, use the `Java2WSDL` command provided with the SOAP Communication Infrastructure.
2. When invoking a service component by invoking standard reception (Web Services) from the service requester, use the WSDL provided with the HCSC server. You can create a stub by using the `WSDL2Java` command based on the WSDL provided with the HCSC server.
3. When invoking standard reception (Web Services) from the service requester, create in advance a request message matching the message format of the service component side. Then, specify the created message in the parameters of standard reception (Web Services) (parameters of the user message) defined in the WSDL provided with the HCSC server, and then request execution of a service component.
4. A request message for invoking a business process from the service requester is the message format defined in the receive activity of the business process. In the service requester, create a message in the message format of the

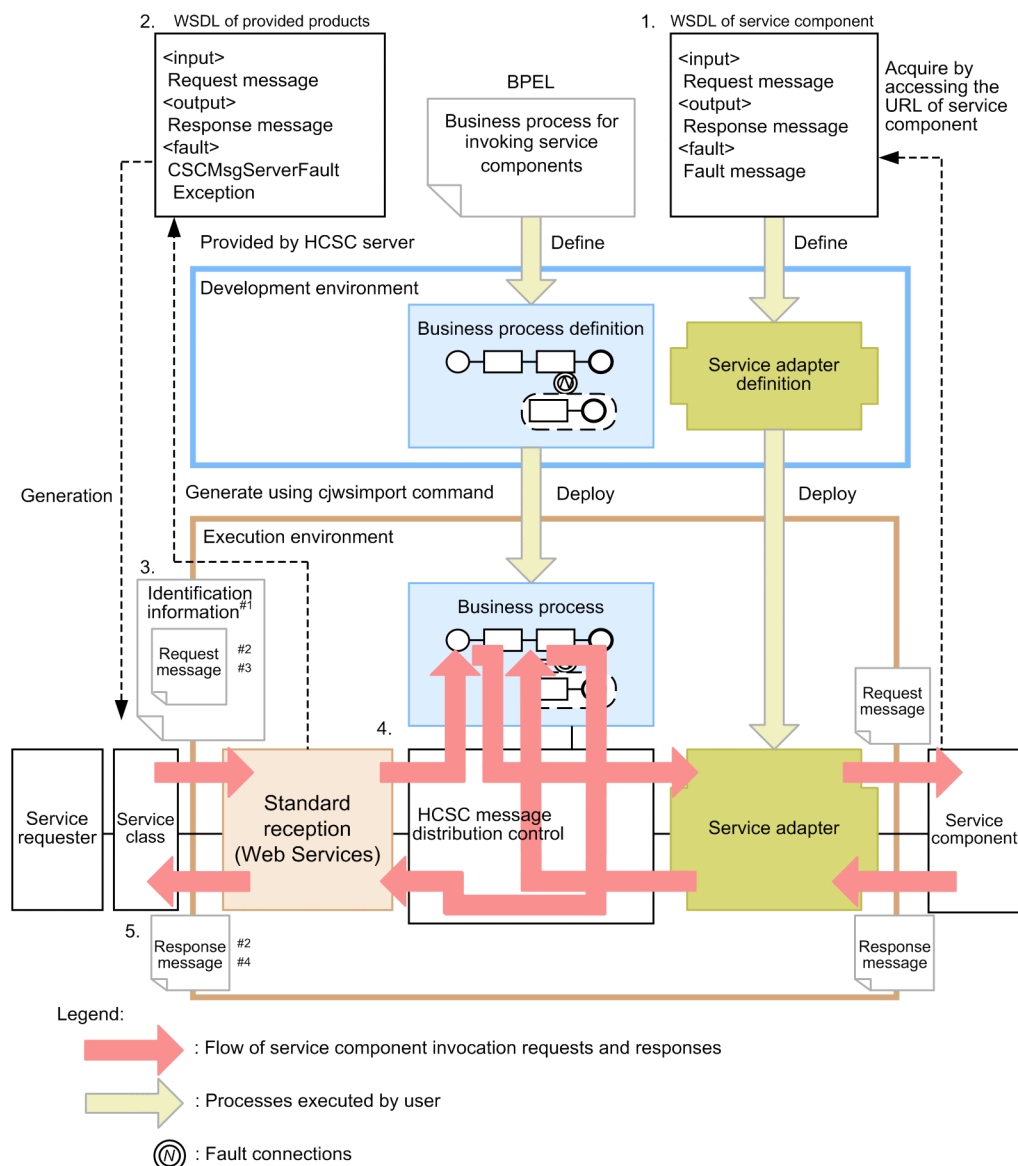
request message defined in the receive activity of the business process. Then, specify the created message in the parameters of standard reception (Web Services) (parameters of the user message), and then request execution of a service component.

5. For the response to the service requester, use the message format of the response message defined in the service adapter.

(b) When the JAX-WS engine is used

The figure below shows the relationship between user messages and the WSDL during SOAP communication (standard reception (Web Services)) when the JAX-WS engine is used as the communication infrastructure. You can also use a configuration in which the JAX-WS engine is used at the service requester side or the service component side, and the SOAP Communication Infrastructure is used at the other side. For details about the side on which the SOAP Communication Infrastructure is used, see (a) *When the SOAP Communication Infrastructure is used*.

Figure 2–11: Relationship between user messages and the WSDL during SOAP communication (standard reception (Web Services)) when the JAX-WS engine is used



1. When defining a service adapter, define the WSDL of the service component as the message format of the service component side (formats of the request message, response message, and fault message are defined in this WSDL). The WSDL of a service component is acquired by accessing the URL prepared at the service component side.
2. When invoking a service component by invoking standard reception (Web Services) from the service requester, use the WSDL provided with the HCSC server. You can create a service class by using the `cjwsimport` command based on the WSDL provided with the HCSC server.
3. When invoking standard reception (Web Services) from the service requester, create in advance a request message matching the message format of the service component side. Then, specify the created message in the parameters of standard reception (Web Services) (parameters of the user message) defined in the WSDL provided with the HCSC server, and then request execution of a service component.
4. A request message for invoking a business process from the service requester is the message format defined in the receive activity of the business process. In the service requester, create a message in the message format of the request message defined in the receive activity of the business process. Then, specify the created message in the parameters of standard reception (Web Services) (parameters of the user message), and then request execution of a service component.
5. For the response to the service requester, use the message format of the response message defined in the service adapter.

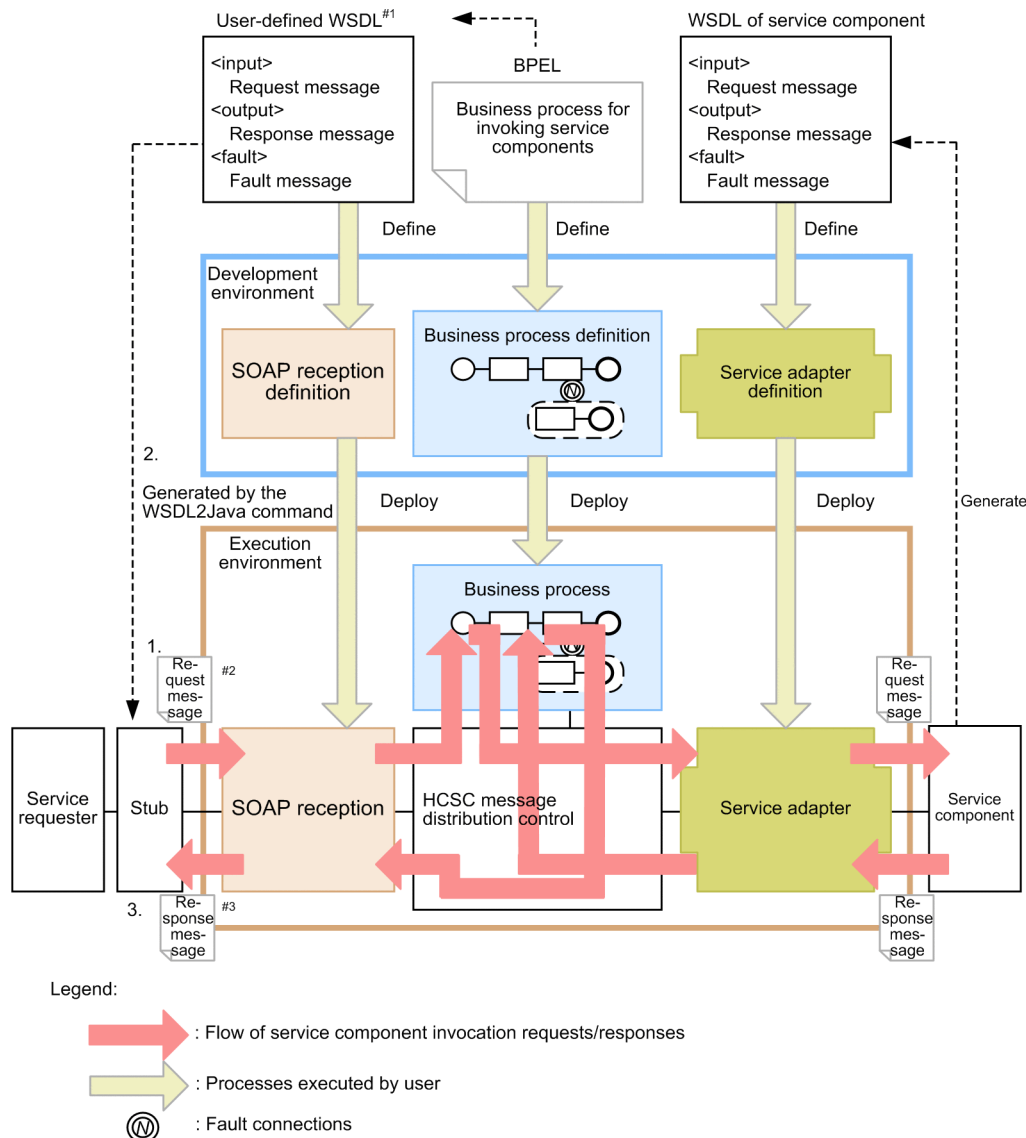
(2) For user-defined reception

The following describes the relationship between user messages and the WSDL during SOAP communication (SOAP reception).

(a) When the SOAP Communication Infrastructure is used

The following figure shows the relationship between user messages and the WSDL during SOAP communication (SOAP reception) when the SOAP Communication Infrastructure is used as the communication infrastructure.

Figure 2–12: Relationship between user messages and the WSDL during SOAP communication (SOAP reception) when the SOAP Communication Infrastructure is used



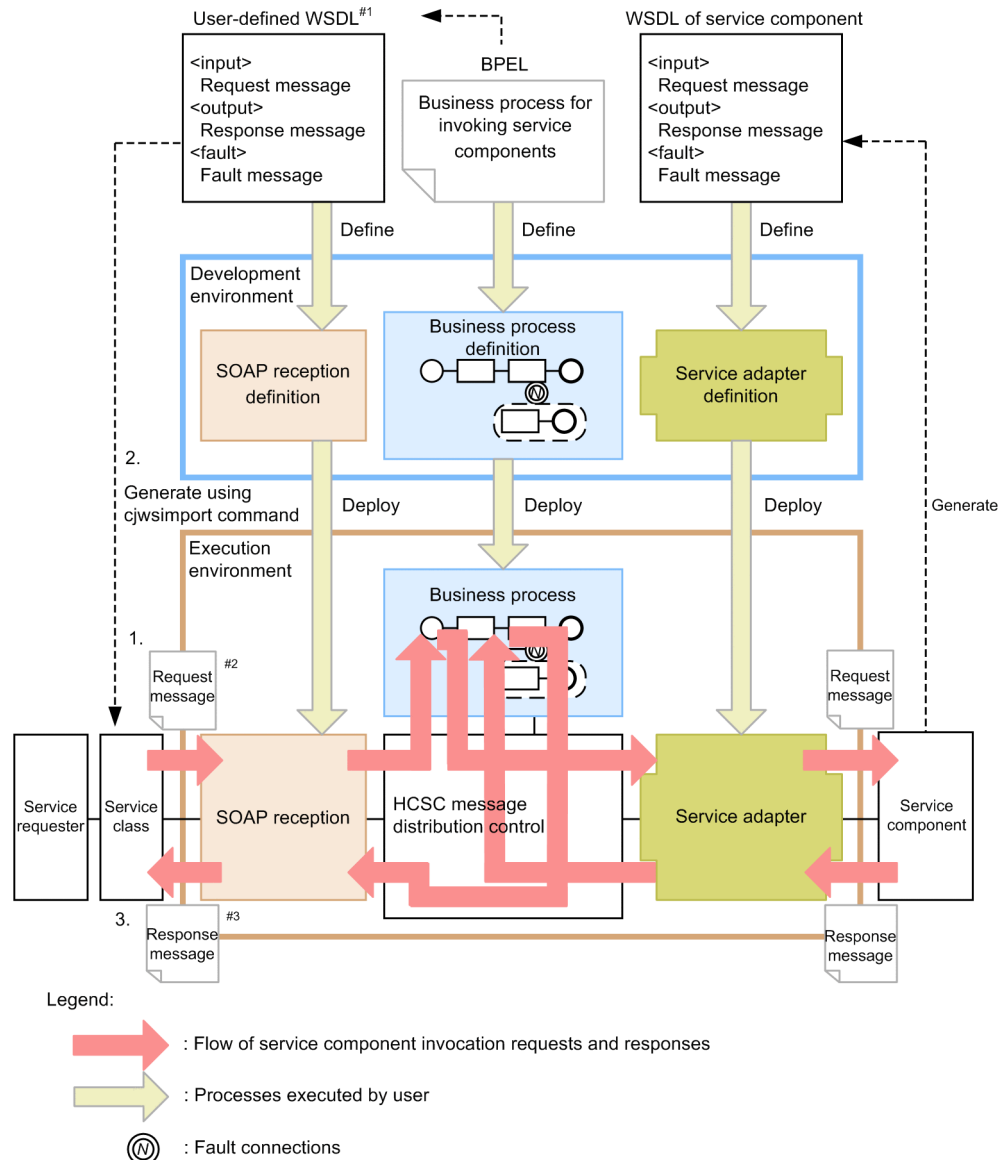
#1: WSDL matching the format defined in the reception activity and response activity of the business process.
 #2: Request message defined in the user-defined WSDL.
 #3: Response message defined in the user-defined WSDL.

1. For a SOAP reception, a request message specified in the service requester is sent in a message format defined in the receive activity of the business process. (Create a WSDL matching the message format defined in the receive activity, and then define the WSDL in the SOAP reception.)
2. A stub can be created by using the WSDL2Java command provided with the SOAP application development support functionality based on the user-defined WSDL. In contrast to standard reception (Web Services), you need not create an XML of the request message in the program at the service requester side. When the service requester sends a request to the stub, set a value in each parameter to request execution of a service component (create a SOAP message within the stub).
3. A response to the service requester is sent in the message format of the response message defined in the reply activity of the business process. (Create a WSDL matching the message format defined in the reply activity, and then define the WSDL in the SOAP reception.)

(b) When the JAX-WS engine is used

The following figure shows the relationship between user messages and the WSDL during SOAP communication (SOAP reception) when the JAX-WS engine is used as the communication infrastructure.

Figure 2-13: Relationship between user messages and the WSDL during SOAP communication (SOAP reception) when the JAX-WS engine is used



#1: WSDL matched with the format defined in the receive and reply activities of the business process.

#2: Request messages that are defined in the user-defined WSDL.

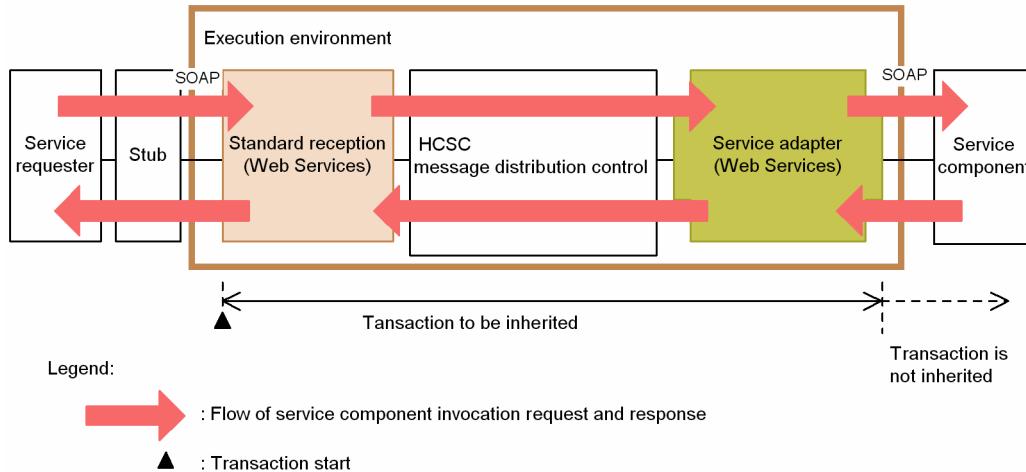
#3: Response messages that are defined in the user-defined WSDL.

1. For a SOAP reception, a request message specified in the service requester is sent in a message format defined in the receive activity of the business process. (Create a WSDL matching the message format defined in the receive activity, and then define the WSDL in the SOAP reception.)
2. You can create a service class by using the `cjwsimport` command based on the WSDL provided with the JAX-WS. In contrast to standard reception (Web Services), you need not create an XML of the request message in the program at the service requester side. When the service requester sends a request to the service class, set a value in each parameter to request execution of a service component (create a SOAP message within the service class).
3. A response to the service requester is sent in the message format of the response message defined in the reply activity of the business process. (Create a WSDL matching the message format defined in the reply activity, and then define the WSDL in the SOAP reception.)

2.2.3 Transactions when using Web Services (SOAP communication)

When you use a Web Service (SOAP communication), the transactions managed within an HCSC server are not inherited in the service. The following figure shows the transactions when using a Web Service (SOAP communication):

Figure 2–14: Transactions when using a Web Service (SOAP communication)



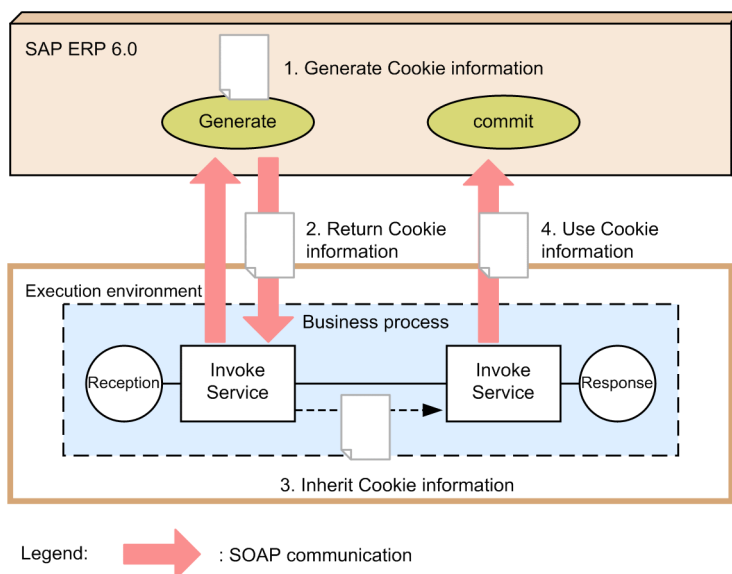
2.2.4 Cookie information in Web service (SOAP communication)

To invoke a service provided in SAP ERP 6.0 from a business process, the Cookie information returned by the update process must be inherited. When invoking a service other than those provided in SAP ERP 6.0, the Cookie information need not be inherited.

There are 2 methods to inherit the Cookie information while invoking a service provided in SAP ERP 6.0. One is to enable the Cookie information inheritance by specification of cookie-parsing property of HCSC runtime definition file and another is to set HTTP header name in the header allocation variable of a business process. For details on how to set HTTP header name in the header allocation variable of a business process, see "2.2.5 Inheriting HTTP header in Web service (SOAP communication)".

The following figure shows the flow of inheriting the Cookie information:

Figure 2–15: Inheriting the Cookie information in service invocation of SAP ERP 6.0



Set-Cookie header information is inherited when a service component is issued as Cookie information.

If all the following conditions for inheriting the Cookie information are met, Set-Cookie header attribute is set in Cookie header.

- Set only Cookie name of Set-Cookie header and its value (NAME=VALUE).
- If there are multiple Cookie information items, separate them by ";" and set.

If conditions are not met, nothing is set in the Cookie header.

The following points give details on inheritance range of the Cookie information, inheritance conditions, and log output. Points to be considered for inheriting the Cookie information are also described here.

(1) Inheritance range of the Cookie information

The range of business processes inheriting the Cookie information is described here.

1. From the first service invocation till execution of receive activity and reply activity
2. From the first service invocation till the business process instance ends

In the range of 1., the Cookie information is also inherited in the following cases:

- If the business processes are invoked hierarchically
The cookie information is inherited even if receive activity and reply activity exist in the business process, which is being invoked.
- If a business process containing asynchronous receive activity is invoked
If the communication model is asynchronous, reply activity cannot be deployed for the operation name defined in the receive activity. In such cases, the Cookie information is inherited only when there is no receive activity between invoke service activities.
- If a business process is invoked in which the activity is defined after the reply activity
In such cases, Cookie information is inherited only when there is no receive activity between invoke service activities.
- If a business process is invoked in which wait activity is defined
If there is a wait activity after a reply activity is executed, the Cookie information is inherited even if there is invoke service activity before and after the wait activity.

However, the Cookie information is not inherited in the following cases regardless of inheritance range:

- If the execution environment process is down
- If a request is to be processed in multiple execution environment processes
- If a process instance is re-executed

(2) Inheritance conditions of the Cookie information

The Cookie information is inherited if all the following conditions are met for information on invoking of service components and for the information on the invoked service components:

- ON is specified in the cookie-parsing property of HCSC runtime definition file or the Cookie information is set in the header allocation variable in the invoke service activity that invokes a SOAP adapter.
- Character string of "Service component host name: Port number" matches completely.
Each different host name specification format is handled as a separate URL. For example, if the same host is specified, when items specified in the host name and items specified in IP address are mixed, each is handled as a separate URL. At this stage, the Cookie information is not inherited. Also, if the port number is not set, there is comparison with a character string that does not contain ":Port number".
- path attributes matches in front.

Examples of the cases when the front is considered as matching and as not matching are as follows:

(Example) When front is considered as matching

/sap and /sap, /sap and /sap/aaa, /sap and /sapaaa/bbb

(Example) When front is considered as not matching

/sap and /aaa, /sap and /aaasap/bbb, /sap and /aaa/sap

Notes

- The Cookie information inheritance is determined by "path attribute" of Set-Cookie header specifications of SAP ERP 6.0. However, "path attribute" is not considered while using SOAP communication infrastructure (SOAP1.1 mode) and only "Host name:Port number" is considered. Host name is considered, if port number is not specified.
- If Set-Cookie header does not contain path attribute, "path=/" is set as default value.
- If multiple "path" attributes are set as the attribute name in Set-Cookie header, as the following example shows, operation is to be performed by considering that "path=/" is set in path attribute.

(Example)

Set-Cookie: NAME=value; path=/sap/aaa; path=/sap/bbb

- If the Cookie information containing multiple different path attribute values is set in HTTP header, path attribute value is set in detailed order in HTTP header independent of Set-Cookie reading order.
For example, if the next Set-Cookie header is received, "cookie:NAME3=value3; NAME2=value2; NAME1=value1" is set in HTTP header.

(Example)

Set-Cookie: NAME1=value1; path=/
Set-Cookie: NAME2=value2; path=/sap
Set-Cookie: NAME3=value3; path=/sap/aaa

(3) Log output

If a value above "2" is set in methodtrace-level property of HCSC server runtime definition file, log is output in method trace.

(4) Notes

(a) Unsupported specifications and attributes

- "domain attribute", "expires attribute" and "secure attribute" that are Set-Cookie header specifications in Netscape Communications are not supported.
- "Comment attribute", "Domain attribute", "Max-Age attribute" and "Secure attribute" that are Set-Cookie header specifications in RFC2109 are not supported.
- Set-Cookie2 specifications supported in RFC2965 are not supported.
- "JSESSIONID" is not supported. Session is managed in SAP ERP 6.0 by "sap-contextid attribute".

(b) Version attributes

- You can set Cookie version only in "0" (Set-Cookie specifications of Netscape Communication) or "1" (Set-Cookie specifications of RFC2109). Version is decided in the following order:
 1. If "expires attribute" is defined in Set-Cookie header, version is 0.
 2. If "Version attribute" is defined in Set-Cookie header, version is 1.
 3. If "Max-Age attribute" is defined in Set-Cookie header, version is 1.
 4. If none from 1-3 fits, version is 0.
- Set-Cookie header issued by SAP ERP 6.0 does not contain "Version attribute".
- If "Version attribute" differs in multiple Cookie information items, "Version attribute" value of the first Cookie information read is used.
- In case of operations by Cookie specifications supported by RFC2109, it is necessary to set "Version attribute" setup value in "cookie-version attribute" of Cookie header.

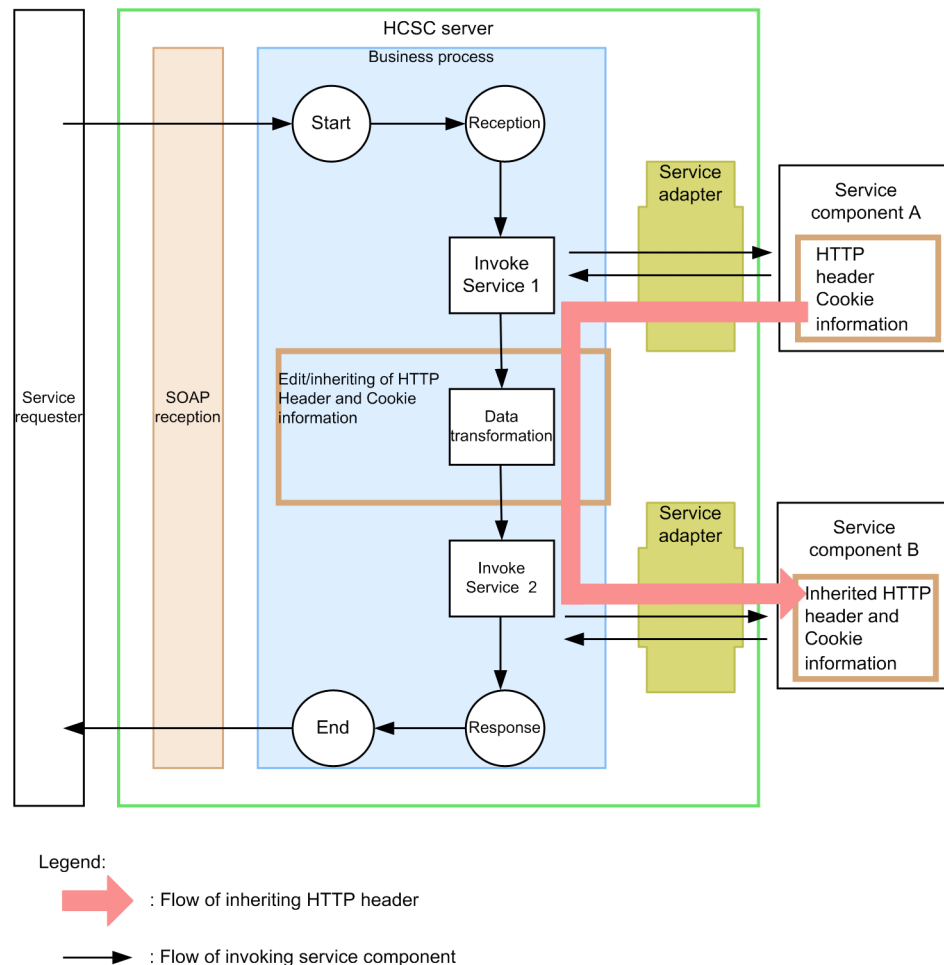
2.2.5 Inheriting HTTP header in Web service (SOAP communication)

HTTP header inheritance functionality is the inheritance functionality of HTTP header acquired when a different service is executed in SAP ERP 6.0 in identical business processes. This functionality is not required when a service other than SAP ERP 6.0 is invoked.

(1) Overview of HTTP header inheritance functionality

The following figure shows the overview of HTTP header inheritance functionality:

Figure 2–16: Overview of HTTP header inheritance functionality



(2) Inheritance method of HTTP header inheritance functionality

HTTP header and Cookie information are inherited by allocation in the header area of invoke service activity. For details on the allocation method in the header area, see "Appendix F Inheriting HTTP header and Cookie information in which service adapter is used" in "Service Platform Basic Development Guide".

(3) Range of inheritance information of HTTP header inheritance functionality

Inheritance information differs depending on the value of cookie-parsing property in HCSC runtime definition file and on whether to allocate information in the header area of the invoke service activity.

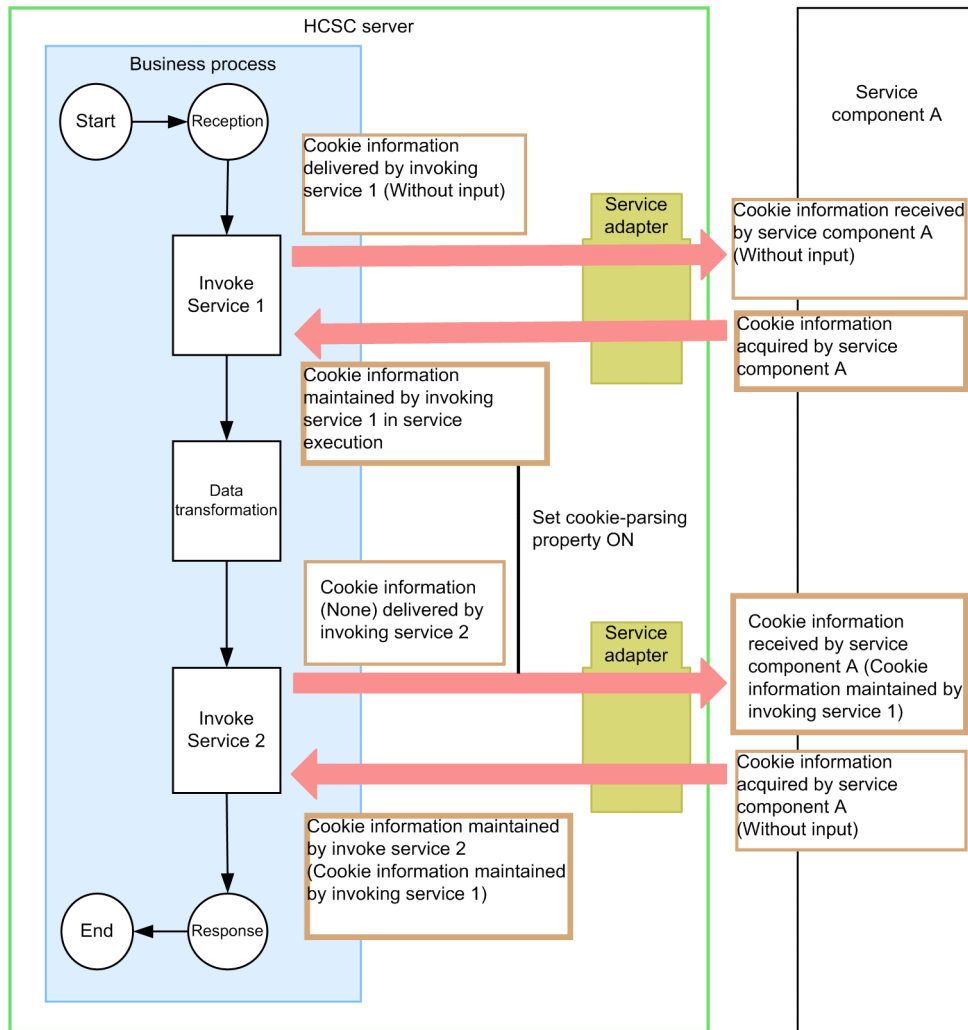
The Cookie information is not sent, if the host name, IP address or path attribute of a service component for which the Cookie information is sent and received does not match.

If cookie-parsing property is ON in HCSC runtime definition file and the Cookie information is not set in header allocation variable

All Cookie information is inherited. Information acquired by service component invocation 1 is inherited in service component invocation 2.

The following figure shows inheritance contents:

Figure 2–17: Inheritance contents if cookie-parsing property is ON in HCSC runtime definition file and the Cookie information is not set in header allocation variable

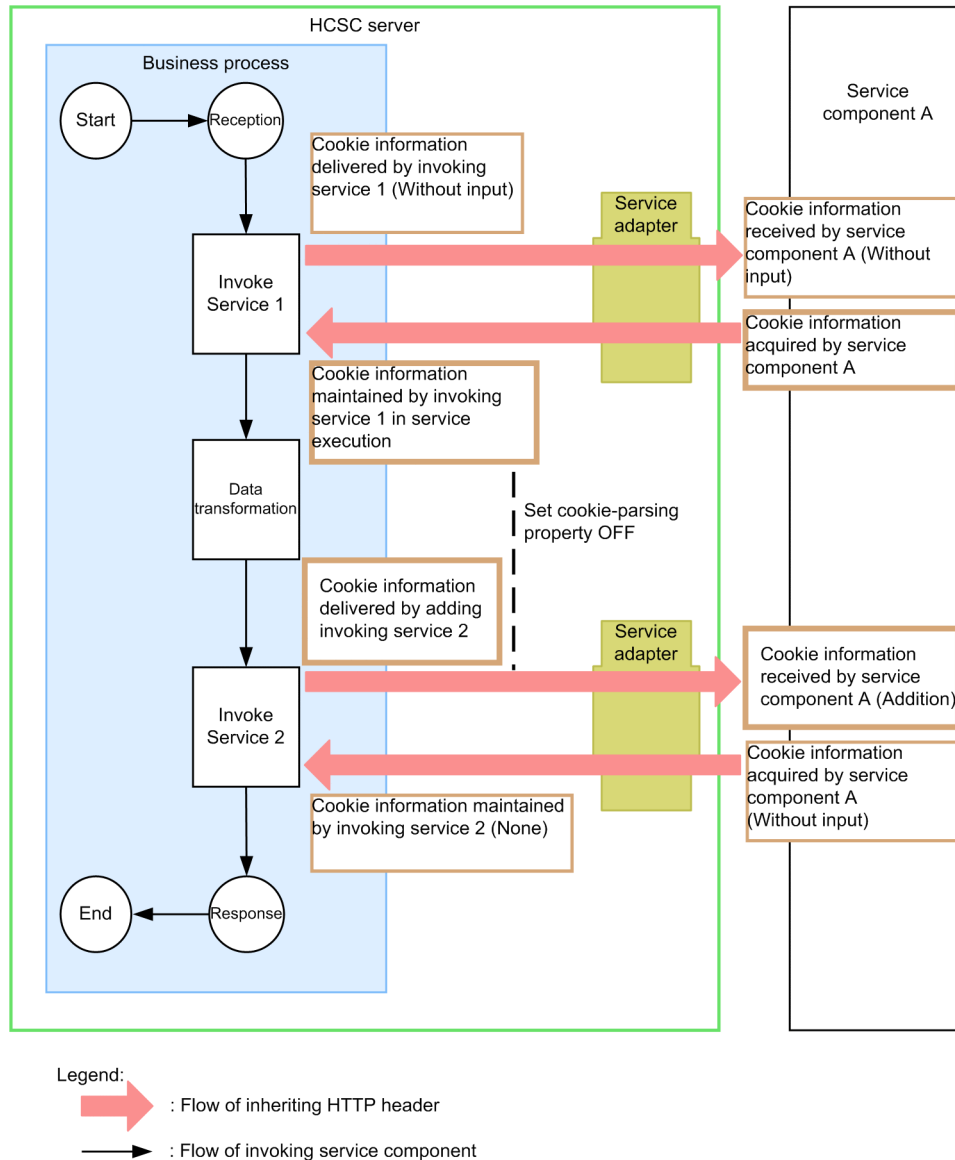


If cookie-parsing property is OFF in HCSC runtime definition file and the Cookie information is set in header allocation variable

The Cookie information set in the header area is inherited by executing service component invocation 2. However, information is not returned in service component invocation 2.

The following figure shows the inheritance contents:

Figure 2–18: Inheritance contents if cookie-parsing property is OFF in HCSC runtime definition file and the Cookie information is set in header allocation variable



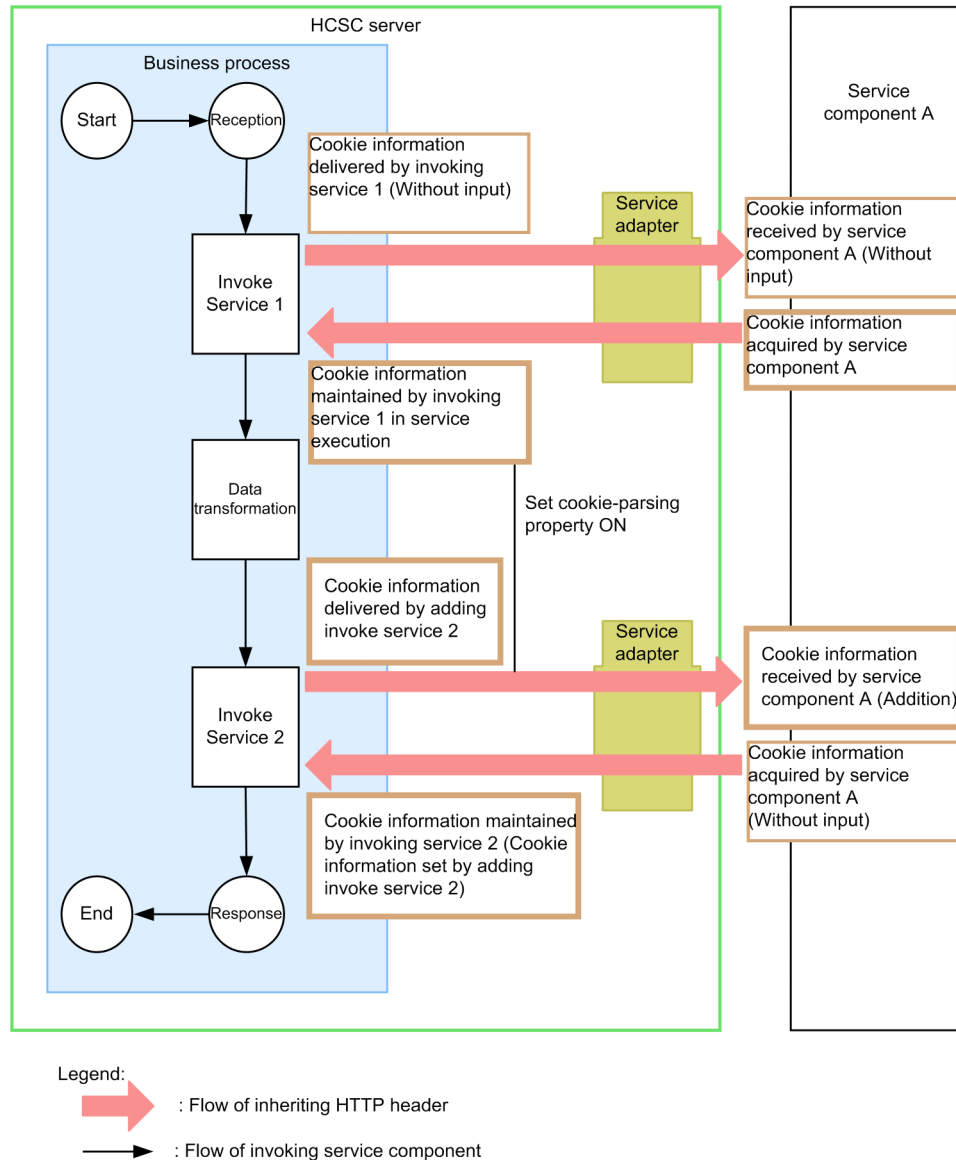
If cookie-parsing property is ON in HCSC runtime definition file and the Cookie information is set in header allocation variable

If the Cookie information in which name attribute, path attribute and host attribute are same is set in the header area, Cookie information acquired by executing service component invocation 1 is overwritten by Cookie information set in the header area and inherited by executing service component invocation 2.

However, if name attribute is same but path attribute or host attribute is different, the Cookie information acquired by service component invocation 1 and Cookie information set in the header area are inherited by executing service component invocation 2.

The following figure shows the inheritance contents:

Figure 2–19: Inheritance contents if cookie-parsing property is ON in HCSC runtime definition file and the Cookie information is set in header allocation variable



2.2.6 SOAP message configuration in Web service (SOAP communication)

SOAP messages handled as request messages and response messages in Web service (SOAP communication) are configured from headers (SOAP header) and body (SOAP body). SOAP header contains additional information of the message (such as identification information) and SOAP body contains the main text of the message.

The following figure shows an example of SOAP message configuration:

Figure 2–20: Example of SOAP message configuration

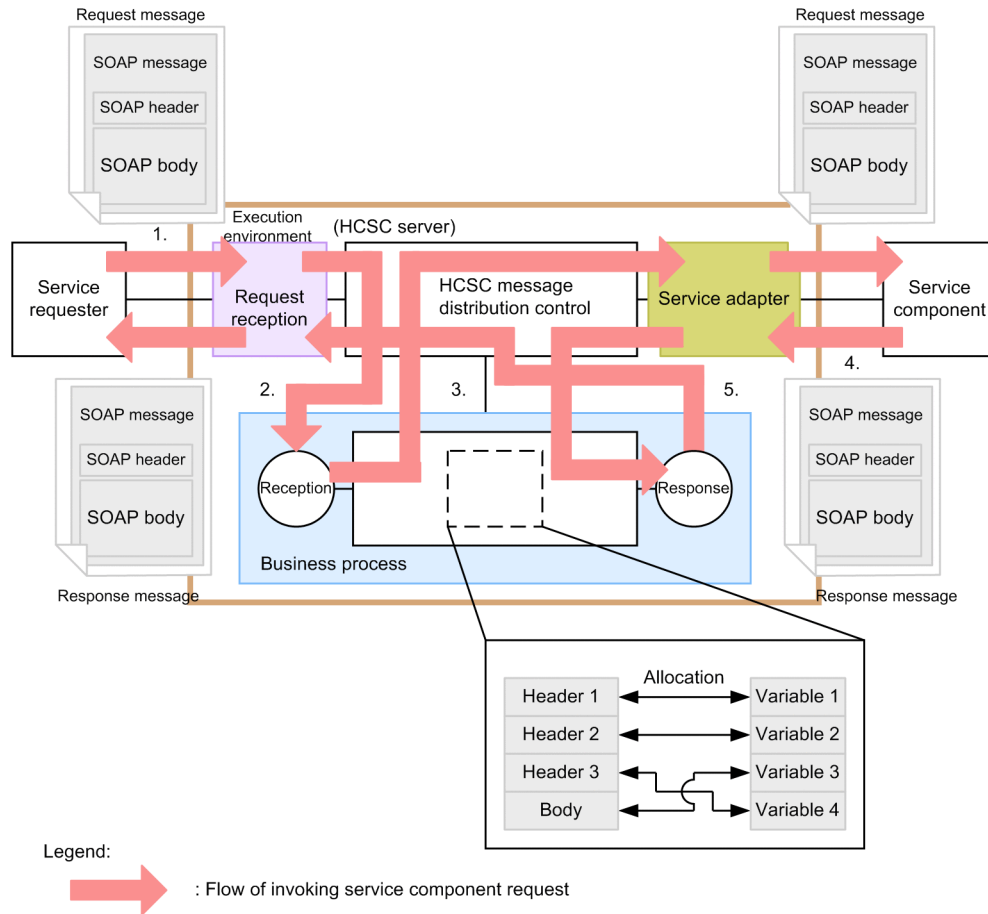


(1) Relation between SOAP messages and business processes

In SOAP messages, header information and body information of request messages is allocated in business process variables so that this information can be used in a business process. When this information is returned as a response message, header information and body information are regenerated from the variable.

The following figure shows the relation between a SOAP message and a business process:

Figure 2–21: Relation between SOAP message and business process



Steps 1 to 5 given in the figure are described here.

Description

1. For reception from the service requester, a request message (SOAP message) setting the parameters required for the business process is returned.
2. A business process is started from reception. At this stage, header information and body information set in the SOAP message received by reception is allocated in the business process variable so that this information can be used in the business process.
3. The business process executes an operation process. When a service is invoked, SOAP message for which parameters required for service execution from the variable is set is generated and this SOAP message is sent to the service component through the service adapter.
4. When the process is finished, the service component returns the result as a response message (SOAP message). The business process receives this SOAP message through the service adapter and re-allocates header and body information in the variable.
5. When processing of the business process is finished, SOAP message is generated from the variable and the result is sent to the service requester through reception.

(2) Information of SOAP messages output in user message trace

In SOAP messages, the following information included in the user message is output in service message trace data:

- Multiple child elements containing soap:Header element
- 1 child element of soap:Body element

If soap:Header element does not exist, only the child element of soap:Body element is output as information.

Since operation is not guaranteed in the following cases, attention is required while creating a request message.

- If a child element containing name space and element specified in user-defined reception nests in the child element under SOAPHeader
- If multiple child elements with identical operation names are set in the child element under SOAPBody

For soap:Header element and soap:Body element, see "2.6.1 Applicability of the service components that use Web service" in "Service Platform Basic Development Guide" .

2.2.7 Dynamic change in connection destination in Web service (SOAP communication)

You can dynamically change SOAP adapter communication destination by using the connection destination URL set in the business process. If you set in advance the communication destination URL in the business process, the communication destination changes when the service adapter is executed. This enables reduction in the number of created service adapters.

If the connection destination URL is not set in the business process, perform SOAP communication in the connection destination URL of Web service set in the development environment.

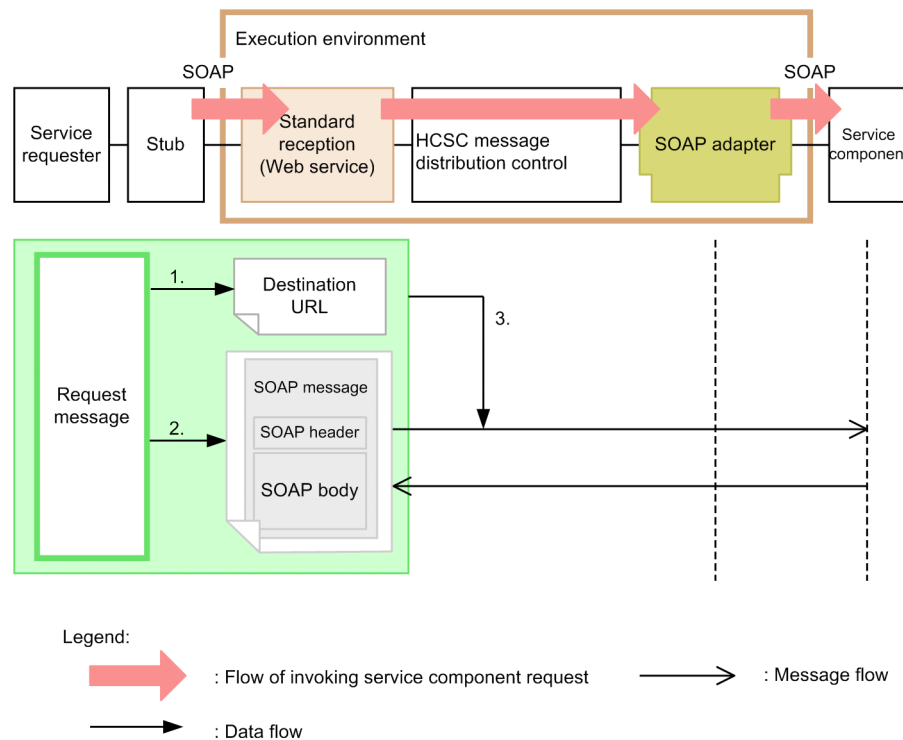
(1) Flow of dynamic change process in communication destination

The following points describe the flow of dynamic change process in communication destination in SOAP communications in each communication infrastructure:

(a) When using SOAP communication infrastructure

The following figure shows the flow of dynamic change process in the communication destination when SOAP communication infrastructure is used in communication infrastructure:

Figure 2–22: Flow of dynamic change process in communication destination when SOAP communication infrastructure is used in communication infrastructure



Description

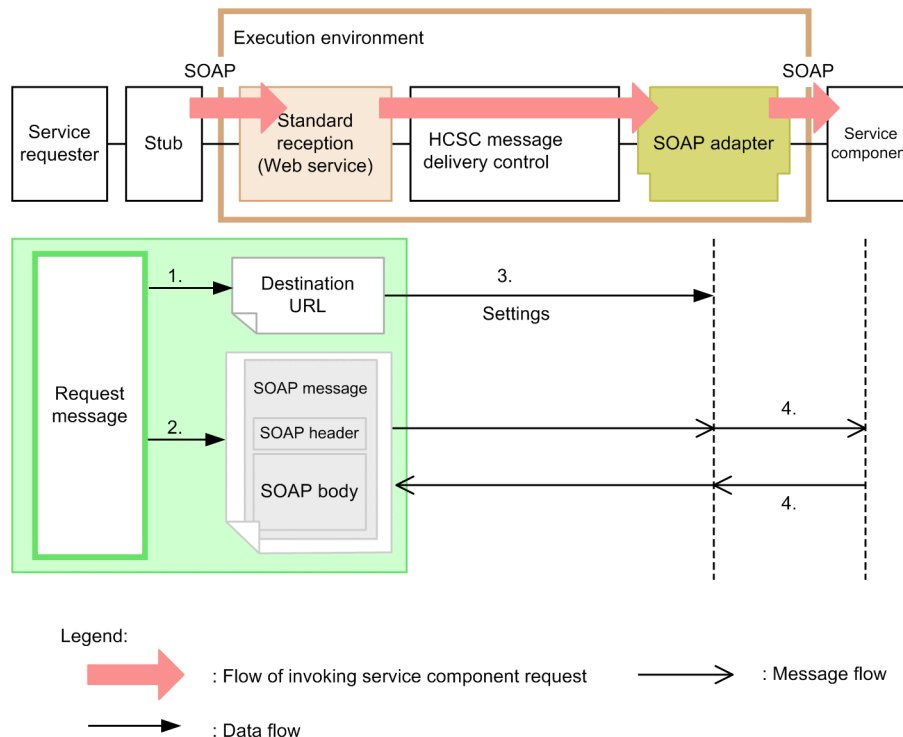
The numbers in the figure correspond to the following numbers:

1. Acquire connection destination URL from the header message in the request message.
2. Create SOAP message on the basis of request message information.
3. Communicate with the service in the acquired connection destination URL.

(b) When using JAX-WS engine

The following figure shows the flow of dynamic change process in the communication destination when JAX-WS engine is used in the communication infrastructure:

Figure 2-23: Flow of dynamic change process in communication destination when JAX-WS engine is used in communication infrastructure



Description

The numbers in the figure correspond to the following numbers:

1. Acquire connection destination URL from the header message in the request message.
2. Create SOAP message on the basis of request message information.
3. Set the connection destination URL.
4. Communicate with the service of the set connection destination URL.

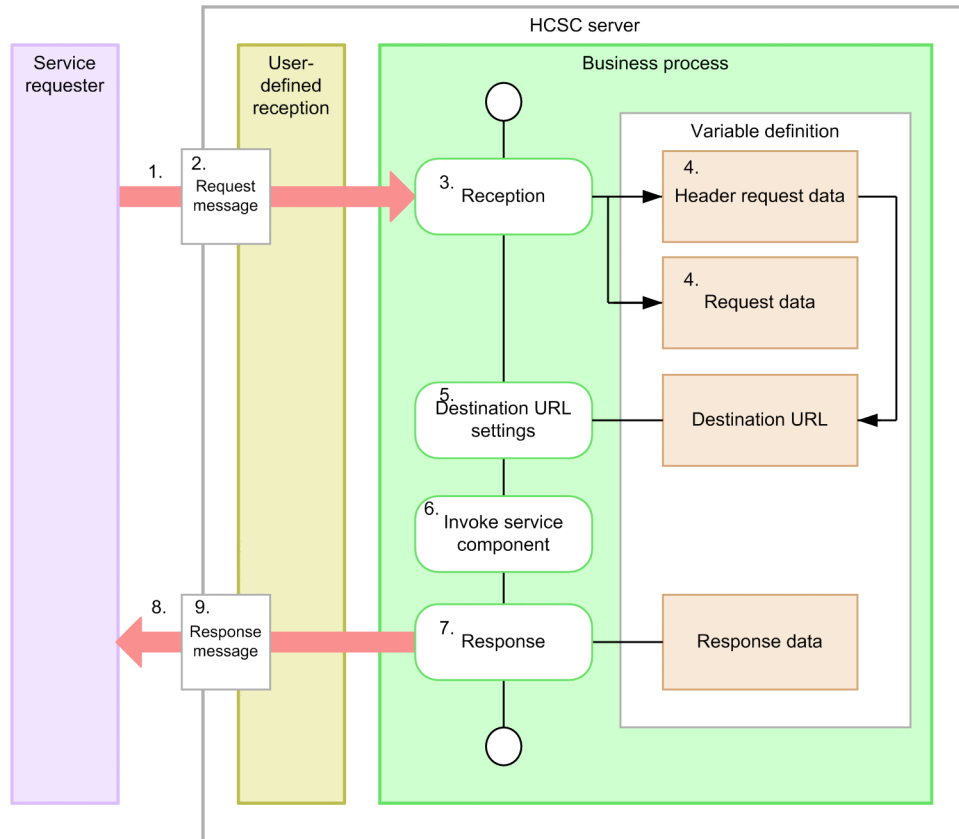
(2) Example of dynamic change of connection destination information

The following points describe an example of dynamic change of connection destination information by using the connection destination information file.

(a) Example of a business process

The following figure shows an example of a business process using connection destination information file:

Figure 2–24: Example of business process using connection destination information file (in SOAP adapter)



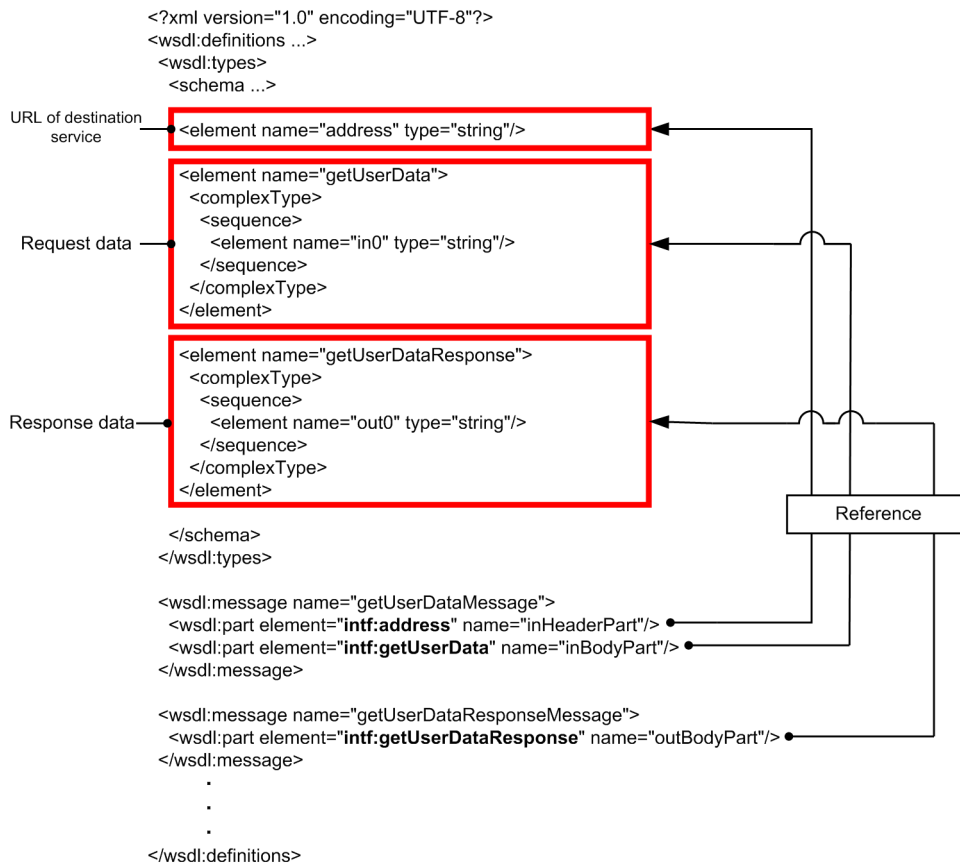
1. The service requester requests for execution of a service component in the business process.
2. Set the connection destination URL in the header message in the request message sent from the service requester.
3. The business process receives XML request messages by the receive activity.
4. The business process generates header request data and request data.
5. Set the connection destination URL of the header request data in the connection destination URL variable by data transformation activity.
6. The business process acquires the connection destination URL from the connection destination URL variable and invokes the service component.
7. The business process configures the service component execution result as an XML response message and returns it by a reply activity.
8. Response is returned to the service requester.
9. XML response message is sent in the response to the service requester.

(3) Example of WSDL creation generating user-defined reception

The following figure shows an example of creating WSDL that generates user-defined reception:

In this example, service component URL to be connected is sent by SOAP header.

2. Functionality for Connecting to Various Types of Systems

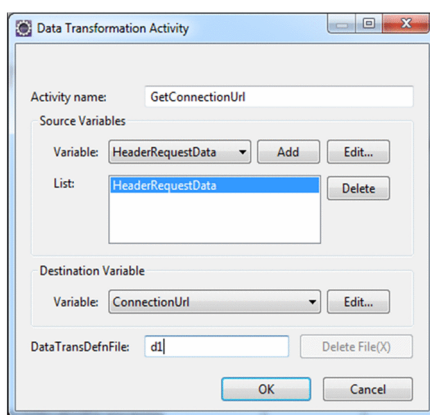


(a) Example of setting data transformation activities

The following figure shows an example of setting data transformation activities used in the business process shown in "Figure 2-24 Example of business process using connection destination information file (in SOAP adapter)":

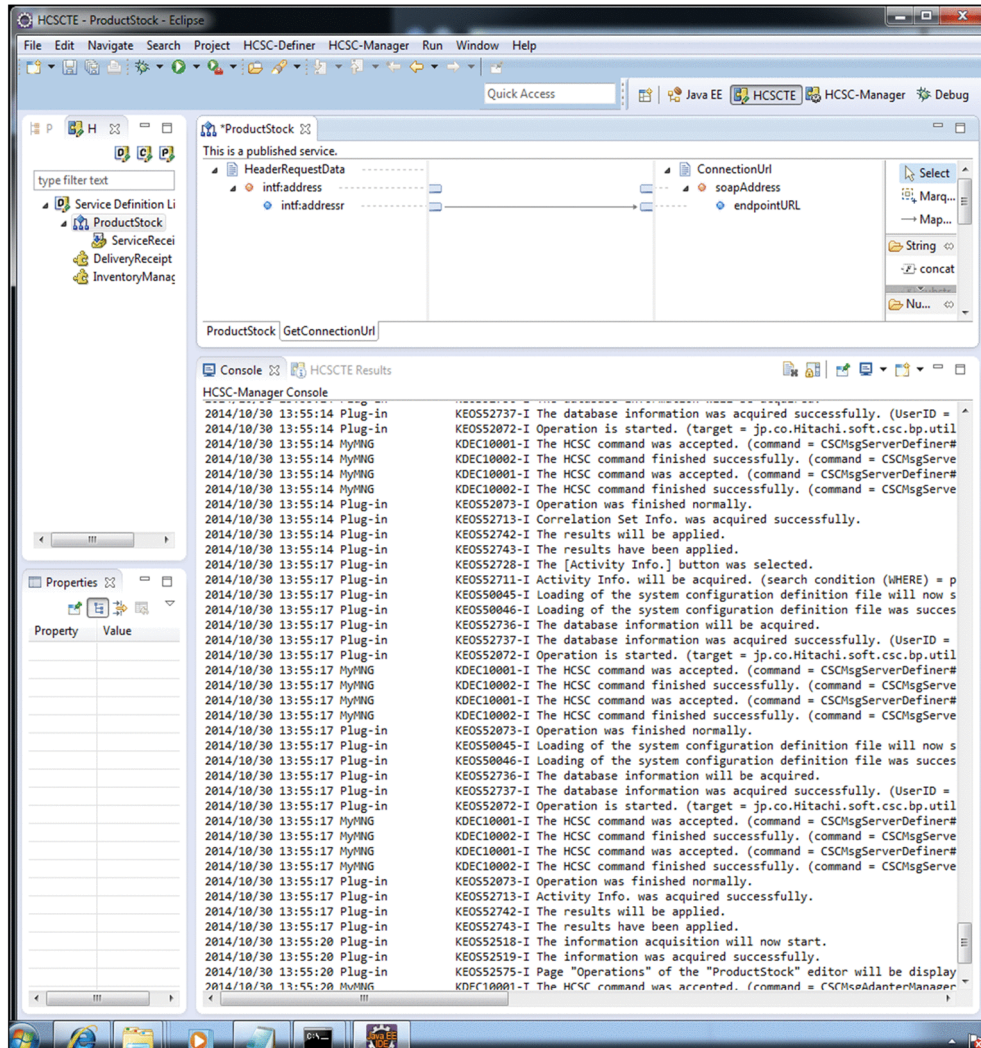
Here, "Header request data" is specified in the transformation source and "Connection destination URL" is specified in the transformation destination.

Figure 2–25: Example of setting data transformation activities (in SOAP adapter)



The following figure shows the definition of transformation mapping:

Figure 2-26: Example of setting transformation mapping (in SOAP adapter)



(b) Defining connection destination information file

Define URL for connecting to Web service in the connection destination information file (XML schema) used for performing setup in a service adapter.

Connection destination information file is provided in the service platform.

- **Storage location of connection destination information file**

<Installation directory of the service platform>/CSC/schema/connection/connection_soap_adapter.xsd

- ! **Important note**

Use the connection destination information file provided in the service platform to dynamically change connection destination information of SOAP adapter. Operation is not guaranteed if you do not use the provided connection destination information file.

- **Format of connection destination information file**

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
DO NOT EDIT THIS FILE.
-->
<xsd:schema
  elementFormDefault="qualified"
  targetNamespace="http://www.hitachi.co.jp/soft/xml/cosminexus/csc/connection/change/
  soap"
```

```

xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="soapAddress">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="endpointURL" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>

```

- **Element that can be set**

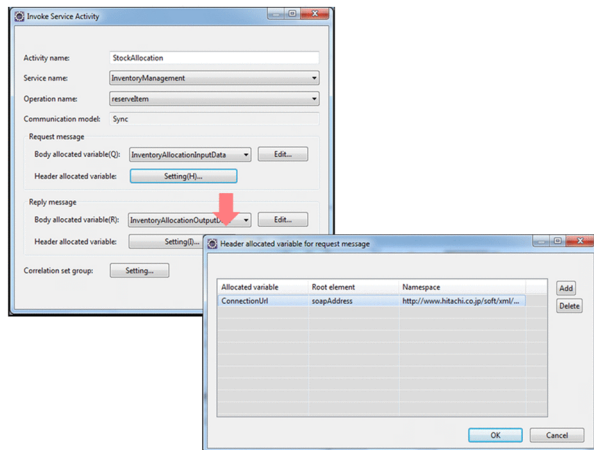
- endpointURL~<Character string>**

Specify connection destination URL of the service component.

(c) Setting connection destination information

The following figure shows an example of setting connection destination information used in the business process shown in "Figure2-24 Example of business process using connection destination information file (in SOAP adapter)". Set connection destination information in the header assigned variable of the invoke service activity. You can see the information set here in SOAP adapter.

Figure 2–27: Example of setting connection destination information (setting invoke service activity)



(4) Notes

- You cannot dynamically change possibility of use of basic authentication and user ID and password used in basic authentication by using the connection destination URL. Information related to basic authentication is enabled by information set from the development environment screen.
- Error message is output and abnormal termination occurs in the following cases:
 - If multiple connection destination information items in the development environment are set in the header assigned variable
If multiple system-specific names spaces and element names of XML data exist, error occurs because you cannot specify the connection destination.
 - If the set connection destination information is invalid
If there is an error in the connection destination URL, error occurs during service invocation.
 - If endpointURL element of XML data showing connection destination information does not exist
If endpointURL element does not exist, error occurs because you cannot specify the connection destination.

2.3 Connecting to services using Session Beans

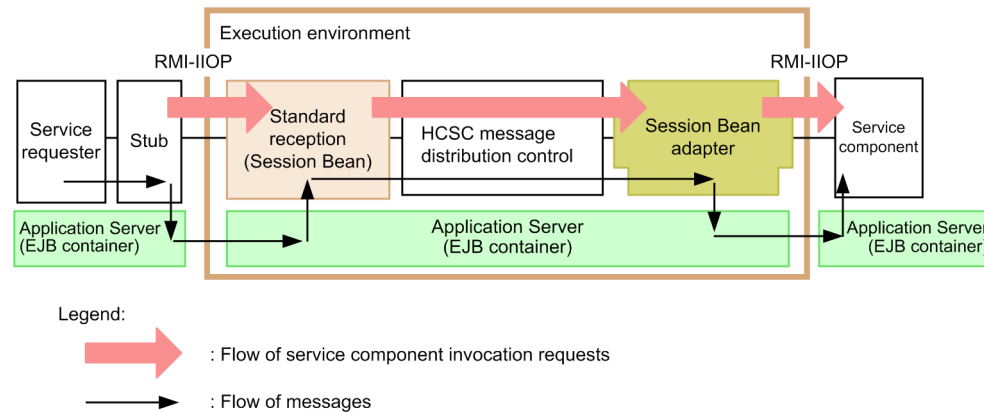
2.3.1 Relationship with Application Server (EJB containers)

The HCSC server in the execution environment operates with EJB containers of Application Server.

The request reception of the HCSC server is provided as a remote interface of EJB (Stateless Session Bean), and is connected with RMI-IIOP.

The service components created in the EJB (Stateless Session Beans or Stateful Session Bean) are executed from the service adapter of the HCSC server. The following figure shows the relationship with Application Server (EJB container).

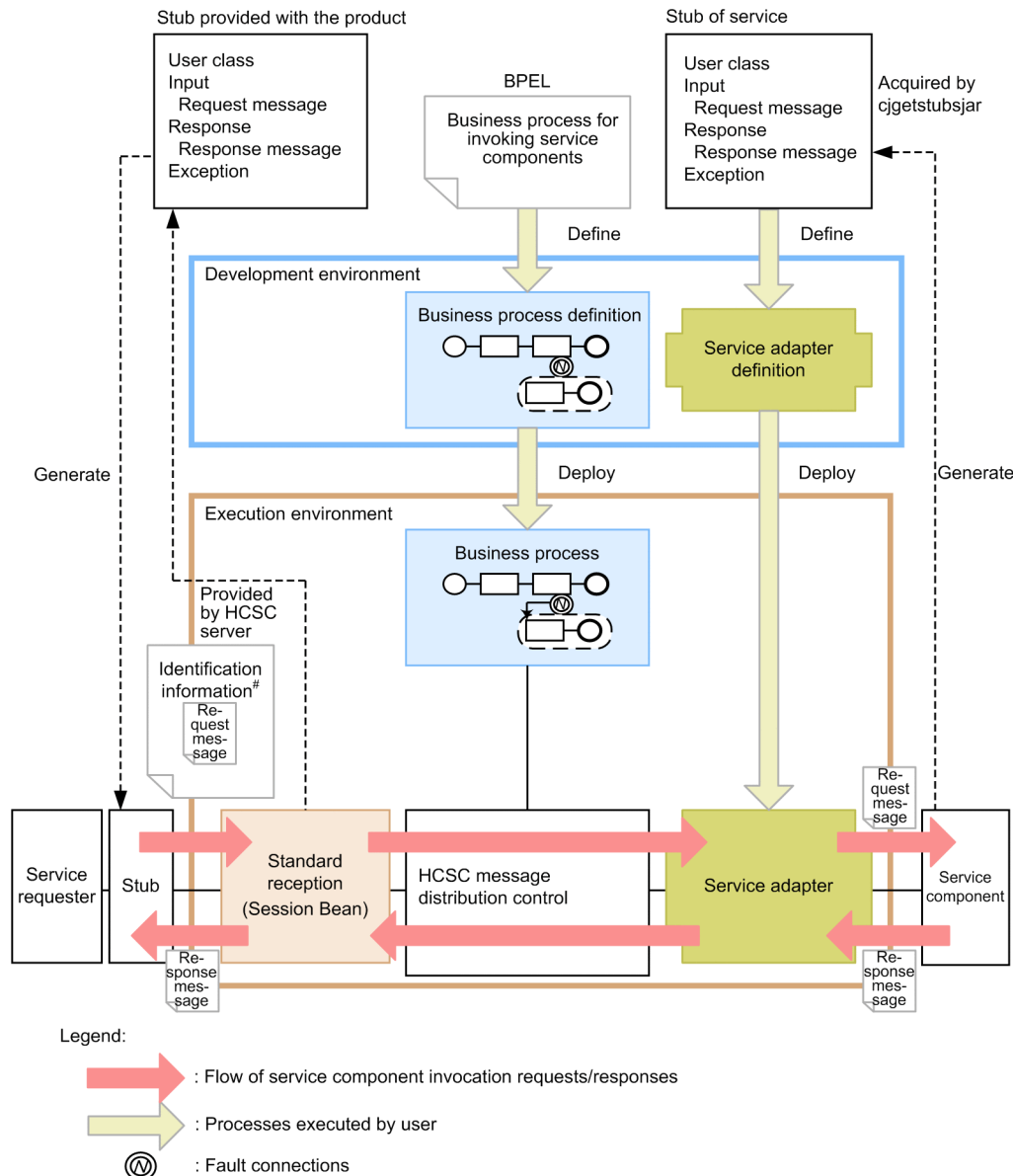
Figure 2–28: Relationship with Application Server (EJB container)



2.3.2 Relationship between user messages and stubs when a service component is invoked

The following figure shows the relationship between user messages and stubs when a Session Bean is used.

Figure 2–29: Relationship between user messages and stubs when a Session Bean is used



#: Identification information includes details such as service names and client correlation IDs.

When defining a service adapter, define the stub of the service component. (The stub of the service component includes the user class of the created service component, and the types of request message, response message, and exception are also defined.)

If you use Application Server with the J2EE server on the service component side, use the `cjgetstubsjar` command of the J2EE server to acquire the stub of the service component.

When invoking a service component from the service requester, use the stub provided with the product. The stub provided with the product is the interface of standard reception (Session Bean). When invoking standard reception (Session Bean) from the service requester, create in advance a request message matching the message format of the service component side. Then, specify the created message in the parameters of standard reception (Session Bean) (parameters of the user message), and then send the request.

A request message for invoking a business process from the service requester is the message format defined in the receive activity of the business process. In the service requester, create a message in the message format of the request message defined in the receive activity of the business process. Then, specify the created message in the parameters of standard reception (Session Bean) (parameters of the user message), and then send the request.

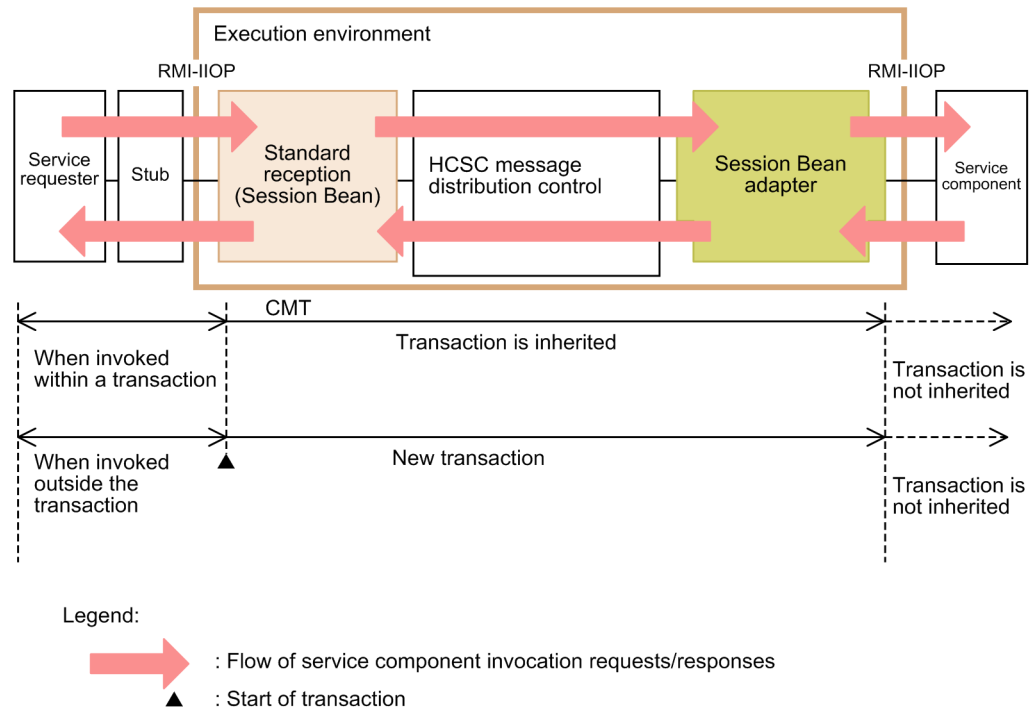
For the response to the service requester, use the message format of the response message defined in the service adapter.

For details about the `cjgetstubsjar` command, see *cjgetstubsjar (Obtaining the RMI-IIOP stub and interface of applications)* in the *Application Server Command Reference Guide*.

2.3.3 Transactions when using Session Bean

The following figure shows the transactions when a Session Bean is used.

Figure 2–30: Transactions when a Session Bean is used

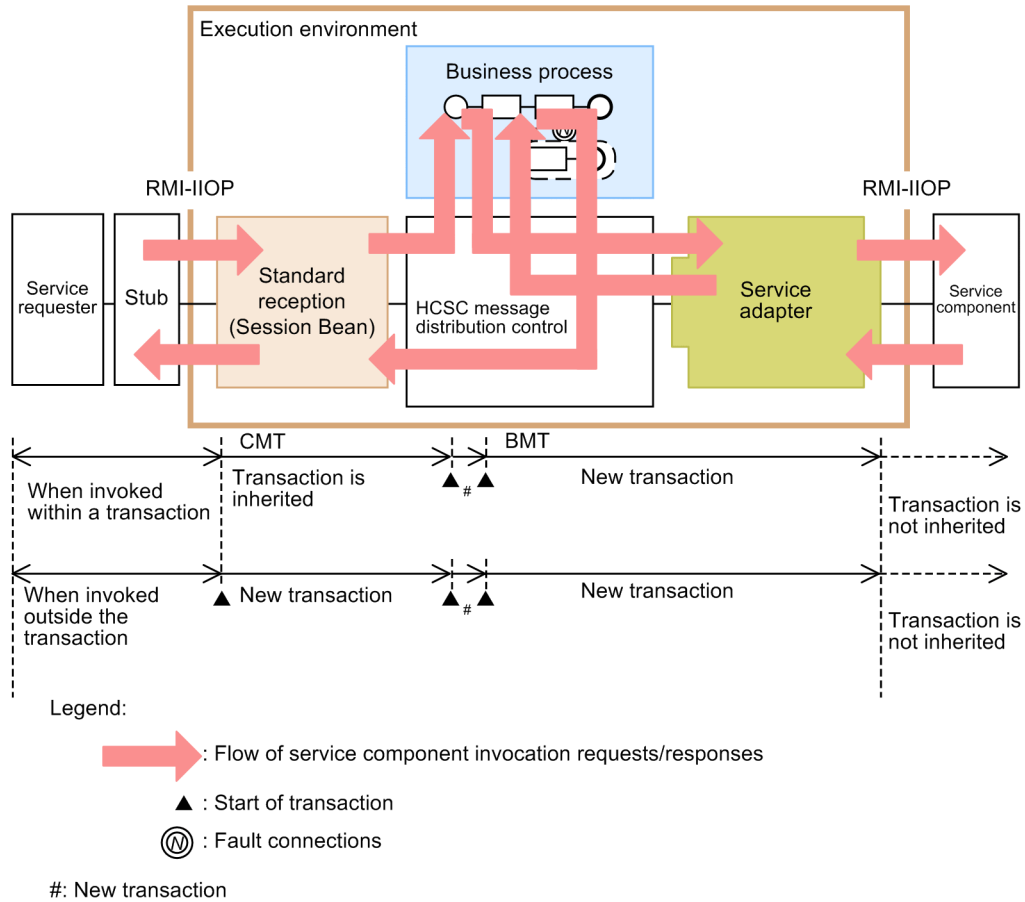


When you use a Session Bean, operations are performed within the HCSC server with a CMT (Container Managed Transaction). Also, the transaction attribute of standard reception (Session Bean) is `Required`. Therefore, if the service requester invokes standard reception (Session Bean) of the HCSC server within the transaction scope, the transaction context is inherited on the HCSC server. Then, the process within the HCSC server enters the transaction scope at the service requester side. On the contrary, if the service requester invokes standard reception (Session Bean) of the HCSC server outside the transaction scope, a new transaction starts on the HCSC server.

Note that the transaction attribute of the service component must be `RequiresNew` or `NotSupported`. Therefore, the transaction is not inherited in the service component. For details about whether you can specify the transaction attribute of service components using a Session Bean, see 2.6.2 *Applicable scope of service components using a Session Bean* in the *Service Platform Basic Development Guide*.

A business process operates by using a BMT (Bean Managed Transaction). The business process itself is controlled by using a new transaction. If you invoke a business process, each service component invocation process defined in the invoke service activity of the business process is controlled by using a new transaction. Therefore, even when the service requester invokes standard reception (Session Bean) of the HCSC server within the transaction scope, the transaction is not inherited up to the service adapter. The following figure shows the transactions passing through the business process.

Figure 2–31: Transactions passing through the business process



For persistent business processes, see the description of the transactions during execution of business processes in 3.4 *Transaction of a business process*.

2.3.4 Points to be considered when using SessionBean adapter

If you restart SessionBean service in an environment in which naming caching is enabled (ON), when SessionBean service is first invoked after restart, KDECO3002-E message might be output and an error might occur. Take the following actions to avoid this error:

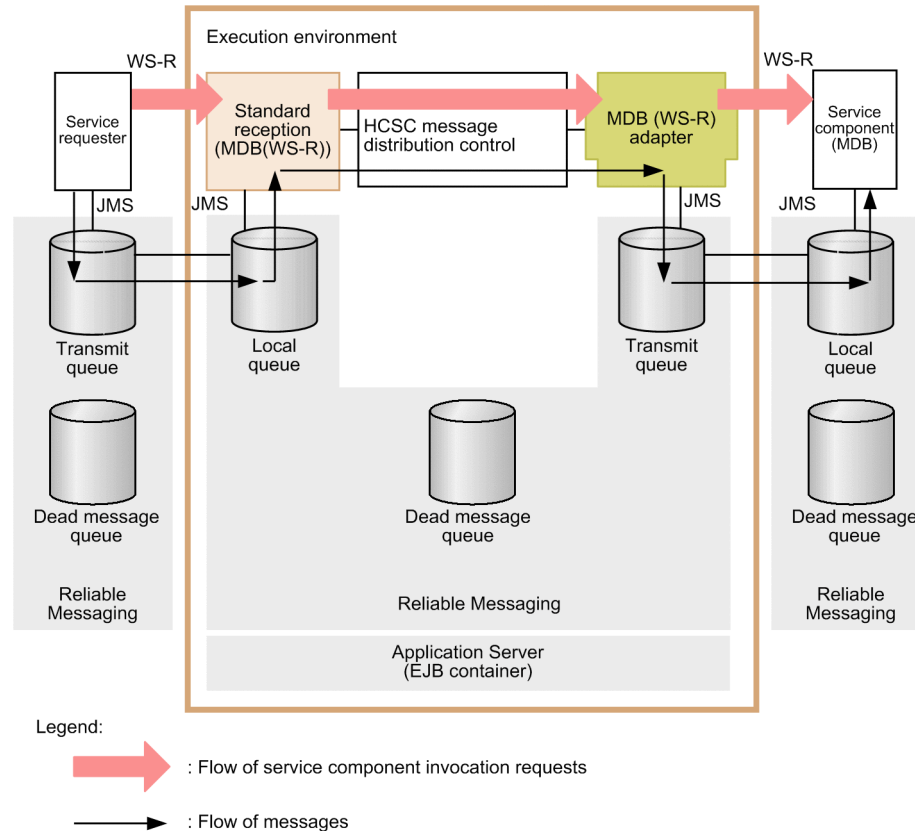
- In reference system: Enable the reconnect to EJB home object functionality.
- In other than reference systems: Restart HCSC server.

2.4 Connecting to services Using WS-R (WS-Reliability)

2.4.1 Relationship with Reliable Messaging

Use Reliable Messaging to invoke an asynchronous service component. In addition, use WS-R (WS-Reliability) between a transmit queue of the service requester and a local queue of the HCSC server, and between a transmit queue of the HCSC server and a local queue of the service component. Reliable Messaging operates on the EJB container of Application Server. The following figure shows the relationship with Reliable Messaging.

Figure 2-32: Relationship with Reliable Messaging



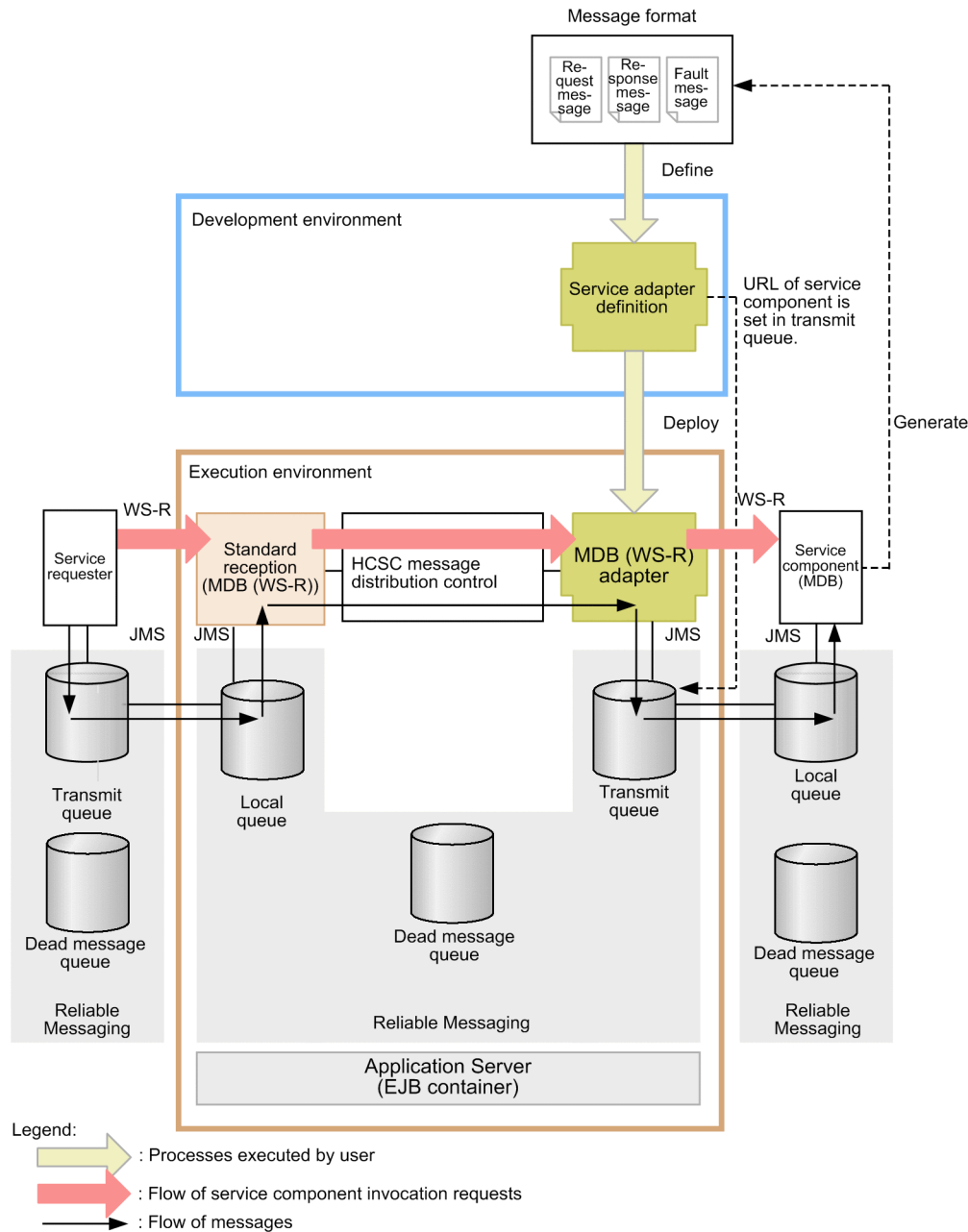
2.4.2 Flow of messages when invoking service components

Standard reception (MDB (WS-R)) of an HCSC server operates by using an MDB (Message Driven Bean) and monitors the local queue of the HCSC server side. When a message is sent to the local queue of the HCSC server side, standard reception (MDB (WS-R)) extracts the message and invokes the service component.

In the transmit queue of the service requester side, specify the destination address of the HCSC server and the queue name of the local queue of the transmit destination. From the service requester, send a JMS message to the transmit queue of the service requester side to invoke standard reception (MDB (WS-R)) of the HCSC server.

When invoking a service component, send the message to the transmit queue of the service adapter side. The transmit queue then sends the message to the local queue of the service component side defined in the Service Adapter Settings window in the development environment. (In this window, the transmit destination queue name is specified for the address in the Basic window and the destination URL is specified in the Details window.) By creating the service component by using an MDB, the process can be executed by extracting the message sent to the local queue of the service component side. The following figure shows the specification of the destination for an asynchronous service component (MDB (WS-R)).

Figure 2–33: Specification of the destination for an asynchronous service component (MDB (WS-R))

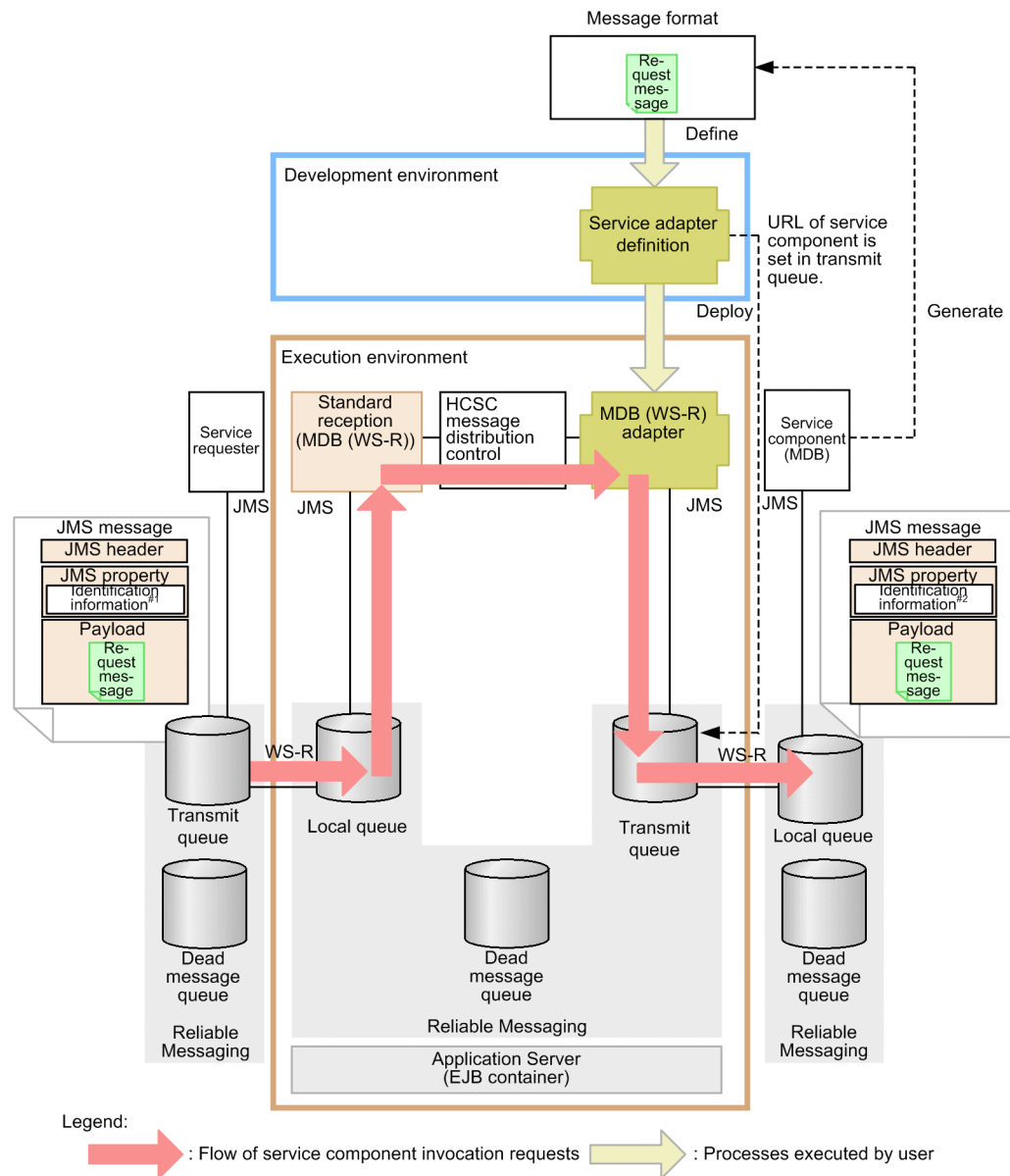


For details about how to create a transmit queue, see 8.5.2 *Creating a transmit queue* in the *Service Platform Basic Development Guide*. For details about how to create a JMS message, see 8.5.3 *Creating a JMS message* in the *Service Platform Basic Development Guide*.

When a service component is invoked from the service requester, the format of the JMS message to be sent to the transmit queue is the interface of standard reception (MDB (WS-R)).

The contents specified in the payload (the main data excluding the header) of the JMS message are transmitted to the service component as the user message. Therefore, when invoking standard reception (MDB (WS-R)) from the service requester, create a request message matching the message format of the service component side in advance, and then send that message set in the payload. The following figure shows the relationship between user messages and JMS messages when an asynchronous service component is invoked.

Figure 2-34: Relationship between user messages and JMS messages when an asynchronous service component is invoked

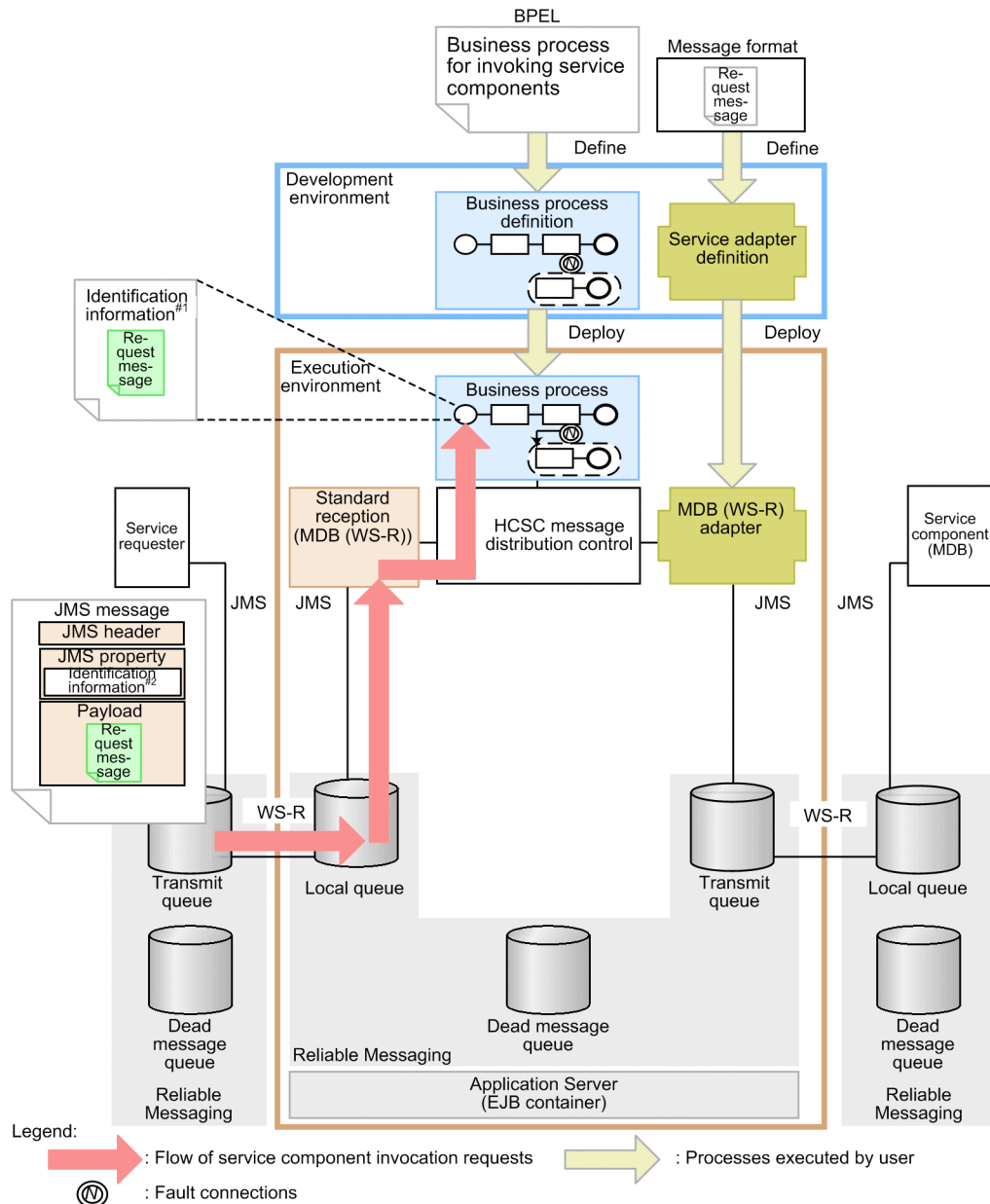


#1: Identification information includes service names, client correlation IDs, and user-defined properties.

#2: Identification information includes user-defined properties.

A request message for invoking a business process from the service requester is the message format defined in the receive activity of the business process. In the service requester, create a message in the message format of the request message defined in the receive activity of the business process, and then send the message set in the payload. The following figure shows the relationship between user messages and JMS messages when a business process is invoked.

Figure 2-35: Relationship between user messages and JMS messages when a business process is invoked



2.4.3 Handling response messages when a synchronous service component is invoked

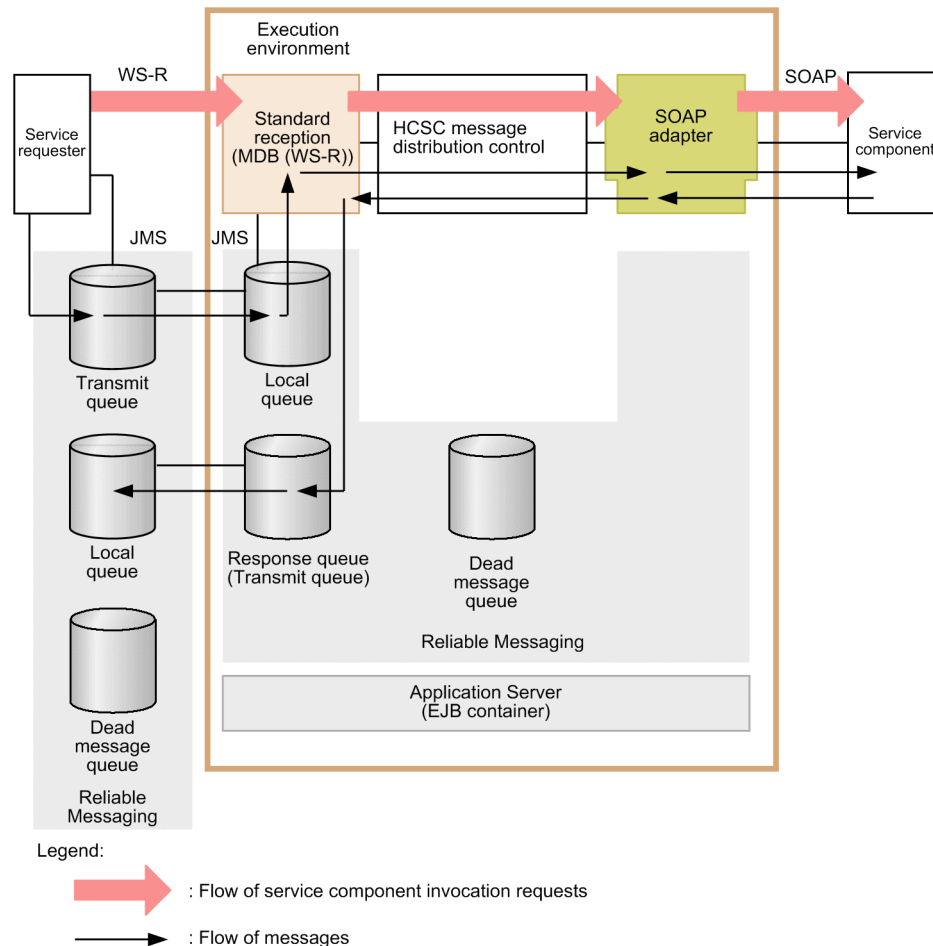
Standard reception (MDB (WS-R)) is provided for asynchronous request messages and is basically used when no response is sent from the service component. However, if a synchronous service component (such as Web Service or Session Bean) or business process is requested from asynchronous (MDB (WS-R)) standard reception, a response might be returned from the synchronous service component. Therefore, the HCSC server provides a way to send the response to a response queue.

To use the response queue, create a transmit queue (response queue) at the HCSC server side when setting up the environment of the HCSC server. Then, for the transmit queue, specify the destination address of the Web application used for inter-queue transfer on the service requester machine and the transmit destination queue name.

When sending a request for invoking a service component from the service requester to standard reception (MDB (WS-R)), specify the name of the created transmit queue (response queue) in the parameters of standard reception (MDB (WS-R)). This specification causes a JMS message to be sent to the specified queue when a response is returned from the service component.

The JMS message sent to the transmit queue (response queue) is transmitted to the local queue of the service requester side. Therefore, the service requester can acquire the response message from the service component by extracting the message from the local queue. The following figure shows how a response is handled when a synchronous service component is invoked.

Figure 2-36: Response handling when a synchronous service component is invoked



For details about how to extract responses, see 8.5.8 *Extracting responses* in the *Service Platform Basic Development Guide*.

Even for a synchronous service component, whether a message is to be sent to the response queue differs depending on whether a response exists, as shown in the table below.

Note that even if the response queue name is specified by the service requester, messages cannot be sent to the response queue if a queue with the specified name is not created at the HCSC server side. In this case, an error occurs during the service component invocation process.

Table 2-1: Cases when messages are output to the response queue

No.	Cases when messages are output to the response queue	Existence of a response message			
		A response message exists	No response message exists	A user-defined exception occurs from the service component	An error other than a user-defined exception occurs
1	If the invoked service component is a synchronous service component (such as Web Service or Session Bean)	A response message set in the payload is sent.	A message without a payload is sent.	A response message set in the payload is sent.	N [#]
2	If the invoked service component is an asynchronous service component (such as an MDB (WS-R) and MDB (DB queue))	N	N	--	N [#]
3	If the invoked service component is a business process	A response message set in the payload is sent.	A message without a payload is sent.	A response message set in the payload is sent.	N [#]

Legend:

N: No message is sent.

--: Not applicable because a user-defined exception is not generated from an asynchronous service component.

#

Error information is output to the log and trace.

If the response queue name is not specified in the request from the service requester, the contents of the response message will be discarded (when the response message is discarded, the message KDEC00031-W is output).

To set up a correlation on the service requester side, the JMS property specified by the service requester is inherited. For details about the inherited header and properties, see *8.5.8 Extracting responses* in the *Service Platform Basic Development Guide*.

2.4.4 Transactions when using an MDB (WS-R)

For an MDB (WS-R), different transactions are used between the service requester and the HCSC server, and between service adapters and service components.

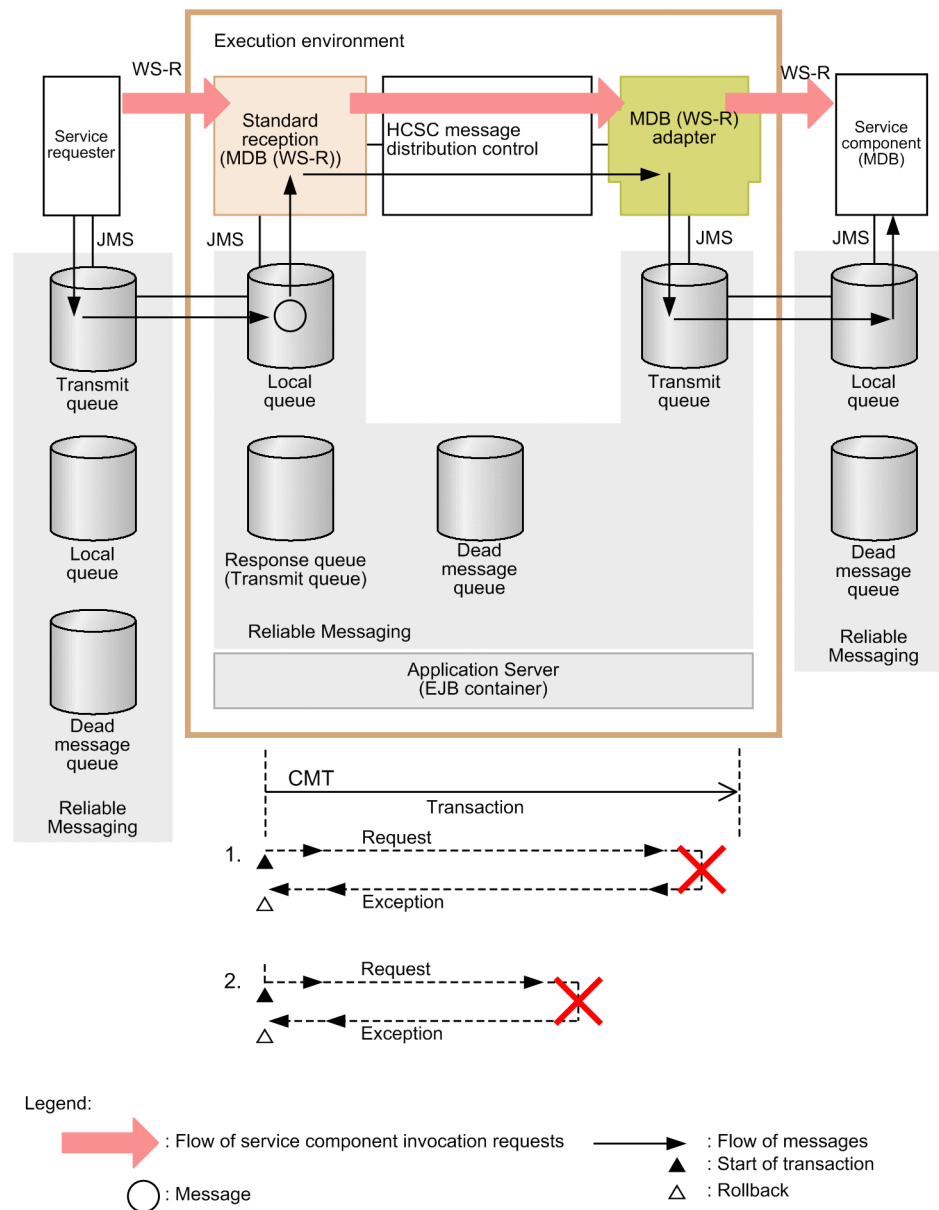
(1) When sending a message to an asynchronous service adapter

When asynchronous standard reception (MDB (WS-R)) receives a service component invocation request and then sends a message to an asynchronous MDB (WS-R) adapter, the following operations are processed in one transaction: Extraction of the message from the local queue by standard reception (MDB (WS-R)), and transmission of the message to the transmit queue by the service adapter.

If an error occurs, such as an attempt to send a message to the transmit queue fails or the specified service adapter is not running, the transaction rolls back.

When the transaction rolls back, the message extracted by standard reception (MDB (WS-R)) is recovered, and then the service component is invoked after the message is re-extracted from the local queue. The following figure shows a transaction in which the process starting from reception until service component invocation is asynchronous.

Figure 2-37: Transaction in which the process starting from reception until service component invocation is asynchronous



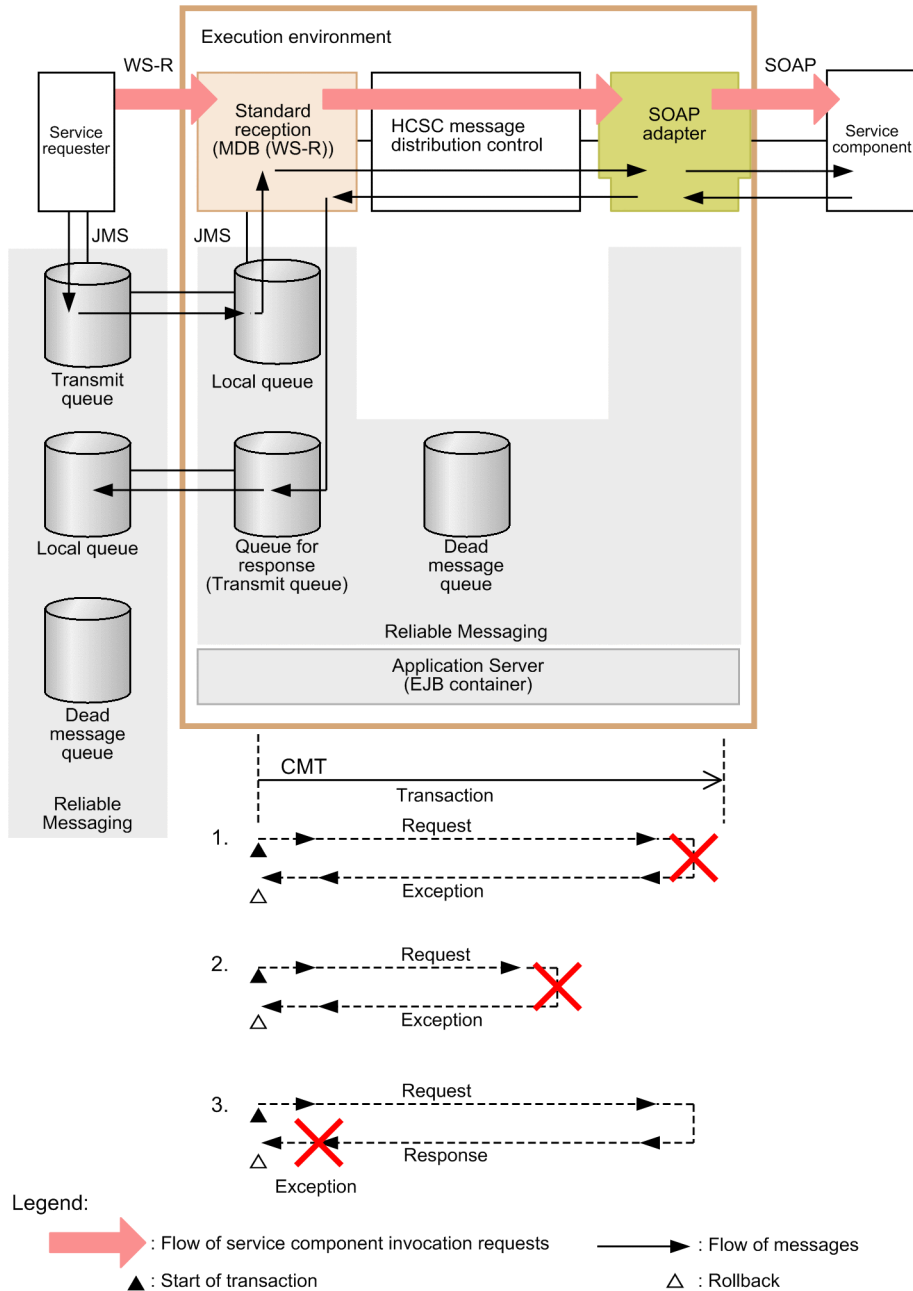
For details about actions to be taken if an error occurs, see *7.7.3 Troubleshooting during the execution of an MDB (WS-R) in the Service Platform Setup and Operation Guide*.

(2) When sending a message to a synchronous service adapter

When asynchronous (MDB (WS-R)) standard reception receives a service component invocation request and then sends a message to a synchronous service adapter, the following operations are processed in one transaction: Extraction of the message from the local queue by standard reception (MDB (WS-R)), invocation of the service component by the service adapter, and transmission of the message to the response queue after a response is returned.

If an error occurs, such as an attempt to send a message to the transmit queue or response queue fails or the specified service adapter is not running, the transaction rolls back. When the transaction rolls back, the message extracted by standard reception (MDB (WS-R)) is recovered, and then the service component is invoked after the message is re-extracted from the local queue. The following figure shows a transaction in which the reception process is asynchronous but the service component is synchronous.

Figure 2–38: Transaction in which the reception process is asynchronous but the service component is synchronous



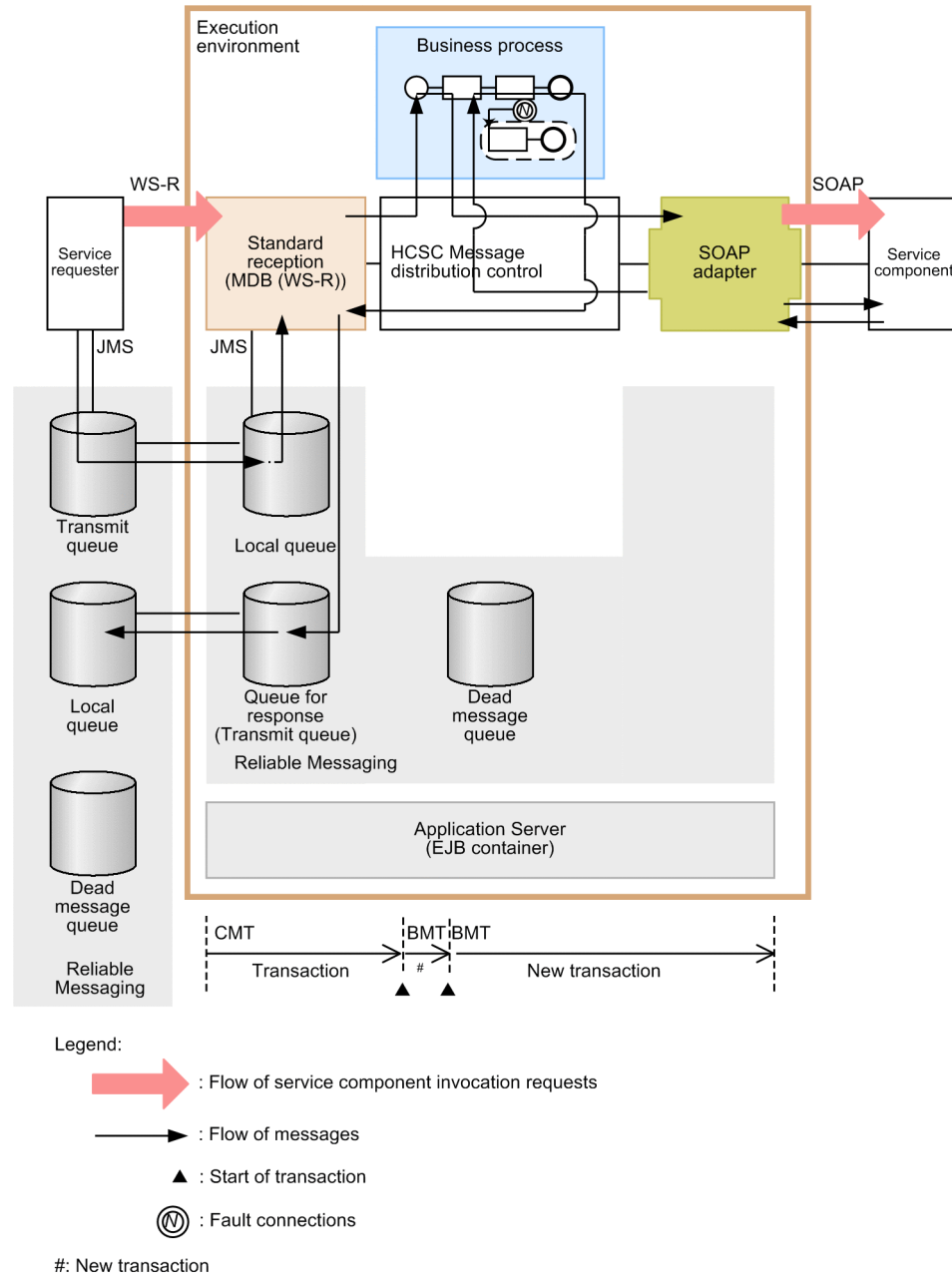
Operations performed if a transaction fails are described below. The numbers correspond to the numbers in the figure.

1. If an attempt to send a message to the transmit queue fails
If an attempt to invoke a service component from the service adapter using SOAP fails, the message is returned to the local queue and then re-extracted by standard reception (MDB (WS-R)).
2. If the specified service adapter is not running
If the specified service adapter is not running, the message is returned to the local queue and then re-extracted by standard reception (MDB (WS-R)).
3. If an attempt to send a message to the response queue fails
If a response is returned and an attempt to send a message to the response queue fails, the message is returned to the local queue and then re-extracted by standard reception (MDB (WS-R)).

For details about actions to be taken if an error occurs, see 7.7.3 *Troubleshooting during the execution of an MDB (WS-R)* in the *Service Platform Setup and Operation Guide*.

A business process operates by using a BMT (Bean Managed Transaction). The business process itself is controlled by using a new transaction. When you invoke a business process, each service component invocation process defined in the invoke service activity of the business process is controlled by using a new transaction. The following figure shows the transactions passing through the business process.

Figure 2-39: Transactions passing through the business process



For persistent business processes, see the description of the transactions during execution of business processes in 3.4 *Transaction of a business process*.

2.4.5 Dead message queue

If invocation of a service component rolls back on the machine running the HCSC server, and the rollback count (message delivery frequency) reaches the maximum, the message is moved to the dead message queue. The message is also moved to the dead message queue if no limit is set for the delivery frequency.

For the mechanism of transiting to the dead message queue when an error occurs, see 7.7.3 *Troubleshooting during the execution of an MDB (WS-R)* in the *Service Platform Setup and Operation Guide*.

2.4.6 Names and number of queues used in an MDB (WS-R)

The queues described in the following table are used in the execution environment of an HCSC server.

Table 2–2: Names and number of queues used in an MDB (WS-R)

No.	Use	Description	Queue type	Queue type	Number of queues	Remarks
1	For asynchronous (MDB (WS-R)) standard reception	Queue used as the destination of the transmit queue of the service requester side	Local queue	<ul style="list-style-type: none"> In the case of a single configuration and load balance cluster <i>CSC</i><i>HCSC-server-name</i><i>ACPT_RCVQ</i> For an HA cluster <i>CSC</i><i>cluster-name</i><i>ACPT_RCVQ</i> 	1	--
2	For MDB (WS-R) adapters	Queue that is transmitted to the local queue of the service component side	Transmit queue	<i>CSC_cluster-name adapter-name</i>	Number of asynchronous MDB (WS-R) adapters	--
3	Dead message queue	Shared with the dead message queue for an MDB (DB queue)	Local queue or transmit queue	Any name	One per system	Same as the dead message queue in a database queue
4	Response queue	Used when receiving a response of service invocation	Local queue or transmit queue	Any name	Any number	--

Legend:

--: Not applicable

2.5 Connecting to services by using a database queue

2.5.1 Relationship with Reliable Messaging

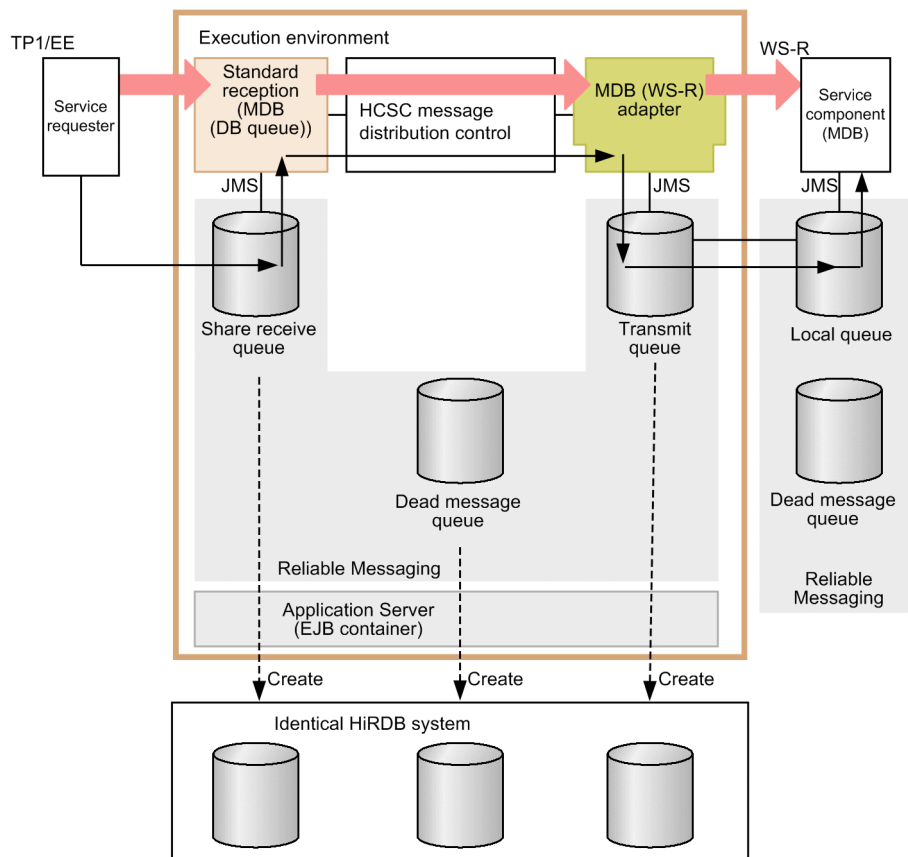
You can use Reliable Messaging, which is usually used for integrating with TP1/EE, to invoke an asynchronous service component by using a database queue. (The shared receive queue and shared send queue are referred to as *database queue* at the TP1/EE side.)

The shared receive queue is used for the interface with the TP1/EE system of the service requester side, and the shared send queue is used for the interface with the TP1/EE system of the service component side.

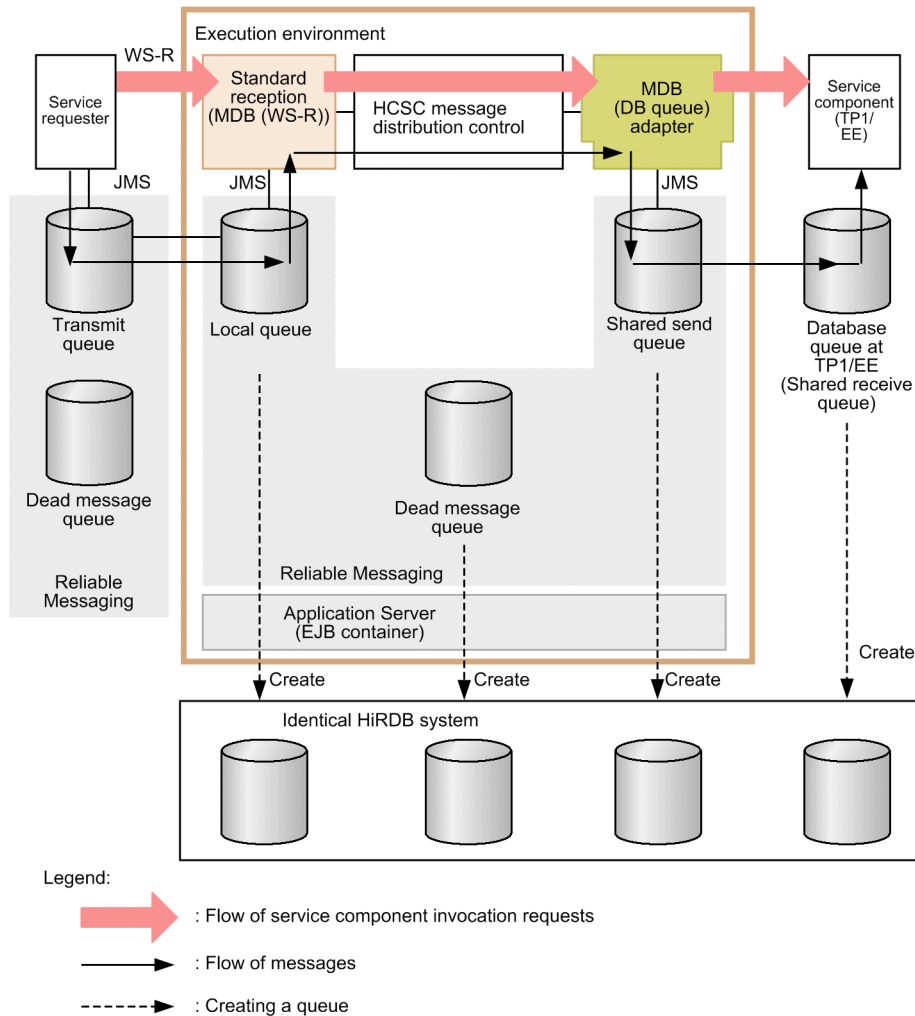
Note that you cannot invoke an asynchronous MDB (DB queue) adapter from asynchronous (MDB (DB queue)) standard reception. You must always use a combination of request reception and service adapter of different protocols. The following figure shows the relationship with Reliable Messaging.

Figure 2–40: Relationship with Reliable Messaging

- When service requester is TP1/EE:



- When the service component is TP1/EE:



! Important note

Note that different queues are not created in the execution environment and HiRDB. Because the queues of a database are created in the same system, the queues in the execution environment and the queues of HiRDB are the same. The database queues of the TP1/EE side are also the same.

2.5.2 Flow of messages when invoking service components

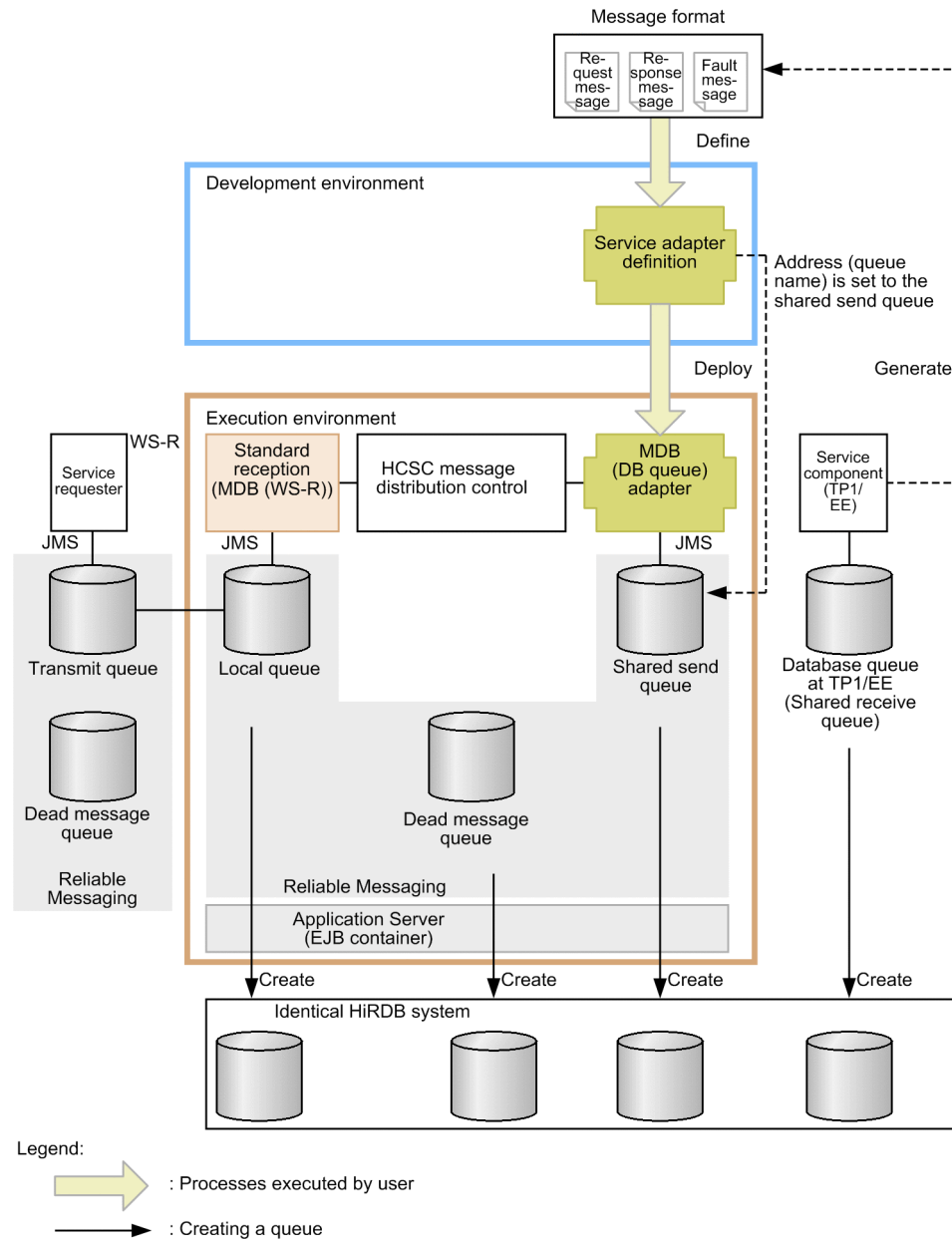
A request reception of an HCSC server operates by using an MDB (Message Driven Bean) and monitors the shared receive queue of the HCSC server side. When a message is sent to the shared receive queue of the HCSC server side, the message is extracted and then a service component is invoked.

The service requester of the TP1/EE side sends a message by directly accessing the database in which the shared receive queue exists.

For the shared send queue of the service requester side, specify the destination queue name in the Service Adapter Settings window in the development environment.

Use the commands of TP1/EE to create the database queue (shared receive queue) of the TP1/EE side to be used as the destination. Note that the database queue of the TP1/EE side must be created in the same database (the same HiRDB system) in which the shared send queue exists. The following figure shows the specification of the destination for an asynchronous (MDB (DB queue)) service component.

Figure 2-41: Specification of the destination for an asynchronous (MDB (DB queue)) service component

**Important note**

Note that different queues are not created in the execution environment and HiRDB. Because the queues of a database are created in the same system, the queues in the execution environment and the queues of HiRDB are the same. The database queues of the TP1/EE side are also the same.

For details about creating a database queue, see the description about the data queue in the *TP1/Server Base Enterprise Option Usage Guide*.

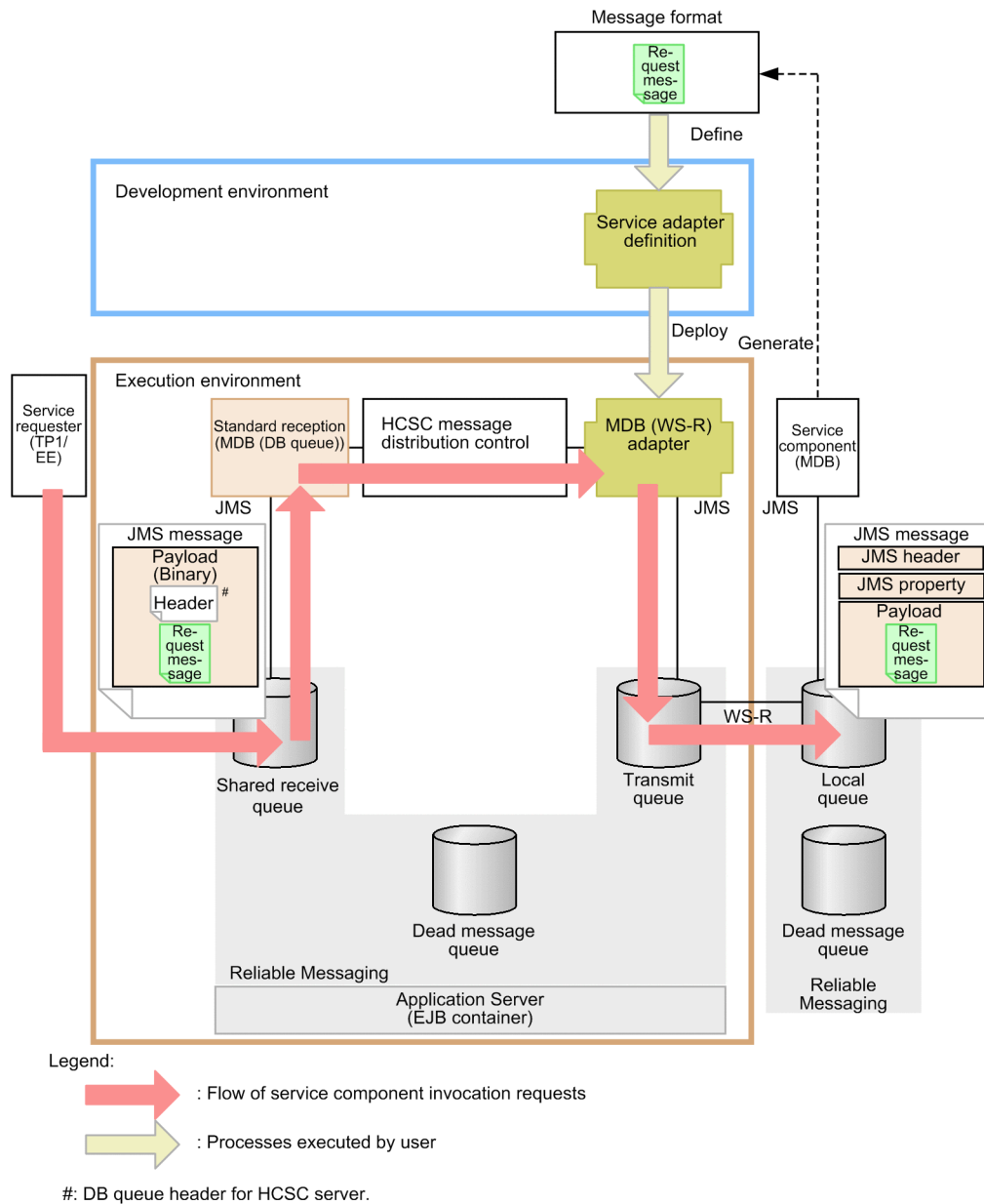
(1) When a service component is invoked from TP1/EE

When a service component is invoked from the service requester, the format of the message to be sent to the shared receive queue is `BytesMessage`.

The database queue header for an HCSC server is created in binary format by specifying parameters such as the service name to be invoked and client correlation ID. Then, the request message (binary format) is set up. This allows the contents of the request message (binary format) to be transmitted to the service component as the user message.

When invoking standard reception (MDB (DB queue)) from the service requester, create in advance a request message matching the message format of the service component side, and then send that message set in the payload. The following figure shows the relationship between the user message and message format when a service component is invoked from TP1/EE.

Figure 2–42: Relationship between the user message and message format when a service component is invoked from TP1/EE



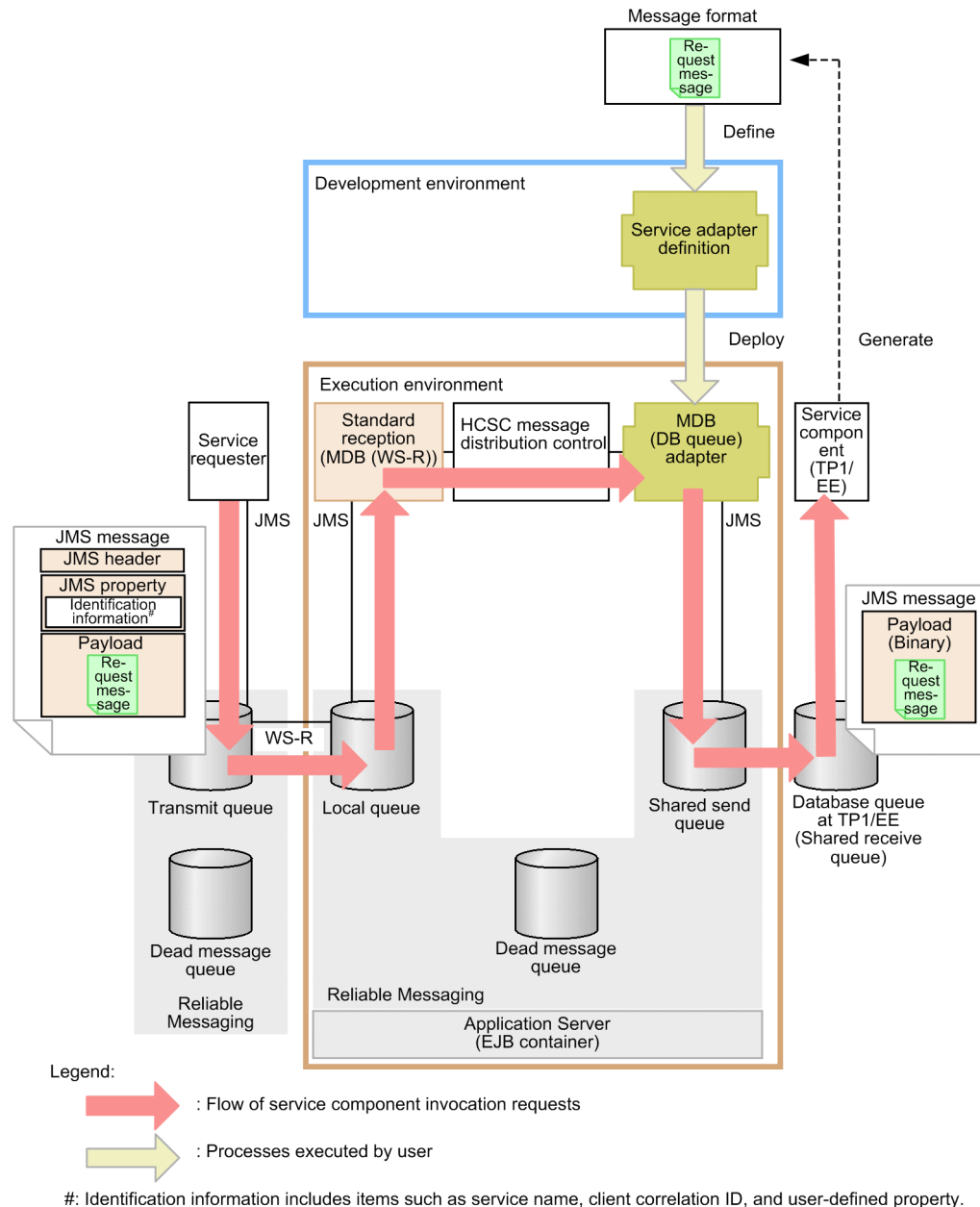
For details about how to create binary data, see 8.6.4 *Creating binary data (TP1/EE and JMS)* in the *Service Platform Basic Development Guide*.

(2) When a service component of TP1/EE is invoked

When a service component is invoked from the service adapter, the format of the message to be sent to the shared send queue is `BytesMessage`.

A request message matching the message format defined in the service adapter must be specified as a binary message. The following figure shows the relationship between the user message and message format when a service component of TP1/EE is invoked.

Figure 2-43: Relationship between the user message and message format when a service component of TP1/EE is invoked



2.5.3 Handling response messages when a synchronous service component is invoked

Asynchronous (MDB (DB queue)) standard reception is provided for asynchronous request messages and is basically used when no response is sent from the service component. However, if a synchronous service component (such as Web Services or Session Bean) or business process is requested from asynchronous (MDB (DB queue)) standard reception, a response might be returned from the synchronous service component.

Therefore, the HCSC server provides a way to send the response to a response queue.

To use the response queue, create a shared send queue at the HCSC server side when setting up the environment of the HCSC server, and specify the send destination queue name for the shared send queue.

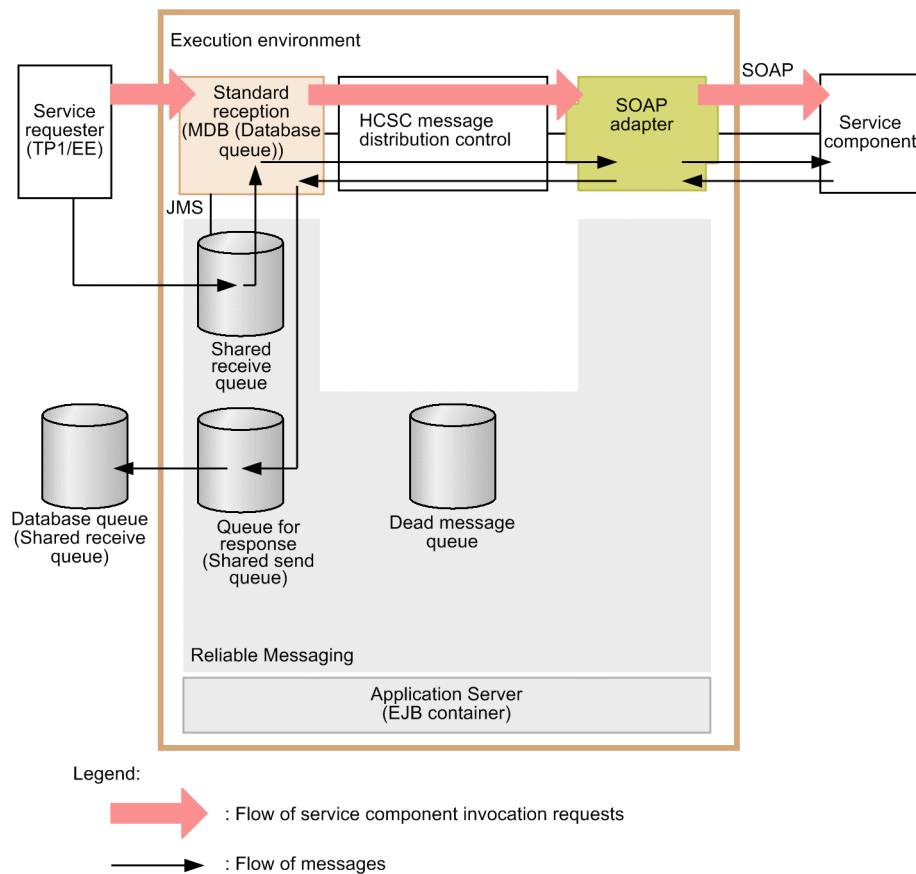
When sending a request for invoking a service component from the service requester to standard reception (MDB (DB queue)), specify the name of the shared send queue in the parameters of standard reception (MDB (DB queue)). This specification causes a JMS message to be sent to the specified queue when a response is returned from the service component.

The message is transmitted from the shared send queue to the database queue (shared receive queue) of the service requester side. Therefore, you can acquire the response message from the service component by extracting the message from the database queue (shared receive queue).

For details about how to extract responses, see *8.6.10 Extracting responses (TP1/EE and JMS)* in the *Service Platform Basic Development Guide*.

The following figure shows how a response is handled when a synchronous service component is invoked.

Figure 2-44: Response handling when a synchronous service component is invoked



Messages are output to the response queue in the cases shown in the table below.

Even for a synchronous service component, messages are sent to the response queue in the format shown below depending on whether a response exists.

Note that even if the response queue name is specified by the service requester, messages cannot be sent to the response queue if a queue with the specified name is not created at the HCSC server side. In this case, an error occurs during the service component invocation process.

Table 2–3: Cases when messages are output in the response queue

No.	Cases when messages are output in the response queue	Existence of a response message			
		A response message exists	No response message exists	A user-defined exception occurs from the service component	An error other than a user-defined exception occurs
1	If the invoked service component is a synchronous service component (such as Web Service, Session Bean, or custom adapter)	A response message set in the payload is sent.	A message without the payload is sent.	A response message set in the payload is sent.	N [#]
2	If the invoked service component is an asynchronous service component (such as an MDB (WS-R) and MDB (DB queue))	N	N	--	N [#]
3	If the invoked service component is a business process	A response message set in the payload is sent.	A message without the payload is sent.	A response message set in the payload is sent.	N [#]

Legend:

N: No message is sent.

--: Not applicable because a user-defined exception is not generated from an asynchronous service component.

#

Error information is output to the log and trace.

If the response queue name is not specified in the request from the service requester, the contents of the response message will be discarded. (The message KDECC00031-W indicating that the response message was discarded is output.)

Unlike the case of an MDB (WS-R), the JMS properties cannot be inherited. To set up a correlation for service component invocation, you need to embed the information showing the correlation within the user message.

2.5.4 Transactions when using an MDB (DB queue)

For an MDB (DB queue), different transactions are used between the service requester and the HCSC server, and between the service adapter and a service component.

Extraction of the message from the local queue by standard reception (MDB (DB queue)), and transmission of the message to the transmit queue by the service adapter are processed in one transaction. The transaction process is the same as that for an MDB (WS-R).

For persistent business processes, see the description of the transactions during execution of business processes in 3.4 *Transaction of a business process*.

2.5.5 Dead message queue

If invocation of a service component rolls back on the machine running the HCSC server, and the rollback count (message delivery frequency) reaches the maximum, the message is moved to the dead message queue. The message is also moved to the dead message queue if no limit is set for the delivery frequency.

For the mechanism of transiting to the dead message queue when an error occurs, see 7.7.4 *Troubleshooting when executing MDB (DB queue)* in the *Service Platform Setup and Operation Guide*.

2.5.6 Names and number of queues used in an MDB (DB queue)

The queue names and the number of queues described in the table below are used in the execution environment of an HCSC server. The following table shows the names and number of queues used in a database (database queue).

Table 2–4: Names and number of queues used in a database (database queue)

No.	Use	Description	Queue type	Queue type	Number of queues	Remarks
1	For asynchronous (MDB (DB queue)) standard reception	Queue that receives requests from the TP1/EE system of the service requester side	Shared receive queue	<ul style="list-style-type: none"> In the case of a single configuration and load balance cluster <i>CSCHCSC-server-name</i>ACPT_DBQ For an HA cluster <i>CSCcluster-name</i>ACPT_DBQ 	1	--
2	For MDB (DB queue) adapters	Queue that is transmitted to the database queue of the service component side	Shared send queue	<i>CSC_cluster-name adapter-name</i>	Number of asynchronous MDB (DB queue) adapters	--
3	Dead message queue	Shared with the dead message queue for an MDB (WS-R)	Local queue or transmit queue	Any name	One per system	Same as the dead message queue in WS-R
4	Response queue	Used when receiving a response of service component invocation	Shared send queue	Any name	Any number	--

Legend:

--: Not applicable

2.6 Connecting to a database

When invoking a database as a service component, you use a DB adapter. This section describes the settings for accessing a database when the DB adapter is used.

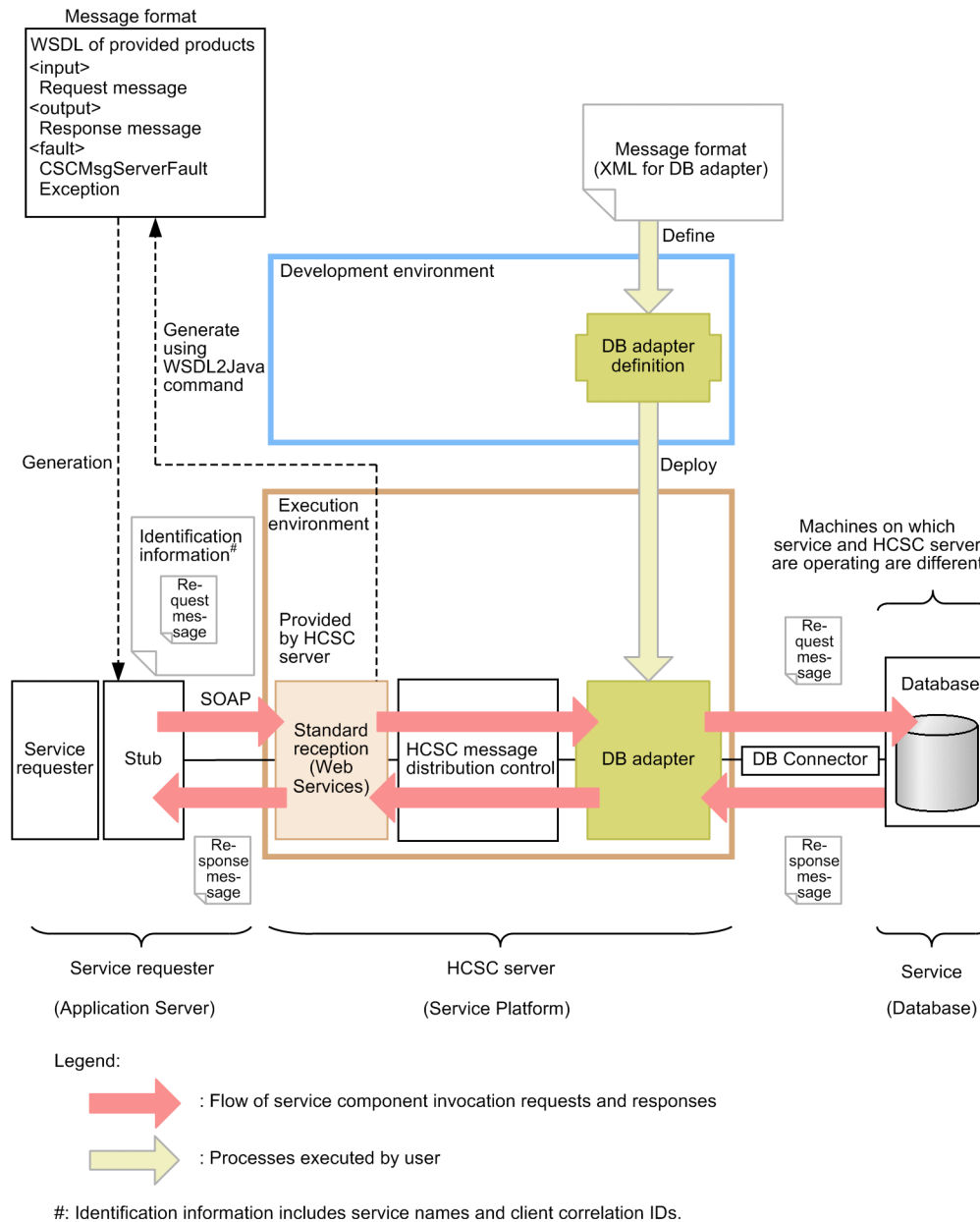
2.6.1 Accessing a database using DB adapters

A DB adapter enables direct access to a database when receiving requests from a service requester and a business process. Database access follows the SQL statement defined in the database adapter.

(1) Executing SQL statements by using database adapters

You can define the DB adapter in the development environment. In the service requester that sends a request to the DB adapter, create an XML document writing the SQL statement to be executed. Then, specify the created XML message in the request message, and then execute the process. The following figure shows the execution of an SQL by using a DB adapter.

Figure 2-45: Executing an SQL by using a DB adapter



For details about how to define the DB adapter and how to create a service requester that sends requests to the DB adapter, see 3.3.5 *Defining a database adapter* in the *Service Platform Basic Development Guide*.

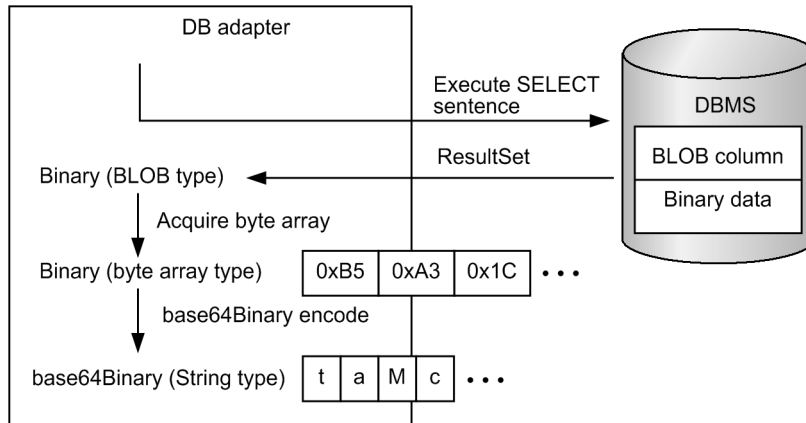
(2) Binary data operations

In a DB adapter, binary data is handled as a `byte` array type, and `base64Binary` and `hexBinary` types are handled as `String` type. The following describes examples of operating the BLOB type for executing the `SELECT` statement and for executing the `INSERT` statement.

(a) Processing when executing the `SELECT` statement

If you execute the `SELECT` statement for the BLOB column, the DB adapter converts (encodes) the `byte` array to the `base64Binary` or `hexBinary` type of XML Schema. Then, the value of the BLOB column is returned to the request source.

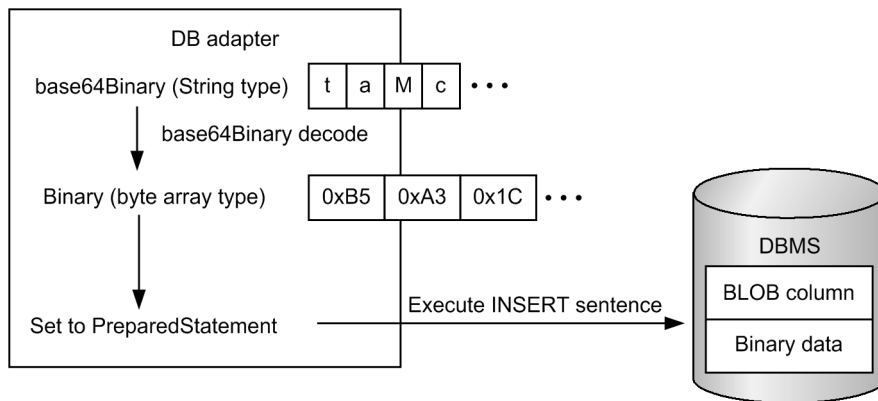
Figure 2-46: Processing when executing the SELECT statement (conversion to the base64Binary type)



(b) Processing when executing the INSERT statement

If you execute the `INSERT` statement for the BLOB column, the request source passes the `base64Binary` or `hexBinary` type data of XML Schema to the DB adapter. The DB adapter converts (decodes) the `base64Binary` or `hexBinary` type data to a byte array, and then the `INSERT` statement is executed.

Figure 2-47: Processing when executing the INSERT statement (conversion from the base64Binary type)



If an error occurs during conversion (decoding) of `xsd:hexBinary` or `xsd:base64Binary` type data, the message KDEC63001-E is output (with cause code 22 for the `xsd:hexBinary` type, or cause code 23 for the `xsd:base64Binary` type). If the **Convert a system exception into a fault message** option is enabled during definition of the DB adapter, a fault message will be output to the caller of the DB adapter. If this option is disabled, a system exception will be sent.

2.6.2 Setting up a DB connector to be used by DB adapters

To access the database, you need to prepare a DB connector for accessing the machine on which the service components are running (machine on which the database is running), which is different from the DB connector defined when setting up the HCSC server.

(1) Usable DB connectors

The usable DB connectors are limited depending on the transaction attribute of the DB connector defined when setting up the HCSC server.

The DB connectors that can be used by a DB adapter depend on the value of `XATransaction` or `LocalTransaction` specified in the `dbcon-xadisplayname` property of the HCSC server setup definition file. Therefore, you must consider the value of `XATransaction` or `LocalTransaction` specified in the `dbcon-xadisplayname` property when setting up the HCSC server.

Table 2–5: Usable DB connectors

DB connector being used	DB adapter		Usability	Reason for non-usability
	Usable DB Connector	DB adapter connection destination		
LocalTransaction	LocalTransaction	Same database as that used by the HCSC server	Yes	--
	LocalTransaction	Different database from that used by the HCSC server	No	Because multiple DB Connectors cannot be mixed up in a local transaction
	XATransaction	Different database from that used by the HCSC server	No	Because XA transactions (global transactions) cannot be mixed up in a local transaction
XATransaction	LocalTransaction	Different database from that used by the HCSC server	No	Because multiple DB Connectors cannot be mixed up in an XA transaction
	XATransaction	Same database as that used by the HCSC server	Yes	--
	XATransaction	Different database from that used by the HCSC server	Yes	--

Legend:

--: Not applicable

(2) Importing and deploying a DB connector

Import and deploy the DB connector to be used by the DB adapter on the HCSC server (J2EE server).

For details about how to select the RAR file to be imported when using a DB adapter, see the contents about setting up a DB connector in the manual *Cosminexus Service Platform System Setup and Operation Guide*.

1. Importing DB Connectors

Import the DB connector using the following command:

```
Cosminexus-installation-directory\CC\admin\bin\cjimportres J2EE-server-name -
nameserver corbaname::localhost:port-number-for-in-process-naming-service-type rar
-f Cosminexus-installation-directory\CC\DBConnector\RAR-file-to-be-imported
```

2. Deploying DB Connectors

Deploy the DB connector using the following command:

```
Cosminexus-installation-directory\CC\admin\bin\cjdeployrar J2EE-server-name -
nameserver corbaname::localhost:port-number-for-in-process-naming-service -resname
imported-DB-Connector-name
```

(3) Defining the properties of a DB connector

After importing the DB connector, you define properties. For details about the property definition file, see the contents about the HITACHI Connector Property file in the manual *Cosminexus Application Server Application and Resource Definitions*.

Reference note

When setting up properties for the deployed DB connector, instead of creating a new XML for the property definition file, if you acquire the properties of the deployed DB connector and edit the acquired XML, you can efficiently create an XML.

1. Acquire the file for property definition

Acquire the file for defining properties of the DB connector using the following command:

```
Cosminexus-installation-directory\CC\admin\bin\cjgetrarprop J2EE-server-name -
nameserver corbaname::localhost:port-number-for-in-process-naming-service -resname
imported-DB-Connector-name -c property.xml
```

2. Edit the file for property definition

Change the value of each attribute of the acquired XML file for property definition.

For details about the settings of the XML file for property definition (HITACHI Connector Property file), see the contents about setting up a DB connector in the manual *Cosminexus Service Platform System Setup and Operation Guide*.

For details about the HITACHI Connector Property file, see the description about defining properties of a DB connector in the manual *Cosminexus Application Server Application Setup Guide*, and also see the manual *Cosminexus Application Server Application and Resource Definitions*.

3. Set up the edited file for property definition

Set up the contents of the edited XML file for property definition in the deployed DB connector.

Set up the definition contents in the DB connector using the following command:

```
Cosminexus-installation-directory\CC\admin\bin\cjsetrarprop J2EE-server-name -
nameserver corbaname::localhost:port-number-for-in-process-naming-service -resname
imported-DB-Connector-name -c property.xml
```

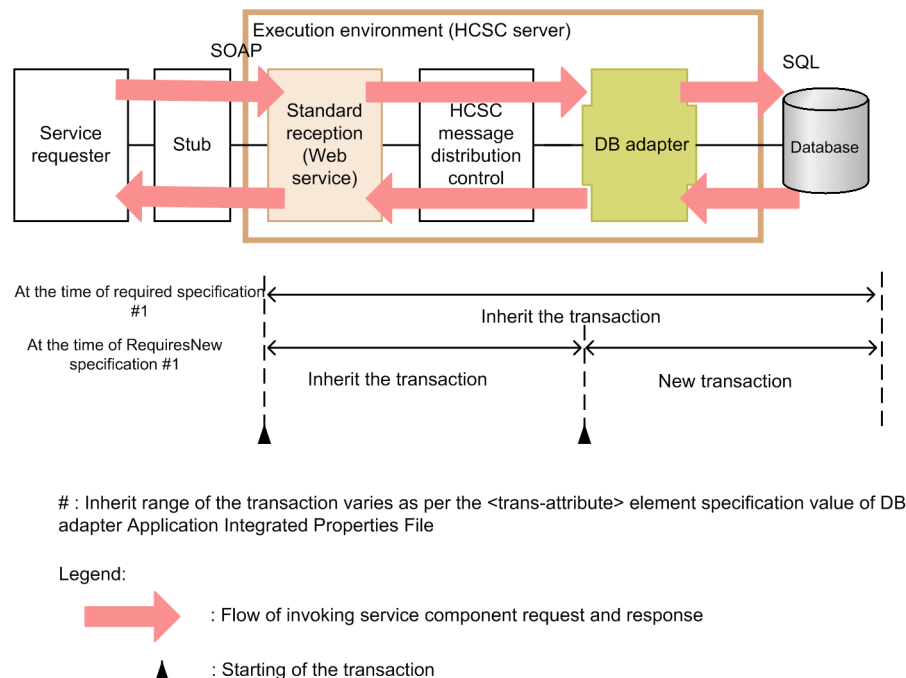
#: The XML file for property definition (property.xml) is the property definition file after the value of each attribute of the acquired property definition file has been changed.

2.6.3 Transactions while using DB adapter

In DB adapter, the database side inherits the transactions managed in HCSC server. For transactions between HCSC servers from the service requester, see the description of each standard reception transaction control.

The following figure shows transactions in DB adapter with an example of a service requester using SOAP communication:

Figure 2-48: Transactions in DB adapter



: Inherit range of the transaction varies as per the <trans-attribute> element specification value of DB adapter Application Integrated Properties File

2.6.4 Dynamic change in connection destination information for DB adapters

You can dynamically change the DB adapter connection destination by setting a resource alias for the connection destination DB Connector in the DB adapter. If the resource alias for the connection destination DB Connector has been set in advance in the business process, the connection destination changes when the DB adapter is executed. This can reduce the number of DB adapters to be created.

(1) Procedure for dynamically changing the connection destination

The following describes the procedure for dynamically changing the DB adapter connection destination.

1. Acquire connection destination information from the header message in the request message.
2. Establish a connection with the database based on the acquired information.

If an attempt to acquire the connection destination information fails because, for example, the connection destination is not set in the header message, a request process causes an error.

(2) Preconditions for dynamically changing the connection destination

The following describes the preconditions for dynamically changing the connection destination of the DB adapter.

(a) Assigning a resource alias

Assign a resource alias to the connection destination DB Connector.

For details about the rules for assigning aliases, see 2.6.2 *Rules for assigning the optional names* in the *Application Server Common Container Functionality Guide*.

(b) Setting up the SQL operation definition file

Set the `dynamic` attribute to `Y` in the SQL operation definition file.

(c) Selecting the DB Connector type

Make sure that the connection destination DB Connector type is the same as the database type specified in the SQL operation definition file.

(d) Specifying the transaction support level

You can specify `LocalTransaction` or `XATransaction` as the transaction support level of the connection destination DB Connector.

The following table shows the specifiable transaction support levels and required conditions.

Table 2–6: Specifiable transaction support levels and required conditions

Specifiable transaction support level	Required conditions
<code>LocalTransaction</code> or <code>XATransaction</code>	<ul style="list-style-type: none"> • The status is not made persistent in the business process. • The transaction of the DB adapter starts separately from the business process. • The status is made persistent in the business process, and the transaction of the DB adapter starts separately from the business process.
<code>XATransaction</code>	Any condition other than the above.

(e) Setting up a business process

Specify the connection destination information in a business process.

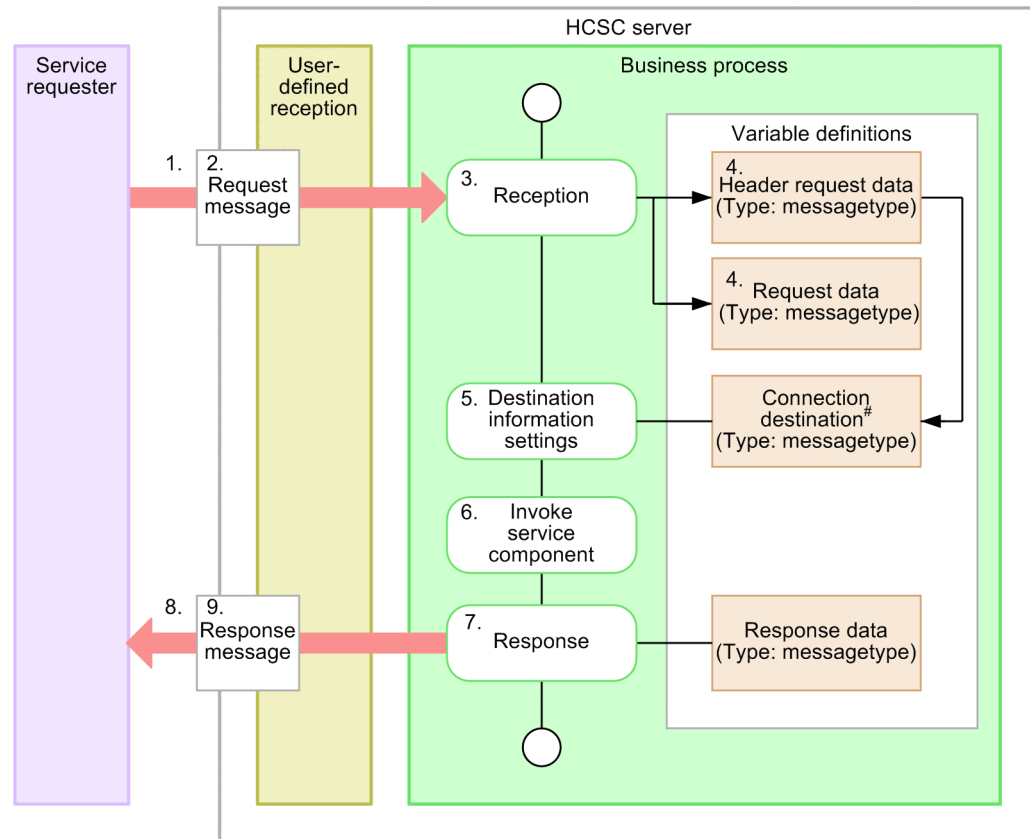
(3) Example of dynamic change of connection destination information

The following describes an example of dynamically changing connection destination information by using the connection destination information file.

(a) Example of a business process

The following figure shows an example of a business process using the connection destination information file.

Figure 2-49: Example of a business process using the connection destination information file (in a DB adapter)



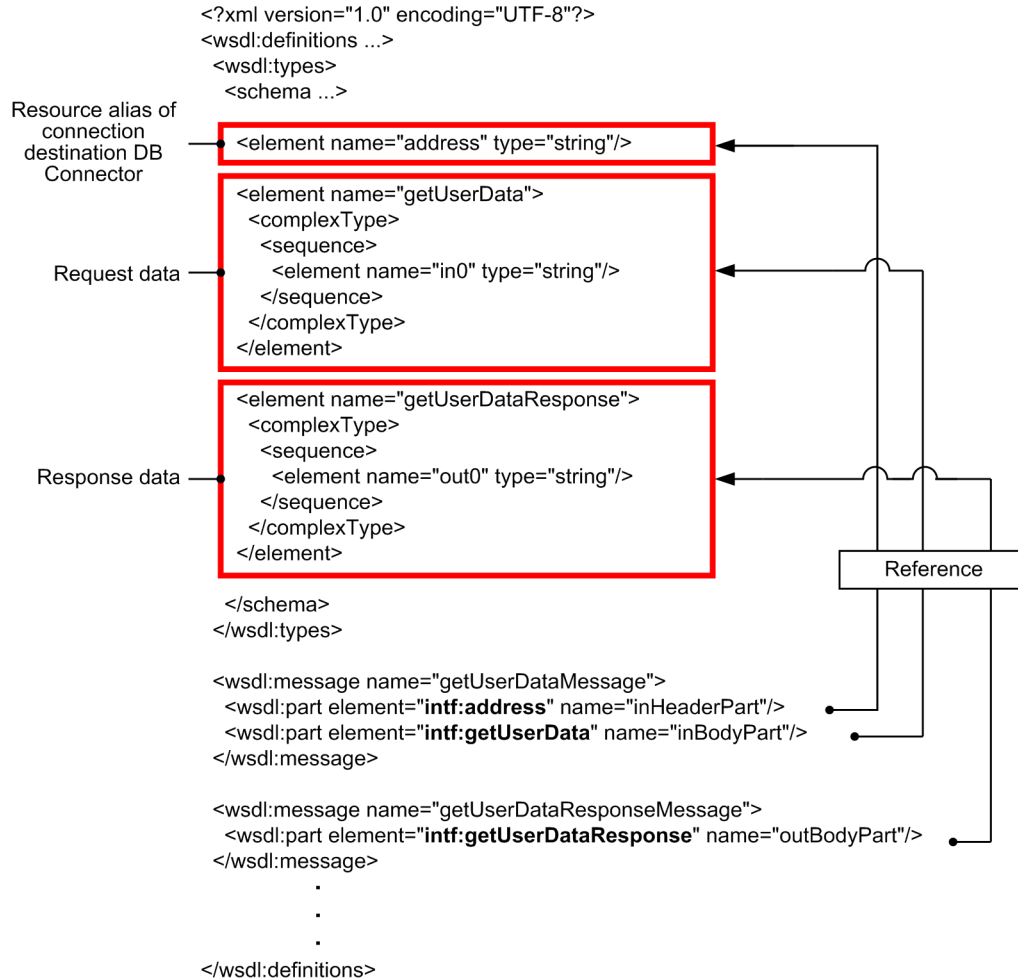
#: Variables defined by using the connection destination information file

1. The service requester requests execution of a service component in the business process.
2. Set the connection destination information (resource alias for the DB Connector) in the header message in the request message sent from the service requester.
3. The business process receives an XML request message by the receive activity.
4. The business process generates header request data and request data.
5. Set the connection destination information of the header request data in the connection destination information variables by using a data transformation activity.
6. The business process sets the connection destination information contained in the connection destination information variables to the DB adapter, and then invokes the service component.
7. The business process configures the service component execution result as an XML response message, and then returns it by using a reply activity.
8. A response is returned to the service requester.
9. An XML response message is sent in response to the service requester.

(4) Example of WSDL creation generating user-defined reception

The figure below shows an example of creating a WSDL that generates user-defined reception.

In this example, the resource alias for the DB Connector to be connected is sent in the SOAP header.

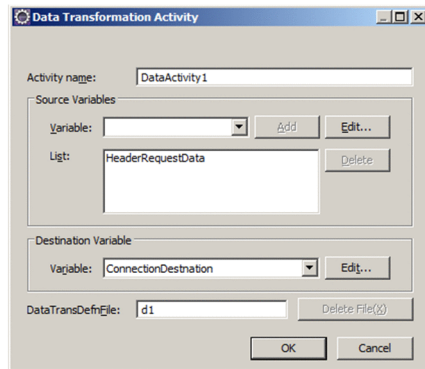


(a) Example of setting data transformation activities

The figure below shows an example of setting data transformation activities used in the business process shown in *Figure 2-49 Example of a business process using the connection destination information file (in a DB adapter)*.

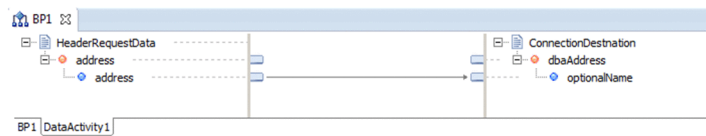
Here, *Header request data* is specified in the transformation source and *Connection destination* is specified in the transformation destination.

Figure 2–50: Example of setting data transformation activities (in a DB adapter)



The following figure shows the definition of transformation mapping.

Figure 2–51: Example of setting transformation mapping (in a DB adapter)



(b) Defining a connection destination information file

Define the resource alias of the DB Connector in the connection destination information file (XML schema) used for setting up a DB adapter.

The connection destination information file is provided in Service Platform.

- Storage location of the connection destination information file

installation-directory-of-the-service-platform/CSC/schema/connection/connection_db_adapter.xsd

! Important note

Use the connection destination information file provided in Service Platform to dynamically change connection destination information of the DB adapter. Operation is not guaranteed if you do not use the provided connection destination information file.

- Format of the connection destination information file

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
DO NOT EDIT THIS FILE.
-->
<xsd:schema
  elementFormDefault="qualified"
  targetNamespace="http://www.hitachi.co.jp/soft/xml/cosminexus/csc/connection/change/dba"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="dbaAddress">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="optionalName" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

- Element that can be set

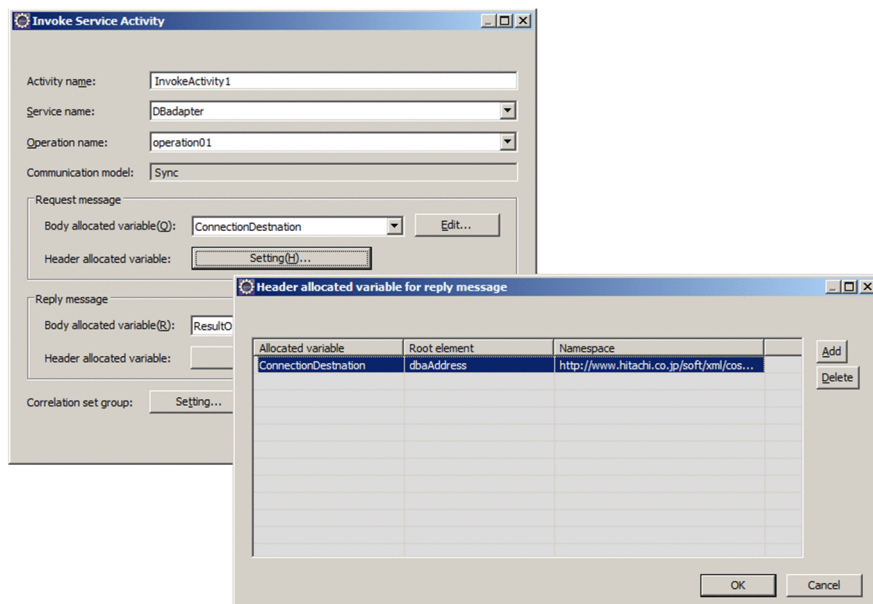
optionalName

Specify the resource alias set in the connection destination DB Connector.

(c) Setting connection destination information

The following figure shows an example of setting connection destination information used in the business process shown in *Figure 2-49 Example of business process using connection destination information file (in a DB adapter)*. Set connection destination information in the header assigned variable of the invoke service activity. The information set here can be viewed in the DB adapter.

Figure 2-52: Example of setting connection destination information (setting invoke service activity)



2.7 Connecting to OpenTP1

INTENTIONALLY DELETED

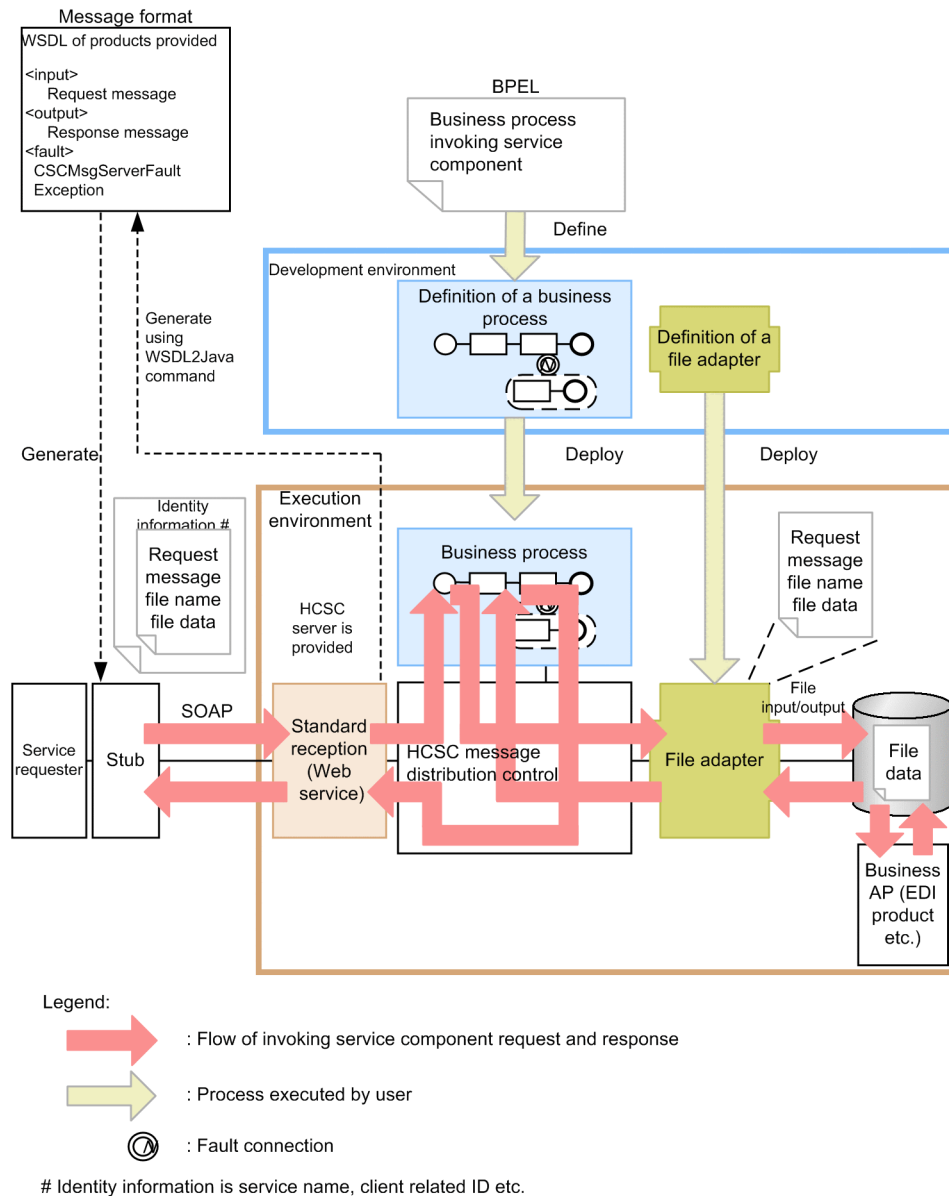
2.8 Connecting to a system handling a file

Use the file adapter for direct I/O of files for a local disk on HCSC server when a request is received from a service requester or business process.

2.8.1 Accessing files using file adapter

The following figure shows execution of file access by the file adapter:

Figure 2-53: Execution of file access by the file adapter



Define the file adapter in the development environment. For details on the procedure for defining file adapters, see "3. Defining Adapters" in "Service Platform Reception and Adapter Definition Guide".

2.8.2 Reading and writing files using adapter

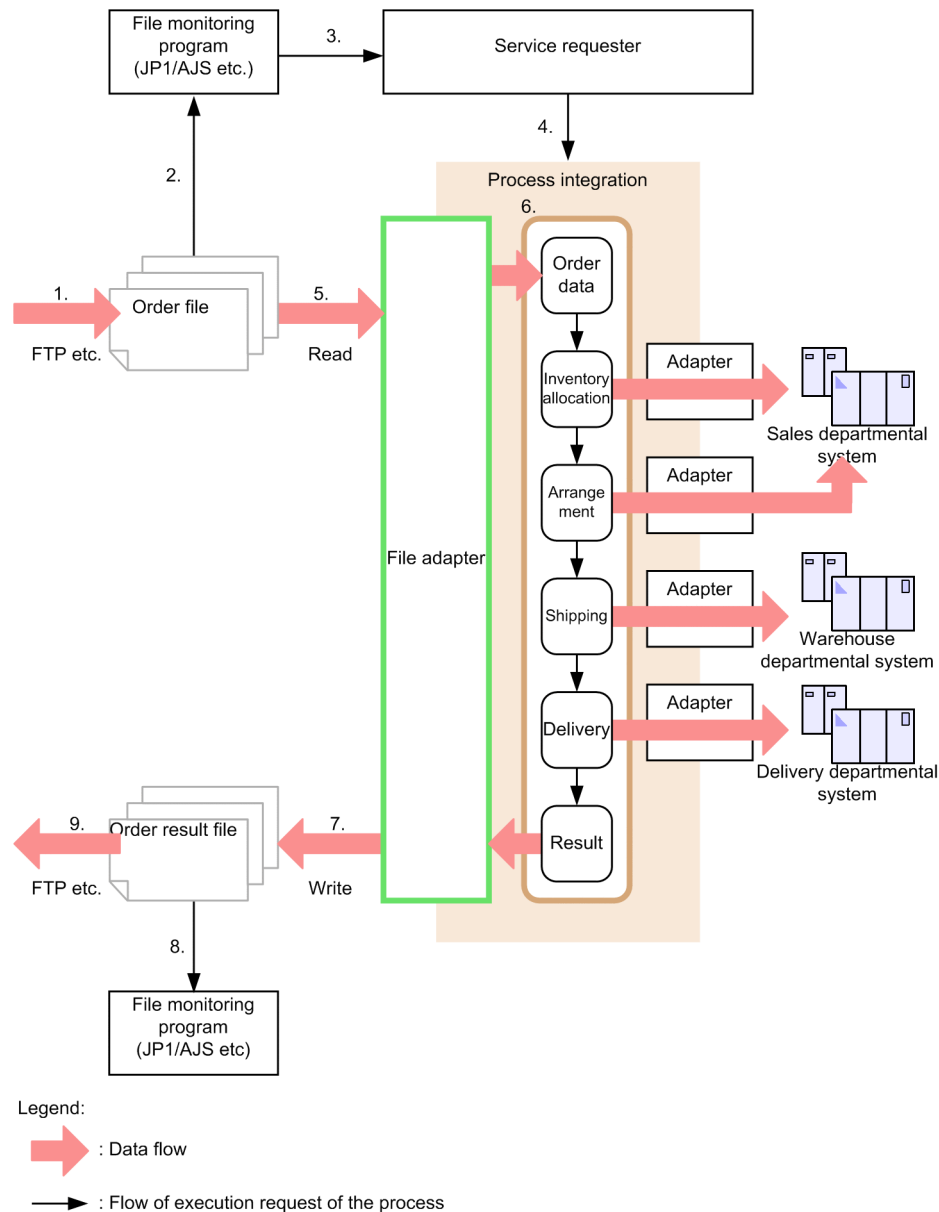
The file adapter is a service adapter that reads the data from files and writes that data in files in the service platform.

The purpose of the file adapter is to apply program resources of the business process system to the system through files. By using the file adapter as a business process activity, you can implement business processes based on file contents such as linkage with existing systems according to file I/O in SOA.

A prerequisite for the purpose of using the file adapter in process integration is invocation from the business process and not direct invocation from the service requester.

The following figure shows an example of system operation using the file adapter:

Figure 2–54: Example of file adapter operation



The following points describe each process shown in the figure:

1. Order files are received from other companies using FTP.
2. File reception is detected by file monitoring programs such as JP1/AJS.
3. Service requester is started from the file monitoring program through JP1/AJS.

4. Business process is started from the service requester through SOAP.
5. Order file data is read by the file adapter invoked from the business process.
6. Each business service is invoked.
7. Service invocation result is written in the order results file by the file adapter.
8. Writing in the order results file is detected by file monitoring programs such as JP1/AJS.
9. Order results file is transferred using FTP.

Such systems enable linkage with other companies in which SOA is not supported.

2.8.3 File formats that can be used in adapters

You can use the following file formats in the file adapter:

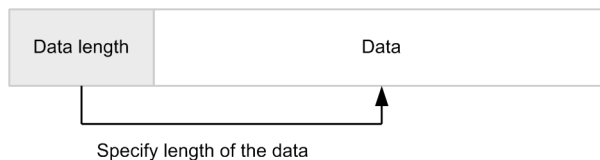
(1) File format handled as XML data in file adapter

- XML format
This file format is described in the markup language established by W3C (Extensible Markup Language). The application range of XML schema that you can use in HCSC server supports XML format files. For details, see "2.6.5 Scoping of XML schema" in "Service Platform Basic Development Guide" .

(2) File format handled as binary data in file adapter

- Fixed length format
This file format contains the length decided for each data element in the record.
- Length tag format
This file format handles variable length data by adding numeric value information representing the length for each data element in the record.

Figure 2–55: Length tag format



- Separator format
This file format handles variable length data by using a separator (delimiter) for separation of each data element in the record. CSV file exists as the general format.

2.8.4 Modes for file operations

This section describes the modes for file operations in the file adapter.

(1) Read mode

The mode used for reading files in the file adapter is described here. The maximum size of a file that can be read is 64MB. You can change the settings in the property file of the file adapter execution environment. 0 byte files cannot be read. Read files in the file adapter using permission of HCSC server administrator.

- Batch
The file to be read is entirely read.

(2) Write mode

The modes used for writing files in the file adapter are described here. Write files in the file adapter using permission of HCSC server administrator.

- **New**
A file is created for writing and data is written. If a file to be written already exists, it is overwritten by the data to be written. If an error occurs in the writing process, the data gathered until just before the error occurrence is written. A blank file might be created depending on the time at which the error occurred.
- **Add**
Data to be written is written by adding it in an existing file. If a file to be written does not exist, a file is created and the data to be written is written. You cannot specify add in the writing mode in XML format file. If an error occurs in the writing process, data is written until just before the error occurrence. A blank file might be created depending on the time at which the error occurred.

2.8.5 Exclusive control while reading and writing files

Exclusive control while operating files in the file adapter prevents conflict and data inconsistencies in the process.

Exclusive control occurs in file units that are to be processed in 1 file adapter. Accordingly, if a file being processed in the file adapter is operated from an external application, there is no exclusive control. If a single HCSC server uses multiple file adapters, there is no exclusive control between file adapters. Use of 1 file adapter is recommended for files in identical formats.

The following table describes exclusive control during reading and writing of files:

Table 2–7: Exclusive control during reading and writing of files

Exclusive control	Reading	Writing
Exclusive reference (while reading)	Y	N
Exclusive update (while writing)	N	N

Legend:

Y: Can be processed.

N: Cannot be processed (standby status).

(1) Exclusive reference

Exclusive reference occurs if a file is read in the file adapter. You can read a file in exclusive reference but you cannot write in it. If you write in a file in exclusive reference, standby status occurs till the reading process being executed ends.

(2) Exclusive update

Exclusive update occurs if a file is written in the file adapter. You cannot read and write in a file in exclusive update. If you read or write in a file in exclusive update, standby status occurs till the writing process being executed ends.

2.9 Connection with Object Wrapper system

INTENTIONALLY DELETED

2.9.1 Access to Object Wrapper system using Object Access adapter

INTENTIONALLY DELETED

2.9.2 Message format transformation at the time of connection

INTENTIONALLY DELETED

2.10 Connecting to Message Queues

Use Message Queue reception to send and receive messages from JMS provider message queue (IBM WebSphere MQ system) through MQ resource adapter.

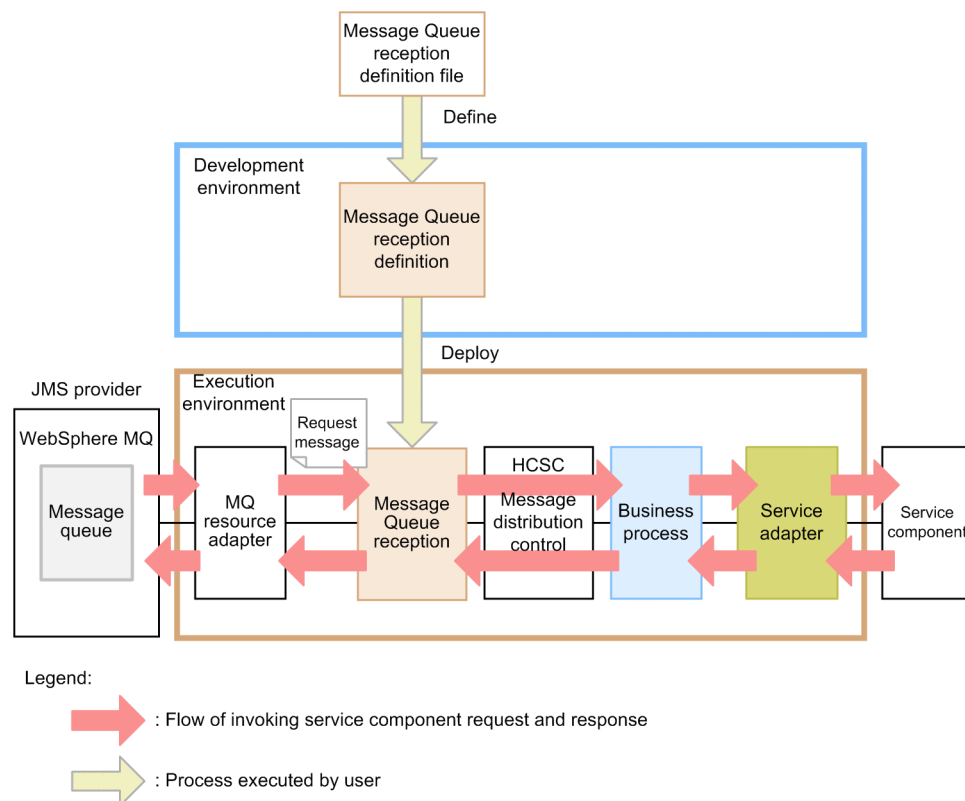
Use Message Queue adapter to send and receive messages for existing message queues (IBM WebSphere MQ system).

2.10.1 Invoking service components using Message Queue reception

You can use Message Queue reception to invoke a business process from JMS provider message queue (IBM WebSphere MQ) through MQ resource adapter.

The following figure shows the flow of service component invocation using Message Queue reception:

Figure 2–56: Invoking a service component from JMS provider (message queue) using Message Queue reception



Message Queue reception receives request messages through MQ resource adapter. Message Queue reception receiving a service component execution request invokes the business process and executes all types of service components through the service adapter.

Use MDB (Message-Driven Bean) to receive a request message.

A response message is not returned from Message Queue reception.

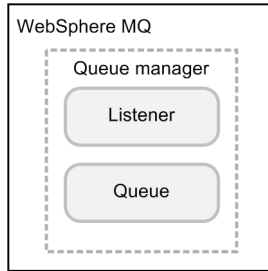
(1) Overview of WebSphere MQ

Use WebSphere MQ (manufactured by IBM) as JMS provider in Message Queue reception. WebSphere MQ is asynchronous communication infrastructure of message queueing type for data communication between system programs of different models.

Message Queue reception is supported in WebSphere MQ version 7.5.

The following figure shows WebSphere MQ configuration:

Figure 2–57: WebSphere MQ configuration



Legend:

Queue manager: Management program organizing queues and listeners

Queue: Queue object organizing JMS objects

Listener: Listener to receive or send messages by queue manager

! Important note

Queue manager of WebSphere MQ cannot access the server by channel authentication in initial status. For this, it is necessary to change the setting so that channel authentication is not used or to change the channel authentication rules set by default. For WebSphere MQ setting methods, see IBM manual.

(2) Send option of Message Queue reception

You can specify the send option (JMS header and properties) to send messages. The following shows details of send options of Message Queue reception:

(a) Details of JMS header

All setting values of the send option (JMS header) are transformed to String type and are set in the header message. At this stage, if value is null or blank, the tag is blank.

The following table describes the details of JMS header:

Table 2–8: Details of JMS header

Header name	Setting contents	Setting time	Possibility of setting in client
JMSDestination	Queue name in which message is registered	Set when QueueSession.createSender(queue) method is issued.	N
JMSDeliveryMode	Value showing message persistence	Set when the message is sent.	N
JMSMessageID	Message identifier to uniquely identify each message sent by the provider	Set when the message is sent.	N
JMSTimestamp	Value showing message sending time (unit: milliseconds)	Set when the message is sent.	N
JMSExpiration	Value showing time for which message is valid (unit: milliseconds)	Set when the message is sent.	N
JMSRedelivered	Value showing whether the message is being redelivered	Set when a message delivered to the application needs to be redelivered by recovery.	N
JMSPriority	Message priority	You can specify by QueueSender.setPriority() method, or QueueSender.send() method.	Y

Header name	Setting contents	Setting time	Possibility of setting in client
JMSReplyTo	Response destination address	Set when Message.setJMSReplyTo() method is issued.	Y
JMSCorrelationID	Correlation identifier to correlate a message to another message	Set when Message.setJMSCorrelationID() method is issued.	Y
JMSType	Message type provided by the application when a message is sent	Set when Message.setJMSType() method is issued.	Y

Legend:

Y: Can be specified.

N: Cannot be specified.

(b) Details of properties

Message properties are configured from the properties set by JMS provider and properties that the user can set in the client.

Properties set by JMS provider

Properties set by JMS provider contain JMS standard properties and properties set by WebSphere MQ. All items in which either property is set are set in the header message.

- **JMS standard properties**

JMS standard properties are prefixed with the item name "JMSX".

The following table lists and describes JMS standard properties:

Table 2–9: List of JMS standard properties

Item name	Description
JMSXUserID	User ID of message sender
JMSXAppID	ID of message sending application
JMSXDeliveryCount	Number of times a message is re-received
JMSXGroupID	ID of group to which the message belongs
JMSXGroupSeq	Message order number in the group
JMSXProducerTXID	Transaction identifier
JMSXConsumerTXID	Transaction identifier
JMSXRecvTimestamp	Message reception time
JMSXState	Item used in the provider

- **Properties set by WebSphere MQ**

Properties set by WebSphere MQ are included in the properties for provider. Properties for provider are prefixed with the item name "JMS_". Among these properties, properties prefixed with the item name "JMS_IBM_" are WebSphere MQ properties.

Properties set by user

Properties that are optional to the user are set by using "set<Type name>Property(String name,<Type name>value)" method.

All values set in a property are transformed to String type and set in the header message.

The following table lists the data types that you can set in a property:

Table 2–10: List of data types you can set in a property

Data type	Setting method
Boolean	setBooleanProperty(String name, boolean value)
Byte ^{#1}	setByteProperty(String name, byte value)
Short	setShortProperty(String name, short value)
Integer	setIntProperty(String name, int value)
Long	setLongProperty(String name, long value)
Float	setFloatProperty(String name, float value)
Double	setDoubleProperty(String name, double value)
String	setStringProperty(String name, String value)
Object ^{#2}	setObjectProperty(String name, Object value)

Note#1

If you set Byte type value, the item transformed to Integer type numeric value is set in the header message as a character string. The range of numeric values set is -127-127.

Note#2

If Object is set in data type, you can use all data types (Boolean, Byte, Short, Integer, Long, Float, Double and String).

(3) Relation between Message Queue reception and MQ resource adapter

MQ resource adapter is used for passing messages from JMS provider (IBM WebSphere MQ system) to Message Queue reception.

The connection relation between MQ resource adapter and Message Queue reception is that you can correlate multiple receptions for 1 resource adapter but you cannot correlate multiple resource adapters for 1 reception.

Specify the correlation of MQ resource adapter and Message Queue reception in Message Queue reception definition file. The specified contents are enabled when Message Queue reception starts. At this stage, you must start MQ resource adapter in advance, which is correlated with Message Queue reception. Setting contents of Message Queue reception are not enabled, since Message Queue reception cannot start if the correlated MQ resource adapter is not started.

The connection relation between Message Queue reception and MQ resource adapter is described here.

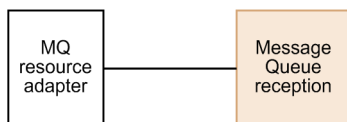
(a) Connection relations that can be set up (in MQ resource adapter)

You can set up one of the following connection relations:

- If connection relation between resource adapter and reception is 1:1
- If connection relation between resource adapter and reception is 1:n

The following figures show the examples:

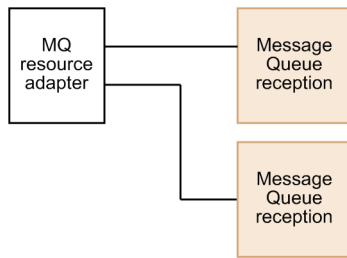
(Example 1) If connection relation between resource adapter and reception is 1:1



Description

- 1 Message Queue reception correlates to 1 MQ resource adapter
- 1 MQ resource adapter correlates to 1 Message Queue reception

(Example 2) If connection relation between resource adapter and reception is 1:n

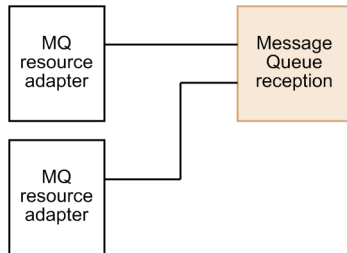
**Description**

- Multiple Message Queue receptions correlate to 1 MQ resource adapter
- 1 MQ resource adapter correlates to 1 Message Queue reception

(b) Connection relations that cannot be set up (in MQ resource adapter)

You cannot set n:1 configuration for connection relation between the resource adapter and reception, since you cannot specify multiple resource adapters for 1 reception.

The following figure shows an example:

(Example)**Description**

- 1 Message Queue reception correlates to 1 MQ resource adapter
- Multiple MQ resource adapters correlate to 1 Message Queue reception

(4) Relation between Message Queue reception and queues

Queue is a queue object compiling JMS objects and is managed by the queue manager.

The connection relation between a queue and Message Queue reception is that multiple receptions can correlate to 1 queue but multiple queues cannot correlate to 1 reception.

The connection relation between Message Queue reception and queue is described here.

(a) Connection relations that can be set up (in queues)

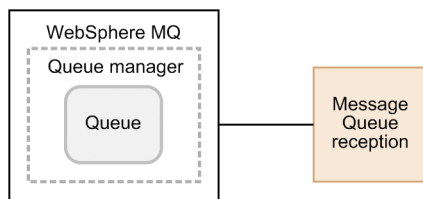
You can set up one of the following connection relation:

- If the connection relation between a queue and reception is 1:1
- If the connection relation between a queue and reception is 1:n

The following figures show the examples:

(Example 1) If connection relation between a queue and reception is 1:1

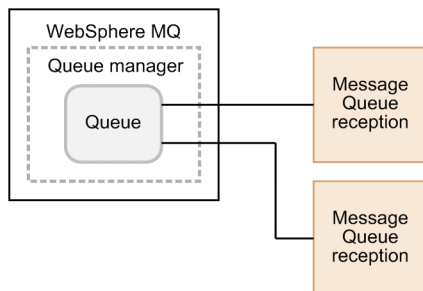
2. Functionality for Connecting to Various Types of Systems



Description

- 1 queue exists in 1 queue manager
- 1 Message Queue reception correlates to 1 queue

(Example 2) If connection relation between a queue and reception is 1:n



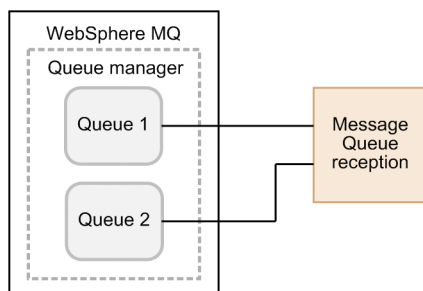
Description

- 1 queue exists in 1 queue manager
- 2 Message Queue receptions correlate to 1 queue

(b) Connection relations that cannot be set up (in queues)

You cannot set up n:1 configuration for connection relation between a queue and reception, since you cannot specify multiple queues for 1 reception.

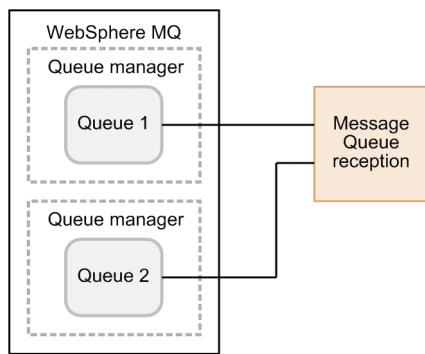
(Example 1) If there is 1 queue manager



Description

- 2 queues exist in 1 queue manager
- 1 Message Queue reception correlates to queue 1 and queue 2

(Example 2) If there are multiple queue managers

**Description**

- 2 queue managers exist
- 1 queue exists in each queue manager
- 1 Message Queue reception correlates to queue 1 and queue 2

(5) Relation between Message Queue reception and business processes

While creating a business process, you must match the receive activity communication model with the communication model of Message Queue reception.

You can set business process persistence as persistent or non-persistent.

However, please note that if you set non-persistent, there may be duplicate message delivery when a fault occurs.

The connection relation between a business process and Message Queue reception is that multiple receptions can correlate to 1 business process but multiple business processes cannot correlate to 1 reception.

The connection relation between Message Queue reception and business processes is described here.

(a) Connection relations that can be set up (in business processes)

You can set up one of the following connection relations:

- If connection relation between a business process and reception is 1:1
- If connection relation between a business process and reception is 1:n

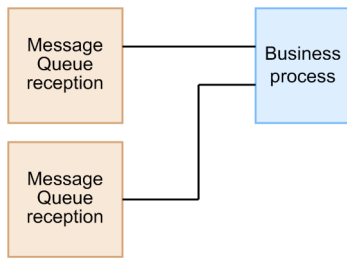
The following figures show the examples:

(Example 1) If connection relation between a business process and reception is 1:1

**Description**

- 1 business process correlates to 1 Message Queue reception
- 1 Message Queue reception correlates to 1 business process

(Example 2) If connection relation between a business process and reception is 1:n



Description

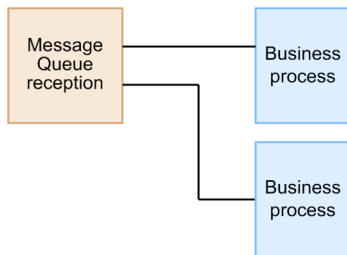
- 1 business process correlates to 1 Message Queue reception
- Multiple Message Queue receptions correlate to 1 business process

(b) Connection relations that cannot be set up (in business processes)

You cannot set up n:1 configuration for connection relation between a business process and reception, since you cannot define multiple business processes for 1 reception.

The following figure shows an example:

(Example)



Description

- Multiple business processes correlate to 1 Message Queue reception
- 1 Message Queue reception correlates to 1 business process

(6) Example of Message Queue reception configuration

You can configure various systems according to the communication model used in Message Queue reception.

Configuration examples in the following cases are explained here.

- If Message Queue reception is used in an asynchronous communication model
- If Message Queue reception is used in a synchronous communication model

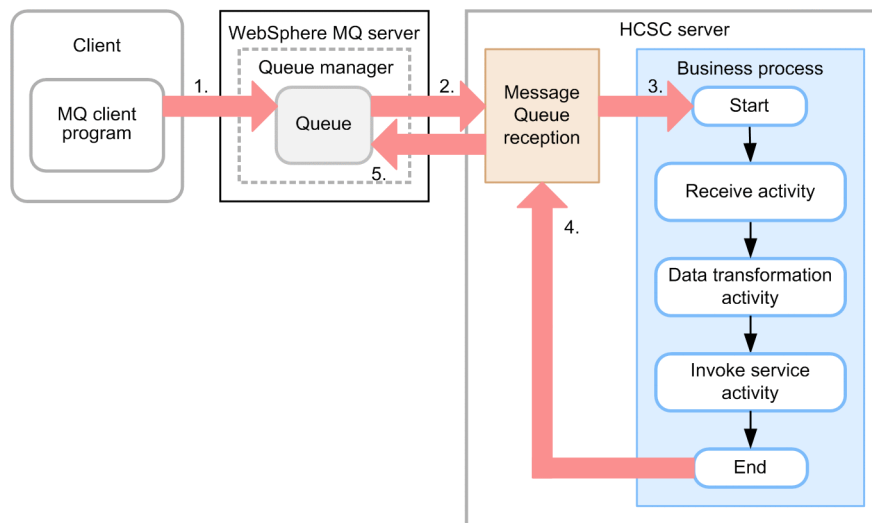
(a) If Message Queue reception is used in asynchronous communication model

Preconditions

1. A business process that defines Message Queue reception is created.
2. Communication model of Message Queue reception is defined as "Asynchronous".

Example of configuration

The following figure shows an example of configuration of Message Queue reception and business process:



Description

1. Register the message in WebSphere MQ server queue from MQ client program.
2. Message is sent from WebSphere MQ server to Message Queue reception (HCSC server).
3. Message Queue reception invokes the business process.
4. After all activity processes end, Message Queue reception process also ends.
5. WebSphere MQ server commits the transaction and deletes the message from the queue.

Reference note

In this configuration example, the message is maintained till the request process is finished. Due to this, the message is not deleted from the queue while the business process is being processed. However, if the duplicate delivery prevention process is executed or if the message is moved by rollback restrictions, the message is not deleted from the queue. To prevent duplicate delivery of messages, see "(8) Preventing duplicate message delivery while using Message Queue reception".

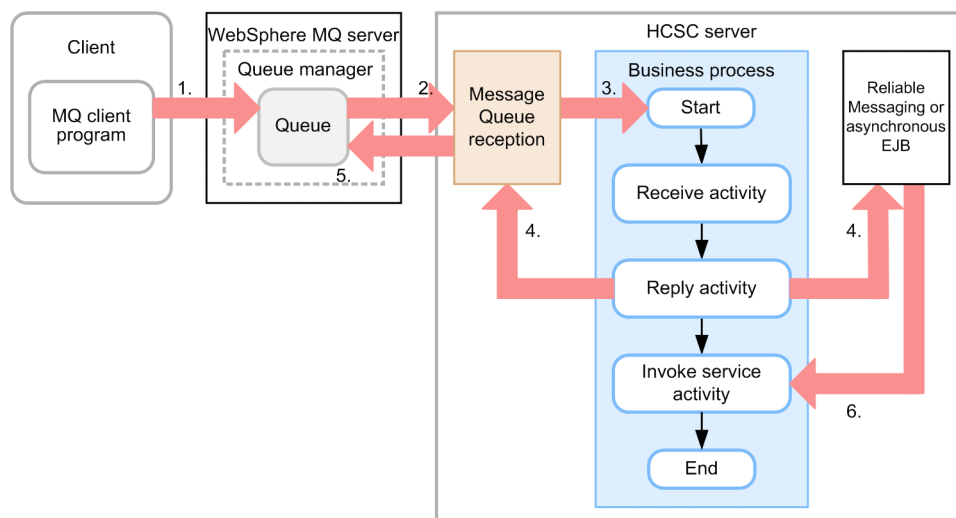
(b) Using Message Queue reception in synchronous communication model

Preconditions

1. A business process that defines Message Queue reception is created.
2. Communication model of Message Queue reception is defined as "Synchronous".

Example of configuration

The following figure shows an example of configuration of Message Queue reception and business process:



Description

1. Register the message in WebSphere MQ server queue from MQ client program.
2. Message is sent from WebSphere MQ server to Message Queue reception (HCSC server).
3. Message Queue reception invokes the business process.
4. After the reply activity process ends, the information required for execution after reply activity is stored in Reliable Messaging or asynchronous EJB and Message Queue reception process also ends.
5. WebSphere MQ server commits the transaction and deletes the message from the queue.
6. The activity set after reply activity is executed according to the information stored in Reliable Messaging or asynchronous EJB.

Reference note

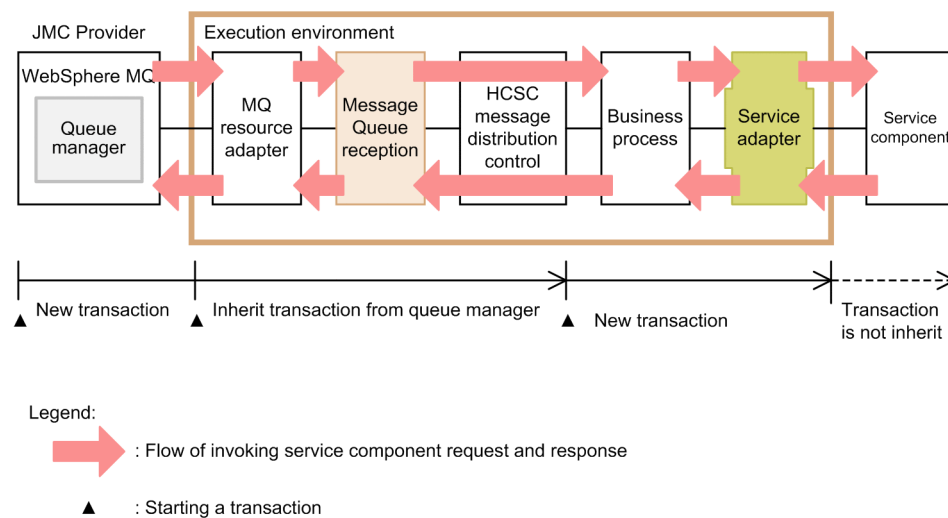
In this configuration example, you can delete the message from the queue before processing of the business process ends. However, if an exception occurs in the processing of the business process after the reply activity, it is necessary to resend the request because the message is deleted from the queue.

(7) Transactions performed while using Message Queue reception

You must use transactions same as those of WebSphere MQ queue manager, since the queue rolls back when an exception occurs in Message Queue reception. For this, use a container management transaction for Message Queue reception transaction. By this, the transaction is committed at the same time as when WebSphere MQ queue is committed.

The following figure shows the range of transactions performed when Message Queue reception is used:

Figure 2–58: Range of transactions performed when Message Queue reception is used



Message Queue reception is processed by using transactions same as those of WebSphere MQ queue manager. However, the business process is processed by transactions different from WebSphere MQ queue manager. Due to this, the process terminates abnormally in Message Queue reception and even when the message returns to the beginning of the queue to execute rollback, the business process transaction might end normally. In this status, if the message is resent by rollback, the same process is re-executed in the business process in which the process was executed once. To prevent such processes, perform a process to prevent duplicate delivery of messages. To prevent duplicate delivery of messages, see "(8) Preventing duplicate message delivery while using Message Queue reception".

(8) Preventing duplicate message delivery while using Message Queue reception

If the request process ends abnormally, the business process for which the process is executed once is reprocessed when the message to execute rollback returns to the beginning of the queue and the message is resent.

Perform settings for preventing duplicate message delivery in Message Queue reception. With these settings, you can prevent re-execution of a business process that is already executed due to resending of a message.

(a) Preconditions to prevent duplicate delivery

If all the following conditions are met, check prevention duplicate delivery:

- Database is used.
- Execution history is valid (status persistence in business process properties is "yes").

(b) Points to be considered when using prevention of duplicate delivery

The points to be considered when prevention of duplicate delivery is set are as follows:

- You must estimate the database capacity, since execution history is too managed.
- A business process in which system exception has occurred must be re-executed.

(c) Measures to be taken when a fault occurs in a business process

Examples of the main faults occurring in business processes and the operations and actions to be performed when prevention of duplicate delivery is set are described here.

(Example 1) If a fault occurs before registering to the message correlation information table

Details on the display of execution history, request message status and actions to be taken in this case are as follows:

- **Display of execution history**

Not executed

- **Request message status**

Rolled back and request message is resent.

If the cause is an invalid message (such as null), continue the rollback only for the number of executions of the specified rollback.

- **Action**

Set the number of rollback restrictions.

Remove the message in which the fault occurred.

(Example 2) If a fault occurs during process execution of a business process

Details on the display of execution history, request message status and actions to be taken in this case are as follows:

- **Display of execution history**

Executing or error

- **Request message status**

Rolled back when fault occurred and committed as normal termination by duplicate delivery prevention check when the 2nd request message is sent.

- **Action**

Use the re-execution command or operating environment screen and re-execute the business process.

(Example 3) If a fault occurs after processing of a business process ends

The following shows execution history display, request message status and action in this case:

- **Execution history display**

Normal termination

- **Request message status**

Rolled back when fault occurred and committed as normal termination by duplicate delivery prevention check when the 2nd request message is sent.

- **Action**

Not required.

(Example 4) If a fault occurs after reply activity process of a business process ends

Details on the display of execution history, request message status and actions to be taken in this case are as follows:

- **Display of execution history**

Executing

- **Request message status**

Request message is not sent because the queue is committed after the reply activity process ends.

- **Action**

Use the re-execution command or operating environment screen and re-execute the activity after the reply activity.

(9) Setting number of times of message rollback

Always roll back if a fault occurs during the request process in Message Queue reception. This is to prevent unlimited loops by moving messages to the queue specified in the dead letter queue of the queue manager or the back out re-queue of the queue.

Set the number of rollbacks in WebSphere MQ queue manager and the queue.

For details on the setting methods, see WebSphere MQ manual.

Items set in queue manager

The following table describes the setting item correlated to rollback set in the queue manager:

Table 2–11: Setting item correlated to rollback set in the queue manager

Setting item	Setting contents	Value that can be set	Default value
DEADQ	Dead letter queue name	Queue name	SYSTEM.DEAD.LETTER.QUEUE

Items set in queue

The following table describes the setting items correlated to rollback set in queue:

Table 2–12: Setting items correlated to rollback set in queue

Setting item	Setting contents	Value that can be set	Default value
BOTHRESH	Backout threshold value	0-999999999	0
BOQNAME	Backout re-queue name	Queue name	None

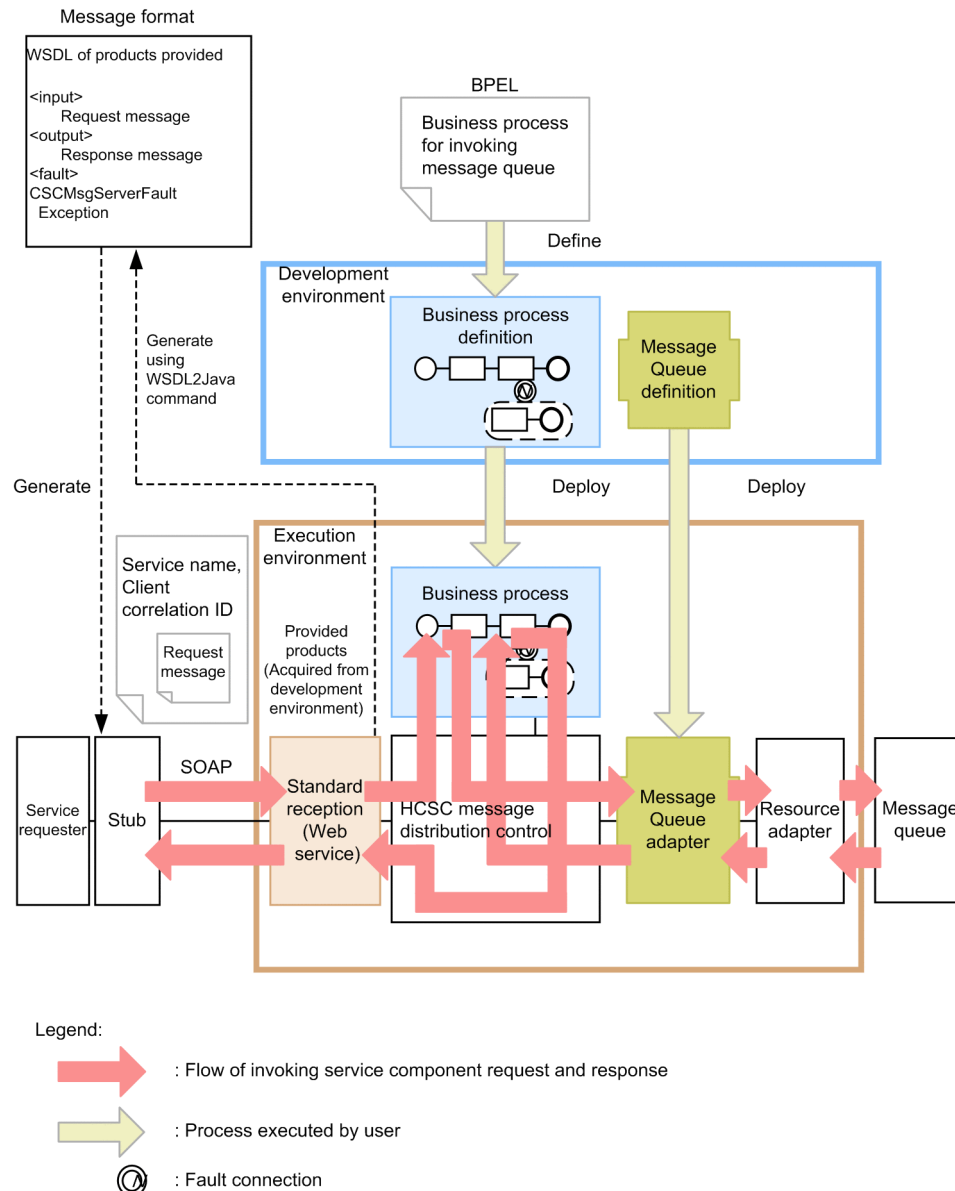
! Important note

You must take care while specifying a value (time for which message is maintained) in QueueSender#send parameter set in the client. Even if the message is moved to the dead letter queue, it is deleted, if the time specified is exceeded.

2.10.2 Access to message queues using Message Queue adapter

The following figure shows sending and receiving of messages by Message Queue adapter:

Figure 2-59: Execution of sending and receiving of messages by Message Queue adapter

**! Important note**

- Message Queue adapter executes the connection release process for each request from the business process. However, actually, the connection between fixed time (default 5 minutes) resource adapter and WebSphere MQ is not disconnected by the resource adapter connection pool functionality. This depends on the specification in Connector attribute file (default is not disconnected).
- If connection release fails in the connection release process, Message Queue adapter is in connection release status. However, connection release might not be detected in the connection destination JMS provider system. This causes inconsistency in the connection status and the next communication process might fail. In such cases, it is necessary to support 1 initialization of connection establishment status in the connection destination JMS provider system.

(1) Send option of Message Queue adapter

The message sending functionality of Message Queue adapter sets connection and sessions with JMS provider through the resource adapter according to the send message request from the business process. It also sends messages to JMS provider.

2. Functionality for Connecting to Various Types of Systems

The message types you can send are XML format messages and binary format messages. However, even if the sent message is in XML format, it is stored as a binary format message in the sending queue.

You can specify the send option (JMS provider) while sending messages. The following table describes the list of send options of Message Queue adapter:

Table 2–13: List of send options of Message Queue adapter

Item	Overview of send option	Specification method	
		Message	Definition
Message sending destination (JMSTDestination)	Message sending queue.	Y	Y
Message maintenance time (JMSExpiration)	Message maintenance time. Message maintenance time refers to the time of continuous maintenance in the sending queue of JMS provider.	N	Y
Message persistence (JMSDeliveryMode)	Message persistence. Message persistence refers to specification of whether to maintain a message in the queue after a system fault or when JMS provider is restarted.	N	Y
Message identification ID (JMSMessageID)	ID to identify a message. Any information is set if WebSphere MQ Resource Adapter is used.	-	-
Time stamp (JMSTimestamp)	Message time stamp.	-	-
Report message sending destination (JMSReplyTo)	Report message sending queue.	N	N
Message type (JMSType)	Message type.	N	Y
Correlation identifier (JMSCorrelationID)	Correlation identifier of message.	N	A
Priority (JMSPriority)	Message priority. 4 is set in WebSphere MQ Resource Adapter.	-	-

Legend:

Specification method:

Message: Existence of specification according to the request message from the business process.

Definition: Existence of specification according to communication configuration definition file of Message Queue adapter.

Y: Can be specified.

A: You can only specify the prefix of the message correlation identifier.

Message Queue adapter is auto-generated while sending a message correlation identifier.

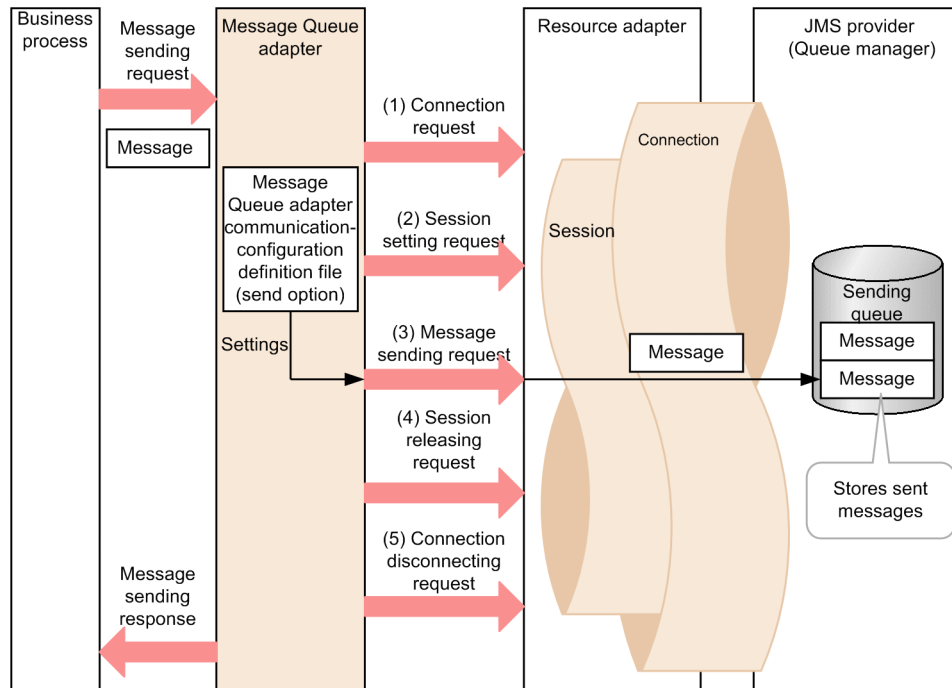
N: Cannot be specified.

-: Message Queue adapter is not set. It is dependent on resource adapter specifications.

For details on sending of messages by setting the message correlation identifier, see "2.10.2(2) Method of generating message correlation identifier".

The following figure gives an overview of the message sending functionality:

Figure 2-60: Message sending functionality



1. Message Queue adapter receives a message sending request from the business process and connects the connection.
 2. After connection, a session is set with the sending queue.
 3. Execute the message sending request and store the message in the sending queue. If you specify the send option in the communication configuration definition file of Message Queue adapter, the send option is set and the message is sent.
 4. Release the session after message sending is finished.
 5. Disconnect the connection with JMS provider.
- After the connection is disconnected, return the message sending response to the business process.

(2) Method of generating message correlation identifier

If Type2 is specified in the <id_type> tag of the communication configuration definition file of Message Queue adapter, Message Queue adapter automatically generates the message correlation identifier and provides it to the sent message.

Message correlation identifier is configured from the prefix and Message Queue adapter-specific information. There are 2 methods namely, when the generation method of message correlation identifier provides the prefix and when it does not provide a prefix.

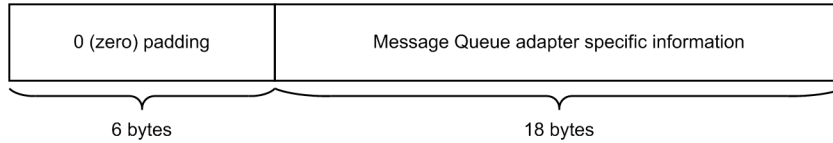
Message Queue adapter-specific information is set by Message Queue adapter.

(a) When there is no prefix

There is no prefix provided in the message correlation identifier when there is no specification in <JMSCorrelID_PFX> tag.

In such cases, the first 6 bytes have 0 (zero) padding and the message correlation identifier matches with the Message Queue adapter-specific information.

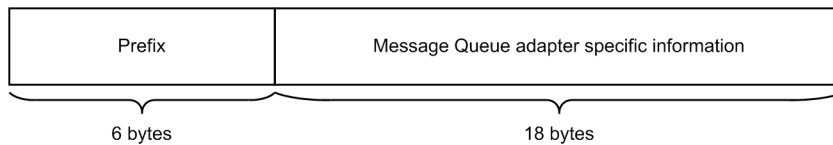
Figure 2–61: Method of generating message correlation identifier (without prefix)

**(b) When there is a prefix**

If the message correlation identifier is provided with a prefix and if the `<JMSCorrelID_PFX>` tag is specified, the `<JMSCorrelID_PFX>` tag value is a part of the message correlation identifier. You can specify a maximum of 6 bytes for the prefix. However, if the prefix is less than 6 bytes, there is 0 (zero) padding from the beginning to make the prefix 6 bytes. For example, if the specified value of the `<JMSCorrelID_PFX>` tag is "ABC", "000ABC" is set in the prefix part.

This prefix and Message Queue adapter-specific information match to form the message correlation identifier.

Figure 2–62: Method of generating message correlation identifier (with prefix)

**Reference note**

If message sending fails when the message correlation identifier is set, you can specify the sent message and message correlation identifier from data trace.

(3) Transactions to be performed while using Message Queue adapter

For the transaction support level provided in the resource adapter, specify "LocalTransaction" or "XATransaction". However, you can only specify "LocalTransaction" when business process status is not persistent. You cannot specify "NoTransaction".

If you specify "LocalTransaction" or "XATransaction", correlation occurs with JTA (Java Transaction API). For example, if the specified value of JTA transaction timeout is shorter than the message reception response monitoring time (receive_timeout) of the communication configuration definition file of Message Queue adapter, timeout occurs in the specified value of JTA transaction timeout.

If a request to receive messages or a request to send and receive messages is received from the business process, Message Queue adapter issues a request to receive messages. If the resource adapter cannot receive the message of this request to receive messages within the fixed time, timeout of the message reception response monitoring time occurs in the resource adapter. Message Queue adapter notifies the business process about fault response information according to the notification from the resource adapter. Moreover, a request to browse and receive a message is received from the business process and this request is made. However, even if the message does not exist in the reception queue, the business process is notified about fault response information.

For details on defining the process to be followed when a fault occurs, see "5.4.3 Defining Fault Handling" in "Service Platform Basic Development Guide".

2.11 Connecting to FTP Client and FTP Server

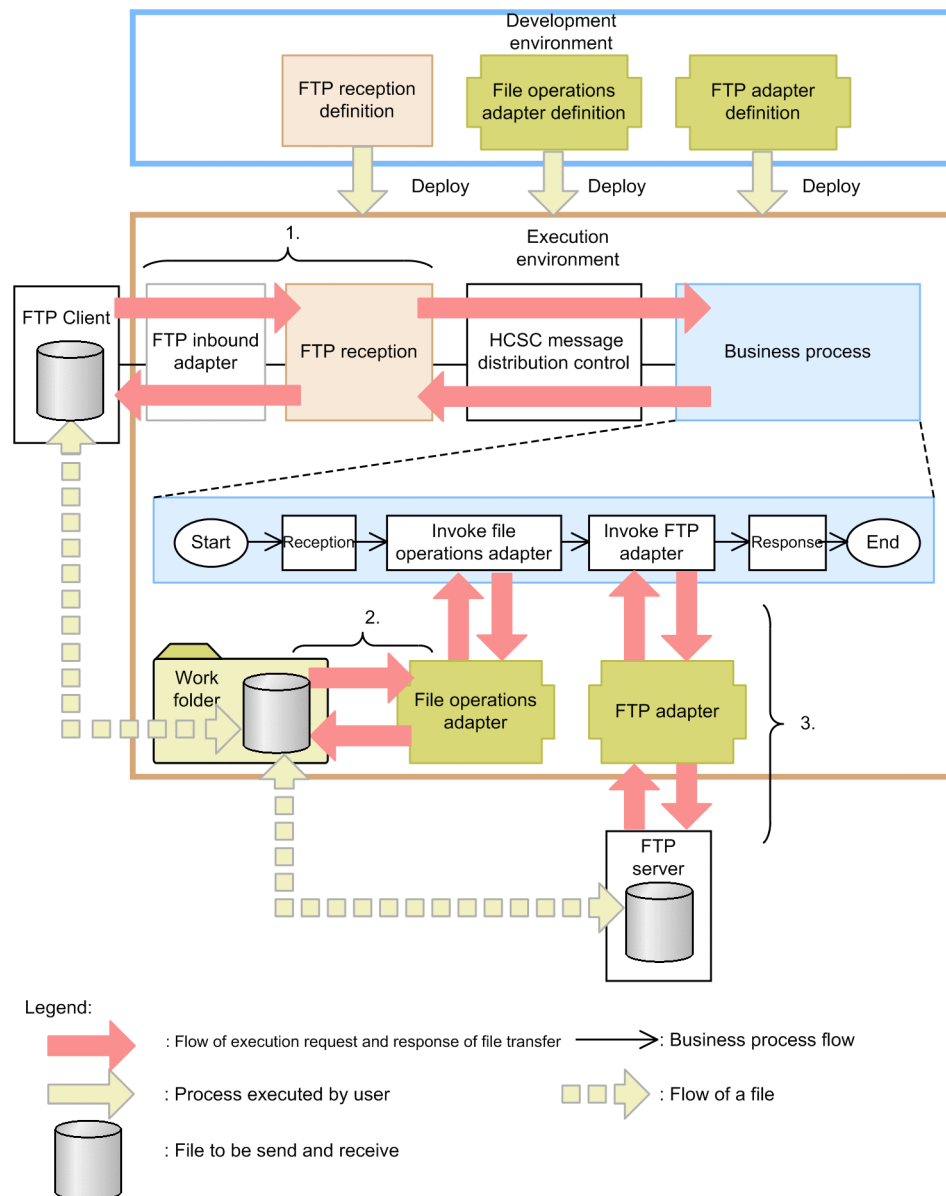
You can use FTP reception, FTP adapter and file operations adapter to broadcast sending and receiving of files using FTP between clients and servers in the service platform.

For details on FTP reception, FTP adapter, and file operations adapter functionality, see "8. Functionality to Transfer Files by Integrating with FTP (FTP Integration)". For details on FTP reception definitions, see "2.4 Defining FTP reception" in "Service Platform Reception and Adapter Definition Guide". For details on defining an FTP adapter and a file operations adapter, see "3. Defining Adapters" in "Service Platform Reception and Adapter Definition Guide".

2.11.1 File transfer integrated with FTP

The following figure shows the flow of file transfer using FTP:

Figure 2–63: Flow of file transfer using FTP



2. Functionality for Connecting to Various Types of Systems

1. Request reception from FTP client

FTP inbound adapter and FTP reception are used to receive file transfer requests from FTP client. Processes such as FTP client authentication, file transfer process are performed in FTP inbound adapter. File transfer requests from FTP client are sent to FTP reception through FTP inbound adapter. After that, the business process is invoked from FTP reception.

Files are sent and received using the working folder in HCSC server created by FTP reception.

2. Transforming, duplicating and deleting data in files

If the file formats of the sending side and receiving side are different, use the file operations adapter to transform file layout and the character code.

File operations adapter accesses the working folder and other folders on HCSC server and performs operations such as duplication and deletion of files between folders, deletion of working folders.

File operations adapter is invoked from the business process.

3. Connecting to FTP server

Use FTP adapter to connect to FTP server and for file transfer between the working folder and FTP server.

FTP adapter is invoked from the business process.

2.12 Connecting to the mail server

You can use the mail adapter to connect the mail server and Service Platform.

2.12.1 Preconditions

The preconditions to use the mail adapter are as follows:

- The mail server and CSC server must be deployed in the same LAN.
- The mail adapter supports RFC821-compliant mail servers that support the SMTP protocol according to the JavaMail specifications.

2.12.2 System using the mail adapter

The mail adapter can invoke a mail server that supports the SMTP protocol as a service in Service Platform.

You can use the mail adapter to send notification email messages to users and a system administrator when a business process ends normally or if a system failure occurs. The mail adapter can also be linked with FTP reception and an FTP adapter to add any file to email.

Examples of a system using the mail adapter are as follows:

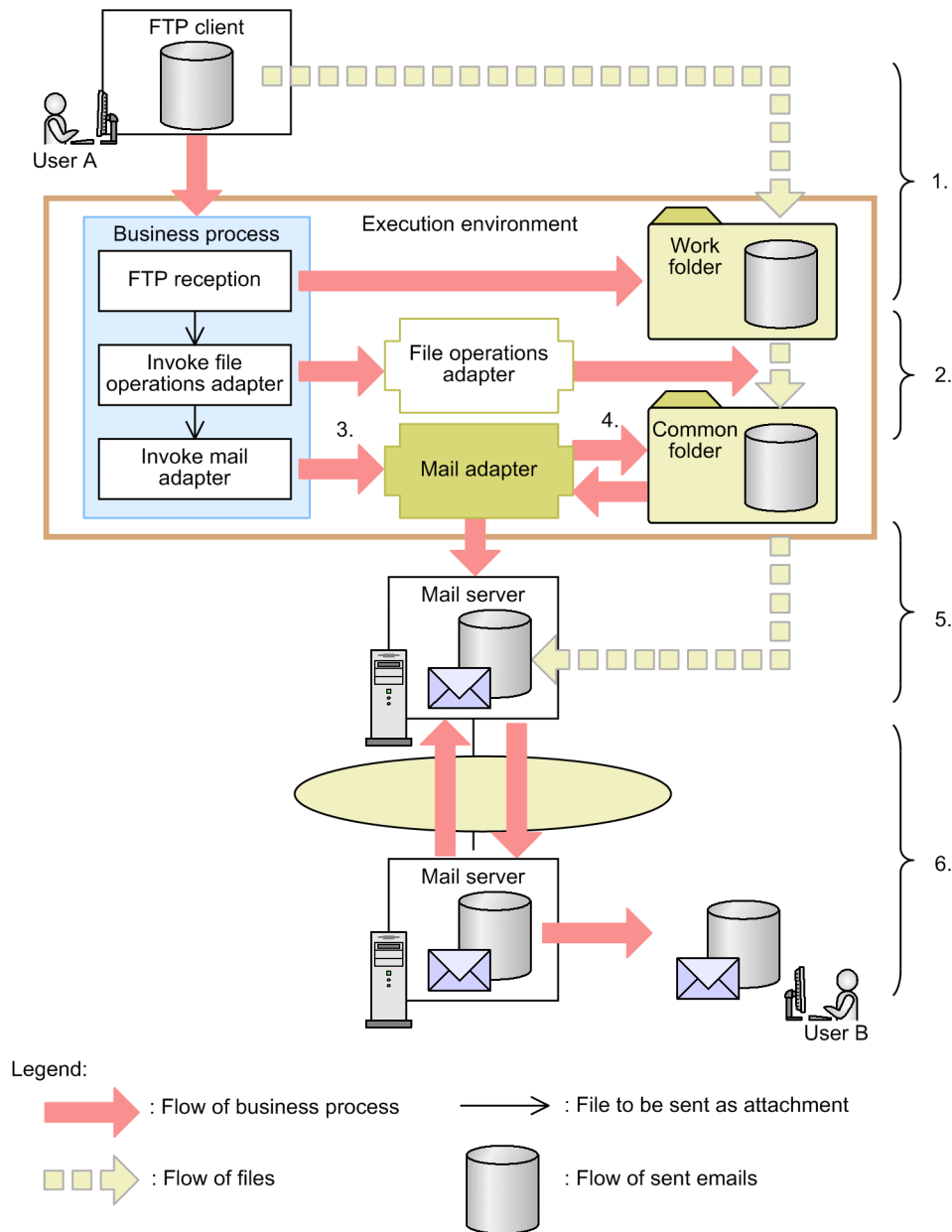
- Sending email with a file transferred from an FTP client attached
- Sending email with an attached file (acquired from an FTP server)
- Sending an authorization request for a workflow item by email
- Sending a completion notification of an asynchronous business process by email
- Sending email to notify that an error occurred during execution of a business process

Each example in the system is described below. For details about FTP reception and FTP adapter functionality, see *Chapter 8. Functionality to Transfer Files by Integrating with FTP (FTP Integration)*.

(1) Sending email with a file transferred from an FTP client attached

In this example, a user attaches a file transferred from an FTP client to an email message, and then sends it to another user.

Figure 2–64: Example of a system using the mail adapter (1)



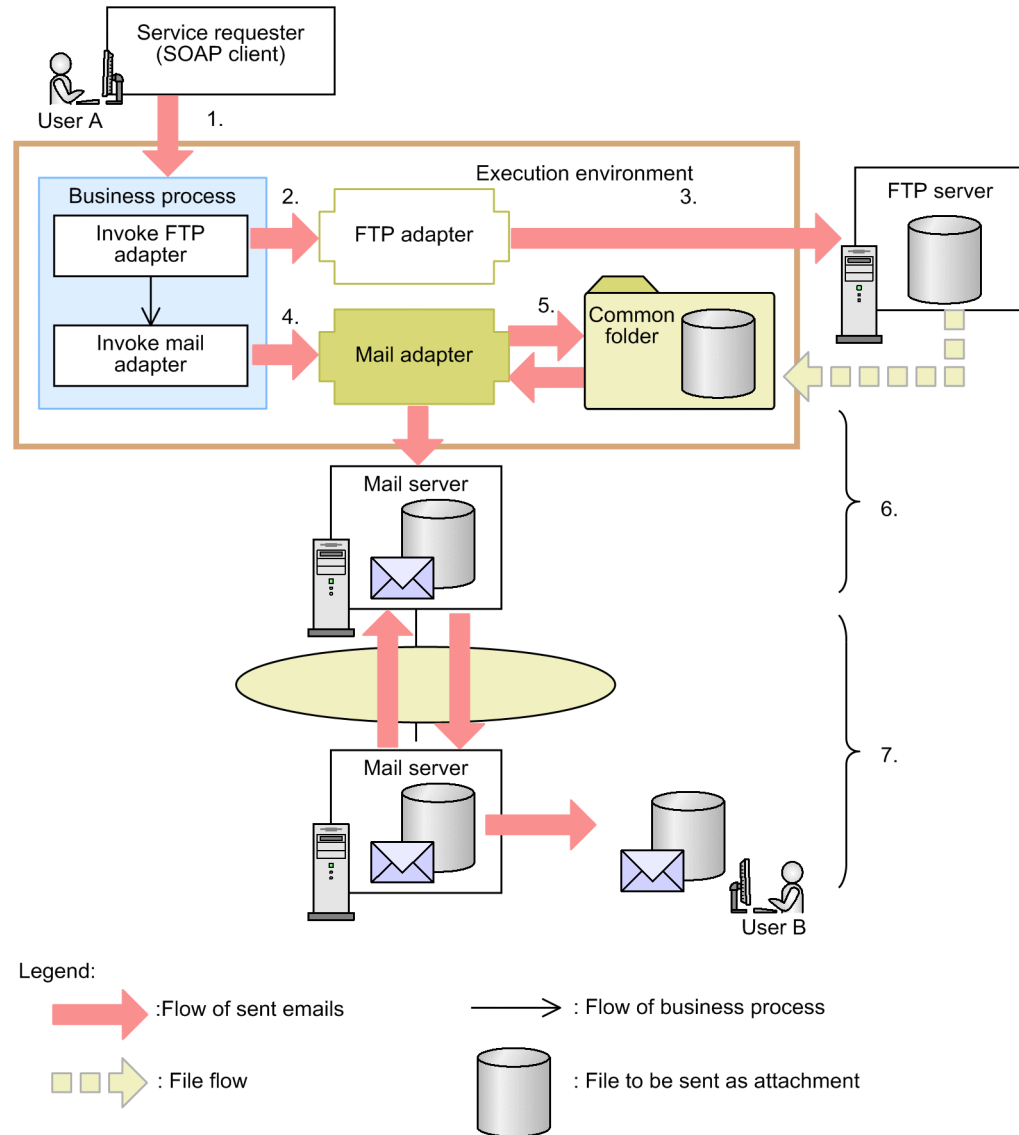
The figure below shows the flow of the email send process and the flow of a file. The following numbers correspond to the numbers in the figure.

1. The FTP client of user A transfers a file to Service Platform by using the PUT operation. The transferred file is saved in the work folder on the HCSC server created by FTP reception.
2. The file in the work folder is copied to the common folder by the file operations adapter invoked from the business process. If necessary, the file operations adapter converts the format or file.
3. The business process invokes the mail adapter.
4. The file in the common folder is set as an email attachment.
5. The mail adapter sends an email message to the mail server in the same LAN.
6. The email message with an attachment is sent from the mail server to user B through the network.

(2) Sending email with an attached file (acquired from an FTP server)

In this example, after the SOAP client starts a business process, a file acquired from the FTP server via the FTP adapter is attached to an email message, and then the email message is sent to another user.

Figure 2–65: Example of a system using the mail adapter (2)



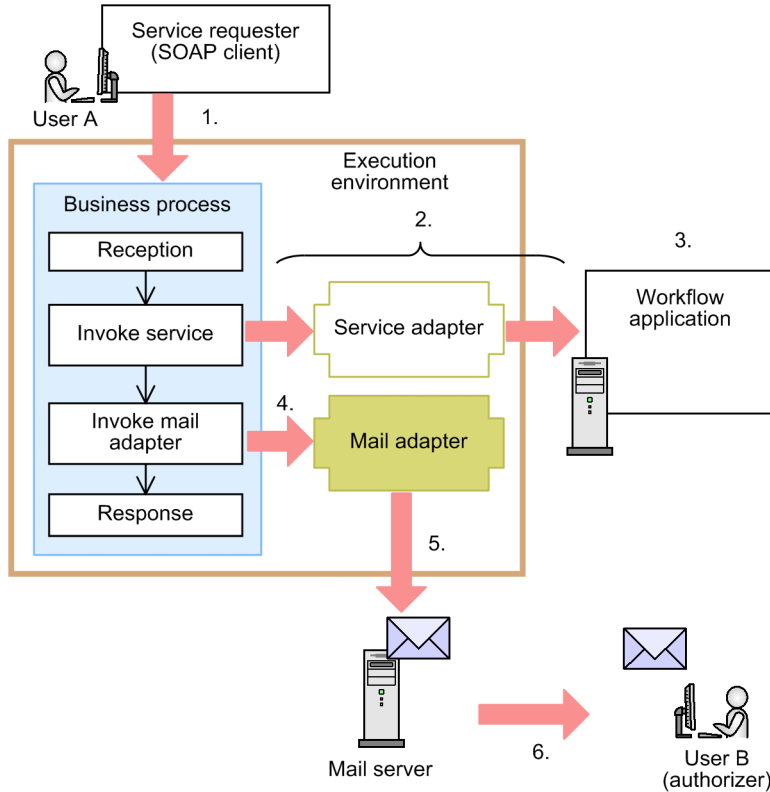
The figure below shows the flow of the email send process and the flow of a file. The following numbers correspond to the numbers in the figure.

1. The service requester (SOAP client) of user A invokes Web Services in Service Platform.
2. The FTP adapter is invoked from the business process.
3. The file is transferred from the FTP server to the common folder by the GET operation of the FTP adapter.
4. The mail adapter is invoked from the business process.
5. The file saved in the common folder is set as an email attachment.
6. The mail adapter sends an email message to the mail server in the same LAN.
7. The email message with an attachment is sent from the mail server to user B through the network.

(3) Sending an authorization request for a workflow item by email

In this example, after a workflow application is invoked from the business process via the service adapter, an authorization request email message is sent to the authorizer, and then the workflow process is performed.

Figure 2–66: Example of a system using the mail adapter (3)



Legend:



: Flow of sent emails



: Flow of business process

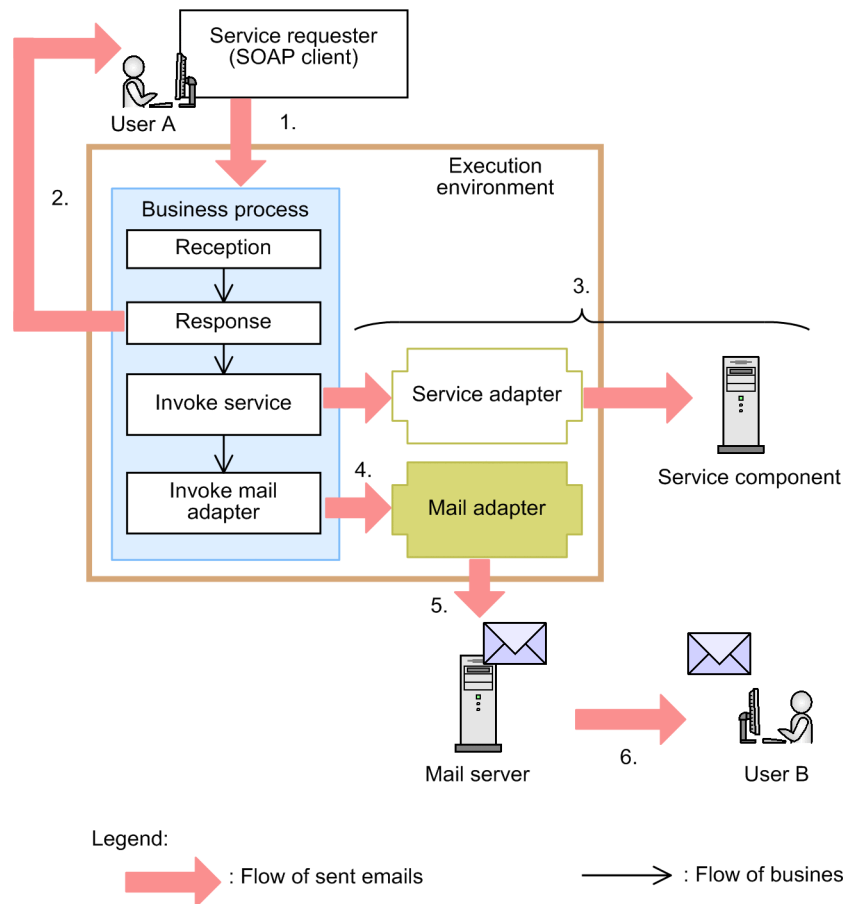
The figure below shows the flow of the email send process. The following numbers correspond to the numbers in the figure.

1. The service requester (SOAP client) of user A invokes Web Services in Service Platform.
2. A workflow application is invoked from the business process via the service adapter.
3. The workflow process starts, and then enters authorization process wait mode.
4. The mail adapter is invoked from the business process.
5. The URL of the authorization site is set in the email body.
6. After the email message is sent to user B (authorizer), user B clicks the URL in the body of the email message to perform the authorization process.

(4) Sending a completion notification of an asynchronous business process by email

In the case of asynchronous business processes, even if processing of a business process continues after a response is sent to the service requester, the user will not be notified of completion of subsequent processing. In this example, the mail adapter is used to notify the user by email that an asynchronous business process has completed.

Figure 2–67: Example of a system using the mail adapter (4)



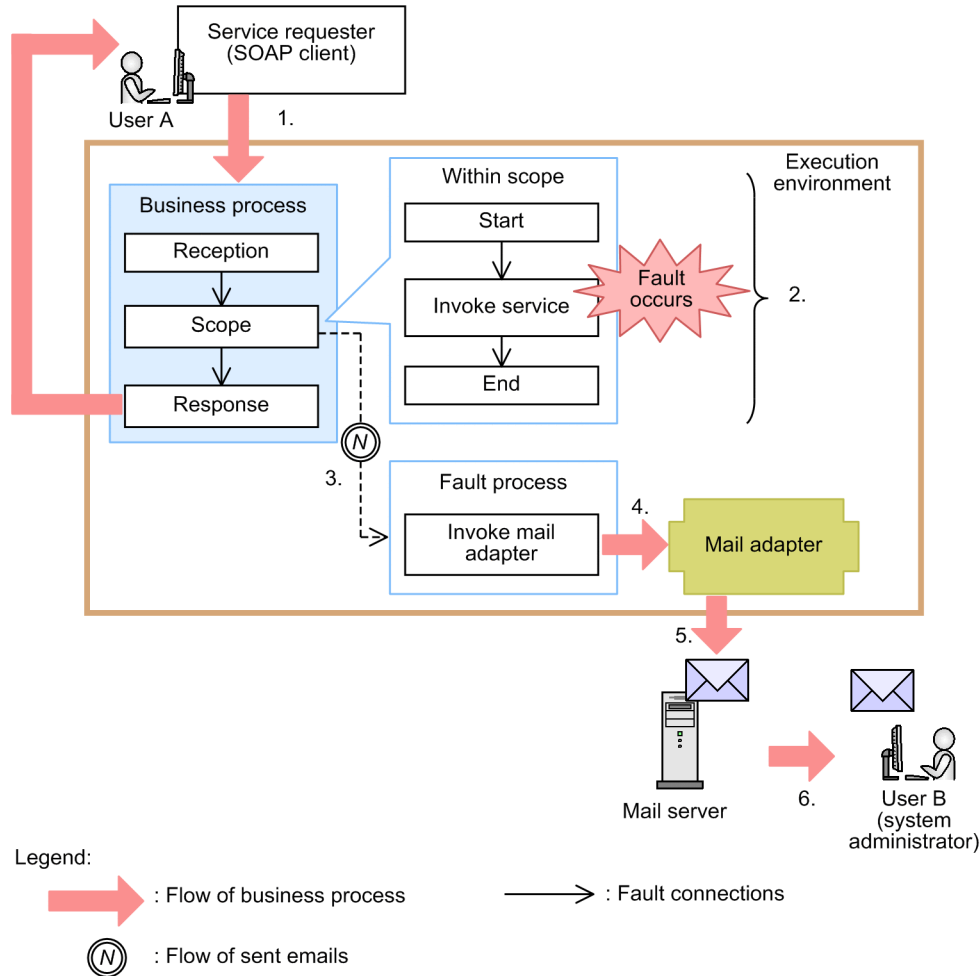
The figure below shows the flow of the email send process. The following numbers correspond to the numbers in the figure.

1. The service requester (SOAP client) of user A invokes Web Services in Service Platform.
2. A response message is returned to the service requester of user A.
3. The service adapter is invoked from the asynchronous business process, and then the service is executed.
4. The mail adapter is invoked from the asynchronous business process.
5. Text indicating that the service invocation process has completed normally is entered in the body of the email message.
6. After the email message is sent to user B, user B confirms that the process has completed.

(5) Sending email to notify that an error occurred during execution of a business process

In this example, if an error occurs during service invocation in the business process, the system administrator is notified of the error by email.

Figure 2–68: Example of a system using the mail adapter (5)



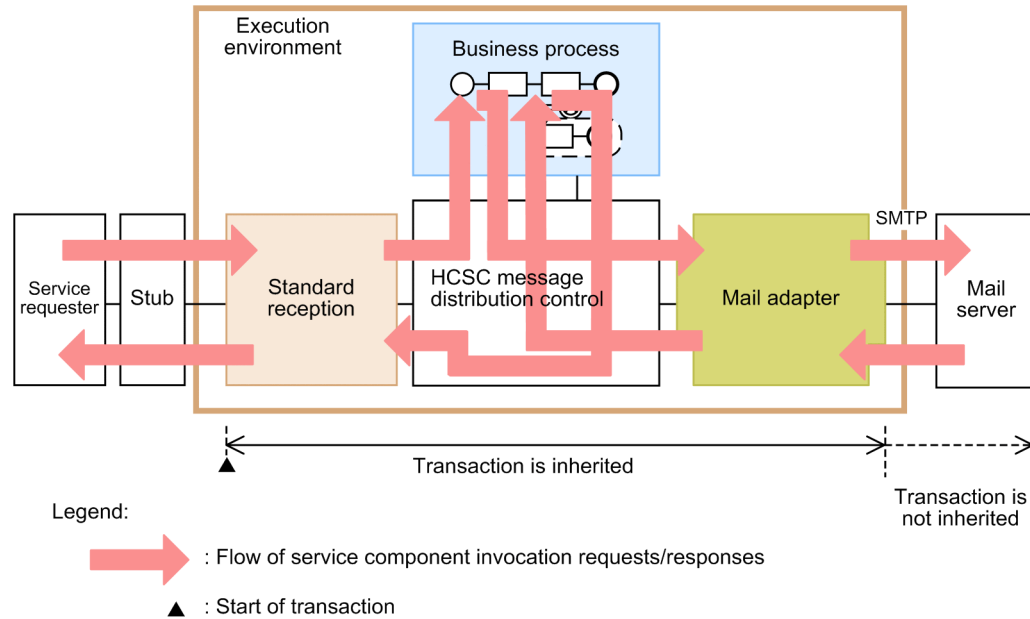
The figure below shows the flow of the email send process. The following numbers correspond to the numbers in the figure.

1. The service requester (SOAP client) of user A invokes Web Services in Service Platform.
2. An error occurs during service invocation within the scope of the business process, and then a fault message is returned to the service requester.
3. Processing of the business process moves to the fault process, which is the flow for handling errors.
4. The mail adapter is invoked in the fault process.
5. An error message is entered in the body of the email message.
6. After the email message is sent to user B (system administrator), user B confirms that an error occurred, and then performs troubleshooting.

2.12.3 Transactions when using the mail adapter

The following figure shows the range of a transaction when the mail adapter is used.

Figure 2–69: Range of a transaction when the mail adapter is used



If an error occurs in the processing of the business process after the processing to send email messages to the mail server is complete, the transaction is rolled back, and then an error is returned to the business process. However, the email messages sent to the mail server cannot be retrieved.

2.12.4 Mail information that can be set on the mail adapter

For the mail adapter, information required for sending an email message can be set in a request message or the mail adapter runtime environment property file. If information is specified in both the request message and the mail adapter runtime environment property file, the request message has priority.

The following table shows the mail information that you can set in the mail adapter.

Table 2–14: Mail information that you can set

Category	Subcategory	Settings
Address	Address type	Specify the address type (TO, CC, or BCC), and the email address to be specified as the destination.
Email subject	--	Specify the subject of the email message.
Body of the email message	Contents of the email body	You can use either of the following ways to set the body of the email message: <ul style="list-style-type: none"> Directly enter the email body Specify the path to the file containing the email body
	Email format	Specify the format of the email message. The mail adapter supports the following formats: <ul style="list-style-type: none"> Text HTML XML
	Encode	Specify the encoding used when sending the email message. If you do not specify any encoding, ISO-2022-JP is used.
	Character set	If you set the body of the email message by reading a file containing the email text, you can specify the character set of the text.

Category	Subcategory	Settings
Body of the email message	Character set	If you do not specify a character set, UTF-8 is used.
Attachment	--	Specify the path to the file to be attached to the email message.
Mail header	--	Specify the user-defined mail header options, such as the precedence of email and whether to send a response request.

Legend:

--: No applicable

For details about the items you can set in a request message, see the description of `adpmail_smtp_request.xsd` (request message format file) in 3.3.12 *Defining mail adapters* in the *Service Platform Reception and Adapter Definition Guide*. For details about the items you can set in the mail adapter runtime environment property file, see *Mail adapter runtime environment property file* in the *Service Platform Reference Guide*.

2.12.5 Managing the mail adapter timer

This subsection provides an overview of managing the mail adapter timer, and describes how to set timeout values.

You can set reconnection timeout and exclusive management timeout in the mail adapter. Each timeout value can be set in the mail adapter runtime environment property file. For details about the mail adapter runtime environment property file, see *Mail adapter runtime environment property file* in the *Service Platform Reference Guide*.

(1) Reconnection timeout

If the mail server goes down or if a network error occurs during connection to the mail server or during email sending, the mail adapter retries to connect to the mail server until the mail server connection timeout expires. You can also set the email send timeout, which is the time that can elapse before email sending fails.

The following table shows the parameters you can set.

Table 2–15: Mail server connection timeout and email sending timeout settings

Value	Key name	Default value
Mail server connection timeout (seconds)	<code>mailadp.smtp.connectiontimeout</code>	180
Email sending timeout (seconds)	<code>mailadp.smtp.timeout</code>	180

(2) Exclusive management timeout

When sending an email message with an attachment, if you access the path to an attachment in the common folder specified in the request message, a shared lock is set for the specified file.

A shared lock is not set if you specify the work folder in the request message or if you access the path to a file specified in the mail adapter runtime environment property file.

For exclusive management timeout, you can set the lock retry frequency and the retry interval used if the mail adapter fails to acquire the shared lock.

The retry status that exists when the mail adapter fails to acquire the shared lock is output to the maintenance log. For details about the maintenance log, see 7.3.1 *How to acquire log or trace output by Cosminexus Service Platform* in the *Service Platform Setup and Operation Guide*.

The following table shows the parameters you can set.

Table 2–16: Exclusive management timeout settings

Value	Key name	Default value
Lock retry frequency	<code>mailadp.read-lock.retry.count</code>	0
Lock retry interval (seconds)	<code>mailadp.read-lock.retry.interval</code>	1

2.12.6 Authentication when sending email messages from the mail adapter

This subsection describes how to specify SMTP authentication when sending email messages from the mail adapter, and how to set a user name and password necessary for authentication.

(1) Specifying SMTP authentication

Two types of SMTP authentication exist: LOGIN and PLAIN.

LOGIN

The user name and password specified in the mail adapter account definition file are used for authentication.

PLAIN

The user is not authenticated when sending an email message.

You can set the authentication type in either a request message or the mail adapter runtime environment property file. For a request message, specify with the `<mail-auth>` tag. For the mail adapter runtime environment property file, specify with the `mailadp.smtp.auth` key.

For details about the items you can set in a request message, see the description of `adpmail_smtp_request.xsd` (request message format file) in 3.3.12 *Defining mail adapters* in the *Service Platform Reception and Adapter Definition Guide*. For details about the items you can set in the mail adapter runtime environment property file, see *Mail adapter runtime environment property file* in the *Service Platform Reference Guide*.

If information is specified in both the request message and the mail adapter runtime environment property file, the request message has priority.

If information is not specified in the request message or the mail adapter runtime environment property file, PLAIN authentication is used for sending an email message.

(2) Setting the user name and password

The user name and password necessary for LOGIN authentication can be set by using a request message or property file. The following describes how to set the user name and password.

User name

You can set the user name in a request message or the mail adapter runtime environment property file.

For a request message, specify with the `<mail-user>` tag. For the mail adapter runtime environment property file, specify with the `mailadp.user` key.

Password

Use the `csmmailaddusr` command to add the password for the user name in the mail adapter account definition file.

2.13 Connecting to client using HTTP communication

You can use HTTP reception to invoke a business process by using HTTP communication from HTTP client.

If you use HTTP reception when there is a request for connection from a mobile terminal and Web browser to a service platform, invoke a business process directly and not through Web front system and SOAP reception.

! Important note

While using HTTP reception, connect from HTTP client to HTTP reception using IPv4.

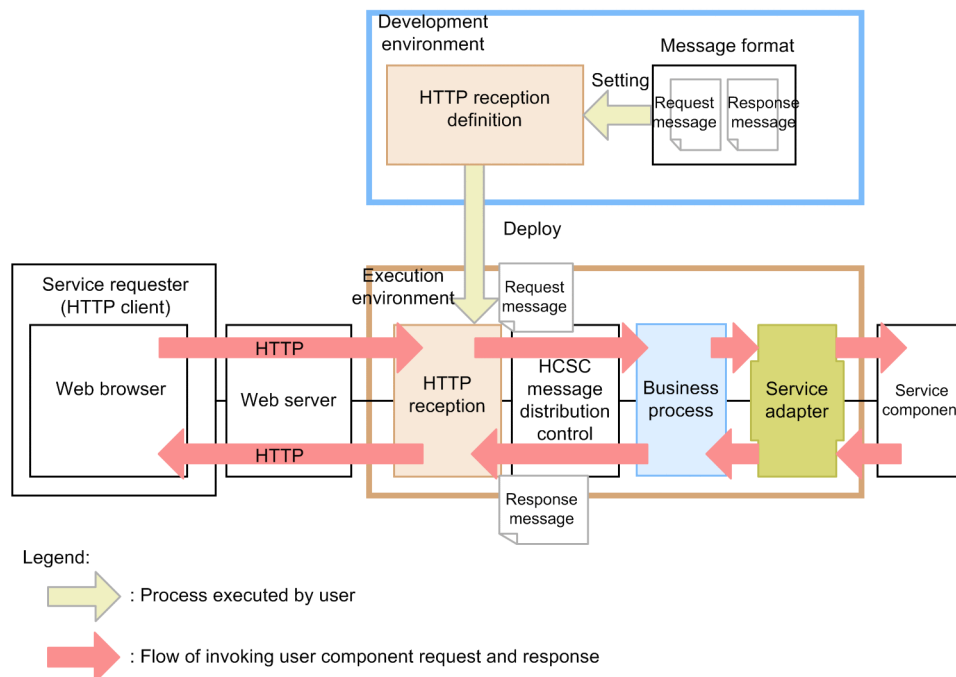
2.13.1 Invoking a service component using HTTP reception

HTTP requests sent from Web browser used as HTTP client are received in HTTP reception through Web server. After that, HTTP request is transformed to a request message (service component invocation request) and the service component is invoked in a business process.

During response, the response message in the business process is transformed to HTTP response format in HTTP reception and response returns to Web browser.

The following figure shows service component invocation request and response using HTTP reception:

Figure 2–70: Flow of service component invocation request and response using HTTP reception



Methods supported in HTTP reception

HTTP reception supports the following 2 methods:

GET method

This method is for passing query string information from HTTP client to the service platform and for downloading text data and file data.

POST method

This method is for sending text data from HTTP client to the service platform and for uploading file data. You can also receive text data as a response and download file data.

HTTP client and Web server supported by HTTP reception

The Web server and HTTP client supported in HTTP reception are described here.

Web server

Web server operating in Component Container compliant with HTTP 1.1 is supported.

HTTP client

The HTTP client supported by the above mentioned Web server is supported.

2.13.2 Example of online system using HTTP reception

This section is an introduction to HTTP reception usage method providing examples of systems handling text data, files and binary data.

(1) System in which text data is sent and received

This section describes examples of systems buying products online from Web browser of the tablet terminal.

Web browser used in this example is configured from [Login screen] to authenticate user information, [Input screen] to enter information such as the products to be bought and [Results screen] to display buying results. If a user buys a product, send a request from Web browser as per the following flow:

1. Send GET method request and acquire the list of products.
2. Send POST method request, send the products selected from the list and invoke the buy process.

The relation between each request process and HTTP reception is described here.

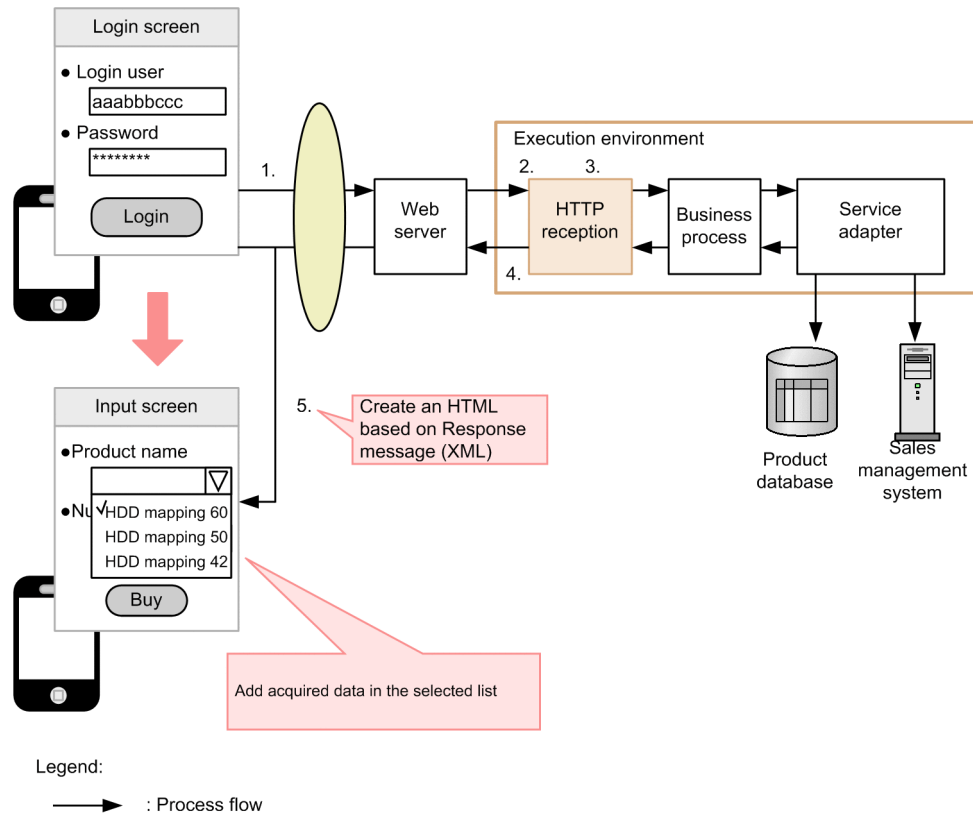
(a) Example of sending GET method request

In this request example, after the information entered in Web browser is sent to the service platform as GET method request, the screen generated on the basis of the corresponding HTTP response data is executed by the process displayed in Web browser.

The flow of sending GET method request is described here. The following numbers correspond to the numbers in the figure:

1. Information entered by the user in [Login screen] of Web browser is sent to Web server.
2. HTTP request is passed from Web server to HTTP reception.
3. HTTP header and HTTP body are transformed to request message (header) and request message (body) respectively in HTTP reception. Transformed data is passed to the business process in the service platform and various services such as product database are invoked from the business process. After that, the response message is transformed to HTTP response format in HTTP reception.
4. Response message in XML format is set in HTTP body of HTTP response.
5. HTML is created (using JavaScript or through front end server) on the basis of response message contents and it is displayed at client side. In this example, acquired data is added in the selection list of [Input screen].

Figure 2-71: Example of system using HTTP reception (if GET method request is sent)



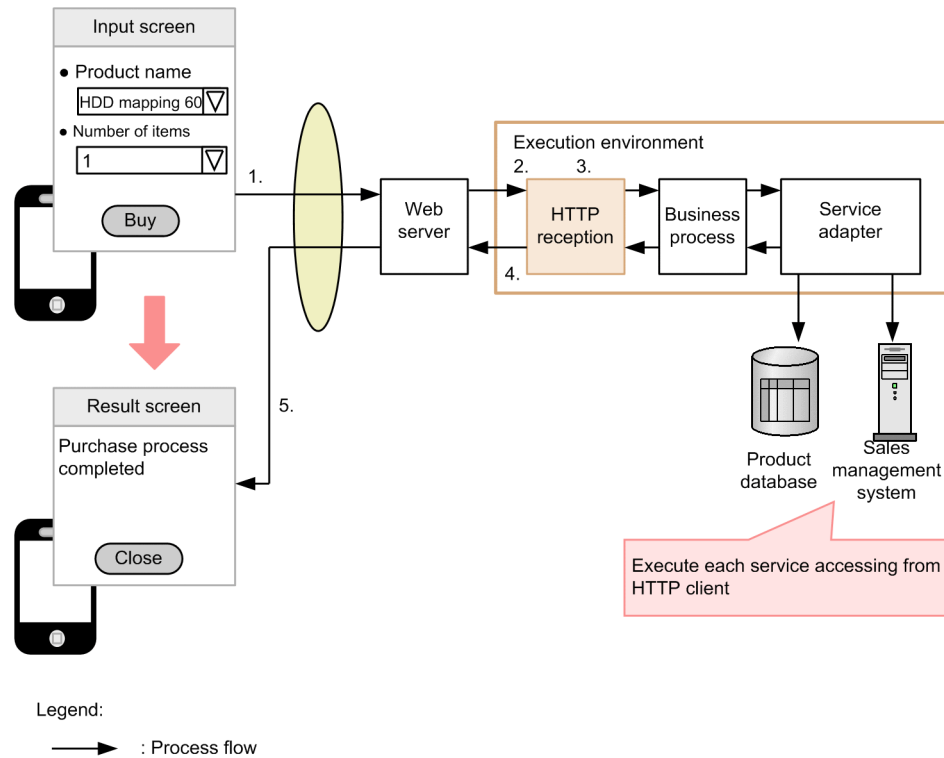
(b) Example of sending POST method request

In this request example, after the information entered in Web browser is sent to the service platform as POST method request, the screen generated on the basis of the corresponding HTTP response data is executed by the process displayed in Web browser.

The flow of sending POST method request is described here. The following numbers correspond to the numbers in the figure:

1. Data entered by the user in [Input screen] in Web browser is sent to Web server.
2. HTTP request is passed from Web server to HTTP reception.
3. Similar to the case in which GET method request is sent, HTTP header and HTTP body are transformed to request message (header) and request message (body) respectively in HTTP reception and various services are invoked through the business process. After that, the response message is transformed to HTTP response format in HTTP reception.
4. Response message in XML format is set in HTTP body of HTTP response.
5. HTML is created (using JavaScript or through front end server) on the basis of response message contents and it is displayed at client side. In this example, [Results screen] showing that the product buying process is complete is displayed.

Figure 2–72: Example of system using HTTP reception (if POST method request is sent)



(2) System in which files are sent and received

The following examples describe the system that sends and receives data between HTTP client, service platform, and servers using the file transfer functionality of HTTP reception.

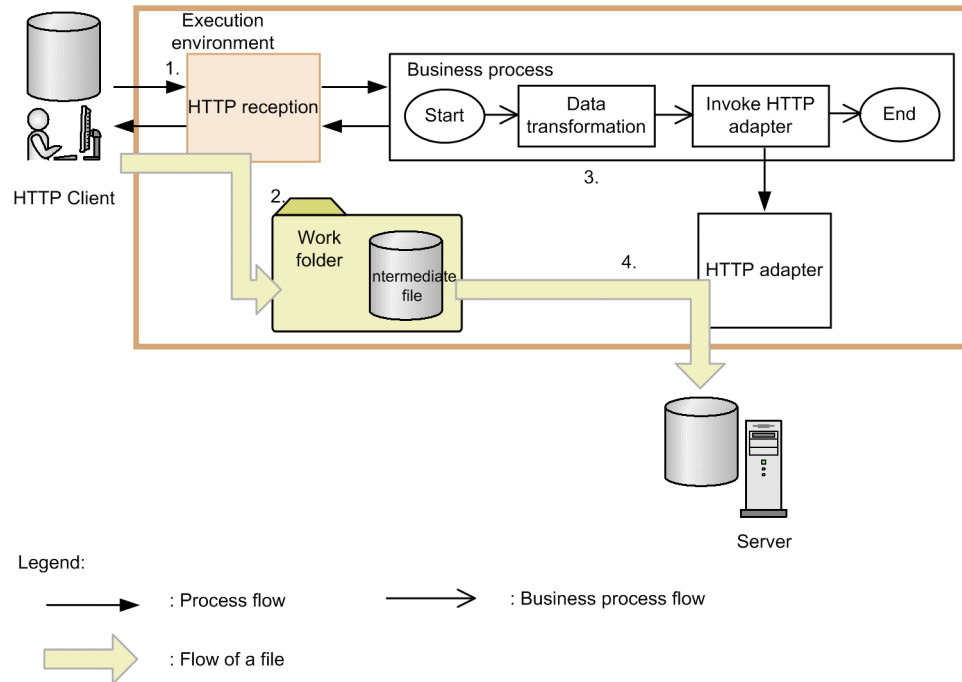
(a) Example of sending files from HTTP client to server

In this request, the process is executed to send files uploaded using POST method from HTTP client to the destination server from HTTP adapter invoked through a business process.

The flow of sending files from HTTP client to the server is as follows. The following numbers correspond to the numbers in the figure:

1. Send HTTP request by POST method from HTTP client program.
2. HTTP reception analyses the received HTTP request and creates a working folder to process files. Uploaded files are generated in the working folder as intermediate files.
3. Specify information of destination server and sent data (intermediate file saved in the working folder by HTTP reception) in HTTP adapter request message using data transformation activity in a business process.
4. HTTP adapter invoked from a business process sends a file from the working folder to the destination server.

Figure 2–73: Example of system using HTTP reception (while sending files from HTTP client to server)



For HTTP adapter functionality, see "2.14 Connecting to services using HTTP communication".

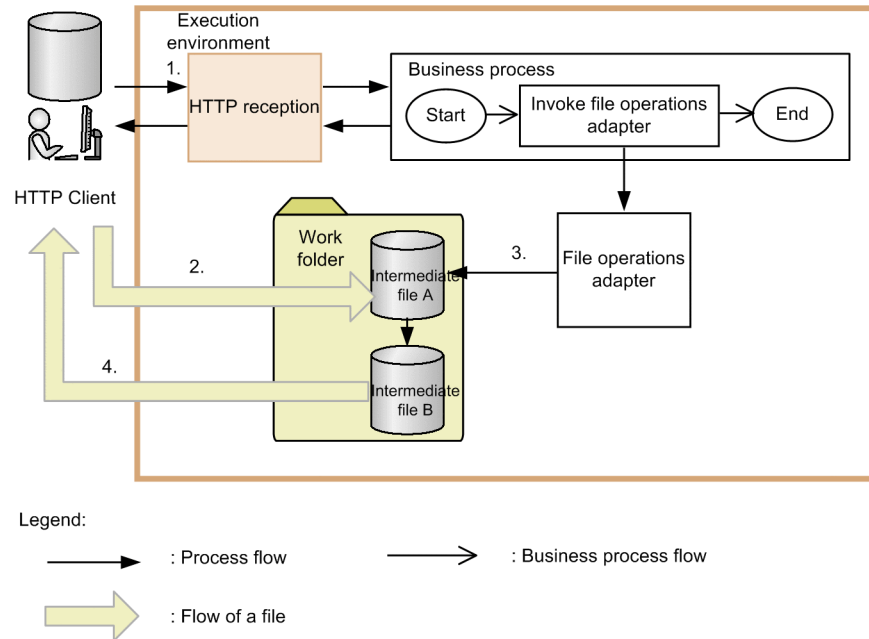
(b) Example of transforming files sent from HTTP client

In this request, files uploaded using POST method from HTTP client are transformed by the file operations adapter invoked through the business process and the process is then executed to download the files in HTTP client.

The flow of sending files using HTTP reception is as follows. The following numbers correspond to the numbers in the figure:

1. Send HTTP request using POST method from HTTP client program.
2. HTTP reception analyses the received HTTP request and creates a working folder to process files. The uploaded file is generated in the working folder as intermediate file A.
3. The file operations adapter invoked from the business process transforms the intermediate file A in the working folder to intermediate file B.
4. HTTP reception downloads the intermediate file B after transformation in HTTP client.

Figure 2–74: Example of system using HTTP reception (while transforming files sent from HTTP client)



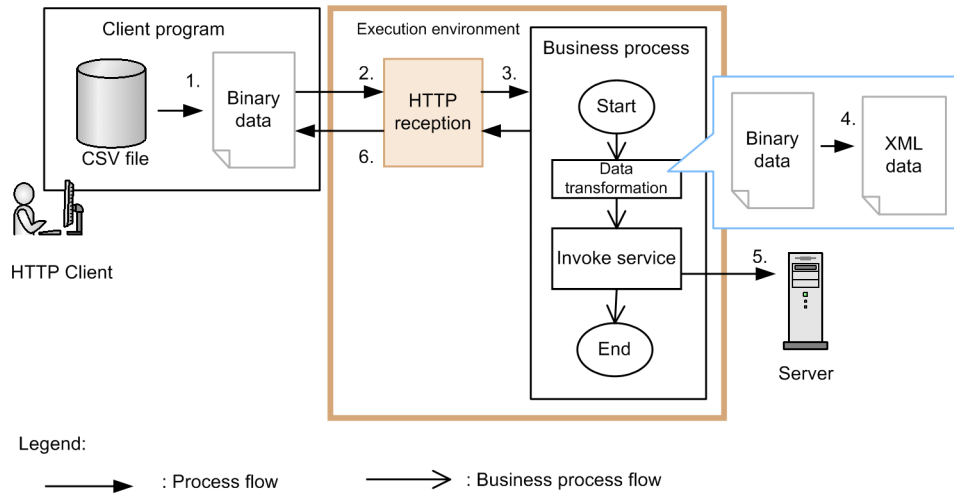
(3) System in which binary data is sent and received

In this request, a process is executed to send the binary data in which CSV format files are transformed to a business process from an HTTP client, as an example of processing binary data in a business process.

The flow of sending binary data from an HTTP client to a business process is described here. The following numbers correspond to the numbers in the figure:

1. CSV format files in HTTP client programs are transformed to binary data and this data is set in HTTP request body.
2. Send HTTP request using POST method from HTTP client program.
3. HTTP reception analyses the received HTTP request and invokes a business process.
4. Transform binary data set in the request message using the data transformation activity in the business process to XML data (XML message).
5. Use the invoke service activity and send XML data (XML message) to each service.
6. HTTP reception returns HTTP response to HTTP client.

Figure 2–75: Example of system using HTTP reception (while sending binary data from an HTTP client to a business process)



2.13.3 Example of HTTP reception configuration

In HTTP reception, you can execute various system configurations according to the following data format to be sent and received:

- To send and receive text data or binary data
- To send and receive file data

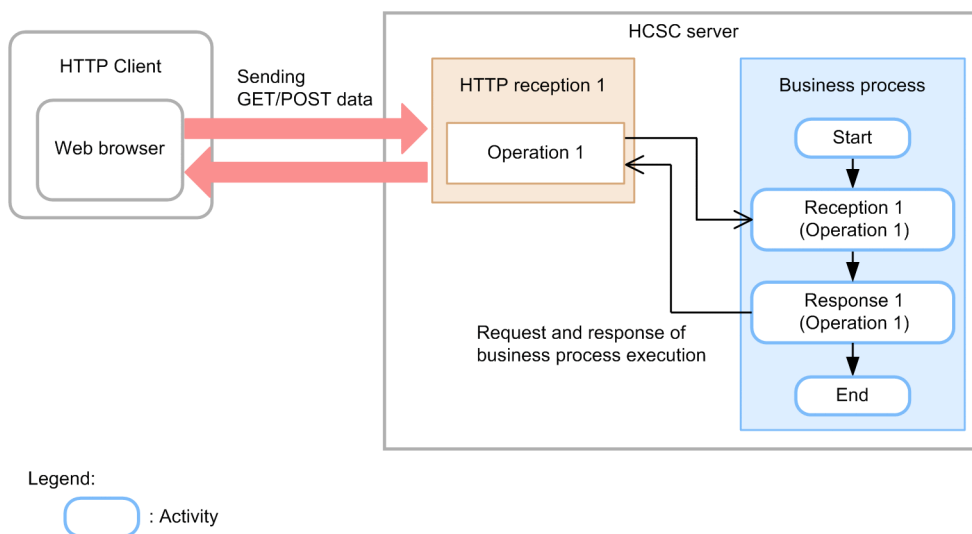
Examples of configuring HTTP reception and a business process are as follows:

(1) Sending and receiving text data or binary data

(a) Example of business process and single HTTP reception configuration

The following figure shows an example of single HTTP reception and business process configuration:

Figure 2–76: Correlating single HTTP reception to a business process

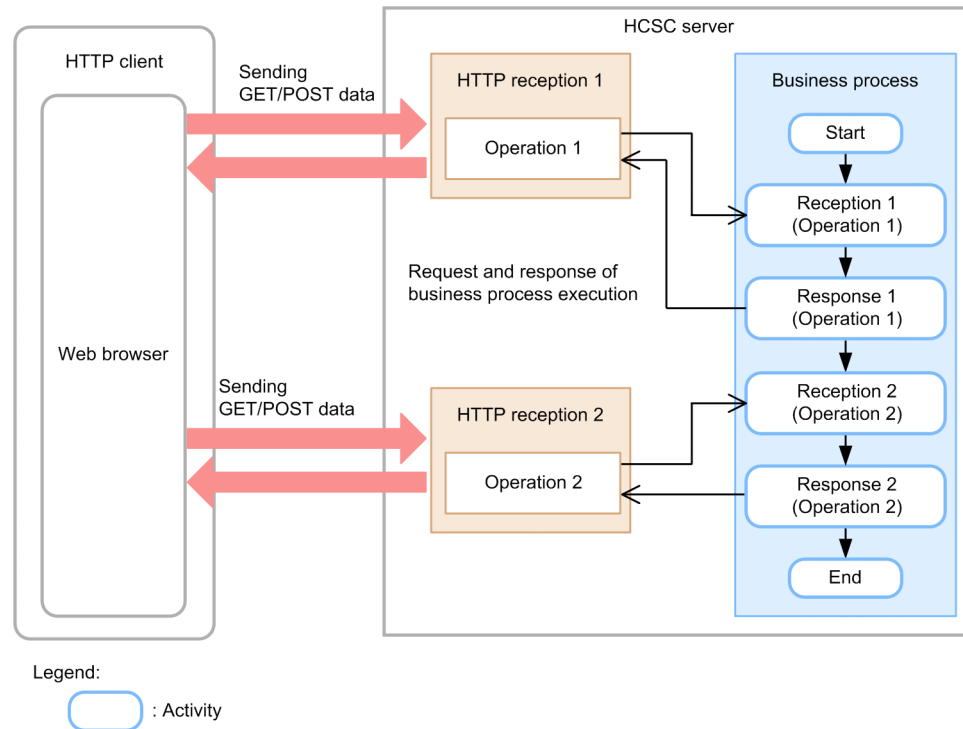


(b) Example of business process and multiple HTTP reception configuration

If multiple request information items are to be managed in the business process, you can correlate multiple HTTP receptions to a business process.

The following figure shows an example of multiple HTTP receptions and business process configuration:

Figure 2-77: Correlating multiple HTTP receptions to a business process



In this configuration, HTTP reception operation correlated to a business process is invoked according to the context root and operation name specified in URL of HTTP request.

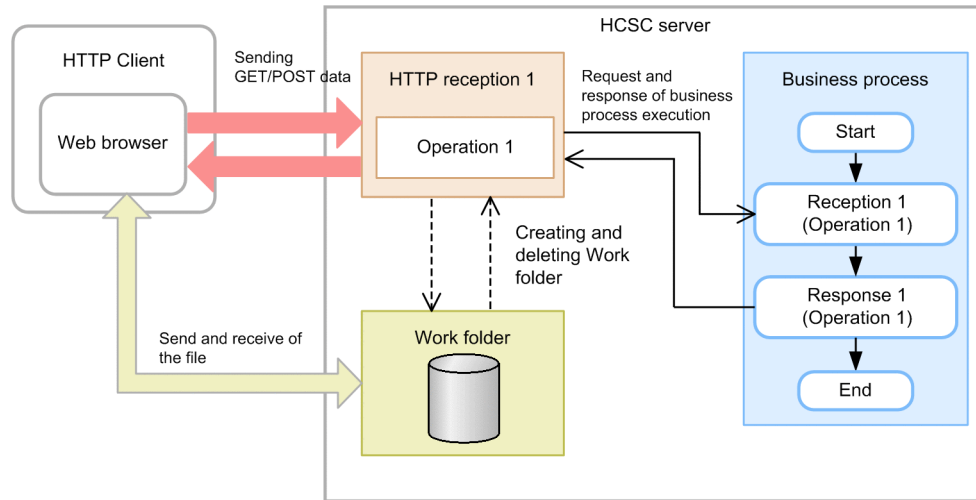
If you select this configuration, you must define it as a persistent business process.

(2) Sending and receiving file data

(a) Example of business process and single HTTP reception (single operation) configuration

The following figure shows an example of configuration of single HTTP reception containing a single operation and a business process:

Figure 2–78: Correlating single HTTP reception (single operation) to a business process



Legend:

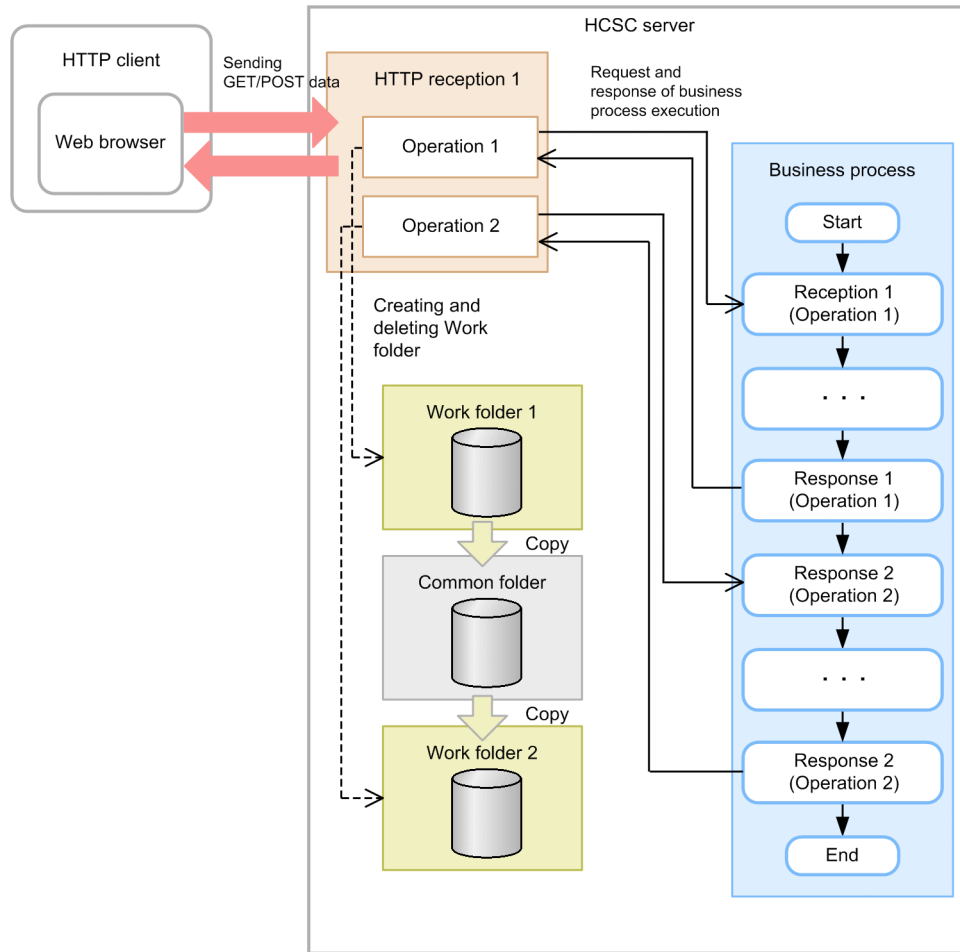
 : Activity

If HTTP reception receives HTTP request from HTTP client, create a working folder for processing file data in HCSC server as an intermediate file. The working folder is automatically deleted after HTTP response process is completed. If an error occurs while processing a business process or HTTP reception, the file is not deleted.

(b) Example of business process and single HTTP reception (multiple operations) configuration

The following figure shows an example of configuration of a single HTTP reception containing multiple operations and a business process:

Figure 2–79: Correlating single HTTP reception (multiple operations) to a business process



Legend:

: Activity

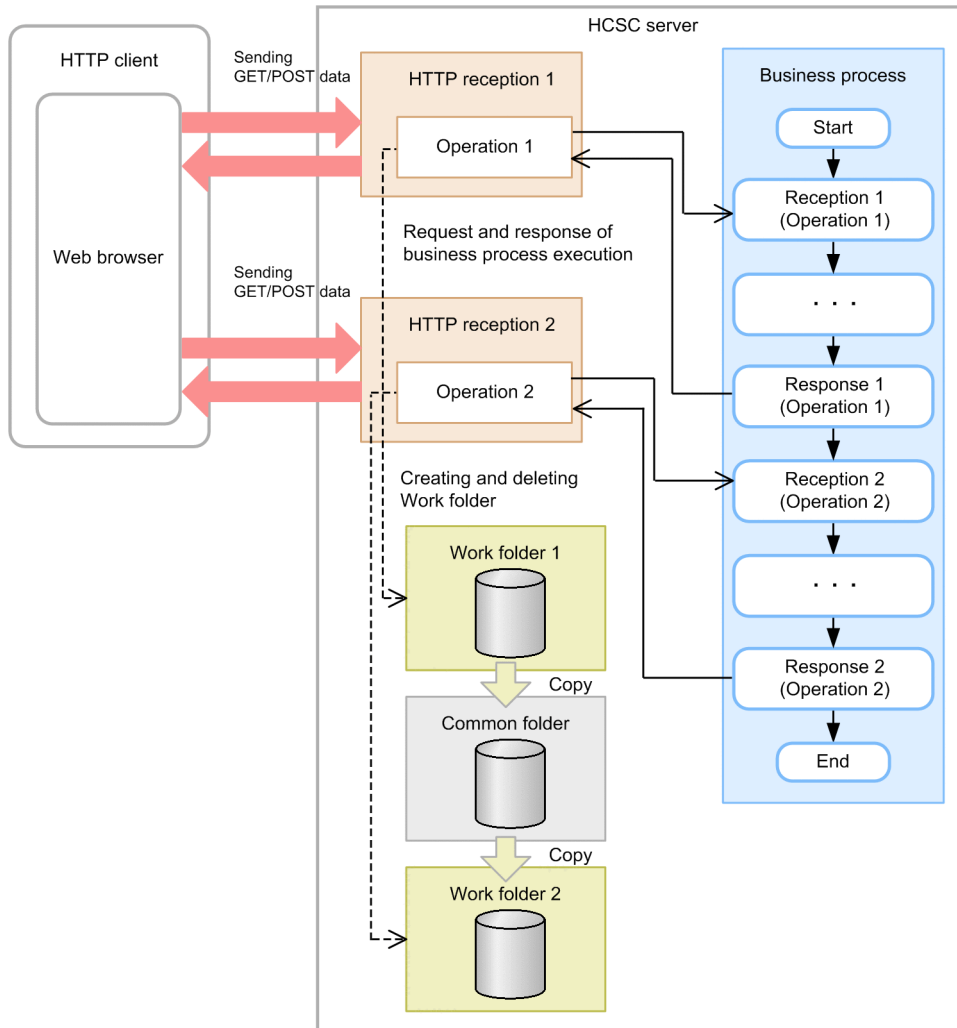
In this configuration example, the created working folder is different for each operation (HTTP request). If the file generated in the working folder by the 1st operation is to be referenced in the 2nd operation process, you must use the file operations adapter and copy the files between the working folder and common folder.

For file operations adapter, see "8.7 File operations adapter functionality". For common folders, see "8.8.2 File access using the common folder".

(c) Example of business process and multiple HTTP reception configuration

The following figure shows an example of multiple HTTP receptions and business process configuration:

Figure 2–80: Correlating multiple HTTP receptions to a business process



Legend:

 : Activity

In this configuration example, same as in case of single HTTP reception configuration containing multiple operations, the created working folder is different for each operation (HTTP request). Due to this, if identical files are to be shared in each HTTP request, you must use the file operations adapter and copy the files between the working folder and common folder.

2.13.4 HTTP reception request process

This section describes how HTTP request is processed when HTTP reception is executed.

HTTP request sent by HTTP client is partitioned in the following manner and it is processed in HTTP reception.

Request line

Request line is positioned in the first line of HTTP request message and HTTP method, request URL and HTTP version are stored.

HTTP request header

Information such as language, authentication information received by HTTP client is specified in HTTP header field.

HTTP request body

Message body to be sent is stored in POST method.

In HTTP reception, you can use the message format and definition file to process the request line of HTTP request, HTTP request header, and HTTP request body.

Name space prefix is omitted in the message example described here.

(1) Request line process

This section describes the request line process of HTTP reception.

(a) Context root

You can specify the context root of URL by `urecp-http.context-root` property of HTTP reception definition file.

If you do not specify `urecp-http.context-root` property, reception ID of HTTP reception set in the user-defined reception definition screen is used as the context root. For details on HTTP reception definition file, see "HTTP reception definition file" in "Service Platform Reference Guide".

While sending HTTP request, specify URL in the following format:

- `http://<Host name>:<Port number>#/<Context root>/<HTTP reception operation name>`
- `https://<Host name>:<Port number>#/<Context root>/<HTTP reception operation name>`

Note#

URL of Web server.

An example of URL specification is as follows:

For GET method

If `urecp-http.context-root` property is "Test001", specify URL in the following format:

```
http://<Host name or IP address>/Test001/get?type=A&size=7
```

"get" shows operation name and "type=A&size=7" shows query character string.

For POST method

If `urecp-http.context-root` property is "Test002", specify URL in the following format:

```
http://<Host name or IP address>/Test002/post
```

"post" shows operation name.

Operation name refers to HTTP reception operation name set in the user-defined reception definition screen.

Operation name is used when a business process is invoked.

If "/" is specified after the operation name and if there is a character string after "/", the part after "/" is also handled as the path. For example, if "http://localhost/contextroot/get/abc123?type=A&size=7" is specified in URL, "/contextroot/get/abc123" is stored as the path in location information (<location>).

! Important note

Context root is used as the directory name in the working directory of Component Container. Due to this, you must specify the context root so that the path length of the entire working directory does not reach the maximum in OS. For details, see "2.13.10 Points to be considered when using HTTP reception".

(b) Location information and query character string part

URL specified in the request line is partitioned into the following elements and stored in a request message (header):

- Location information (<location>)^{#1}
- Query character string part (<query>)^{#2}

For example, if the request line is "http://localhost/httprecp/get?type=A&size=7", transformation in the request message is as follows:

```
<url>
<location>/httprecp/get</location>
<query>type=A&size=7</query>
</url>
```

Note#1

Location information refers to the path from the context path to the extension path.

Note#2

If multiple "?" exist in the query character string, the first "?" is used as the separation character of the query character string.

(2) HTTP request header process

All information in HTTP header part received in HTTP reception is stored in http-header element defined in any type and passed to a business process.

An example of transforming HTTP header part is as follows:

- HTTP request header before transformation

```
Accept-Language: ja
Host: aaa.bbb.ccc.ddd
Connection: Keep-Alive
Authorization: Basic aG9nZTpmdWdh
```

- Request message (header) after transformation

```
<http-header>

<content-type>application/x-www-form-urlencoded; charset=utf-8</content-type>
<Accept-Language> ja</Accept-Language>
<Host> aaa.bbb.ccc.ddd </Host>
<Connection> Keep-Alive</Connection>
<Authorization> Basic aG9nZTpmdWdh</Authorization>

</http-header>
```

You can map XML element of HTTP header part transformed in HTTP reception by HTTP header part of SOAP adapter and any type.

(a) Transforming authorization header

Authorization header field of HTTP request header storing user authentication information is encoded in Base64 format. For this, HTTP reception decodes Authorization header field from Base64 format to plain format and sets in XML element of the request message.

Note

- If you specify both user name and password information in HTTP header part, only user name is set in HTTP reception.
- HTTP reception supports only BASIC authentication. If you specify other authentication types, user name is not set.
- If you specify Authorization header field of HTTP header part and do not specify a value, only a blank element of Authorization is generated as a request message and auth element and username element are not created.
- If you specify the password in Authorization header field of HTTP header part and do not specify user name, auth element and username element are not created. However, username element value is blank.

(b) Transforming Content-Type header

Store the media type of the value of received Content-Type header in the request message (header) as content-type element. At this stage, if there is a charset attribute, store as charset attribute of content-type element.

If there is no Content-Type header, content-type element is not generated. The attribute is not generated if you specify only media type of Content-Type header and there is no charset attribute.

Use charset attribute of Content-Type header as the text data character code of query or form data. If charset attribute of Content-Type header does not exist or if Content-Type header does not exist in the request, processing occurs according to the value of httprecp.http.charset property of HTTP reception definition file. However, in such cases also, charset attribute of content-type element is not generated because there is no charset attribute.

Note

If data is received in which content-type element of Content-Type header does not exist or in which charset attribute of content-type element is not generated, when it is necessary to check text data character code, determine using character code of httprecp.http.charset property value.

(3) HTTP request body process

Part classification and data format of HTTP request is processed in HTTP reception in the following manner, according to HTTP request header specification:

- Part classification of HTTP request
If Content-Type header of HTTP request header is multipart/form-data, it is handled as multi-part type. In other cases, it is handled as single part type (text data or binary data). The maximum number of multi-part types that can be processed in HTTP reception is 1,024.
- Data format of HTTP request
If filename attribute of Content-Disposition header exists in each part, it is handled as if the file data is uploaded. If there is no filename attribute, it is handled as text data or binary data.

HTTP request body process for each data format handled in HTTP reception is described here.

(a) When handling as text data

HTTP reception handles parts that do not have filename attribute in Content-Disposition header in query, form data, multi-part type format as text data. In case of handling as text data, after the part is stored in the request message (body) in XML format, it is passed to the business process as a body variable.

If HTTP request data format is only text data format, you can select one of the following modes as the mode of passing message body to a business process:

Standard mode

In a standard mode, message body is transformed to a request message element and passed to a business process. You can customize to any request message format ((details) for body variable) in XML format storing the message body.

Pass-through mode

In pass-through mode, the value corresponding to msg key in the message body is passed to a business process as is. The user must create the request message format in XML format according to the message body format.

Pass-through mode is used when HTTP reception is used as the dummy of SOAP reception while testing a business process.

If HTTP request is multi-part type, the request becomes invalid even if you specify pass-through mode.

You can select the mode while passing the message body to a business process in httprecp.switchover.pass-through.mode property of HTTP reception definition file. For details on HTTP reception definition file, see "HTTP reception definition file" in "Service Platform Reference Guide" .

The following table describes an example of transformation of standard mode and pass-through mode:

Table 2–17: Example of transformation of standard mode and pass-through mode

Mode	Description	HTTP request (message body)	Request message after transformation (body)
Standard mode	Message body is transformed from "key name = value" to "<Key name>Value</Key name>" format.	product1=uCSP&product2=HiRDB%20Single%20Server	<HTTPBody> <product1>uCSP</product1> <product2>HiRDB Single Server</product2> </HTTPBody>

2. Functionality for Connecting to Various Types of Systems

Mode	Description	HTTP request (message body)	Request message after transformation (body)
Pass-through mode	Specify message body part in "msg(fixed)=Value" format. [#] Value part is set in the request message (body) in XML format as in message body.	msg=%3cHTTPBody%3e%3cproduct1%3euCSP%3c%2fproduct1%3e%3cproduct2%3eHiRDB%20Single%20Server%3c%2fproduct2%3e%3c%2fHTTPBody%3e	<HTTPBody> <product1>uCSP</product1> <product2>HiRDB Single Server</product2> </HTTPBody>

Note#

You can set whether to omit msg key specification in httprecp.pass-through.parameter-use property of HTTP reception definition file.

If you specify multiple msg keys, operation is not guaranteed.

In a standard mode, HTTP request is sanitized in HTTP reception during request message transformation and then passed to the business process.

In a pass-through mode, HTTP request is not sanitized. The user must sanitize personally in advance.

(b) When handling as file data

If HTTP request part classification is multi-part type and data of the part in which filename attribute of Content-Disposition header exists is partitioned in each part in HTTP reception, the data is stored in the working folder as the intermediate file for each part.

If the data is processed as file data, header information of each part is stored as meta information of the file in the file element of the request message (header) as described in the following table:

Table 2-18: Header information of each part stored in request message (header)

HTTP request header	Header attribute	Optional/Mandatory	Description
Content-Type	-	Optional	Media type of file specified in each part. You can omit this header.
	char-set	Optional	Character code of file specified in each part. You can omit this attribute.
Content-Disposition ^{#1}	-	Mandatory	Header showing Disposition type. Always specify "form-data". ^{#2}
	name	Mandatory	Attribute showing the name of each part. Always specify this attribute. ^{#3}
	filename	Optional	Attribute showing data file name of uploaded file. You can omit this attribute. If you omit this attribute, the data is handled as text data.

Legend:

-: There is no applicable item.

Note#1

If you specify an attribute other than the attributes shown in Table2-26 such as size attribute, modification-date attribute in Content-Disposition header, the attribute value is ignored.

Note#2

While uploading file data as multi-part type, always specify form-data in Content-Disposition header. Only form-data is to be processed in HTTP reception.

Note#3

If you omit name attribute, data of the omitted part is not to be processed in HTTP reception.

! Important note

- If multi-part characters are used in name attribute and filename attribute in Content-Disposition header, operation is not guaranteed.
- If you specify any attribute beginning with "filename" in Content-Disposition header attribute and if you specify a file name (except the path part) containing "" in filename attribute value, operation is not guaranteed.

When data of multiple files is uploaded, this meta information is repeated in file element of the request message for each file and then it is stored.

(c) When handling as binary data

You can set binary data in HTTP request body to send binary format request messages to a business process. You can apply it to process CSV format data in the business process.

To handle the data as binary data in HTTP reception, it is necessary to meet all the following conditions:

- Pass-through mode is used (httprecp.switchover.pass-through.mode=true is specified in HTTP reception definition file)
- Setting is performed to omit msg key in pass-through mode (httprecp.pass-through.parameter-use=false is specified in HTTP reception definition file)
- Binary format (extension: fdx) or optional format (any format) is specified in the message format of the request message in HTTP reception
- Part classification of HTTP request body is a single part

Character encoding of binary format request messages is performed during data transformation of the business process.

2.13.5 HTTP reception response process

This section describes how to process the response message from a business process returned to HTTP reception as HTTP response.

HTTP response returning the response to HTTP client is partitioned in the following manner and it is processed in HTTP reception:

Status line

HTTP version, status code, and reason phrase are stored.

HTTP response header

Information such as Content-Length header, Content-Type header of HTTP response is stored.

HTTP response body

Text data or binary data returning response to HTTP client is stored.

(1) Status line process

HTTP response status line is output in the following format:

```
[HTTP version#1] [Blank] [Status code#2] [Blank] [Reason phrase]
```

Note#1

HTTP version is fixed in "HTTP/1.1".

Note#2

Value of status-code element of the response message (header) is set in the status code. For details on the status code, see "(4) Status codes".

(2) HTTP response header process

All header information specified in HTTP reception is set in HTTP response header.

HTTP response header process is described here.

(a) Content-Type and charset

Content-Type header and charset attribute used in HTTP response header differ according to the following conditions:

- If text data is returned
- If file data is returned
- If multiple data is returned (multi-part type)

If text data is returned

You can set Content-Type header and charset attribute used in HTTP response header in content-type element, http-header element, HTTP response header definition file of the response message (header).

If the response message format (for header variable) is set in the business process and multiple Content-Type headers and charset attributes are specified, specification of response message format (for header variable) is prioritized in the following manner:

Table 2–19: Content-Type and charset used in HTTP response header (if response message format (for header variable) is set)

Setting item	If content-type element exists in response message ^{#1}		If content-type element exists in http-header element of the response message		If content-type element and charset attribute do not exist in response message
	If charset attribute exists in content-type element	If charset attribute does not exist in content-type element	If charset attribute exists in http-header element	If charset attribute does not exist in http-header element	
Content-Type header	Value of content-type element of response message is valid.	Value of content-type element of response message is valid.	Value of content-type element of http-header element of response message is valid. ^{#3}	Value of content-type element of http-header element of response message is valid. ^{#3}	"text/xml" is set.
charset attribute	Value of charset attribute of content-type element of response message is valid. ^{#2}	Specified value of HTTP reception definition file is valid.	Value of charset attribute of http-header element of response message is valid. ^{#3}	Specified value of HTTP reception definition file is valid.	Specified value of HTTP reception definition file is valid.

Note#1

If the element exists and the value is blank, the element is considered as not existing.

Note#2

If the attribute exists and the value is blank, the attribute is considered as not existing.

Note#3

If multiple content-type elements are specified in http-header element of the response message (header), any 1 of the specified values is valid. However, there is no guarantee as to which value is set.

If you do not set the response message format (for header variable) in the business process, the value of HTTP response header definition file is valid as described in the following table:

Table 2–20: Content-Type header and charset attribute used in HTTP response header (if response message format (for header variable) is not set)

Setting item	If HTTP response header definition file exists			If HTTP response header definition file does not exist
	If content-type key exists in HTTP response header definition file		If content-type key does not exist in HTTP response header definition file	
	charset attribute exists in specified value of content-type key	charset attribute does not exist in the specified value of content-type key	charset key does not exist in HTTP response header definition file	
Content-Type header	Specified value of content-type key is valid. [#]	Specified value of content-type key is valid. [#]	"text/xml" is set.	"text/xml" is set.
charset attribute	Value of charset attribute existing in the specified value of content-type key is valid. [#]	Specified value of HTTP reception definition file is valid.	Specified value of HTTP reception definition file is valid.	Specified value of HTTP reception definition file is valid.

Note#

If you specify multiple content-type key and value pairs in HTTP response header definition file, any 1 of the specified values is valid. However, there is no guarantee as to which value is set.

HTTP response header definition file is any file invoked from HTTP reception definition file. For details on HTTP response header definition file, see "HTTP response header definition file" in "Service Platform Reference Guide".

If file data is returned

You can set Content-Type header and charset attribute used in HTTP response header in content-type element under file element of the response message (header) as described in the following table:

Table 2–21: Content-Type header and charset attribute used in HTTP response header

Setting item	If content-type element exists under file element of response message		If content-type element does not exist under file element of response message
	If charset attribute exists in content-type element under file element	If charset attribute does not exist in content-type element under file element	
Content-Type header	Value of content-type element under file element of response message is valid.	Value of content-type element under file element of response message is valid.	Not set.
charset attribute	Value of charset attribute of content-type element under file element of response message is valid.	Not set.	Not set.

Note

If the element exists and the value is blank, the element is considered as not existing.

If the attribute exists and the value is blank, the attribute is considered as not existing.

If multiple data items are returned (multi-part type)

If the data returned to HTTP client contains multiple items such as text data and data of 1 file or data of 2 files, regardless of specification of response message (header) and HTTP response header definition file, multipart/form-data is set in media type of Content-Type header. Content-Type header of each data is specified in the header of each part of multi-part type response body. Value specified in each data is as described in "If text data is returned" and "If file data is returned".

(b) Content-Disposition

For Content-Disposition header of HTTP response, the following values are set according to HTTP response part classification and the number of data items. If files are not downloaded, Content-Disposition header is not set in HTTP response.

Table 2–22: Value of Content-Disposition header set by HTTP reception

Part classification	Number of data items	Setting location	Value of Content-Disposition header	name attribute	filename attribute
Single part type	1 file	HTTP response header	attachment [#]	Not set.	Value of filename element of response message (header)
Multi-part type	1 file + text data	Header of each part of HTTP response	form-data	Value of part ID element of response message (header)	
	Multiple files				

Note#

You can change this by `httprecp.response.download.disposition-type` property of HTTP reception definition file.

If `http-header` element of response message (header) contains content-disposition element, the value is ignored. If Content-Disposition header is specified in HTTP response header definition file, the value is valid.

(c) Content-Length

If only text data returns in HTTP response, you can specify whether to generate Content-Length header showing contents length in HTTP reception by using `httprecp.response.generate.content-length` property of HTTP reception definition file. You can specify the character code to be used while calculating contents length in `httprecp.http.charset` property of HTTP reception definition file.

For details on HTTP reception definition file, see "HTTP reception definition file" in "Service Platform Reference Guide".

The following table describes the value of Content-Length header passed to HTTP client when specification exists and does not exist in HTTP reception definition file:

Table 2–23: Content-Length header passed to HTTP client

Value of <code>httprecp.response.generate.content-length</code> property	Existence of specification of content-length element of <code>http-header</code> element in content-length key or response message (header) of HTTP response header definition file	Value of Content-Length header passed to HTTP client
true	Y	Value specified in HTTP response definition file or response message (header) is overwritten by data length calculated by HTTP reception.
	N	
false	Y	Data length specified in HTTP response header definition file or response message (header) is set in Content-Length header. [#]
	N	Content-Length header is not passed to HTTP client.

Legend:

Y: If specified.

N: If not specified.

Note#

If `httprecp.response.generate.content-length` property of HTTP reception definition file is "false" and multiple content-length elements are specified in `http-header` element of the response message (header), or if multiple content-length key and value pairs are specified in HTTP response header definition file, 1 of the specified values is set. However, there is no guarantee as to which value is set.

Downloading of more than 1 file is invalid even option of sending Content-Length header to the client is set. Even if the file is not downloaded, Content-Length header is not set even if the setting is performed to ignore the response message (body) in ignore-bodymsg element of the response message (header) or httprecp.response.ignore-bodymsg property of HTTP reception definition file.

(3) HTTP response body process

This section describes HTTP response body process for each data format sent to HTTP client.

(a) In text data only

After processing of the business process is completed, the response message (body) variable returns to HTTP reception as XML type character string and is set in HTTP response body as text data. While setting, encoding occurs according to the specified character code.

If a system exception occurs, error message ID and message contents are set in the response message (body) element.

Text data returns only data that is the response message (body) part specified in the reply activity. Due to this, if multi-part type response is returned, text data exists in only 1 part.

Note

It is necessary to encode each character code specified in HTTP reception definition file for text data set in XML element.

You can set whether HTTP reception is to ignore (not set in HTTP response) the response message (body) received from the business process by ignore-bodymsg element of response message (header) or httprecp.response.ignore-bodymsg property of HTTP reception definition file. If both response message (header) and HTTP reception definition file are specified, specification of response message (header) is prioritized. If this element (or property) is specified and there is no file data to be downloaded, blank HTTP response body is returned to HTTP client.

(b) In file data

To download file data, specify the name of the file to be downloaded in file element of the response message (header). The following table describes the elements specified in the response message (header) and the corresponding HTTP response header:

Table 2–24: Element specified in response message (header)

Element name specified in response message (header)	Description	Corresponding HTTP response header (header corresponding to each part in multi-part type)	Value if omitted
partID	Part identifier name showing each part of multi-part type.	name attribute of Content-Disposition header	Intermediate file name specified in local-file-name element is set. Ignored even if specified if it is not multi-part type and name attribute is not provided.
file-name	Download file name during download in HTTP client.	filename attribute of Content-Disposition header	None (filename attribute is not set).
local-file-name	Intermediate file name to be downloaded in the working folder. If file with the specified name does not exist, error occurs.	None	Cannot be omitted.
content-type [#]	Specify download file media type.	Content-Type header	None (Content-Type header is not set).
charset	Specify download file character code.	charset attribute	None.

Note#

If specification of content-type element is omitted, Content-Type header of the file to be downloaded is not set in HTTP reception. If multiple files are to be downloaded or if file and text data are multi-part type, Content-Type header of HTTP response header is multipart/form-data and Content-Type header of each file data is specified in the header of each part.

Reference note

To download only 1 file without returning text data during HTTP response, you can set whether HTTP reception is to ignore text data specified in the response message (body). You can change this setting by ignore-bodymsg element of response message (header) or httprecp.response.ignore-bodymsg property of HTTP reception definition file. Set by the response message (header) to dynamically change the setting in each HTTP response.

If both response message (header) and HTTP reception definition file are specified, specification of response message (header) is prioritized.

However, even if the setting is to ignore the response message (body), it is necessary to set any variable in advance in the body allocation variable of the business process reply activity. For this, use of the dummy format (urecp_http_dummy_body_response.xsd) of the response message format (for body variable) provided by the service platform as the dummy for the body variable is recommended.

(c) In binary data

You can set binary data in HTTP response body and return the response to the client by specifying binary format or optional format (any format) in the message format of HTTP reception response message.

Binary data is set in HTTP response body only if all the following conditions are met:

- Setting is performed to use the response message (body) received from the business process in HTTP response body (httprecp.response.ignore-bodymsg=false is specified in HTTP reception definition file)
- Binary format (extension: fdx) or optional format (any format) is specified in the message format of HTTP reception response message
- file element does not exist in the response message (header)

! Important note

- If the setting is to ignore the response message (body) of the business process (httprecp.response.ignore-bodymsg=true is specified in HTTP reception definition file), binary data is not set in HTTP response body.
- If XML format (extension: xsd) is specified in the message format of HTTP reception response message, it is processed as text data.
- If file element exists in the response message (header), it is processed as multimedia data.
- Character encoding is not performed for binary data set in HTTP response body.

(4) Status codes

This section describes status codes of HTTP request and HTTP response.

(a) Status code during response

The value set for status code during response differs according to response timing and type. The following table describes the status code setting value and response timing:

Table 2–25: Set status codes

Timing	Response type	Specification of status-code element in response message (header)	
		Exists	Does not exist
Before invoking business process	If error occurs in HTTP reception verification	-	HTTP reception sets status code according to error status.
After invoking business process	In normal termination	Status code set in status-code element of response message (header) is used. ^{#1}	200 ^{#2}

Timing	Response type	Specification of status-code element in response message (header)	
		Exists	Does not exist
After invoking business process	In fault	Status code set in status-code element of response message (header) is used. ^{#1}	200 ^{#3}
	In system exception	-	Value of httprecp.system-exception.status-code property in HTTP reception definition file is valid. ^{#4}

Legend:

-: Not applicable.

Note#1

It is not verified whether status code has terminated normally or is in error. Due to this, if status code is set in error range (example: 400-599) in the response message (header), status code is returned as is in HTTP client.

Note#2

Default value if status code is not set.

Note#3

If status code is not set in fault, it is considered as normal termination and default value is set.

Note#4

If there is no specification in HTTP reception definition file, default value 500 is set.

(b) Status code during error occurrence

The following table describes the correspondence between message ID and status code when error occurs in HTTP request and HTTP response processing:

Table 2–26: List of status codes when error occurs in request processing

Message ID	Status when error occurs	Status code
KDEC80424	If an internal error occurs (error occurs during XML format transformation of HTTP header or HTTP body)	500
KDEC80438	If HTTP method is other than GET or POST	405
KDEC80439	If the operation name specified in URL does not match with the operation name specified in HTTP reception	404
KDEC80440	If an invalid character is specified in the query character string, HTTP body or HTTP header	400
KDEC80441	If msg key is not specified in the request parameter or if an upper case character is specified in pass-through mode	400
KDEC80442	If an error occurs in URL decoding according to the encoding specified in HTTP request header or HTTP reception definition file	500
KDEC80451	If a unique request ID cannot be generated due to one of the following factors: <ul style="list-style-type: none"> Acquisition failed of parameters such as time, date, HTTP client IP address, etc. to generate request ID A large number of requests are received from same IP addresses at same time and there is no ID that can be used 	500
KDEC80452	If the root directory of the working folder specified in HCSC server runtime definition file does not exist	500
KDEC80453	If failure occurs while creating a working folder	500
KDEC80454	If multi-part type HTTP request data reception or analysis fails	400
KDEC80457	If an error occurs during output of an intermediate file	500

Table 2–27: List of status codes when an error occurs in response process

Message ID	Status when an error occurs	Status code
KDEC80424	If an internal error occurs (acquisition of HTTP response body stream fails or an exception occurs in writing value in HTTP response body)	500
KDEC80443	If an error occurs in HTTP header or HTTP body analysis during reception	500
KDEC80444	If an error occurs during response message encoding	500
KDEC80445	If the status code returned from the business process is more than 3 digits or less	500
KDEC80458	If reading of an intermediate file fails	500
KDEC80459	If failure occurs while deleting a working folder	200

2.13.6 Managing HTTP request files

This section describes file management when file data is transferred in HTTP reception.

If HTTP reception receives file data, output the received data as an intermediate file in the working folder. To download the file during response, the user sends the intermediate file in the working folder specified in the response message (header) to each client.

(1) Creating and deleting working folders

The working folder temporarily stores data transferred by HTTP reception.

(a) Creating a working folder

When HTTP request is received from HTTP client, HTTP reception creates a working folder for each request.

You can set the directory creating the working folder (working folder root) in work-folder property of HCSC server runtime definition file. For details, see "HCSC server runtime definition file" in "Service Platform Reference Guide" .

When HTTP request is received from HTTP client, HTTP reception also generates a request ID for each request. The business process is processed by specifying the generated request ID.

Request ID is generated in the following format:

<Reception ID><HTTP client IP address><Date><Time><Additional number><Partition key><HCSC server name>

The following table describes each request ID item:

Table 2–28: Item of request ID

Item	Number of digits	Contents
<Reception ID>	8 digits	Reception ID of HTTP reception. If reception ID is less than 8 digits, "-" is set for the missing part on the right of reception ID.
<HTTP client IP address>	12 digits	IP address of request source HTTP client (10 number notation format of IPv4). If each byte of IP address is less than 3 digits, "0" is set for the missing part on the right of each byte.
<Date>	8 digits	The date on which the request is received by HTTP reception is set in yyyyymmdd format.
<Time>	9 digits	The time at which the request is received by HTTP reception is set in hhmmssSSS format.
<Additional number>	3 digits	Integer value for a unique folder name is set in serial numbers from "000". If <Date><Time><Additional number> overlaps an already existing request ID, 1 additional number is raised. If duplication occurs with an integer value between "001" and "999", change <Date> and <Time> and set "000" in the additional number.
<Partition key>	1 digit	Partition key showing that HCSC server name is added. Single-byte slash (/) is used.

Item	Number of digits	Contents
<HCSC server name>	Maximum 8 digits	Name of HCSC server in which HTTP reception is deployed. If HCSC server name is less than 8 digits, the missing part is not replaced.

Character strings excluding <Reception ID>, <Partition key>, <HCSC server name> from request ID become name of a working folder.

Reference note

Working folder is not created in default status for HTTP requests other than multi-part type. However, you can always set the option to create working folders regardless of multi-part type or not in HTTP response in order to download files in HTTP response for HTTP requests other than multi-part type. You can enable these settings in `httprecp.switchover.file-transfer.mode` property of HTTP reception definition file.

(b) Deleting a working folder

The working folder and intermediate files in the working folder are automatically deleted, if all the following conditions are met:

- Value of `httprecp.work-dir.auto-delete` property in HTTP reception definition file is true
- Value of status-code element of the response message received from the business process in which HTTP reception is invoked is 200 or specification of status-code element is omitted

If the process terminates abnormally and the business process is re-executed by `cscpireexec` command (process instance is re-executed), the working folder and intermediate files in the working folder are not deleted automatically. If the working folder is not deleted automatically, you must delete it using either of the following methods:

- If the value of `httprecp.work-dir.auto-delete` property in HTTP reception definition file is false and the business process is asynchronous type
Invoke the file operations adapter from the business process and delete the working folder and intermediate files. See the working folder by `cscfswls` command, check whether it is to be deleted and then delete the working folder using `cscfswrm` command.
- In other than the above
See the working folder using `cscfswls` command, check whether it is to be deleted and then delete it using `cscfswrm` command.

(2) Creating and deleting temporary files

While sending file data in HTTP request, a temporary file is created for processing to avoid memory pressure. The folder in which temporary files are stored is always created in the following location when HTTP reception starts:

<Working folder root>\<HCSC server name>\<Reception ID>\<00000000000000000000000000000000>

! Important note

Temporary files are automatically deleted at the end of the request process. However, temporary files are not deleted in the following cases:

- If the temporary file is open
- If J2EE server did not stop normally
- If the system is down

If a temporary file is not deleted, delete the folder in which temporary files are stored using `cscfswrm` command. Delete the folder in which temporary files are stored after stopping HTTP reception. If you delete it when HTTP reception is running, error occurs in all HTTP requests sending file data.

If output in the temporary file is not required in cases such as when small-sized data is sent, you can also set the threshold value of the size for whether to output each part of multi-part type in the temporary file. You can set this threshold value in `httprecp.file-trans.temp-file.partsize-threshold` property of HTTP reception definition file.

(3) Setting maximum size of HTTP request

You can set the maximum size in multi-part type HTTP request to prevent large-sized invalid data from being sent. If HTTP request size exceeds the maximum, request processing is cancelled and error returns in HTTP client.

You can set the maximum size for the following items:

- Whole request
You can set maximum size for the whole HTTP request containing HTTP header and HTTP body (header and body part of whole part). You can set the maximum size of the whole request in `httprecp.file-trans.maxsize.request` property of HTTP reception definition file.
- Part unit
You can set the maximum size of the body part specified in each part unit. Text data also has a similar maximum size. You can set the maximum size of the part unit in `httprecp.file-trans.maxsize.part` property of HTTP reception definition file.

(4) Points to be considered related to path length

If there are restrictions in path length due to OS, set a suitable length of a working folder root path. The following table describes each path length:

Table 2–29: Length of working folder root path

Item	Path length
Working folder root	Set in working-folder property of HCSC server runtime definition file.
HCSC server name	Maximum 8 digits.
Reception ID	Maximum 8 digits.
Working folder name	Fixed at 32 digits.
Intermediate file	Maximum 23 digits.
Name of the folder in which temporary files are stored	Fixed at 32 digits.
Temporary file	Maximum 53.

2.13.7 Encoding and decoding HTTP requests and HTTP responses

This section describes encoding and decoding of HTTP request and HTTP response.

The character codes used during HTTP request, HTTP response and system exceptions are described here.

In HTTP request

You can specify character codes to decode the query character string and message body using the following methods:

- Method to specify by charset attribute of Content-Type header of HTTP request header
- Method to specify by `httprecp.http.charset` property of HTTP reception definition file

If both HTTP request header and HTTP reception definition file are specified, specification of HTTP request header is prioritized.

In HTTP response/in system exception

For character codes used in message body encoding, see "2.13.5(2)(a) Content-Type and charset".

Note

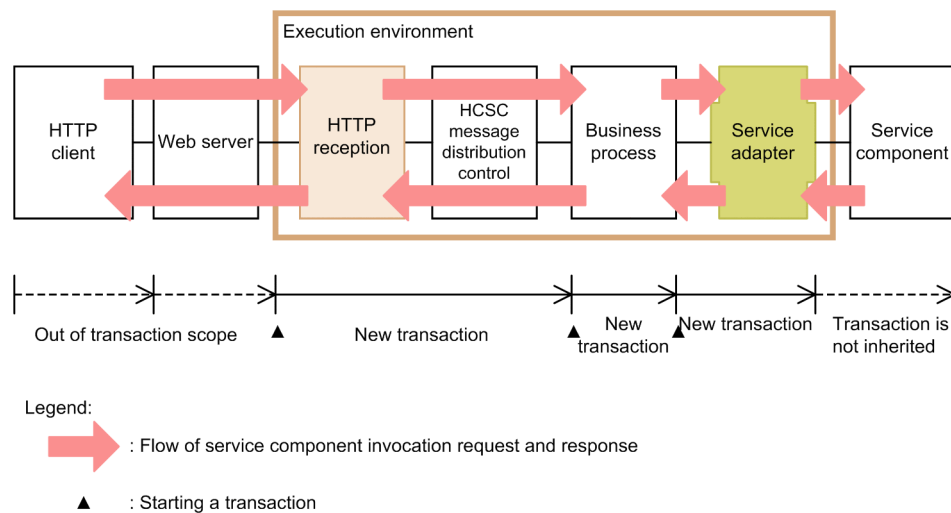
- The same character codes are used to decode URL and message body.
- For response message (body) data used as text data, the character code set in encoding attribute of XML declaration tag is the same as the character code set in HTTP response body and HTTP response header.
- User needs URL encoding for HTTP header value in the following cases:

- To send HTTP request header containing Japanese from HTTP client to HTTP reception
- If the response message (header) contains Japanese HTTP response header
- If the value of HTTP response header definition file contains Japanese HTTP response header
- HTTP header value is not encoded and decoded when HTTP reception passes HTTP request from HTTP client to the business process. HTTP header value is also not encoded and decoded when the response is returned from the business process to HTTP client.
- In pass-through mode, always specify "UTF-8" in the encoding attribute of XML declaration tag of the request message to be sent.
- While transforming a request message in binary format or optional format (any format) generated from binary data to XML format message, you must use data transformation activity for decoding.

2.13.8 Transactions to be performed while using HTTP reception

The following figure shows the range of transactions while using HTTP reception:

Figure 2–81: Range of transactions while using HTTP reception



If HTTP reception is invoked from HTTP client, a new transaction that is different from HTTP client and Web server transactions is started in HTTP reception. A new transaction that is different from HTTP reception transaction is also started in the business process.

2.13.9 Managing HTTP reception timer

In HTTP reception, you can use transaction timeout for time management of the entire HCSC server.

In transaction timeout, you can set the time from invocation of the business process by HTTP reception till the response is returned in HTTP reception in second unit. If processing time exceeds the timeout setting value, processing of the business process is interrupted.

You can set transaction timeout value in HTTP reception definition file.

The following table describes the parameters you can set:

Table 2–30: Setting transaction timeout

Value	Key name	Default value
Transaction timeout (seconds)	urecp-http.ejb-transaction-timeout	0 [#]

Note[#]

If 0 is set, operation is to be performed using the default timeout set in J2EE server.

2.13.10 Points to be considered when using HTTP reception

- HTTP reception does not support IPv6. Connect from HTTP client to HTTP reception by IPv4.
- Set a valid HTTP header to set HTTP response header in http-header element of response message (header). HTTP reception does not verify whether HTTP header is valid.
- HTTP reception does not support HTTP session management.
For session management functionality of the application server, see the following locations:
 - "2.7 Session management functionality" in "Application Server Web Container Functionality Guide"
 - "6. Database Session Failover Functionality" in "Application Server Expansion Guide"
 - "6. Migration Function of Expansion Functionality (Memory Session Failover Functionality)" in "Application Server Maintenance and Migration Guide"
- Even if text data does not exist in HTTP request, set the request message (body) to be set in the receive activity.
- Even if text data is not returned in HTTP response, set the request message (body) to be set in the reply activity. For schema to be set, use the schema that the product provides.
- If pass-through mode is used while passing HTTP request body to the business process and if httprecp.pass-through.parameter-use=true is specified in HTTP reception definition file, the following points are to be considered:
 - Specify only 1 pair of "msg=value". If multiple "msg=value" pairs are specified, operation is not guaranteed. If msg key is not specified, error occurs.
 - Specify all msg keys in lower case. If upper case characters are specified, error occurs.
 - Specify a valid XML character string in HTTP request body. XML documents are not checked in HTTP reception.
 - If multiple "=" are specified in the query character string or message body as in "msg=msg=abc", the first "=" is used as the separation character. In such cases, the part before the first "=" is read as the key and the entire character string after the first "=" is read as the value.
- The following restrictions exist in text data transfer using HTTP reception and in file data upload or download:
 - A business process must return a response to HTTP reception. Due to this, HTTP reception communication model cannot be asynchronous.
 - Parallel processing of BPEL (FLOW) cannot be used.
 - The maximum number of files that can be processed simultaneously with 1 request in HTTP reception is 1,024.
 - If multi-byte characters are specified in part ID element and file-name element of the response message (header), operation is not guaranteed.
 - The maximum number of files that can be specified to send file data in HTTP response is 1,024. If the number of specified files exceeds 1,024, operation is not guaranteed.
- Consider the following points while transferring file data by using HTTP reception and allocate sufficient working folder disk size for each HTTP reception:
 - Number of simultaneous executions
If you can send HTTP request handling large files in large numbers once, consider the number of simultaneous executions to avoid high resource consumption.
 - Assumed maximum HTTP request size
The size and number of temporary files created while processing file data is same as that of intermediate files. Due to this, you must estimate in advance the maximum HTTP request to be sent.
 - Processing of business process correlated to HTTP reception
While generating files in the working folder by using the file operations adapter in the business process invoked from HTTP reception and storing the files acquired by HTTP adapter, in the working folder, you must consider in advance the file size generated according to the process. Consider the processing time till intermediate files and temporary files are deleted and design the business process.
- Context root is used as the directory name in the working directory of Component Container. Therefore, you must specify a context root in which the path length of the entire working directory does not reach upper limit of OS. For maximum path length of OS, see the document on each OS in use.

The following table describes the configuration of the entire working directory of Component Container:

Configuration of entire working directory of Component Container						Remarks
Path till the working directory of Component Container\						-
Working directory of Component Container\						Directory path is as follows: <ul style="list-style-type: none"> In Windows <Service platform installation directory>\CC\server\public In UNIX If slash (/) is specified in /opt/Cosminexus/CC/server/public context root name, value is replaced by 3 digit single-byte character.
web\						-
<J2EE server name>\						-
<Context root name>\						-
WEB-INF\						-
classes\						-
Directory hierarchy of package name of HTTP reception\						Directory hierarchy is jp/co/Hitachi/soft/csc/msg/message/reception/http/impl.
HTTP reception class file						Longest file name is "CSCMsgHTTPServletContextListener.class".

Legend:

-: There is no applicable description.

You can calculate the path length of the entire working directory of Component Container using the following estimation method. Specify the context root so that this path length does not exceed the maximum path length of OS.

Path length of entire working directory of Component Container=A+2×B+C+D+E+F+25

A: Path length till working directory of Component Container

B: Length of working directory name of Component Container

C: Length of longest J2EE server name

D: Context root length (if symbols such as "/" are included, replace them by character x 3)

E: Length of directory hierarchy of package name of HTTP reception

F: Length of longest file name from among HTTP reception class files

2.14 Connecting to services using HTTP communication

You can use HTTP adapter to invoke resources published in Web server and Web services published in REST style (RESTful Web service) from the business process.

2.14.1 Preconditions

The preconditions to use HTTP adapter are as follows:

- While using HTTP adapter, change SOAP mode to SOAP1.1/1.2 combined mode.
 - HTTP adapter uses JAX-RS functionality to communicate with REST service and Web resource.
- The process to generate and send HTTP requests from the parameters set in HTTP adapter and the process to receive and analyze HTTP response are according to JAX-RS functionality operation.

! Important note

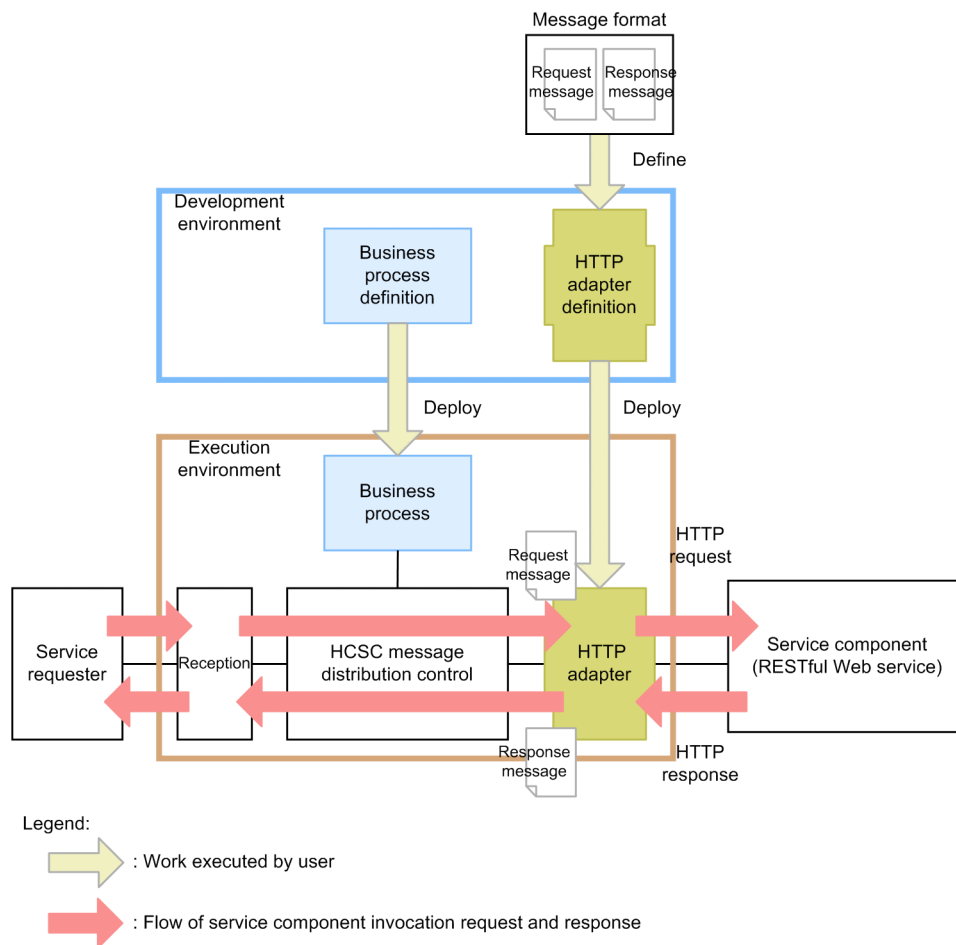
Set HTTP adapter according to the method described in this manual.

If JAX-RS functionality settings used by HTTP adapter are changed directly, operation cannot be guaranteed.

2.14.2 Invoking service components using HTTP adapter

The following figure shows the flow of service component invocation request or response using HTTP adapter:

Figure 2–82: Flow of service component invocation request or response using HTTP adapter



During service component invocation request, the request message passed from the business process is sent to the service component transformed to HTTP request in HTTP adapter. During response, HTTP response sent from the service component is transformed to a response message in HTTP adapter and passed to the business process.

HTTP adapter supports session maintenance using Cookie and file upload and download functions. You can also send files uploaded from HTTP client to the server and files downloaded from the server to HTTP client by using HTTP adapter combined with HTTP reception.

2.14.3 Relation between HTTP requests and request messages

This subsection describes the mechanism to transform from HTTP adapter request message to HTTP request.

Request messages passed from the business process are partitioned in the following manner in HTTP adapter and transformed to HTTP requests:

- Request line
- HTTP request header
- HTTP request body

HTTP request setting methods and details of each item are described here.

Name space prefix is omitted in the message example described here.

(1) HTTP request setting method

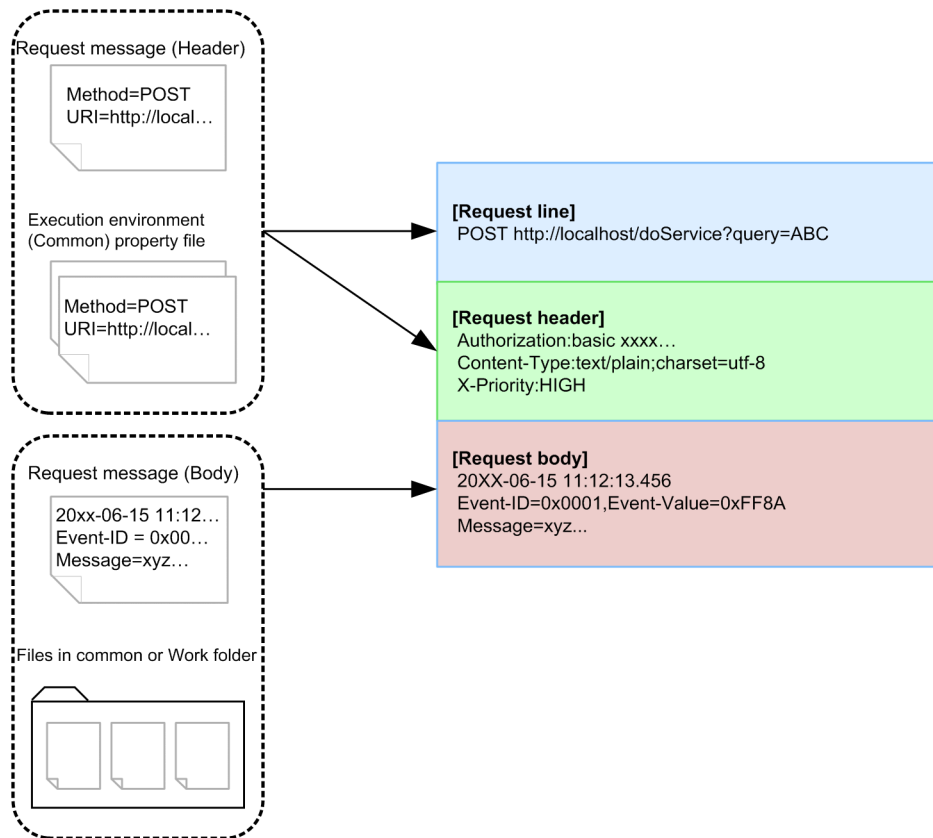
(a) Setting location

HTTP request is set in the following locations for each HTTP request configuration element:

- Request line
 - Request message (header)
 - HTTP-adapter runtime-environment property file
 - HTTP-adapter runtime-environment common property file
- Request header
 - Request message (header)
 - HTTP-adapter runtime-environment property file
 - HTTP-adapter runtime-environment common property file
- Request body
 - Request message (body)
 - Files in the working folder and common folder

Set the request line and request header by using any location from the above 3 setting locations (You can use single locations and multiple location combinations also). Set the request body by using the request message (body) or the files in the working folder and common folder.

Figure 2–83: Example of setting HTTP request



(b) Request line and request header setting method

Set the request line and request header in 3 locations, namely request message (header), HTTP-adapter runtime-environment property file, and HTTP-adapter runtime-environment common property file. If they are set in multiple locations, HTTP adapter is used as high priority setting and processed.

The following table describes the priority of each setting location:

Table 2–31: Request line and request header setting locations and priorities

Priority		Setting location
High	Priority 1	Request message (header)
	Priority 2	HTTP-adapter runtime-environment property file
Low	Priority 3	HTTP-adapter runtime-environment common property file

HTTP request setting is enabled in parameter units in each property file in the request message (header) and each property file (each element in the request message (header) and individual properties in the property file).

Each setting location contains the following features. Reduction in development efforts and easy operations are planned by matching system requirements and combining suitably.

Table 2–32: Setting locations and features of request line and request header

Setting location	Features
Request message (header)	Suitable for parameter definitions required to dynamically change value of each request because setting is enabled during HTTP adapter execution. (Example) Request method and URI specification.

Setting location	Features
HTTP-adapter runtime-environment property file	Suitable for parameter settings decided statically during adapter development (or during operation) because fixed parameters can be set for each definition of HTTP adapter. (Example) Specification of output destination folder during file reception (download).
HTTP-adapter runtime-environment common property file	Suitable for setting common static parameters in the system because common parameters can be set in all HTTP adapters. (Example) Specification of output destination of authentication information and log files.

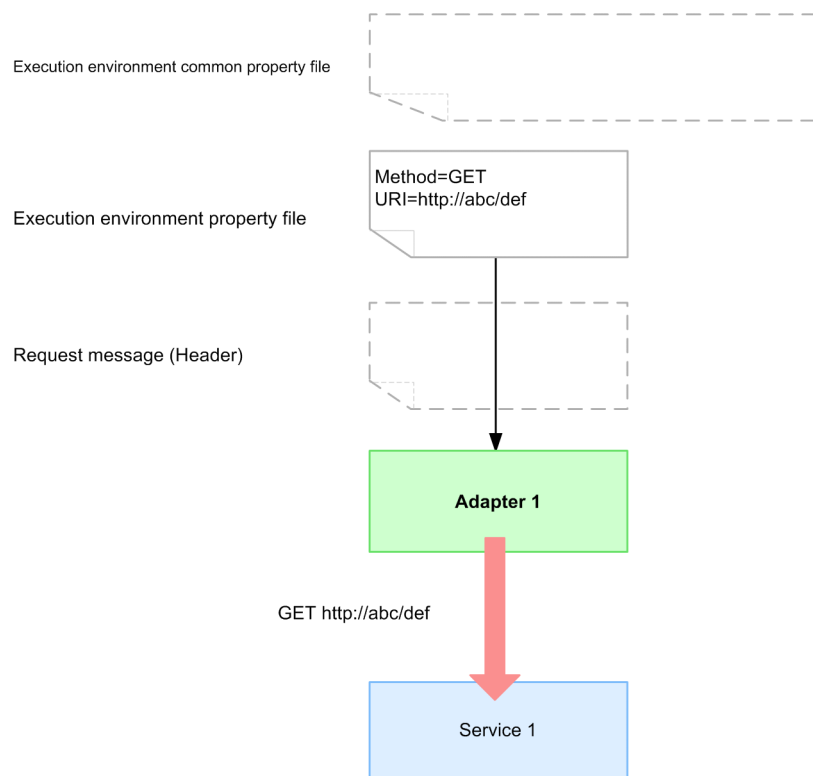
Setting example

The method of setting HTTP request by using basic usage examples is described here.

- Pattern 1 (issuing HTTP request that is always fixed)

The simplest setting method to issue HTTP request that is always fixed from HTTP adapter is to set the request according to the runtime-environment property file. In this method, HTTP adapter reads the runtime-environment property file with the same name as its own service when adapter starts and issues HTTP request on the basis of the contents of the runtime-environment property file every time there is a request for request issue.

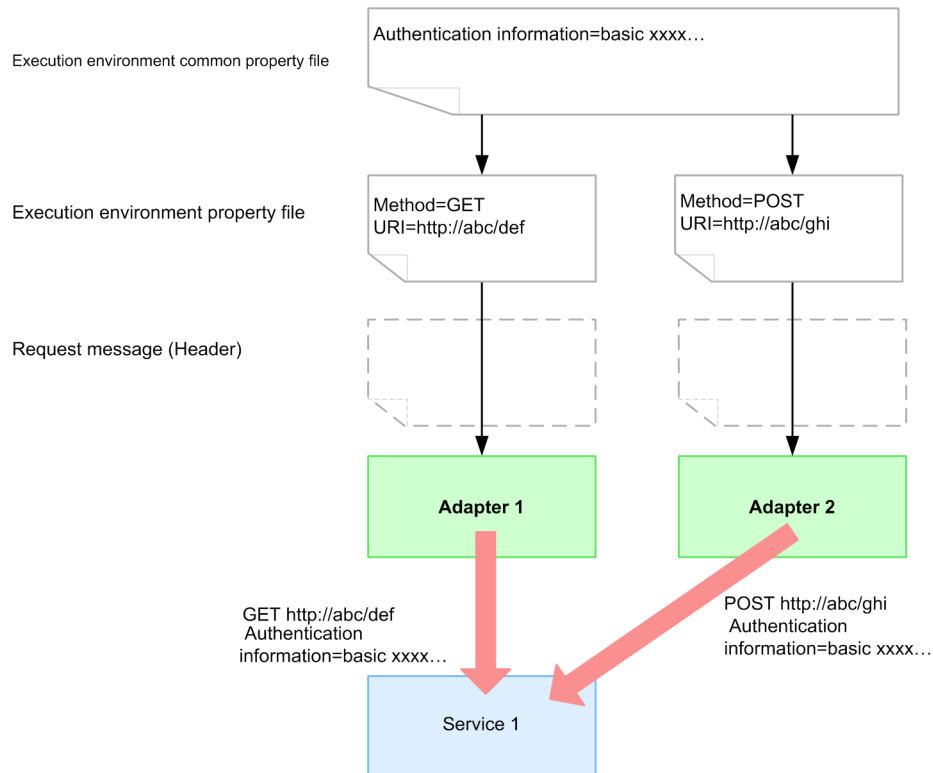
Figure 2–84: Example of setting HTTP request (issuing HTTP request that is always fixed)



- Pattern 2 (sharing some settings between multiple HTTP adapters)

While sharing some settings between multiple HTTP adapters as common system parameters, setting by truncating common system parameters in the runtime-environment common property file is enabled. The following figure contains authentication information of WWW authentication for service 1 as the common parameter of adapter 1 and adapter 2 and this is defined in the runtime-environment common property file:

Figure 2–85: Example of setting HTTP request (sharing some settings between multiple HTTP adapters)



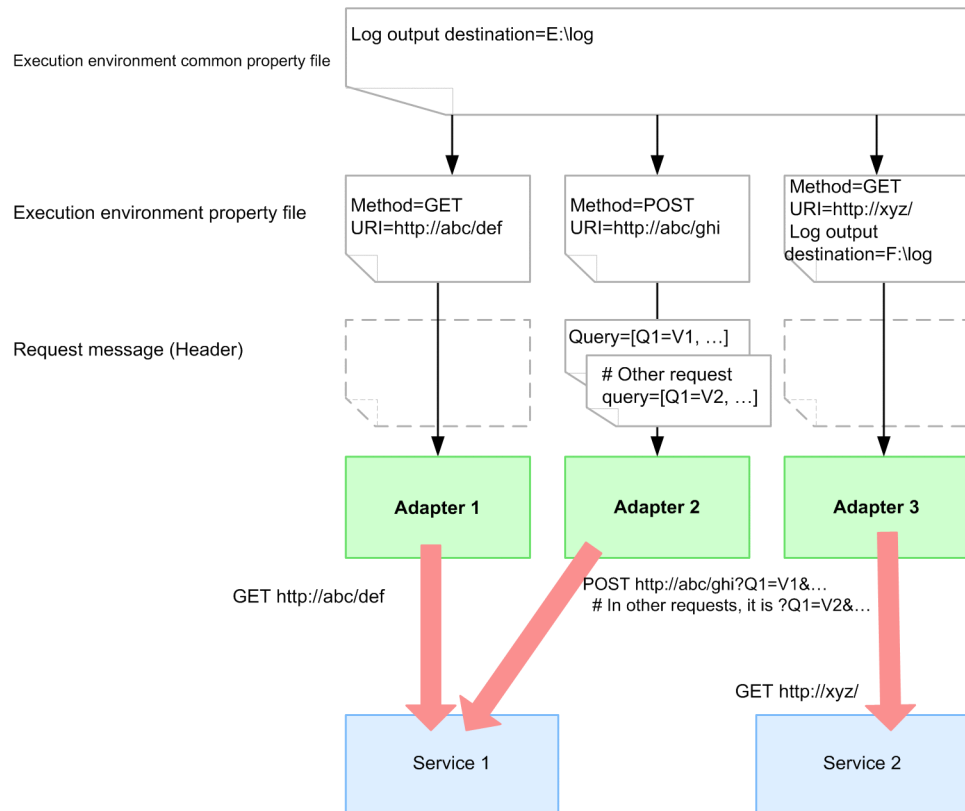
You can reduce development efforts by such setting methods because common parameters need not be defined in duplicate. Also, setting confirmation and change operations (in this example, setting change operation of HTTP adapter according to update of authentication information in service 1) become easy, since common parameters are uniformly managed in 1 property file.

- Pattern 3 (dynamically setting HTTP request contents)

If HTTP request contents are to be set dynamically while issuing a request (if request contents are to be changed according to processing status of the business process), use the request message (header) and set HTTP request. In the following figure, query information is defined in the request message (header) passed from the business process to adapter 2 and query information of the request line is generated dynamically when HTTP request is issued.

Settings dynamically change HTTP request contents when the adapter is executed only when the request message (header) is used.

Figure 2–86: Example of setting HTTP request (dynamically setting HTTP request contents)

**Reference note**

You can replace values in parameters defined in setting locations having low priority with setting locations having high priority (overwriting). For example, in the above figure, all adapter log output destinations are set as "E:\log" by using the runtime-environment common property file. However, for only adapter 3, the log output destination is changed to "F:\log" by using adapter-specific runtime-environment property file. You can also use such setting methods if only some HTTP adapters require exceptional settings or if operations are to be changed only for temporarily specified HTTP adapters.

Specific setting items and setting methods

For specific setting items and setting methods of request lines and request headers, see "(2) Request line" and "(3) HTTP request header".

For details on the elements you can set in the request message (header), see "3.3.13 Defining HTTP adapters" in "Service Platform Reception and Adapter Definition Guide".

For details on properties you can set in HTTP-adapter runtime-environment property file and HTTP-adapter runtime-environment common property file, see "HTTP-adapter runtime-environment property file" in "Service Platform Reference Guide".

For the corresponding relation between request message (header) and HTTP adapter-runtime environment (common) property file, see "2.14.13 Corresponding relation between request message (header) and HTTP-adapter runtime-environment (common) property file".

Notes

- Runtime-environment (common) property file

If runtime-environment property file and HTTP-adapter runtime-environment common property file are not deployed in the prescribed location when HTTP adapter starts, HTTP adapter operates without using the property file.

If both runtime-environment property file and runtime-environment common property file are not deployed, HTTP adapter operates assuming that all default values for these property files are specified. Set a request if required in the request message (header) while executing adapter, since a property that does not have default value is in undefined status.

- Request message (header)

2. Functionality for Connecting to Various Types of Systems

If specification of the header allocation variable for the request message is omitted while developing HTTP adapter, HTTP adapter operates without using the request message (header).

If multiple header allocation variables are specified for the request message while developing HTTP adapter, the header allocation variable containing the name space of HTTP adapter request message (header) is used.

If multiple header allocation variables containing the name space of HTTP adapter request message (header) are specified while developing HTTP adapter, there is uncertainty as to which header allocation variable is used during execution.

- Overall other

You can streamline system development and operations by using multiple setting locations. However, if settings are hierarchical and complicated due to requirements, it is difficult to understand which setting location parameters are used during execution. Analyze system configuration and operation service in the beginning and then design and develop so as to acquire suitable results.

(c) Request body setting method

Specify the request body in pairs of data stored in the request body and data sending types (data storage method). Set each in the following locations:

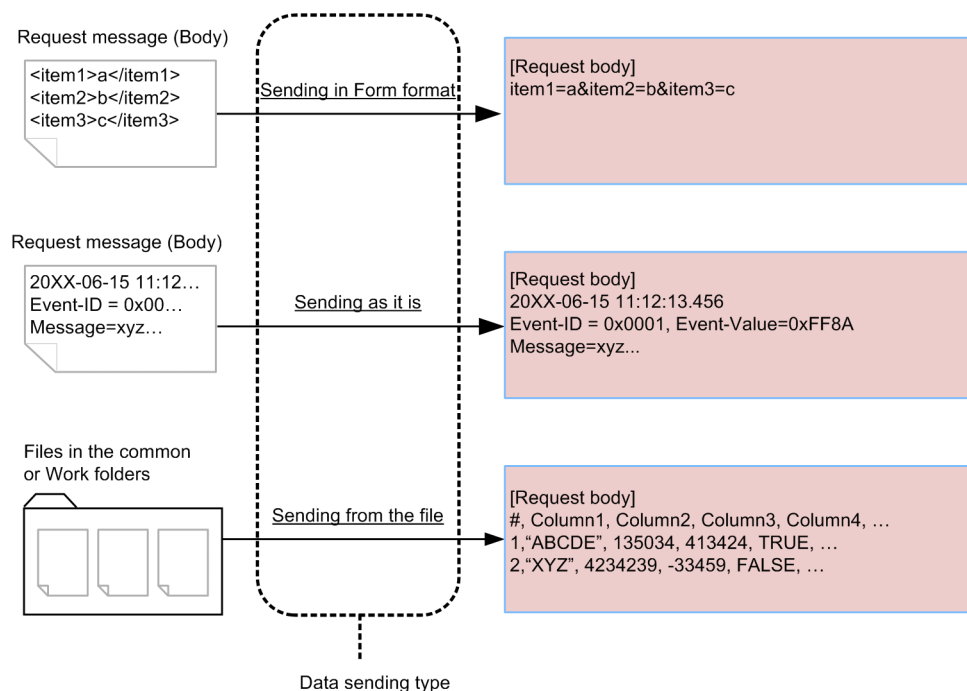
- Stored data

Files in request message (body) or working folder and common folder

- Data sending type

Request message (header) and HTTP adapter-runtime environment (common) property file

Files in request message (body) or working folder and common folder are transformed to a suitable form according to the data sending type, stored in the request body and then issued as requests. The following figure shows setting image when a request body is issued:



For specific setting items and setting methods, see "(4) HTTP request body".

For details on elements you can set in the request message (body), see "3.3.13 Defining HTTP adapters" in "Service Platform Reception and Adapter Definition Guide".

(2) Request line

The request line described in the first line of HTTP request is sent in the following format:

HTTP methodAURIAProtocol#

Legend:

A: Single-byte space.

Note#

Protocol is fixed as "HTTP/1.1".

HTTP methods and URI specification methods are described here.

(a) HTTP method

Specify HTTP method of the request line in the request message (header), HTTP-adapter runtime-environment property file and HTTP-adapter runtime-environment common property file.

The following table describes the specification locations of HTTP methods:

Table 2–33: Specification locations of HTTP methods

Specification method	Item	Specification location
Request message (header)	HTTP method	method element
HTTP-adapter runtime-environment property file, HTTP-adapter runtime-environment common property file	HTTP method	adphttp.request.method property

! Important note

You cannot specify HTTP request body for the following methods:

- GET
- DELETE
- HEAD
- OPTIONS

If you use these methods, always select data type as "No data". For data types of HTTP request body, see "(4)(a) Data types".

If you cannot specify HTTP method in the request message (header), HTTP-adapter runtime-environment property file or HTTP-adapter runtime-environment common property file, an error occurs.

(b) URI

Partition the request line URI into scheme authority, path, and query and then specify these in the request message (header), HTTP-adapter runtime-environment property file and HTTP-adapter runtime-environment common property file.

The following table describes specification locations of URI:

Table 2–34: Specification locations of URI

Specification method	Item	Specification location
Request message (header)	Scheme authority	uri-scheme-authority element
	Path	uri-path element
	Query	uri-query element
HTTP-adapter runtime-environment property file, HTTP-adapter runtime-environment common property file	Scheme authority	adphttp.request.uri-scheme-authority property
	Path	adphttp.request.uri-path property
	Query	adphttp.request.uri-query.<Additional number> property

If scheme authority and path are not specified in the request message (header), HTTP-adapter runtime-environment property file or HTTP-adapter runtime-environment common property file, an error occurs.

If you specify "https" in the scheme part of the scheme authority, use HTTPS to communicate. However, setting is required to use HTTPS. For details, see "Appendix I Security settings required security in the HTTP adapter" in "Service Platform Reception and Adapter Definition Guide".

An example of URI specification is as follows. The generated URI in both cases is "http://localhost:80/test/selectdata?name=sample&id=001".

- If specified in the request message (header)

```
<http-header-request>
  <uri-scheme-authority>http://localhost:80</uri-scheme-authority>
  <uri-path>/test/selectdata</uri-path>
  <uri-query>
    <name>sample</name>
    <id>001</id>
  </uri-query>
</http-header-request>
```

- If specified in HTTP-adapter runtime-environment property file

```
adphttp.request.uri-scheme-authority=http://localhost:80
adphttp.request.uri-path=/test/selectdata
adphttp.request.uri-query.1=name=sample
adphttp.request.uri-query.2=id=001
```

Reference note

You can specify scheme authority, path or query in isolation. You can also dynamically change HTTP request destination by specifying scheme authority in HTTP-adapter runtime-environment property file and by specifying path and query in the request message (header).

! Important note

Set suitable values in scheme authority, path and query. If scheme authority and query are specified in path, and an unsuitable value is specified, operation is not guaranteed.

(c) Query

For URI query, you can customize the detailed message format for query included in the request message (header) and then specify.

An example of specification of query is as follows:

- Example of customization of detailed message format for query
(Original)

```
<xsd:complexType name="uri-query-type">
  <xsd:sequence>
    <xsd:any namespace="##any" processContents="skip" minOccurs="0" maxOccurs="1024" />
  </xsd:sequence>
</xsd:complexType>
```

(Changed)

```
<xsd:complexType name="uri-query-type">
  <xsd:sequence>
    <xsd:element name="parameter1" type="xsd:string" minOccurs="0"/>
    <xsd:element name="parameter2" type="xsd:string" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

In this example, xsd:any type element is changed to xsd:string type element that makes query name a locale name. Moreover, it is optional to the user whether to specify the number of appearances.

- Example of specification of request message (header)

```
<http-header-request>
  <uri-scheme-authority>http://localhost:80</uri-scheme-authority>
  <uri-path>/test/selectdata</uri-path>
  <uri-query>
    <parameter1>value1</parameter1>
    <parameter2>value2</parameter2>
  </uri-query>
</http-header-request>
```

- Generated query

```
http://localhost:80/test/selectdata?parameter1=value1&parameter2=value2
```

! Important note

- Only the value specified under uri-query element is valid and attributes are ignored even if specified.
 - If an element containing hierarchical structure is specified, the value of the query generated by the element is not guaranteed.
 - Appearance order is not guaranteed when multiple identical key names are defined.
 - Even if URI contains basic authentication account information as in "http://user:password@example.jp:80/httpcrp1/op1", it is not used as basic authentication information.
 - If a query is specified in the request message (header), you cannot use an invalid character in the query key name as NCName.
-

(3) HTTP request header

You can specify any extension header for HTTP request header according to http-header element (any type) of the request message (header) and adphttp.request.header.userdef.<Additional number> property of HTTP-adapter runtime-environment property file (or HTTP-adapter runtime-environment common property file).

You can specify any name space, since only locale name is used for element name.

If separate headers are specified in the request message (header) and HTTP-adapter runtime-environment property file (or HTTP-adapter runtime-environment common property file), only the specification in the request message (header) is valid. The header specified in HTTP-adapter runtime-environment property file is not added in the specification in the request message (header).

(a) Example of specification of HTTP header

An example of specification of HTTP header is as follows:

- Example of specification of request message (header)

```
<http-header-request>
  <http-header>
    <X-PRIORITY>HIGH</X-PRIORITY>
    <X-DOCUMENT-ID>12345-67890</X-DOCUMENT-ID>
  </http-header>
</http-header-request>
```

- Example of specification of HTTP-adapter runtime-environment property file

```
adphttp.request.header.userdef.1=X-PRIORITY:HIGH
adphttp.request.header.userdef.2=X-DOCUMENT-ID:12345-67890
```

- Generated HTTP request header

```
X-PRIORITY:HIGH
X-DOCUMENT-ID:12345-67890
```

(b) Special HTTP header specification

HTTP header described in the following table is ignored even if specification is in http-header element of the request message (header) and adphttp.request.header.userdef.<Additional number> property of HTTP-adapter runtime-environment property file (or HTTP-adapter runtime-environment common property file).

Table 2–35: Example of specification in special HTTP header

Header field name	Description
Content-Type	Specify this field in the following locations: <ul style="list-style-type: none"> • Request message (header) • http-header-Content-Type element

2. Functionality for Connecting to Various Types of Systems

Header field name	Description
Content-Type	<ul style="list-style-type: none">HTTP-adapter runtime-environment property file, HTTP-adapter runtime-environment (common) property fileadphttp.request.header.content-type property, adphttp.request.header.content-type.charset property
Cookie	<p>Specify this field in the following location:</p> <ul style="list-style-type: none">Request message (header)Cookies element <p>For details, see "2.14.5 Inheriting Cookie information using HTTP adapter".</p>
Proxy-Authorization	<p>Set Proxy-Authorization header used to connect to the proxy server in JavaVM system properties.</p> <p>For details, see "2.14.6 Communication through proxy servers".</p>
Connection	<p>Set headers related to connection control such as Connection: Keep-Alive in JavaVM system properties.</p> <p>For details, see "2.14.7 Communication using connection continuity (Keep-Alive)".</p>
Content-Length	HTTP adapter is set automatically.
Accept-Encoding	HTTP adapter is set automatically.
Authorization	<p>Specify this field in the following location:</p> <ul style="list-style-type: none">Request message (header)http-header-Authorization elementHTTP-adapter runtime-environment property file, HTTP-adapter runtime-environment (common) property fileadphttp.request.header.authorization property, adphttp.request.header.authorization.type property <p>For details, see "2.14.10 Secure connection using WWW authentication".</p>

(c) Points to be considered

Points to be considered when HTTP header is specified are as follows:

- For the value specified in http-header element, only the value under http-header element is valid and attributes are ignored even if specified.
- If an element containing hierarchical structure is specified in http-header element, value of the header generated by the element is not guaranteed.
- You can specify up to 1,024 request headers under http-header element. If 1,025 or more elements are specified, operation is not guaranteed.
- If HTTP header is specified in http-header element, you cannot use an invalid character in header field name as NCName.
- Description order of HTTP header in HTTP adapter is not guaranteed. If there are multiple header fields with identical names, they are set in HTTP request in the same state.
- If HTTP header and Content-Type header are not specified in the request message (header), HTTP-adapter runtime-environment property file or HTTP-adapter runtime-environment common property file, Content-Type header may be automatically provided depending on the sent contents.
- If a header field name other than extension header is specified, operation is not guaranteed. If a header field name other than extension header is specified, guarantee the operation sufficiently and then use.

(4) HTTP request body

(a) Data types

You can select the data type to be sent in HTTP request body from the following details:

- Form data (normal mode)
Transform data specified in a request message (body) to form data and set this data in HTTP request body.

- Request message (body) (pass-through mode)
Set data specified in a request message (body) as is in HTTP request body.
- File data
File contents are set in HTTP request body.
- No data
HTTP request body is not set. Select GET method, if HTTP request body is not to be sent.

You can specify HTTP request body data type in the binding element of the request message (header) and `adphhttp.request.part.message.binding` property of HTTP-adapter runtime-environment property file (or HTTP-adapter runtime-environment common property file).

However, if `files` element exists in the request message (header), it will be handled as file data even if the above element (or property) is specified. Similarly, if `adphhttp.request.part.file.<Additional number>.input-folder-name` property of HTTP-adapter runtime-environment property file (or HTTP-adapter runtime-environment common property file) exists, it will be handled as file data even if `adphhttp.request.part.message.binding` property is specified.

(b) Form data (normal mode)

To send form data by normal mode, customize and then specify the message format for form data provided by the service platform.

An example of specification of form data is as follows:

- Example of customization of message format for form data
(Original)

```
<xsd:complexType name="http-body-form-data-type">
  <xsd:sequence>
    <xsd:any namespace="##any" processContents="skip" minOccurs="0"
maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>
```

(Changed)

```
<xsd:complexType name="http-body-form-data-type">
  <xsd:sequence>
    <xsd:element name="parameter1" type="xsd:string" minOccurs="0"/>
    <xsd:element name="parameter2" type="xsd:string" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

- Example of specification of request message (header)

```
<http-body-formdata>
  <parameter1>value1</parameter1>
  <parameter2>value2</parameter2>
</http-body-formdata>
```

- Generated HTTP request header and request body

```
Content-Type: application/x-www-form-urlencoded
parameter1=value1&parameter2=value2
```

URL encoding occurs for the specified form data. Default character code is UTF-8.

Important note

- For the value specified in `http-body-formdata` element, only the value under `http-body-formdata` element is valid and attributes are ignored even if specified.
- If an element containing hierarchical structure is specified in `http-body-formdata` element, form data value generated by the element is not guaranteed.
- Appearance order of form data is not guaranteed in HTTP adapter. Appearance order of form data need not always match the order specified in the request message (body).
- For the value specified in `http-body-formdata` element, URL encoding occurs in the character code specified in `charset` attribute of Content-Type header. Always specify in the format before URL encoding.

- If you specify form data in http-body-form data element, you cannot use an invalid character as NCName in key name of form data.

(c) Request message (body) (pass-through mode)

If data specified in the request message (body) in pass-through mode is set as is in HTTP request body, create a message format of the request message (body) that matches HTTP request body format to be sent.

(d) File data

In HTTP adapter, you can send the files in the working folder or common folder to the target server.

Working folder is a temporary folder created on HCSC server for each request by HTTP reception or FTP reception. You can only see the working folder from the business process invoked by the created reception.

Common folder is any folder created by the user during system setup. You can store files shared and used by multiple business processes.

To send file data, specify information related to file data in the request message (header) and HTTP-adapter runtime-environment property file (or HTTP-adapter runtime-environment common property file). You can only use common folders to specify in the property files.

The following table describes the specification locations of file data:

Table 2–36: Specification methods of file data

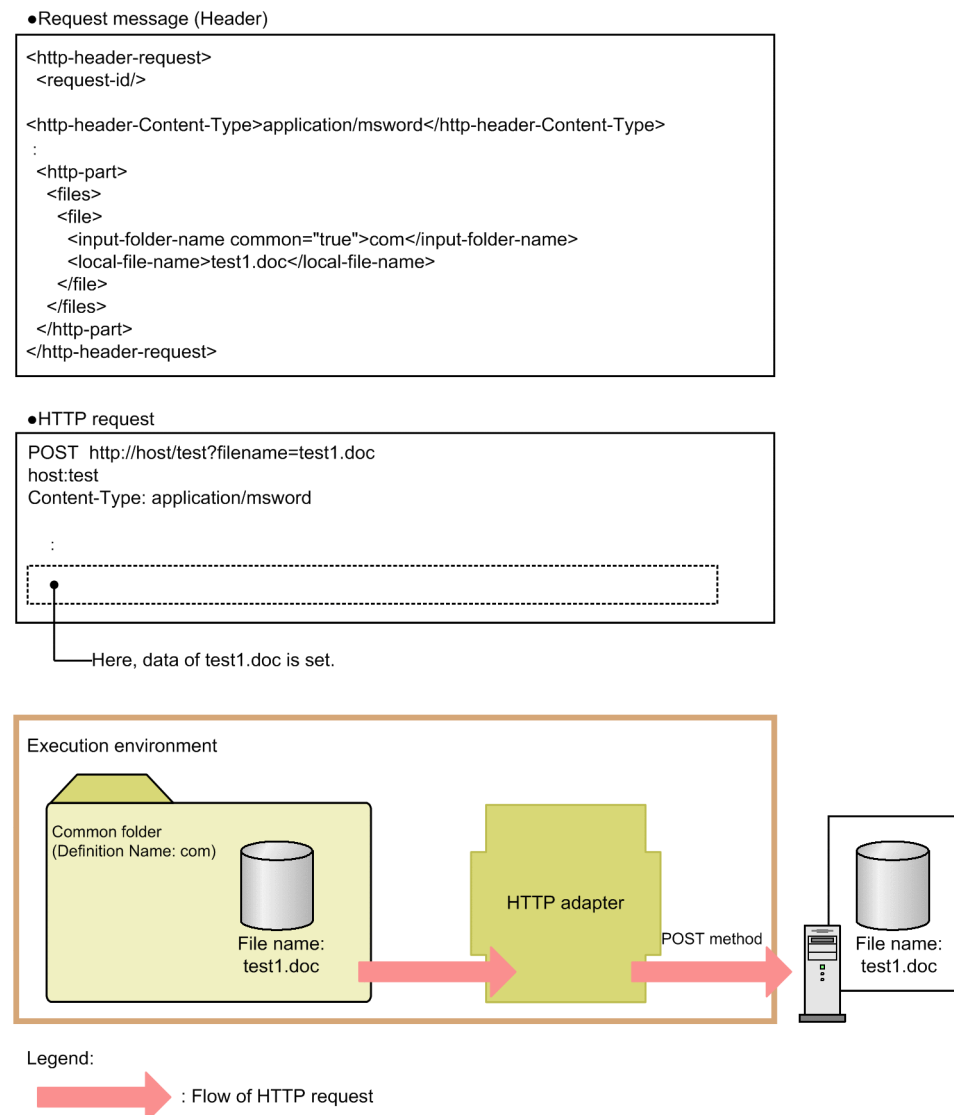
Specification method	Item	Specification location
Request message (header)	Working folder request ID	request-id element
	Used folder	common attribute of input-folder-name element
	Common folder definition name	input-folder-name element
	Name of file to be sent	local-file-name element
	Media type of Content-Type header field of send data	http-header-Content-Type element
	charset attribute value of Content-Type header field of send data	charset attribute of http-header-Content-Type element
HTTP-adapter runtime-environment property file, HTTP-adapter runtime-environment common property file	Working folder request ID	-
	Used folder	-
	Common folder definition name	adphttp.request.part.file.<Additional number>.input-folder-name
	Name of file to be sent	adphttp.request.part.file.<Additional number>.local-file-name
	Media type of Content-Type header field of send data	adphttp.request.header.content-type
	charset attribute value of Content-Type header field of send data	adphttp.request.header.content-type.charset

Legend:

-: There is no applicable specification location.

The following figure shows an example of file sending using HTTP adapter:

Figure 2–87: Example of file sending using HTTP adapter (while using common folder)



The value of the request message (body) passed to HTTP adapter is ignored while sending file data. Use message format for file data to store blank body data in the message format of the specified request message (body).

! Important note

- If only file element is specified in the request message (header) and no lower file element is specified, HTTP request is sent when HTTP request body is in blank status.
- If multiple file elements are specified, the value of the element appearing first is used.
- While sending file data, if the folder in which the file to be read or the file is not specified or if the specified folder and file do not exist, an error occurs.

2.14.4 Relation between HTTP responses and response messages

This section describes the mechanism for transformation from HTTP response of HTTP adapter to the response message.

HTTP response returned from the server is processed for each of the following elements and stored in the response message:

- Status line
- HTTP response header
- HTTP response body

Details of each item are as follows:

Name space prefix is omitted in the message example described here.

(1) Status line

Status line element of HTTP response is stored in the response message as described in the following table:

Table 2–37: Correspondence between status line and response message

Status line element	Correspondence with response message
HTTP version	Not stored in response message.
Status code	Stored in status-code element of response message (header). [#]
Reason phrase	Stored in reason-phrase element of response message (header).

Note[#]

Status codes other than 200 are system exceptions by default. However, you can also set to return faults as normal process or fault response in other than 200. For details, see "(4) Setting normal process or fault process in status code".

(2) HTTP response header

All HTTP headers received in HTTP adapter are stored in http-header element of the response message (header). If HTTP headers with identical names overlap and are received, they are stored in the same state as multiple elements.

An example of HTTP response header and the corresponding response message (header) is as follows:

- Example of HTTP response header

```
Content-Length: 51
Content-Type: image/gif
Last-Modified: Fri, 21 Jul 2006 07:51:32 GMT
Accept-Ranges: bytes
Server: Microsoft-IIS/6.0
Date: Mon, 16 Jul 2012 09:43:57 GMT
```

- Corresponding response message (header)

```
<http-header-response>
  <http-header>
    <Content-Length>51</Content-Length>
    <Content-Type>image/gif</Content-Type>
    <Last-Modified>Fri, 21 Jul 2006 07:51:32 GMT</Last-Modified>
    <Accept-Ranges>bytes</Accept-Ranges>
    <Server>Microsoft-IIS/6.0</Server>
    <Date>Mon, 16 Jul 2012 09:43:57 GMT</Date>
  </http-header>
</http-header-response>
```

! Important note

It is not necessary that the appearance order of HTTP header is the appearance order of XML element. If an invalid character is used as NCName in the header field name of the received HTTP header, a warning message is output, the applicable header field is skipped and processing continues.

Content-Type header and Content-Disposition header of HTTP response header are also stored in the following elements of the response message (header):

Table 2–38: Mapping of Content-Type header, Content-Disposition header and response message (header)

HTTP response header	Attribute	Elements stored in response message (header)
Content-Type	-	Media type part (*/*) of Content-Type header field is stored in http-header-Content-Type element.
	charset	Attribute value is stored in charset attribute of http-header-Content-Type element. Not stored if there is no specification in charset attribute.
Content-Disposition	-	HTTP response is handled as file data if there is specification in HTTP response. Disposition type value is ignored.
	filename	Value is stored as filename element of the child element of file element. Specified value is not transformed for decoding.

Legend:

-: Not applicable.

Only the following values are the elements stored in the response message (header). All other values are ignored.

- Media type of Content-Type header
- charset attribute of Content-Type header
- filename attribute of Content-Disposition header

! Important note

If the response message (header) is omitted, the element stored in the response message (header) from HTTP response cannot be used in the business process.

(3) HTTP response body

HTTP response body is handled in the following manner depending on whether Content-Disposition header is specified in HTTP response header:

- File data
If Content-Disposition header is specified in HTTP response header, any data stored in HTTP response body is output in the file. Output file meta information is stored in the response message (header).
- Raw data (pass-through mode)
If Content-Disposition header is not specified in HTTP response header, any data stored in HTTP response body is stored as is in the response message (body). User creates the message format of the response message (body).

(a) File data

To receive file data, you must specify the output destination directory in the request message (header), HTTP-adapter runtime-environment property file or HTTP-adapter runtime-environment common property file. You can use only a common folder to specify in the property file.

The following table describes the specification location of file data in the request message (header), HTTP-adapter runtime-environment property file, and HTTP-adapter runtime-environment common property file:

Table 2–39: Specification location of file data

Specification method	Specified element or property	Description
Request message (header)	common attribute of request-id element	Set request ID for the working folder. You can omit this if the working folder is not used.
	output-folder-name element	Specify either the common folder or the working folder as the file data output destination.
	output-folder-name element	Specify common folder definition name to output file data. You can omit specification, if the working folder is used.

Specification method	Specified element or property	Description
HTTP-adapter runtime-environment property file, HTTP-adapter runtime-environment common property file	adphttp.request.output-folder-name property	Specify common folder definition name to output file data.

If the output destination directory is not specified in the request message (header), HTTP-adapter runtime-environment property file or HTTP-adapter runtime-environment common property file, an error occurs.

Information such as output destination directory, output file name is stored in the response message (header).

The following table describes file information stored in the response message (header):

Table 2-40: File information stored in response message (header)

Response message (header) element	Description
common attribute of output-folder-name element	Common folder or working folder is stored as folder type used to output file data.
output-folder-name element	Definition name of folder that is used to output file data is stored. If output is in common folder, common folder definition name is stored. If output is in working folder, it is always blank (blank character string).
file-name element	Value of filename attribute of Content-Disposition header specified in HTTP response is stored. However, processes such as decoding do not occur in HTTP adapter. If filename attribute is omitted, element is not generated.
local-file-name element	Name of the intermediate file saved in the output destination folder is stored.

Note 1

The file name output in the common folder by HTTP adapter consists of unique ID numbered in the system. Please note the following points:

- Avoid occurrence of return of time, since server time stamp is used for generating unique ID. If you use NTP software to revise the time, always use slew mode. For details on points to be considered when revising time, see "1.4 Determining the operations policy" in "Service Platform System Setup and Operation Guide" .
- Design the system so that these names and their combinations are not duplicated when same common folders are accessed from multiple HCSC servers, since HCSC server name and cluster name are used for generating unique ID.

Even if HCSC server is in load balance cluster configuration or HA cluster configuration, you must avoid duplication of HCSC server name and cluster name combinations in each HCSC server.

Example: If HCSC server name of HCSC server 1=HCSC1, cluster name=CLS1

- HCSC server name of HCSC server 2=HCSC1, cluster name=CLS1 ? duplicated
- HCSC server name of HCSC server 2=HCSC2, cluster name=CLS1 ? not duplicated
- HCSC server name of HCSC server 2=HCSC1, cluster name=CLS2 ? not duplicated
- HCSC server name of HCSC server 2=HCSC2, cluster name=CLS2 ? not duplicated

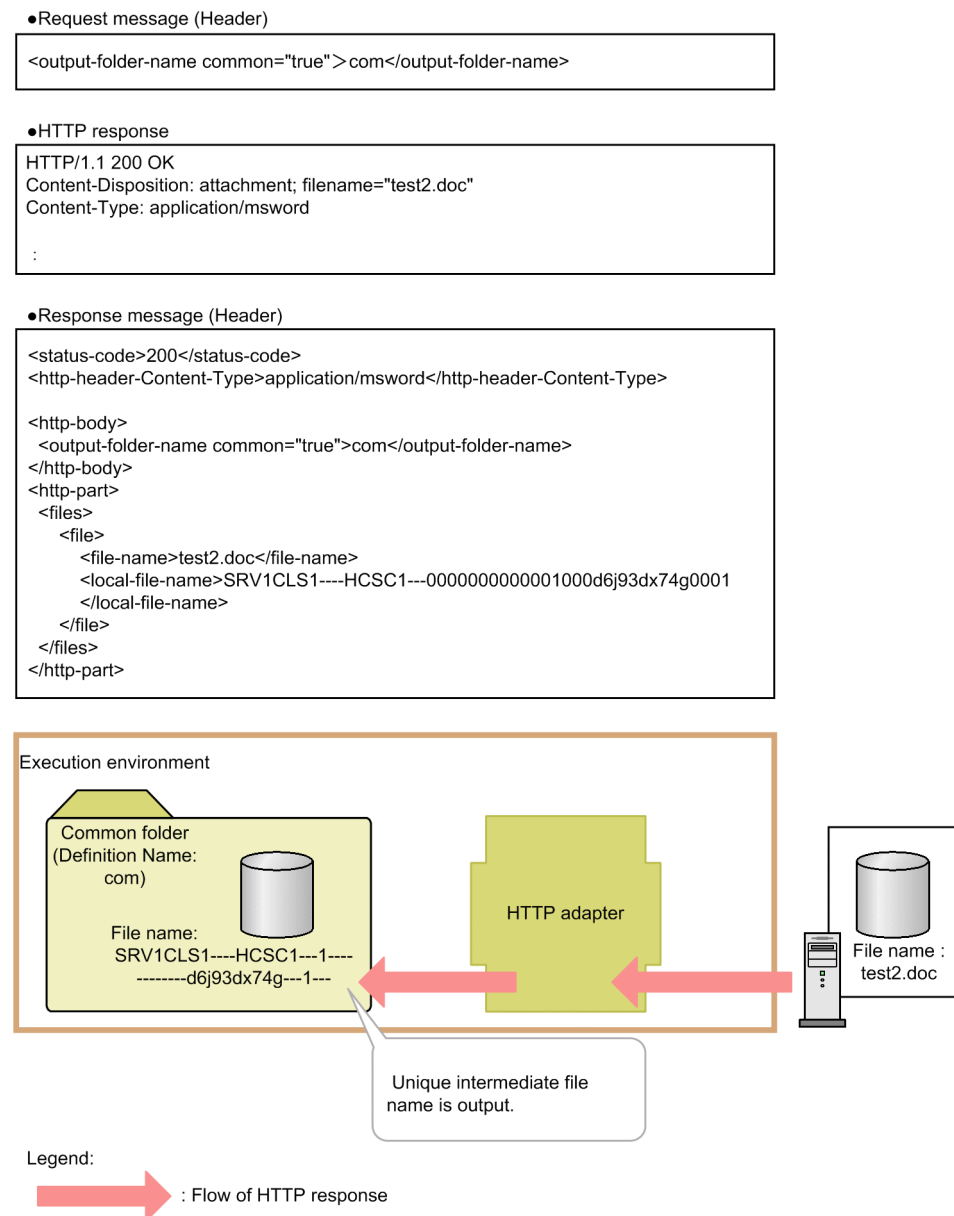
If you cannot meet the above conditions, use only a working folder. If you use a common folder, file name might be duplicated and the file might be overwritten.

Note 2

While using a common folder in the file data output destination, always use the exclusive common folder used only by HTTP adapter. Do not share with common folders used for other usage.

The following figure shows an example of file reception using HTTP adapter:

Figure 2–88: Example of file reception using HTTP adapter (if common folder is used)



(b) Raw data (pass-through mode)

To store HTTP response body data in HTTP adapter response message (body) as is, the user must create an optional message format of the response message (body).

If required, you can also process the received data by invoke java activity, component-common UOC.

(4) Setting normal process or fault process in status code

In HTTP adapter, you can perform settings set to handle the status code specified during HTTP response as normal processing or fault processing.

You can set details in HTTP adapter definition file about which status codes are to be handled as normal processing and as fault processing.

Table 2–41: Settings related to status code processing

Property name	Description
adphttp.response-code.normal	Specify status code to be handled as normal processing using regular expression. Default is 200.
adphttp.response-code.fault	Specify status code to be handled as fault processing using regular expression. Default is a null string.

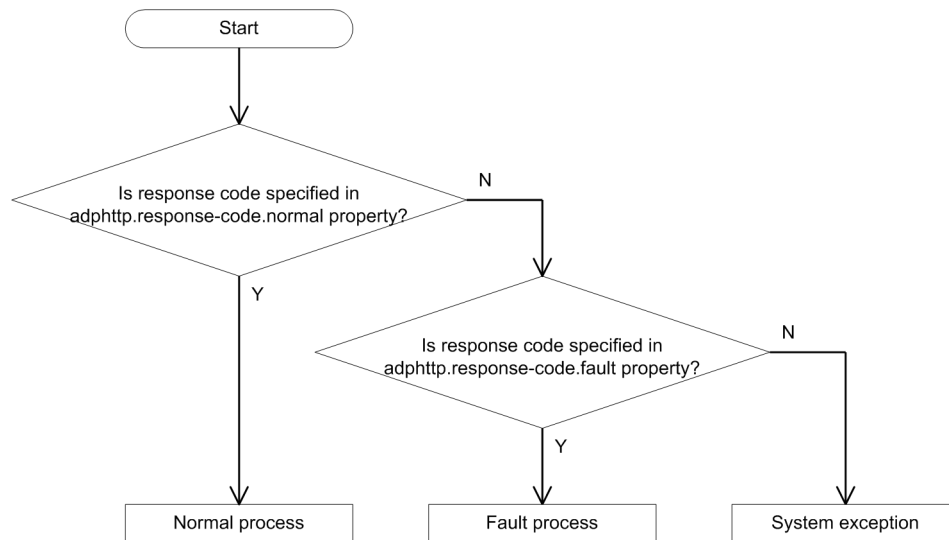
If the status code does not fully match with the regular expression, it is processed as a system exception, since the status code is not returned in HTTP response.

! Important note

You can cache in the business process to handle as fault processing. However, you cannot re-execute. Therefore, while setting fault processing, consider the effects on the business process and then set.

The following figure shows the status code processing flow:

Figure 2–89: Status code processing flow



(5) Setting restriction on response size

In HTTP adapter, to control insufficient memory and disk pressure when an extremely large HTTP response is received, you can set threshold value of HTTP response body size and control the reception size.

You can set threshold value of HTTP response body size in HTTP adapter definition file. The properties to be set are as follows:

Table 2–42: Setting threshold value of HTTP response body size

Item	Property name
Maximum size of data permitted while setting HTTP response in pass-through mode in the response message (body)	adphttp.data-size-limit.body-message
Maximum size of data permitted while writing HTTP response in files	adphttp.data-size-limit.file

After setting threshold value, response size is determined in the following order in HTTP adapter:

1. If HTTP header contains Content-Length header, determine whether header field value exceeds the threshold value. If threshold value is exceeded, a system exception occurs immediately without receiving HTTP response body.

2. If HTTP header does not contain Content-Length header (if chunk encoded data is received), HTTP response body reception is processed, reception process breaks down at the point when threshold value is exceeded and a system exception occurs.

(6) Automatic deletion of incomplete received files

If a problem such as network failure, I/O error, excess of data size limit occur while receiving a file in HTTP adapter, an incomplete received file is generated in the common folder and working folder. However, this incomplete received file can be deleted automatically.

You can set this functionality in `adphttp.auto-delete-incomplete-file` property of HTTP adapter definition file.

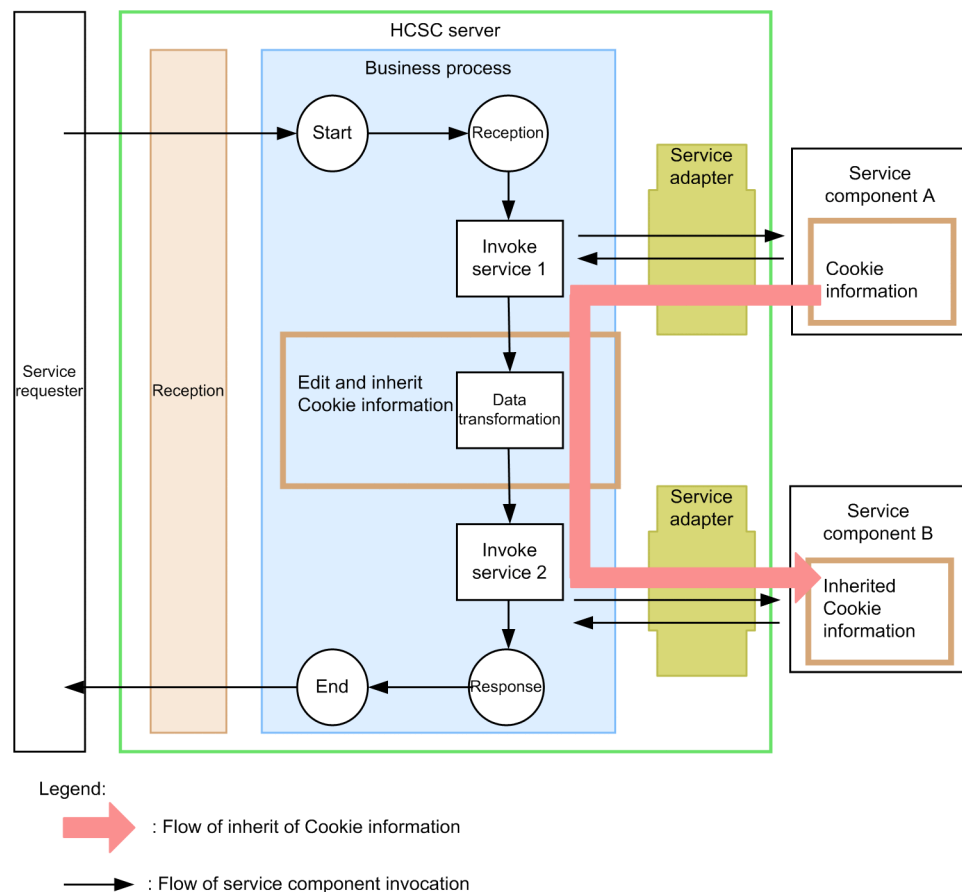
If deletion of incomplete received files fails, message log (KDEC81403-W) is output. In such cases, files which failed to be deleted need to be deleted manually. For deletion methods, see "5.4.20 Deleting working folders" and "5.4.22 Deleting files in the common folder" in "Service Platform System Setup and Operation Guide".

2.14.5 Inheriting Cookie information using HTTP adapter

In HTTP adapter, you can maintain sessions between service invocations (Cookie information inheritance) when multiple services are invoked from a business process.

The following figure shows an overview of Cookie information inheritance using HTTP adapter:

Figure 2–90: Inheriting Cookie information using HTTP adapter



The example in the above figure describes Cookie information inheritance in HTTP adapter. Name space prefix is omitted in the message example described here.

(1) Method of inheriting Cookie information

To inherit Cookie information between 2 services, you can select whether to inherit Cookie information collectively or whether to inherit individually for each Cookie name. The message format for the used header variable differs according to the method selected.

Inheriting Cookie information collectively

- Use `adphttp_header_request1.xsd` in request message format.
- Use `adphttp_header_response1.xsd` in response message format.

Inheriting individually in editable status for each Cookie name

- Use `adphttp_header_request2.xsd` in request message format.
- Use `adphttp_header_response2.xsd` in response message format.

In either case, to inherit Cookie information from service component A to service component B, map Cookie information (Cookie element) included in the response message of service component A in the request message of service component B and then invoke HTTP adapter.

Inheritance method in each case is described here.

(a) Inheriting Cookie information collectively

All Cookie information is represented by any type in the message format in the following manner. The element name showing Cookie element is "Cookie_types".

```
<xsd:complexType name="http-header-Cookies-type">
  <xsd:sequence>
    <xsd:any namespace="##any"
      processContents="skip" minOccurs="0" maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>
```

You can collectively inherit Cookie information by mapping any type element parts from the response message of service component A to the request message of service component B.

(b) Inheriting individually in editable status of each Cookie name

Each attribute of Cookie information in the message format is partitioned in XML element attribute in the following manner. The element name showing Cookie element is "Cookie_types".

```
<xsd:complexType name="cookie-type">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
      <xsd:attribute name="name" type="xsd:string" use="required"/></xsd:attribute>
      <xsd:attribute name="host" type="xsd:string" use="required"/></xsd:attribute>
      <xsd:attribute name="path" type="xsd:string" use="optional"/></xsd:attribute>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

While mapping from the response message of service component A to the request message of service component B, you can see Cookie name and inherit individually.

(2) Transforming HTTP responses and HTTP requests

This section describes transformation from HTTP response header to response message and the process of transformation from the request message to HTTP request header.

Process of transformation from HTTP response header to response message

The value of each Set-Cookie header passed from service component A to HTTP response header is stored in Cookie element of the response message. Each attribute of Set-Cookie header is set in the attribute value of Cookie element.

If multiple Set-Cookie headers exist, Cookie element is generated for each Set-Cookie header.

Examples of transformation from HTTP response header to the response message (header) are as follows:

- Set-Cookie header of HTTP response header

```
Set-Cookie: AAA=1234; expires=Sat, 27-Oct -2012 10:00:00 GMT; path=/
Set-Cookie: BBB=5678; expires=Sat, 27-Oct -2012 10:00:00 GMT;
```

- Response message (header) after generation

```
<Cookies>
  <Cookie name="AAA" path="/" host="hostname:80#">AAA=1234; expires=Sat, 27-Aug-2011
10:00:00 GMT; path=/</Cookie>
  <Cookie name="BBB" host="hostname:80#">BBB=5678; expires=Sat, 27-Aug-2011 10:00:00 GMT</
Cookie>
</Cookies>
```

Note#

Host name and port number used to connect to the service component are stored in host attribute.

If you cannot acquire path attribute from Set-Cookie header, processing occurs without path attribute being added in the response message (header). In such cases, path attribute value is handled as "/".

Process of transformation from request message to HTTP request header

Cookie information passed as the request message of HTTP adapter is set in HTTP request header as the sending target if it meets both of the following conditions:

- Values (host name and port number) set in service component URI and Cookie element host attribute match completely.
- The front part of the path set in Cookie element path attribute matches with service component URI.

Example: If service component URI is http://localhost:80/abc/xyz

- Cookie information host attribute value=localhost:80, path attribute value=/abc ? Sent
- Cookie information host attribute value=somehost:80, path attribute value=/abc ? Not sent
- Cookie information host attribute value=localhost:443, path attribute value=/abc ? Not sent
- Cookie information host attribute value=localhost:80, path attribute value=/xyz ? Not sent
- Cookie information host attribute value=localhost:80, path attribute value=/abc/xyz/def ? Not sent

If multiple Cookie information items exist, they are transformed to collective Cookie header in the following manner.

An example of transformation from the request message (header) to HTTP request header is as follows:

- Cookie element of request message (header)

```
<Cookies>
  <Cookie name="AAA" path="/" host="hostname:80">AAA=1234; expires=Sat, 27-Oct-2012
10:00:00 GMT; path=/</Cookie>
  <Cookie name="BBB" host="hostname:80">BBB=5678; expires=Sat, 27-Oct-2012 10:00:00 GMT</
Cookie>
</Cookies>
```

Cookie element name uses only locale name. If Cookie information in the header request message is collectively inherited from the header response message, name space of Cookie element of the header request message becomes the header response message. However, it is handled correctly as the sending target, since the name space is not seen.

- HTTP request header after generation

```
Cookie: AAA=1234; BBB=5678
```

! Important note

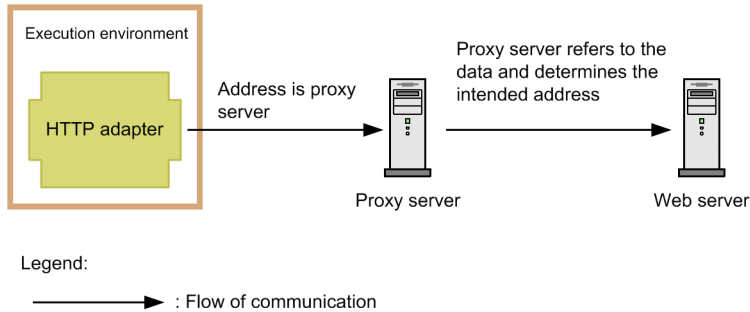
- If you set multiple Cookie elements containing different path attribute values, path attribute value is in detailed order and Cookie information is set in HTTP header.
(Example) If Cookie element containing NAME1=VALUE1 path=/, NAME2=VALUE2 path=/xyz value is set:
Cookie: NAME2=VALUE2; NAME1=VALUE1
 - If Cookie information that cannot be processed normally is received, analysis is skipped and processing continues.
 - Also see the precautions shown in "2.2.4(4) Notes".
-

2.14.6 Communication through proxy servers

In HTTP adapter, you can use services in an external network through the proxy server.

The following figure shows an overview of communication through proxy servers:

Figure 2–91: Communication using proxy server



JavaVM system property settings are required to connect through the proxy server. The following table describes the properties required to be set:

Table 2–43: System properties required for connecting through proxy server

Property name	Specification contents
http.proxyHost	Specify the host name or IP address of the proxy server. If you specify a null character, the proxy server is not connected.
http.proxyPort	Set the port number of the proxy server. If http.proxyHost is set suitably and a null character is specified in http.proxyPort, 80 number port of the host specified in http.proxyHost is accessed. If http.proxyHost is not specified, the proxy server is not connected even if http.proxyPort is specified.
http.nonProxyHosts	Specify the host name group that does not use the proxy server, if required. Proxy server specified in http.proxyHost or https.proxyHost is not used to connect to the host specified in this property. If you specify multiple hosts, separate by " " and set. You cannot specify a character (such as null) other than " " between 2 host names.
https.proxyHost	Specify the host name or IP address of the proxy server used in the connection according to SSL protocol. If you specify a null character, the proxy server is not connected.
https.proxyPort	Set the port number of the proxy server used in the connection according to SSL protocol. If https.proxyHost is set suitably and a null character is specified in https.proxyPort, 443 number port of the host specified in https.proxyHost is accessed. If https.proxyHost is not specified, the proxy server is not connected even if https.proxyPort is specified.

For details of setting methods, see "10.10 Connecting through proxy server" in "Application Server Web Service Development Guide".

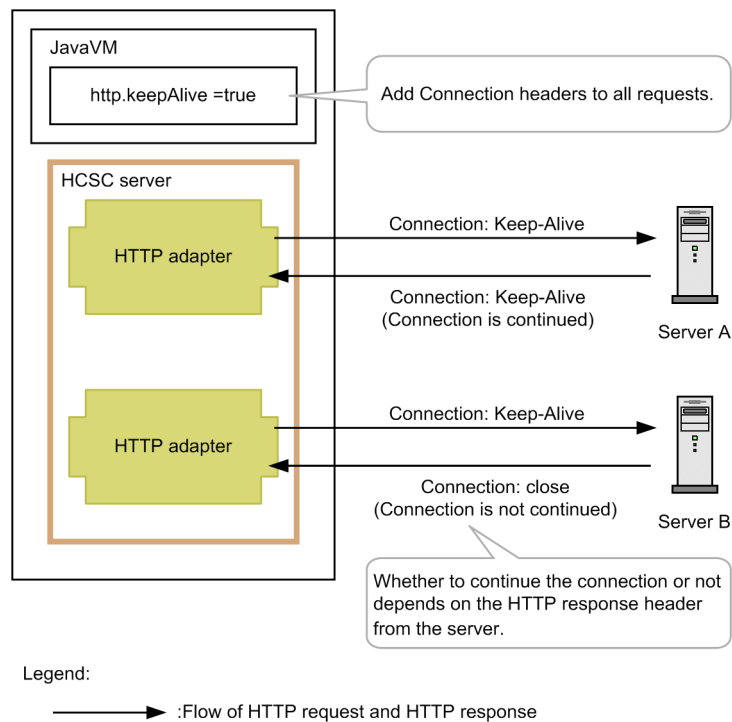
2.14.7 Communication using connection continuity (Keep-Alive)

You can continue using a connection used once by specifying Keep-Alive in Connection header and you can reduce connection continuation cost from next time onwards.

Set whether to use Keep-Alive connection in JavaVM system properties. If the server side setting is performed for not using Keep-Alive, even if you communicate in Keep-Alive connection from HTTP adapter, HTTP response Connection header from the server closes and the connection does not continue.

The following figure shows an overview of connection continuation using Keep-Alive:

Figure 2–92: Overview of Keep-Alive connection



The following figure shows the properties required to be set while using Keep-Alive connection:

Table 2–44: System properties required for Keep-Alive connection

Property name	Specification contents
<code>http.keepAlive</code>	Specify whether to enable Keep-Alive connection by true or false. Default is true.
<code>http.maxConnections</code>	Specify by int type the maximum number of connections of each connection destination maintained in the specific time. Default is 5.

For JavaVM system property settings, see JavaVM documents.

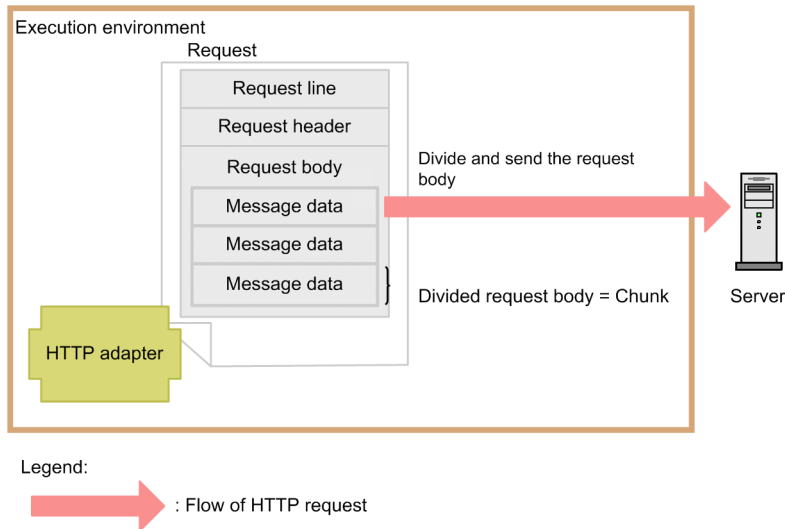
Settings related to Keep-Alive connection are enabled in the entire server and not in HTTP adapter units, since they are set in system properties.

2.14.8 Communication using chunk transmission

While sending HTTP requests in HTTP adapter, you can partition body data into multiple blocks (chunks) and send data for each block.

The following figure shows an overview of chunk transmission:

Figure 2–93: Overview of chunk transmission



You can set existence of use of chunk transmission and chunk size in `adphttp.protocol.chunked-encoding-size` property of HTTP adapter definition file.

For details on HTTP adapter definition file, see "HTTP adapter definition file" in "Service Platform Reference Guide" .

2.14.9 Compressing and extracting HTTP body

If communication data volume is large, you can reduce traffic volume by compressing or extracting HTTP body data to be transferred.

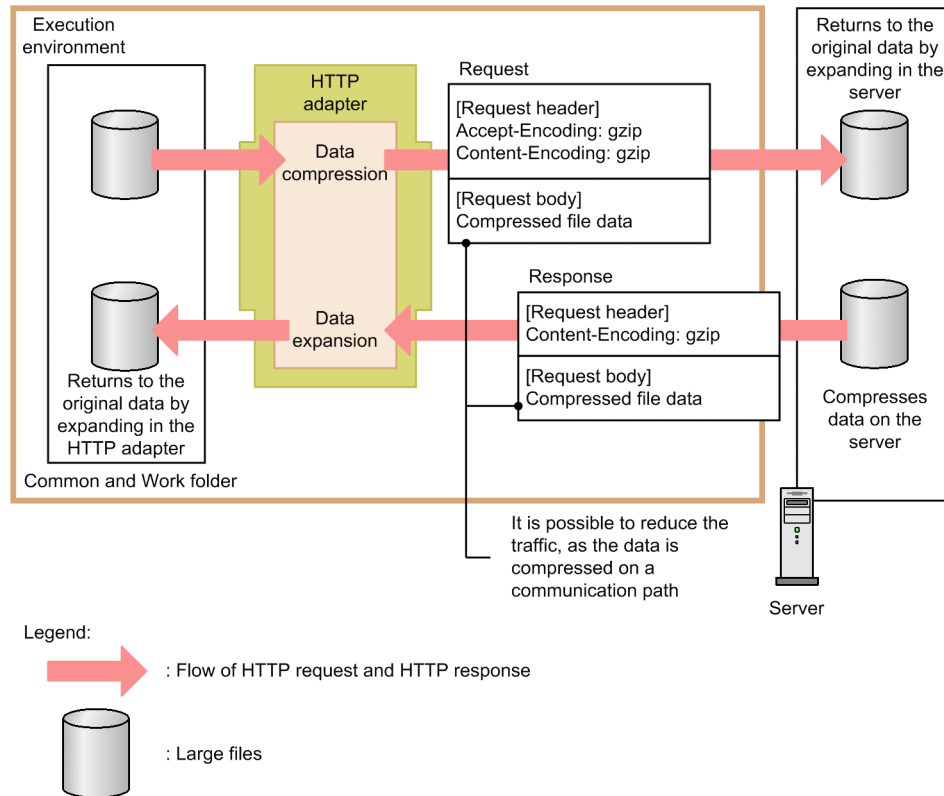
You can set whether to compress or extract HTTP body while transferring HTTP messages by `adphttp.protocol.encoding-type` property of HTTP adapter definition file.

If `gzip` is specified in this property, HTTP body data is encoded or decoded in `gzip` format and then transferred.

For details of HTTP adapter definition file, see "HTTP adapter definition file" in "Service Platform Reference Guide" .

The following figure shows the flow of processing when `gzip` is specified in `adphttp.protocol.encoding-type` property by file linkage with the example of compressing and extracting HTTP body:

Figure 2–94: Example of compressing and extracting HTTP body



2.14.10 Secure connection using WWW authentication

In HTTP adapter, you can safely connect to the target server by using WWW authentication.

To use WWW authentication, you must perform settings in the request message (header), HTTP-adapter runtime-environment property file or HTTP-adapter runtime-environment common property file.

The following figure shows setting locations and setting contents related to WWW authentication:

Table 2–45: Setting items related to WWW authentication

Item	Setting location	Setting item	Value to be set
Authentication type	Request message (header)	http-header-Authorization element type attribute	Set one of the following values: <ul style="list-style-type: none"> none WWW authentication is not used. raw WWW authentication is used.
	HTTP-adapter runtime-environment property file, HTTP-adapter runtime-environment common property file	adphhttp.request.header.authorization.type property	
Authentication information	Request message (header)	http-header-request/http-header-Authorization element	Specify authentication information of WWW authentication.
	HTTP-adapter runtime-environment property file, HTTP-adapter runtime-environment common property file	adphhttp.request.header.authorization property	

2.14.11 Managing HTTP adapter timer

You can set the following timeout duration in HTTP adapter:

- Connection timeout
- Read timeout
- Exclusive management timeout

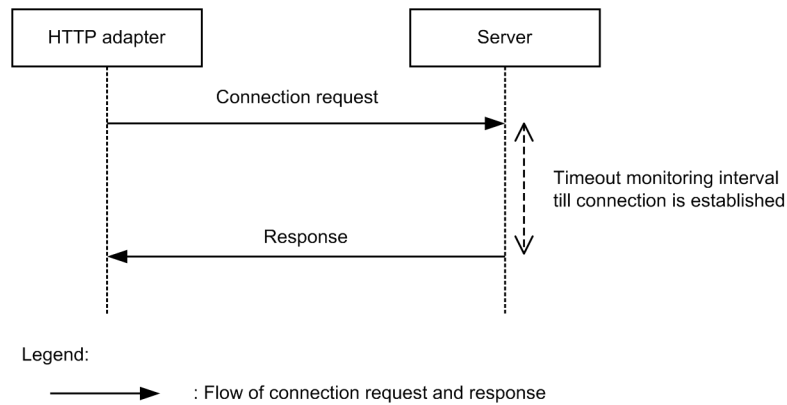
You can set each timeout value in HTTP adapter definition file. For details on HTTP adapter definition file, see "HTTP adapter definition file" in "Service Platform Reference Guide" .

(1) Connection timeout

You can set connection timeout till connection with the server is established.

The following figure gives an overview of connection timeout:

Figure 2–95: Connection timeout



The following table describes the parameters you can set:

Table 2–46: Setting connection timeout

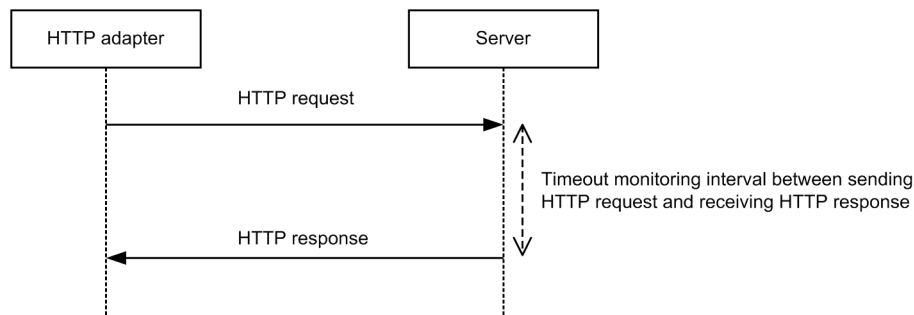
Value	Key name	Default value
Connection timeout (milliseconds)	adphhttp.protocol.connect-timeout	60000

(2) Read timeout

You can set read timeout after sending HTTP request till HTTP response is returned. Monitor timeout for each 1 time if HTTP response is returned by splitting into the number of times after HTTP request is sent.

The following figure gives an overview of read timeout:

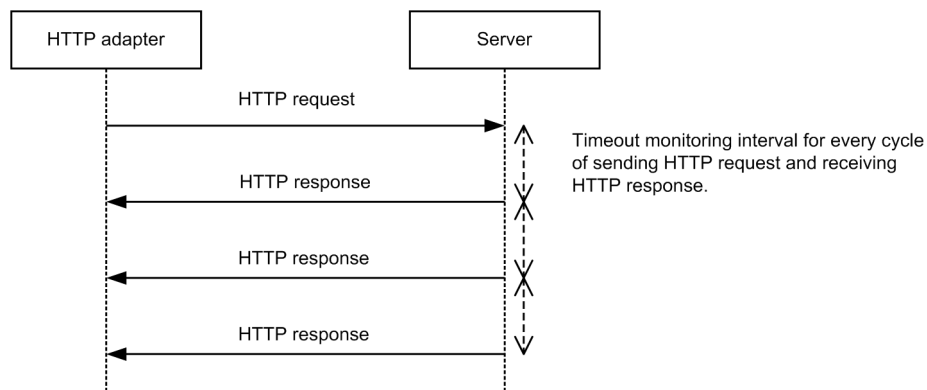
Figure 2–96: Read timeout (if response in 1 time is returned)



Legend:

: Flow of HTTP request and HTTP response

Figure 2–97: Read timeout (if response split 3 times is returned)



Legend:

: Flow of HTTP request and response

The following table describes the parameters you can set:

Table 2–47: Setting read timeout

Value	Key name	Default value
Read timeout (milliseconds)	adphhttp.protocol.read-timeout	300000

(3) Exclusive management timeout

Common lock for the common folder is required to access the common folder from HTTP adapter. At this stage, you can specify lock retry frequency and lock retry interval in HTTP adapter.

The following table shows the parameters you can set:

Table 2–48: Setting read timeout

Value	Key name	Default value
Lock retry frequency (times)	adphhttp.file.read-lock.retry.count	0
Lock retry interval (seconds)	adphhttp.file.read-lock.retry.interval	1

2.14.12 Examples of a system using the HTTP adapter

Examples of a system using the HTTP adapter are as follows:

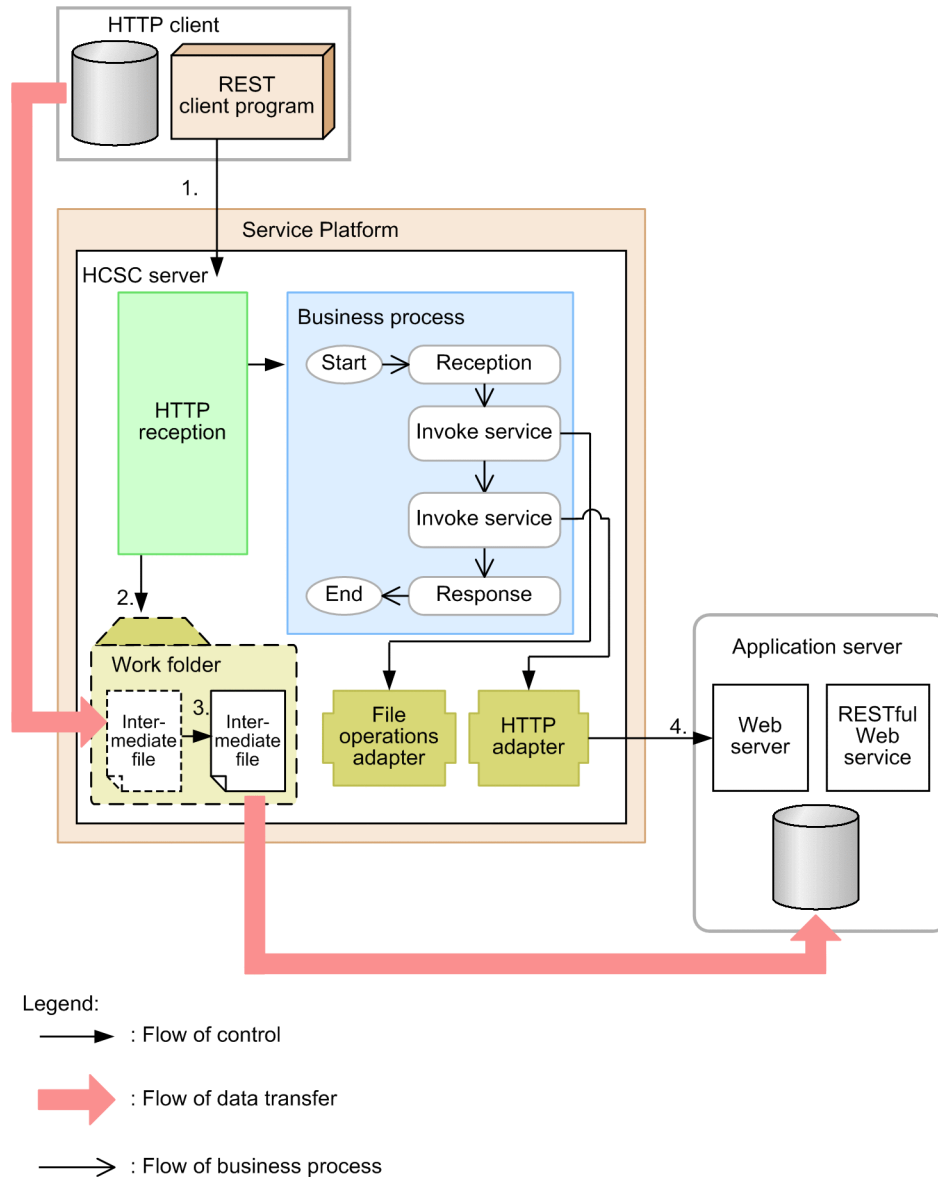
- Uploading a file sent from an HTTP client to the server
- Moving a file saved on a server to another server

The following describes each example

(1) Uploading a file sent from an HTTP client to the server

In this example, a file sent from the HTTP client is converted by the file operations adapter, and then uploaded to the target server.

Figure 2–98: Example of a system using the HTTP adapter (1)



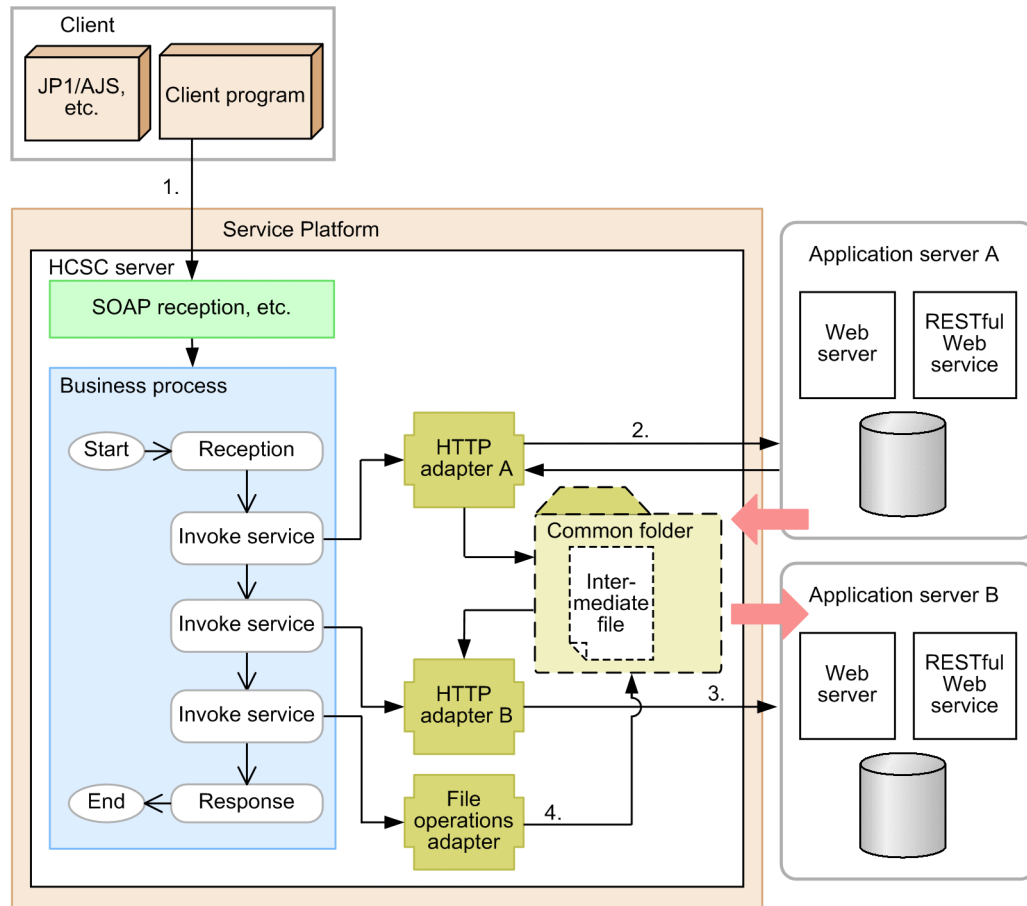
The figure below shows the flow of service component invocation and the flow of a file. The following numbers correspond to numbers in the figure.

1. The service requester (REST client program of the HTTP client) sends a file to the HCSC server.
2. The file received by HTTP reception is generated as an intermediate file in the work folder.
3. The file operations adapter invoked from the business process converts the intermediate file.
4. The HTTP adapter invoked from the business process sends the converted file to the target server.

(2) Moving a file saved on a server to another server

In this example, a file saved on a server is downloaded to the HCSC server, and then sent to another (target) server.

Figure 2-99: Example of a system using the HTTP adapter (2)



Legend:

→ : Flow of control

➡ : Flow of data transfer

→ : Flow of business process

The figure below the flow of service component invocation and the flow of a file. The following numbers correspond to the numbers in the figure.

1. The service requester (client program) sends a request for executing the business process.
2. HTTP adapter A invoked from the business process accesses the target server (application server A) and receives the file, and then saves the file in the common folder.
3. HTTP adapter B invoked from the business process sends the file saved in the common folder to the target server (application server B).
4. The file operations adapter deletes the received file from the common folder.

2.14.13 Corresponding relation between request message (header) and HTTP-adapter runtime-environment (common) property file

This subsection describes the corresponding relation of parameters you can set in the request message (header) of HTTP adapter, HTTP-adapter runtime-environment property file, and HTTP adapter common runtime-environment property file. Element name of the request message (header) here is represented by XPath.

If there are specifications in multiple locations, specification of high priority location is used.

Table 2–49: Request ID

Item	High <-- Priority --> Low			Default value	Possibility of specification
	Request message (header)	HTTP-adapter runtime-environment property file	HTTP-adapter runtime-environment common property file		
Request ID	/http-header-request/request-id	Cannot be specified.	Cannot be specified.	-	Optional

Legend:

-: No default value.

Explanation

Specify a working folder in the input source folder type attribute of a sent file or the output destination folder type attribute of a received file in a request message (header). If the working folder is not used, you can omit this specification.

Table 2–50: Request method

Item	High <-- Priority --> Low			Default value	Possibility of specification
	Request message (header)	HTTP-adapter runtime-environment property file	HTTP-adapter runtime-environment common property file		
Request method	/http-header-request/method	adphttp.request.method	adphttp.request.method	-	Mandatory

Legend:

-: No default value.

Explanation

Always specify request method in a request message (header) or another property file.

Table 2–51: Request URI

Item	High <-- Priority --> Low			Default value	Possibility of specification
	Request message (header)	HTTP-adapter runtime-environment property file	HTTP-adapter runtime-environment common property file		
Scheme authority	/http-header-request/uri-scheme-authority	adphttp.request.uri-scheme-authority	adphttp.request.uri-scheme-authority	-	Mandatory
Path	/http-header-request/uri-path	adphttp.request.uri-path	adphttp.request.uri-path	-	Optional
Query	/http-header-request/uri-query/optional element	adphttp.request.uri-query.<Additional number>	adphttp.request.uri-query.<Additional number>	-	

Legend:

-: No default value.

Explanation

Always specify scheme authority in a request message (header) or a property file.

Specify path in a request message (header) or a property file, if required. If root path ("/") is used, you can omit this specification.

Specify query in a request message (header) or a property file, if required. If query is specified in multiple locations, all queries specified in low priority locations are overwritten by queries specified in high priority locations. You cannot combine queries specified in multiple locations.

Table 2–52: Authentication information

Item	High <-- Priority --> Low			Default value	Possibility of specification
	Request message (header)	HTTP-adapter runtime-environment property file	HTTP-adapter runtime-environment common property file		
Authentication value	/http-header-request/http-header-Authorization	adphhttp.request.header.authorization	adphhttp.request.header.authorization	Null character string	Optional
Authentication type attribute	/http-header-request/http-header-Authorization/@type	adphhttp.request.header.authorization.type	adphhttp.request.header.authorization.type	none	

Explanation

Specify these parameters in the request message (header) or a property file, if required. If authentication information is not used, you can omit this specification (Use default value).

You must always specify these parameters in sets. You cannot specify only authentication value or authentication type attribute in 1 location.

Table 2–53: Content-Type

Item	High <-- Priority --> Low			Default value	Possibility of specification
	Request message (header)	HTTP-adapter runtime-environment property file	HTTP-adapter runtime-environment common property file		
Media type	/http-header-request/http-header-Content-Type	adphhttp.request.header.content-type	adphhttp.request.header.content-type	-	Optional
Character code attribute	/http-header-request/http-header-Content-Type/@charset	adphhttp.request.header.content-type.charset	adphhttp.request.header.content-type.charset	-	

Legend:

-: No default value.

Explanation

Specify these parameters in the request message (header) or a property file, if required. If Content-Type header is not set, you can omit this specification.

If character code attribute is not to be set, you can omit this specification. In such cases, only media type is used.

If only the character code attribute is specified, specification is not enabled.

Table 2–54: Cookie

Item	High <-- Priority --> Low			Default value	Possibility of specification
	Request message (header)	HTTP-adapter runtime-environment property file	HTTP-adapter runtime-environment common property file		
Cookie information	/http-header-request/http-header-Cookies/Cookie	Cannot be specified.	Cannot be specified.	-	Optional
Cookie name attribute	/http-header-request/http-header-Cookies/Cookie/@name	Cannot be specified.	Cannot be specified.	-	

2. Functionality for Connecting to Various Types of Systems

Item	High <-- Priority --> Low			Default value	Possibility of specification
	Request message (header)	HTTP-adapter runtime-environment property file	HTTP-adapter runtime-environment common property file		
Destination host name attribute	/http-header-request/http-header-Cookies/Cookie/@host	Cannot be specified.	Cannot be specified.	-	Optional
Destination path attribute	/http-header-request/http-header-Cookies/Cookie/@path	Cannot be specified.	Cannot be specified.	-	

Legend:

-: No default value.

Explanation

Specify these parameters in the request message (header) or a property file, if required. If Cookie header is not set, you can omit this specification.

You must always specify Cookie information, Cookie name attribute, and destination host name attribute in sets.

If destination path attribute need not be set, you can omit this specification. In such cases, Cookie is sent to all paths, since destination path is not determined.

You cannot specify only Cookie information, Cookie name attribute, destination host name attribute or destination path attribute in 1 location.

Table 2–55: Request header

Item	High <-- Priority --> Low			Default value	Possibility of specification
	Request message (header)	HTTP-adapter runtime-environment property file	HTTP-adapter runtime-environment common property file		
Request header	/http-header-request/http-header/optional element	adphttp.request.header.userdef.<Additional number>	adphttp.request.header.userdef.<Additional number>	-	Optional

Legend:

-: No default value.

Explanation

Specify request header in the request message (header) or a property file, if required.

If request header is specified in multiple locations, all request headers specified in low priority locations are overwritten by request headers specified in high priority locations. You cannot combine request headers specified in multiple locations.

Table 2–56: Request body (data transmission type of request message (body))

Item	High <-- Priority --> Low			Default value	Possibility of specification
	Request message (header)	HTTP-adapter runtime-environment property file	HTTP-adapter runtime-environment common property file		
Request body (data transmission type of request message (body))	/http-header-request/http-part/message/binding	adphttp.request.part.message.binding	adphttp.request.part.message.binding	none	Optional

Explanation

Specify request body in the request message (header) or a property file, if required. If request body is not sent, you can omit this specification (Use default value).

If you specify sent file parameters, parameter specification is ignored.

Table 2–57: Sent file

Item	High <-- Priority --> Low			Default value	Possibility of specification
	Request message (header)	HTTP-adapter runtime-environment property file	HTTP-adapter runtime-environment common property file		
Input source folder name	/http-header-request/http-part/files/file/input-folder-name	adphttp.request.part.file.1.input-folder-name	adphttp.request.part.file.1.input-folder-name	-	Optional
Input source folder type attribute	/http-header-request/http-part/files/file/input-folder-name/@common	Cannot be specified.	Cannot be specified.	-	
Input source file name	/http-header-request/http-part/files/file/local-file-name	adphttp.request.part.file.1.local-file-name	adphttp.request.part.file.1.local-file-name	-	

Legend:

-: No default value.

Explanation

If a file is sent as a request body instead of the request message (body), specify these parameters in the request message (header) or a property file.

If the request body is not sent or if a request message (body) is sent as the request body, you can omit this specification.

You must always specify these parameters in sets. You cannot specify only input source folder name, input source folder type attribute or input source file name in 1 location.

Table 2–58: Output destination of the received file

Item	High <-- Priority --> Low			Default value	Possibility of specification
	Request message (header)	HTTP-adapter runtime-environment property file	HTTP-adapter runtime-environment common property file		
Output destination folder name	/http-header-request/output-folder-name	adphttp.request.output-folder-name	adphttp.request.output-folder-name	-	Optional
Output destination folder type attribute	/http-header-request/output-folder-name/@common	Cannot be specified.	Cannot be specified.	-	

Legend:

-: No default value.

Explanation

If a file is to be received, specify these parameters in the request message (header) or a property file. If a file is not to be received, you can omit this specification.

You must always specify these parameters in sets. You cannot specify only output destination folder name or output destination folder type attribute in 1 location.

2.15 Connecting to a system by using a custom reception

Service Platform provides a custom reception framework so that the user can create any reception. You can develop a reception (custom reception) corresponding to any protocol by operating user-created J2EE applications on this custom reception framework.

2.15.1 Configuring a custom reception

A custom reception is a J2EE application configured from the reception processing created by a developer and the custom reception framework.

The following describes the reception processing created by a developer and the processing executed on the custom reception framework.

Reception processing created by a developer

- Receiving a request to invoke a service component from the service requester
- Passing a request message to the custom reception framework
- Returning the response message returned from the custom reception framework to the service requester

Processing executed on the custom reception framework

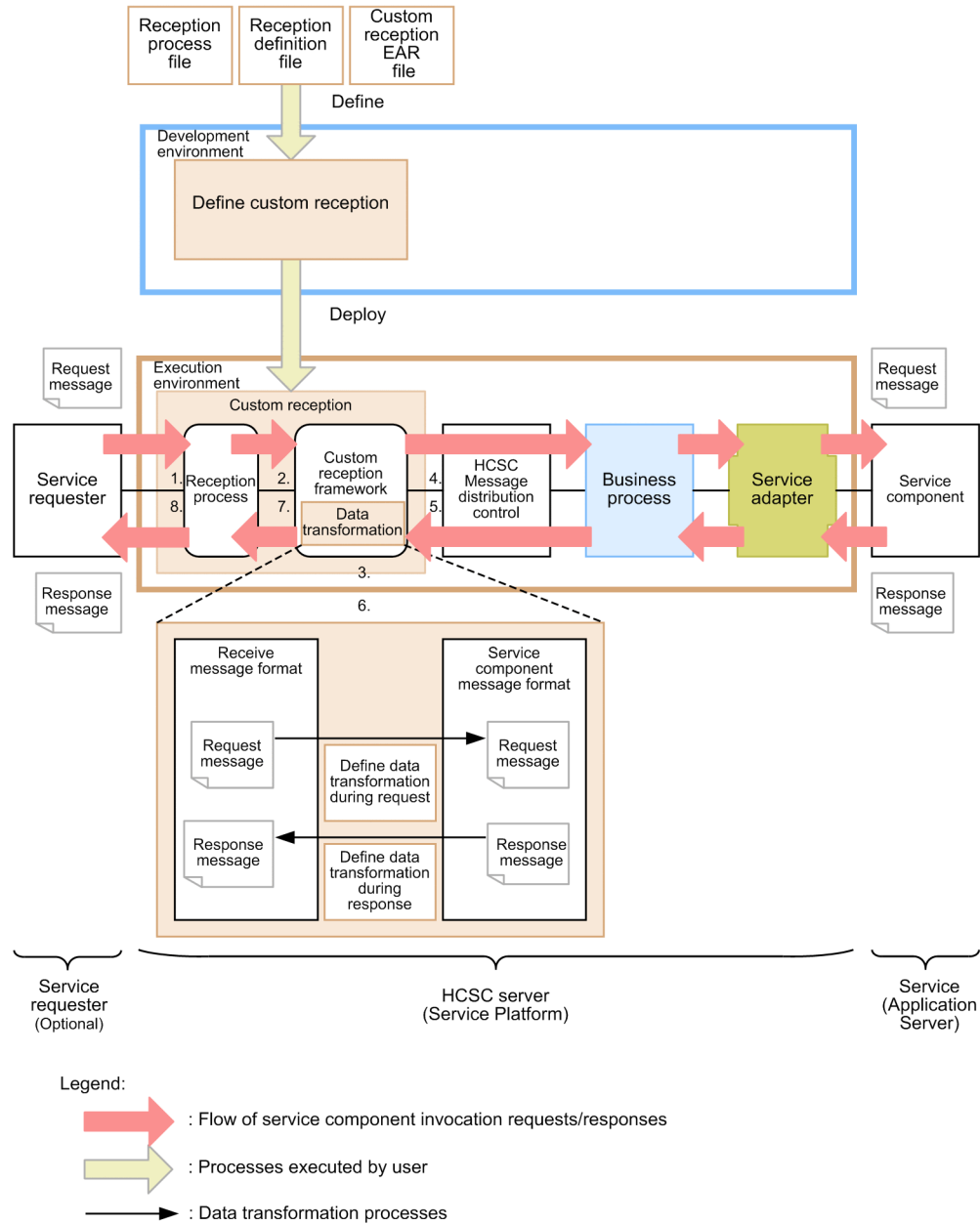
- Passing the request message passed from the reception processing to HCSC message delivery control
- Returning the response message returned from HCSC message delivery control to reception processing
- Performing data transformation for the request or response message according to the data transformation definition

2.15.2 Overview of the custom reception framework

The custom reception framework is used to develop and operate a custom reception. The custom reception framework provides a mechanism to operate a developer-created custom reception as a reception in the execution environment of Service Platform.

The following figure shows invocation of a service component by using a custom reception.

Figure 2–100: Invoking a service component by using a custom reception



Steps 1 to 8 given in the figure are described here.

Description

1. Reception processing receives a service component invocation request from a service requester.
2. Reception processing passes the request message received from the service requester to the custom reception framework.
3. The custom reception framework transforms the request message data.
4. The custom reception framework invokes HCSC message delivery control.
5. The custom reception framework receives a response message from HCSC message delivery control.
6. The custom reception framework transforms the response message data.
7. Reception processing receives the response message from the custom reception framework.
8. Reception processing returns the response message to the service requester.

2.16 Connecting to a system by using a General custom adapter

You need to develop a General custom adapter to invoke and execute service components of a system containing protocols that are not supported with the following adapters provided with Service Platform:

- SOAP adapter
- Session Bean adapter
- MDB (WS-R) adapter
- MDB (DB queue) adapter
- DB adapter
- TP1 adapter
- File adapter
- Object Access adapter
- Message Queue adapter
- FTP adapter
- File operations adapter
- Mail adapter
- HTTP adapter

Service Platform provides a custom adapter development framework for easy development of a General custom adapter.

The following provides an overview of a General custom adapter and the custom adapter development framework.

2.16.1 Overview of General custom adapters

General custom adapters are used for receiving requests from service requesters and invoking the service components. In such a case, a General custom adapter performs the following two roles:

1. Data transformation

If a General custom adapter contains a data transformation definition, the adapter transforms the message data based on the previously defined information.

2. Protocol transformation

The adapter transforms the protocol in accordance with the protocol of the service component, and then invokes a service component.

2.16.2 Overview of the custom adapter development framework

Processes performed by a General custom adapter include data transformation and protocol transformation. Among those processes, the data transformation process that is common to all custom adapters is provided with the custom adapter development framework. Therefore, the user can develop a General custom adapter by implementing the protocol transformation process.

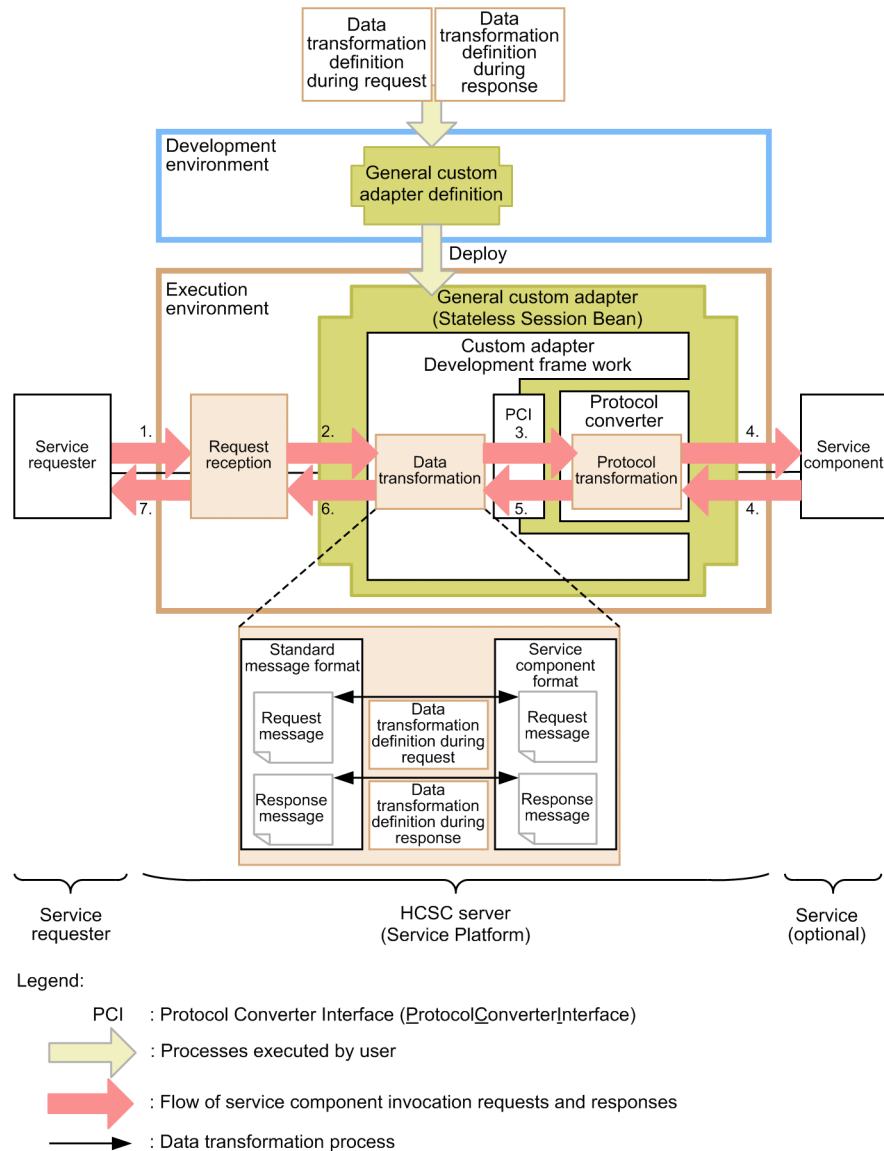
Important note

You must separately define the data transformation.

The General custom adapter consists of a Stateless Session Bean in which the custom adapter development framework is implemented, and a protocol converter. A protocol converter is a Java class that implements the protocol transformation process.

The following figure shows the invocation of service components using a General custom adapter:

Figure 2–101: Invocation of service components using a General custom adapter (when all communications are synchronous)



Steps 1 to 7 given in the figure are described here.

Description

1. The HCSC server receives a request sent to the service component to be invoked via the General custom adapter.
2. The HCSC server uses the EJB remote interface to invoke the General custom adapter implemented in the custom adapter development framework.
3. The custom adapter development framework executes the data transformation process based on previously defined information, and then invokes the protocol converter by using the protocol converter interface.
4. The protocol converter executes a transmission process in accordance with the protocol of the service component.
5. The results of service component invocation are returned to the custom adapter development framework.
6. The custom adapter development framework executes the data transformation process based on previously defined information, and then returns a response to the HCSC server.
7. A response is returned to the service requester.

2. Functionality for Connecting to Various Types of Systems

For details about developing General custom adapters by using the custom adapter development framework, see *Appendix B. Custom Adapter Development Framework* in the *Service Platform Reception and Adapter Definition Guide*.

3

Business Process Functionality

This chapter provides an overview of business processes, and describes the mechanism of business processes, how to make a business process persistent or non-persistent, transaction of business processes, re-execution of business processes, and standby processing of business processes. This chapter also describes process instances, activities, variables, and correlation sets having a deep relationship with business processes.

3.1 Overview of business processes

A business process is a series of tasks in which information such as the order of executing multiple service components and processing conditions are defined.

A business process defines the processing flow in a development environment of Service Platform. The processing flow is defined in the Define Business Process window. In this window, elements known as activities are arranged, linked together with a connection, and then defined.

For details about the Define Business Process window, see *1.2.3 Process definition Window* in the *Service Platform Reference Guide*.

This section describes the following main nine elements configuring a business process:

- Reception and response
- Service invocation
- Java invocation
- Data transformation
- Assignment
- Repetition
- Switch
- Flow
- Standby

3.1.1 Reception and response

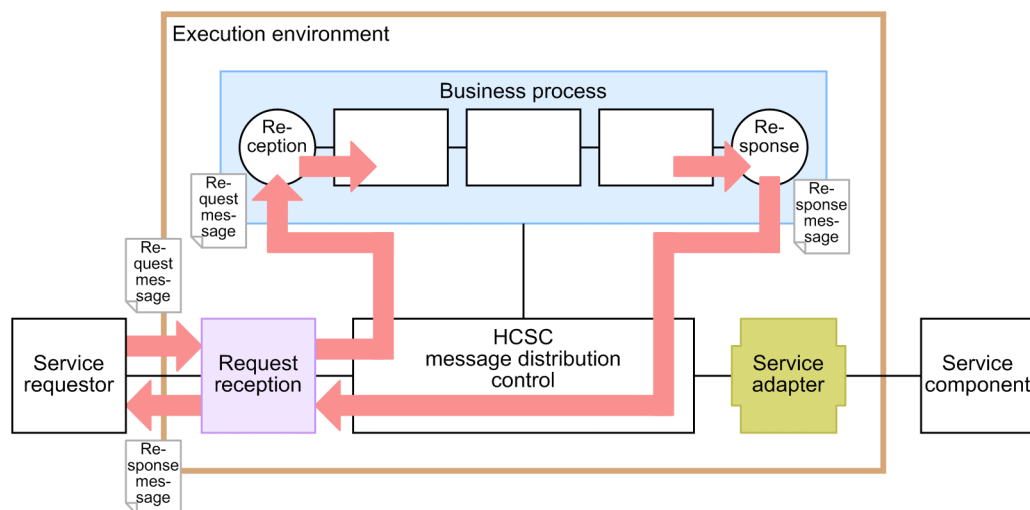
(1) Function


The reception end of a business process receives requests from service requesters, and the response end returns responses from the business process to the service requester.

You can use both standard reception and user-defined reception in the request reception that receives requests from service requesters.

The following figure shows the flow of reception and response of a business process.

Figure 3–1: Flow of reception and response of a business process



Legend:  : Flow of business process reception and response

(2) Activities to be used

Use the receive activity for reception, and use the reply activity for response.

You can define either of the following communication models for reception:

- A synchronous business process in which a response message is returned
- An asynchronous business process in which a response message is not returned

3.1.2 Service invocation

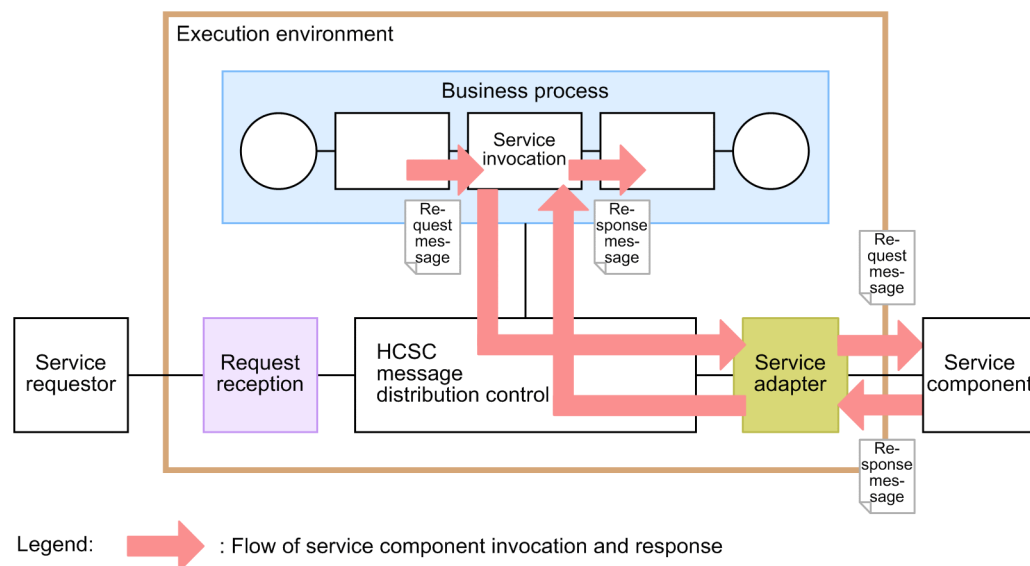
(1) Function

With service invocation, you can execute service components of different service adapters and business processes defined in the development environment of Service Platform.

Depending on the communication model of the service adapter and business process, you can invoke service components for synchronous models in which a response message is returned and for asynchronous models in which no response message is returned.

The following figure shows the process flow when a service adapter is executed with service invocation.

Figure 3–2: Process flow when a service adapter is executed with service invocation



(2) Activity to be used

Use the invoke service activity for service invocation.

3.1.3 Java invocation

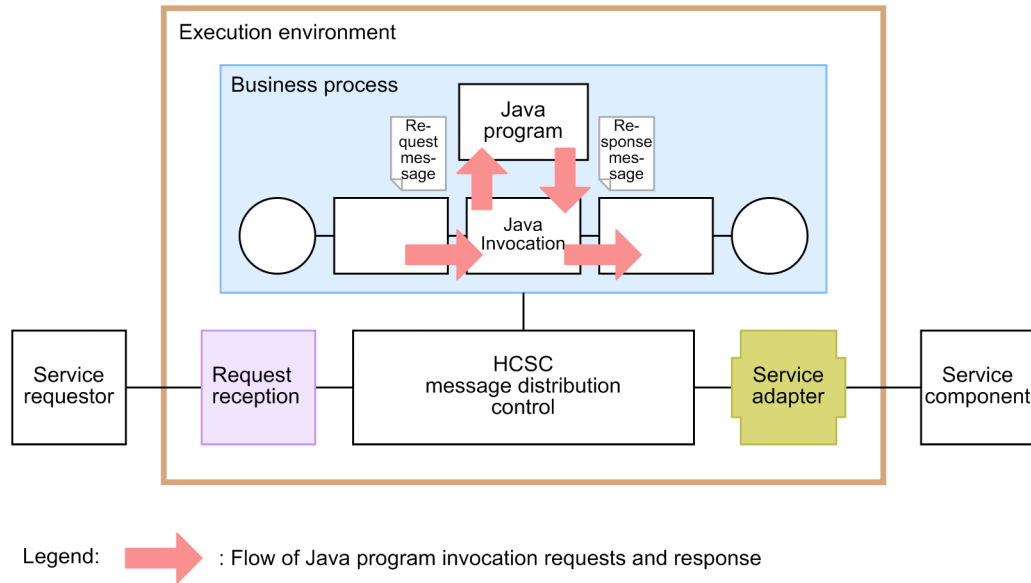
(1) Function

With Java invocation, you can execute the Java class in which an interface provided with Service Platform is implemented.

If you cannot define a business process only by the functionality provided with Service Platform, you can use user-created Java programs to perform processing such as transformation and judgment of XML messages.

The following figure shows the process flow when a service adapter is executed by using Java invocation.

Figure 3–3: Process flow when a service adapter is executed by using Java invocation



(2) Activities to be used

Use the invoke java activity for Java invocation.

Execute a Java program with the message defined in the argument in the invoke java activity and with the variables in the business process.

The return value of the Java program is the message defined in the return value of the invoke java activity and the variables in the business process.

Note that the Java program to be executed must be created on the Java editor by using the invoke java activity in the development environment of Service Platform.

3.1.4 Data transformation

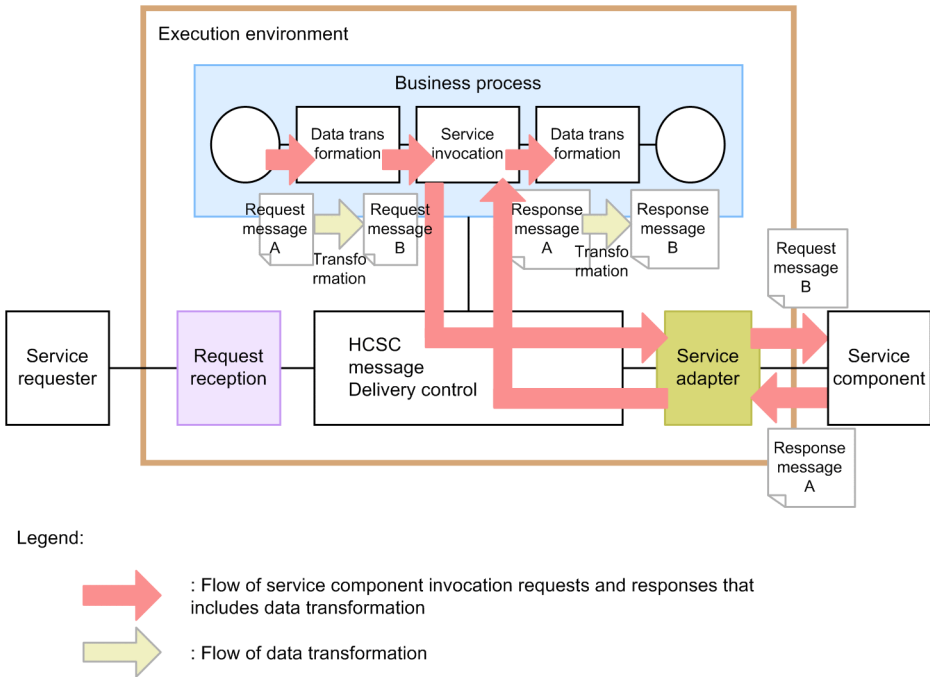
(1) Function

With data transformation, you can convert the message format that flows over the business process to a different message format, and process the values. Furthermore, you can also combine multiple messages to create one message.

Use data transformation in cases such as when you must create a message as per the service component request message format at the time of invoking a service component, and when you want to convert the response message from the service component to a different message format.

The following figure shows the flow of data transformation in a business process:

Figure 3-4: Flow of data transformation in a business process

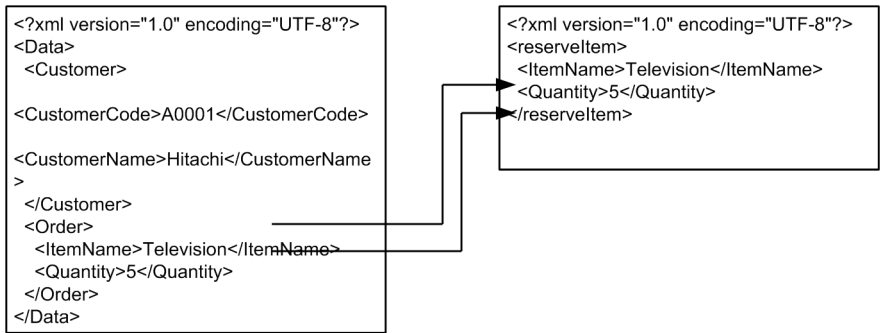


(2) Activity to be used

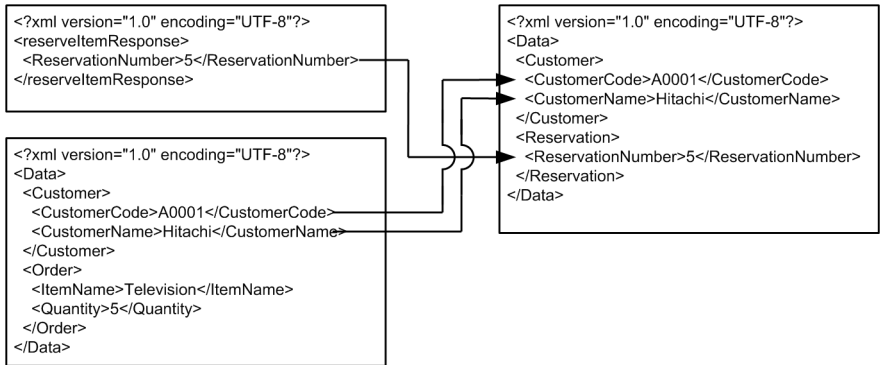
Use the data transformation activity for data transformation.

The following figure shows an example of data transformation:

(Example 1) 1 on 1 data transformation



(Example 2) N to 1 data transformation



The following table describes the processes that can be performed by data transformation:

Table 3–1: List of processes that can be performed by data transformation

Process	Description
Concatenate	Concatenates multiple strings.
Acquire substring	Extracts some portion from one string.
Acquire string length	Assigns string length of the string to the mapping destination.
Check string	Checks if the specified string is included in the string and the string starts with the specified string, and assigns the logical value to the mapping destination.
Trim node	Removes any blank space in the first character and last character of the string. Furthermore, converts the spaces between strings into one-byte space.
Convert number format	Converts the number format. You can change the decimal-point character and the digit separator.
Perform node operation	Does number calculation. (+, -, *, div, mod).
Round node	Rounds up numbers after the decimal. (Round-off, truncate, round-up)
Sum up nodes	Adds up node numbers of multiple node sets.
NOT operation	Executes a logical negation operation.
Logical operation	Executes logical operation. (AND, OR, XOR).
Shift operation	Executes shift operation.
Acquire node count	Assigns a node to the mapping destination.
Acquire node name	Assigns a node name to the mapping destination.
Check node	Checks if there are specified conversion source nodes and assigns the logical value to the mapping destination.
Repeat	Maps repeat. You can also sort instances of conversion destination nodes that appear more than once.
Select	Outputs a different value depending on the conditions.
Change value	Converts the mapping source value based on the specifications in the conversion table, and assigns to mapping destination.
Base conversion	Executes base conversion.
Custom	Can process and create nodes of the value in the Java program created by user.
Set constant	Assigns a specified value to the mapping destination.

3.1.5 ASSIGN

(1) Function

During `assign`, you can transform the message format running on the business process to a different message format.

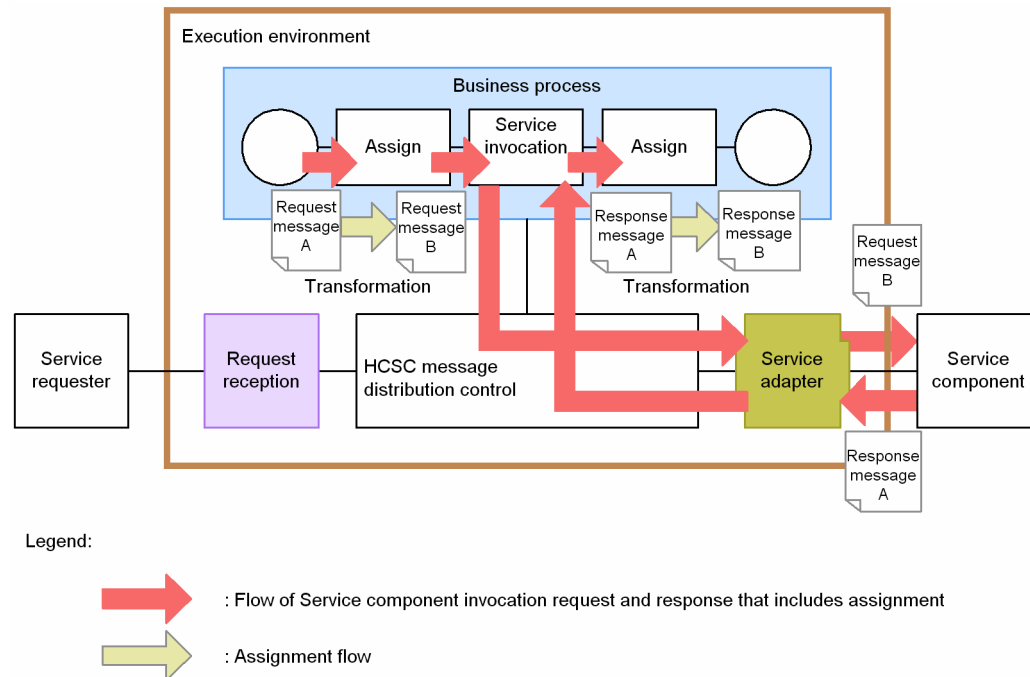
The main functions of `assign` are as follows:

- To copy an XML message to an XML message of another name.
- To copy a part (sub tree) of an XML message to another XML message (insert).
- To copy variables other than XML, such as `boolean`, `numeric`, and `string` to other variables.

Although `assign` has same functions for data transformation, `assign` is applicable when you want to copy an entire XML or a part, rather than processing individual values within an XML message.

The following figure shows the `assign` procedure in a business process:

Figure 3–5: Assign procedure in a business process

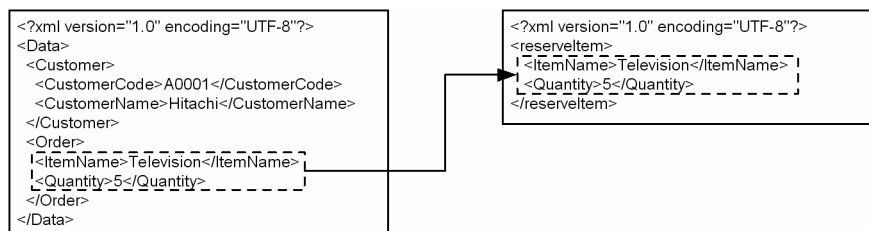


(2) Activity to be used

You use the `assign` activity for assignment.

An example of assignment is shown as follows:

Example: Copying a sub tree



3.1.6 Repetition

(1) Function

With repeat, you can repeat the process (loop process) in the business process under specified conditions.

Deploy one or more activities in the repeat process.

The following are the two methods of the repeat process:

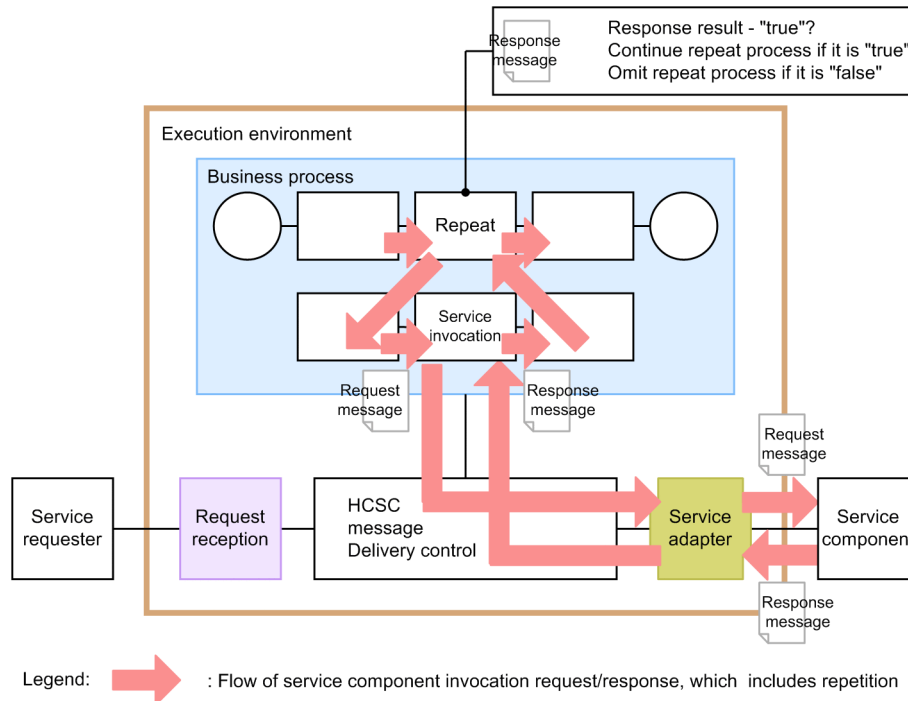
- Repeat condition setting method
Set the repeat conditions. Repeat the process till the repeat conditions are satisfied.
- Specified list repetition method

Extract the list of elements. Repeat the process till there are elements in the list.

In the repeat condition setting method, the judgment to repeat is done beforehand. If the judgment result is true, repetition continues, and in case of false, repetition stops and you transit to the next process.

The following figure shows the flow of repetition in the business process for the repeat condition setting method:

Figure 3–6: Repeat in business process (for the repeat condition setting method)



In the specified list repetition method, a list for repetition is generated based on the variables that define the repeating element. The process is repeatedly executed only for the number of nodes in the repeat list.

For details on the flow of repeat in case of the specified list repetition method, see "5.6.13 While activity" in "BPM/ESB Service Platform Basic Development Guide".

(2) Activity to be used

Use the while activity.

Specify the condition formula to be used for judging repetition and the path of the variable element that defines the repeating element, in XPath for variables such as XML message.

Specify the method to be used in the while activity dialog from among the repeat condition setting method or the specified list repetition method.

In order to avoid falling into an infinite loop with the repetition process, you can specify maximum number of repetitions. However, when you reach the maximum value for the number of repetitions, an error occurs in the business process.

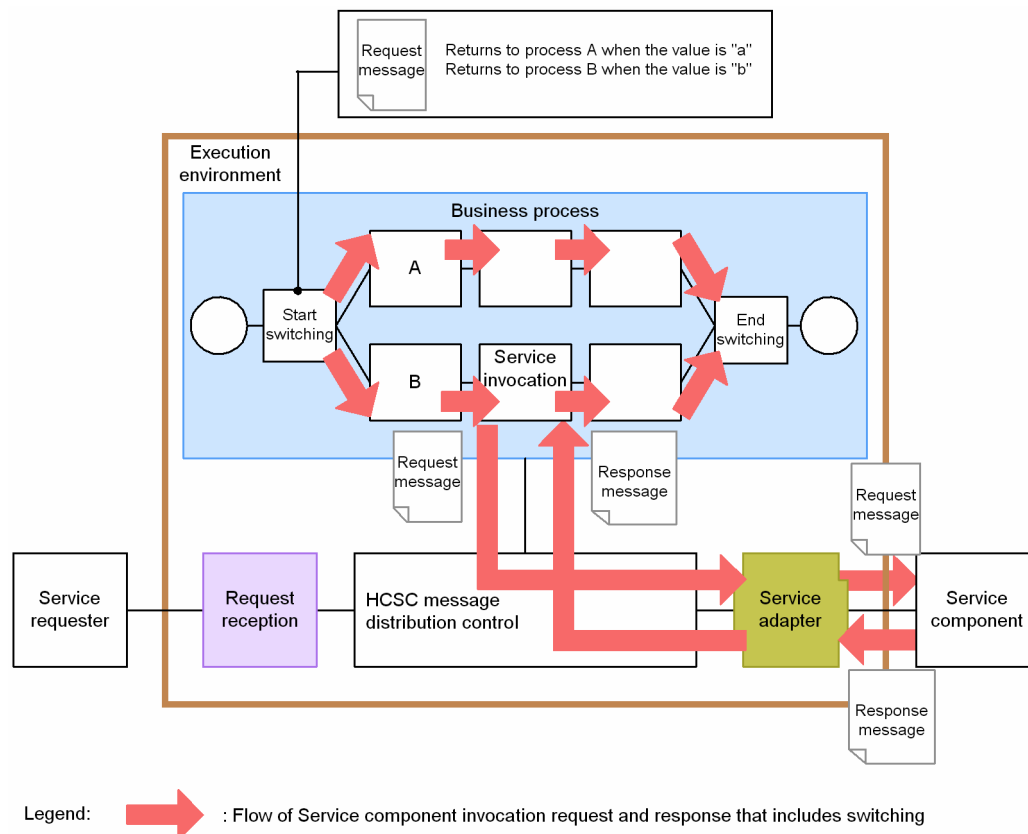
3.1.7 SWITCH

(1) Function

During `switch`, you can switch a processing within a business process depending on the judgment result of the condition. A single or multiple activities are placed in the processing of the `switch` destination.

The following figure shows the `switch` procedure in a business process:

Figure 3-7: Procedure of switch in a business process



(2) Activities to be used

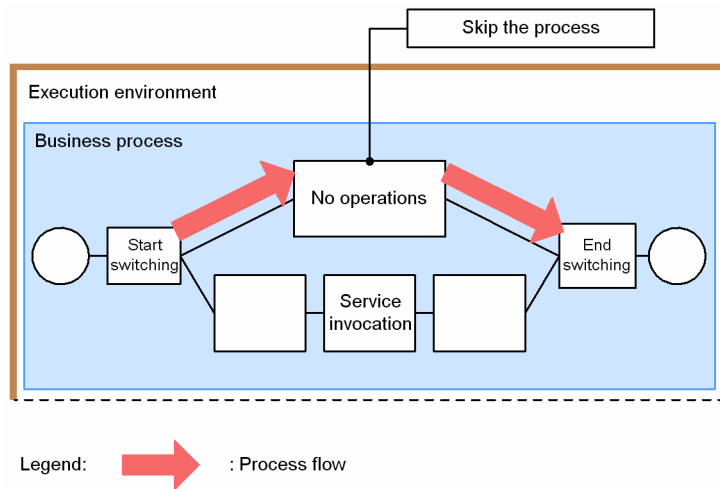
Use the `switch start` activity at the location in which switching starts and the `switch end` activity at the location in which switching ends. You can set up multiple (two or more) switch destinations.

The condition expression used for the judgment of switch is specified as an XPath for variables such as the XML message. Based on this condition expression, specify the activity at each transition destination.

Note that you must set up one or more activities between a `switch start` activity and the corresponding `switch end` activity. Therefore, if you want to transit to the next process without executing any processing in one switch destination, place an empty activity.

The following figure shows an example of arrangement of activities when you want to transit to the next process without executing any processing in one switch destination:

Figure 3–8: Example of arrangement of activities when you want to transit to the next process without executing any processing in one switch destination



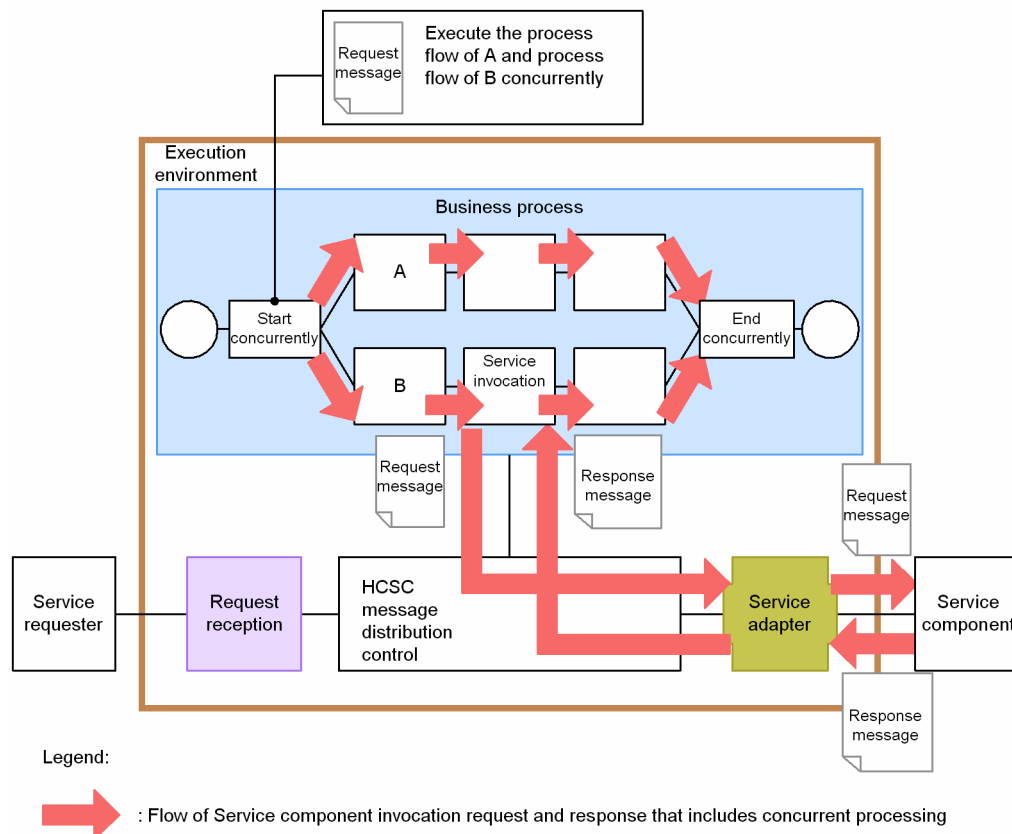
3.1.8 FLOW

(1) Function

During `flow`, you can divide the processing flow into two or more flows and can sequentially proceed with each flow.

The following figure shows the execution of flow in a business process:

Figure 3–9: Execution of flow in a business process



(2) Activities to be used

Use the `flow start` activity at the location in which the flow starts and the `flow end` activity at the location in which the flow ends.

By using a link connection, you can also control the order of the individual process flows to be executed in parallel.

Note that during `flow`, individual process flows are executed sequentially. You cannot execute multiple process flows simultaneously.

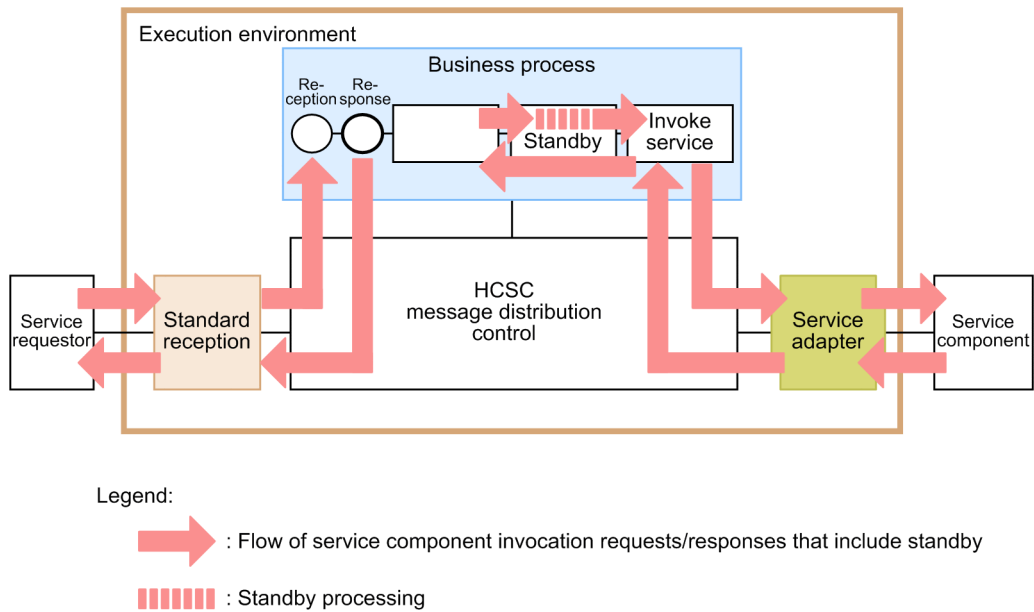
3.1.9 Standby processing

(1) Function

During standby processing, you can wait for the processing of a business process for a specified period of time or until a specified time is reached.

The following figure shows the flow of standby processing in a business process.

Figure 3–10: Flow of standby processing in a business process



(2) Activity to be used

Use the wait activity for standby processing.

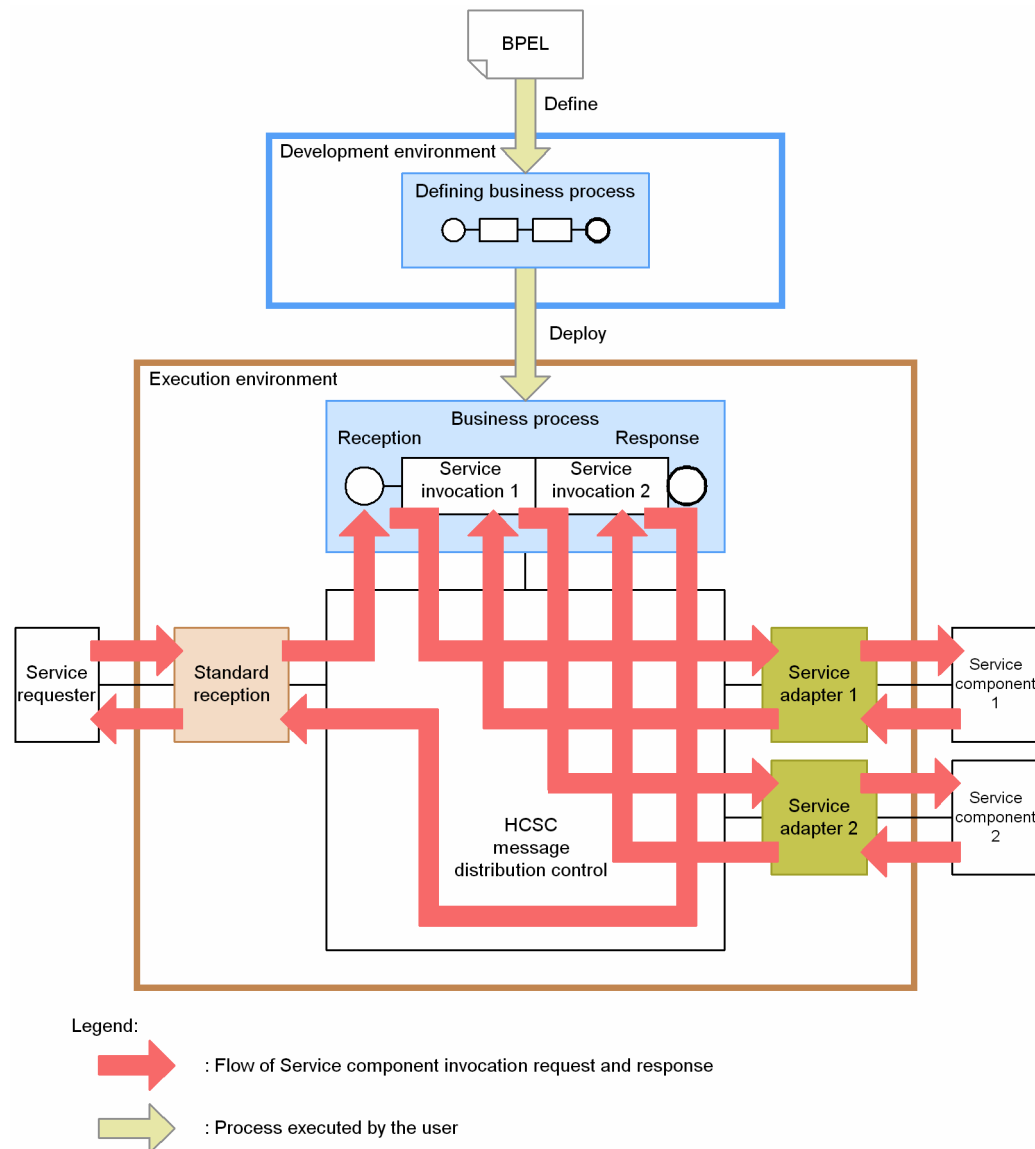
Use the XPath expression to specify the standby interval or standby expiration time. The XPath expression is evaluated during processing of the business process, and the character string obtained as a result of evaluation is used as the standby time.

3.2 Mechanism of business processes

3.2.1 Business process flow

When you invoke a business process from a service requester, the process is performed by executing elements and activities defined in the invoked business process one by one. You define the business process in the development environment. The following figure shows the flow of processing of a business process:

Figure 3–11: Business process flow



3.2.2 Communication models of business processes

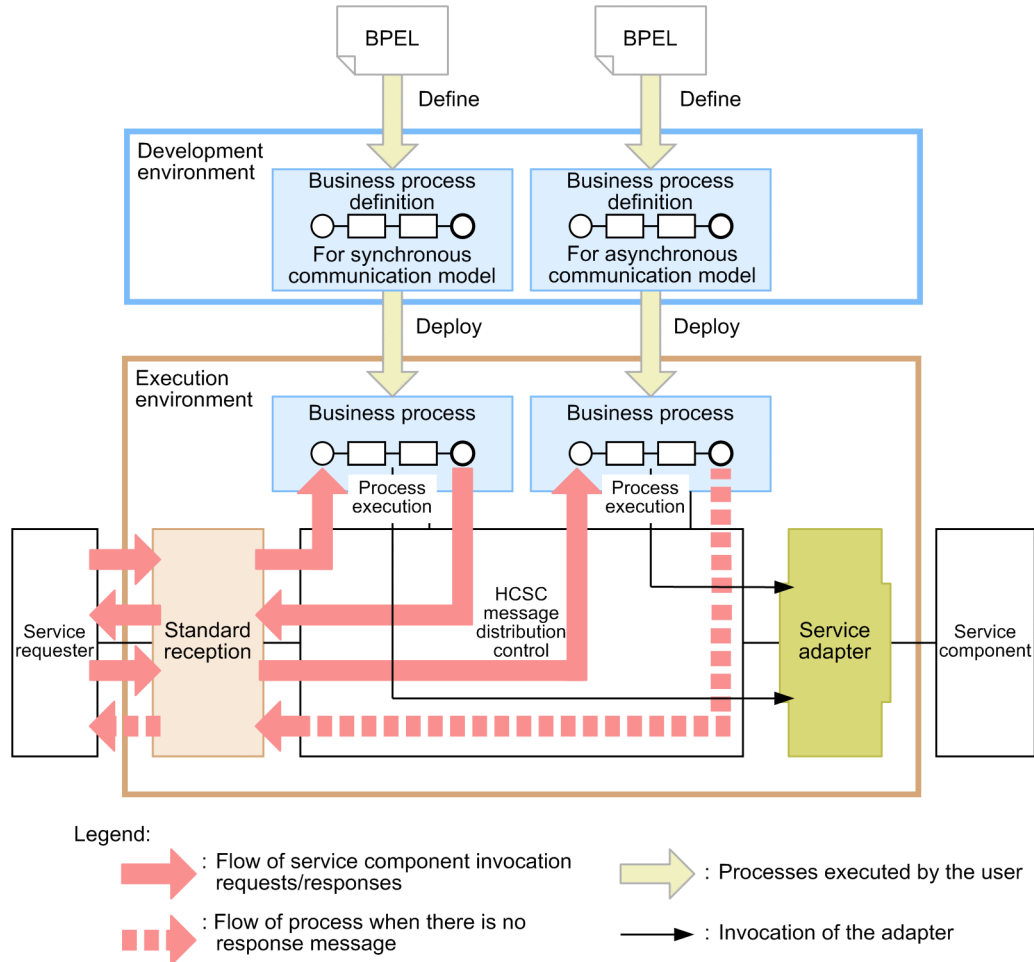
Business processes have two types of communication models: synchronous and asynchronous communication models.

For a synchronous communication model, a response message (in the format defined in the reply activity) is returned to the service requester that invoked the business process.

For an asynchronous communication model, only the processing is returned (no response message is returned) to the service requester that invoked the business process.

The communication model is defined in the Receive Activity dialog box when a business process is defined in the development environment. The following figure shows differences in the communication models of a business process.

Figure 3–12: Differences in the communication models of a business process



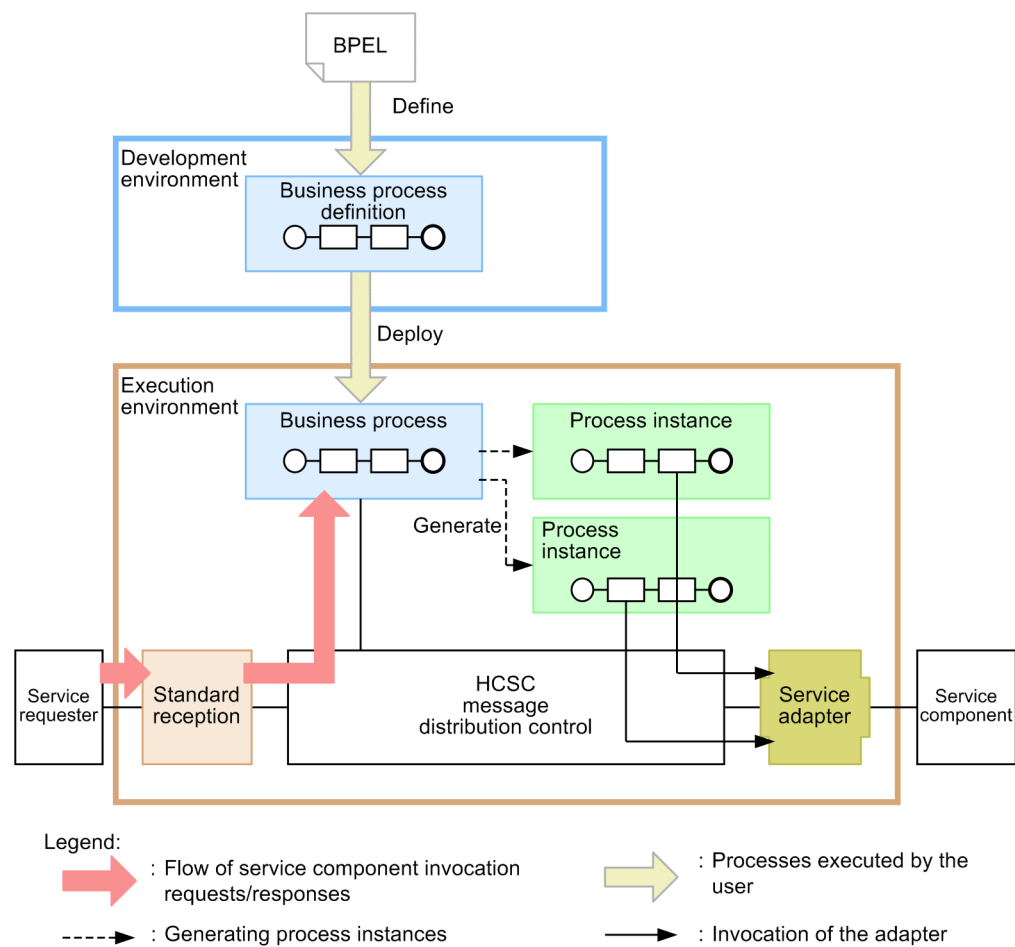
For details about the Receive Activity dialog box, see *1.4.7 Receive Activity dialog box* in the *Service Platform Reference Guide*.

3.2.3 Process instances

A process instance is generated when a business process initially receives a request. To generate process instances, when you define a business process in the development environment, set the instance generation to `yes` in the Receive Activity dialog box that defines the receive activity that initially receives a request.

For a business process for which process instances are to be generated, a process instance is generated each time the business process receives a request. Therefore, even when multiple requests are executed simultaneously, processes are executed independently for each request. The following figure shows the mechanism of process instances in a business process.

Figure 3–13: Mechanism of process instances in a business process



For details about the Receive Activity dialog box, see *1.4.7 Receive Activity dialog box* in the *Service Platform Reference Guide*.

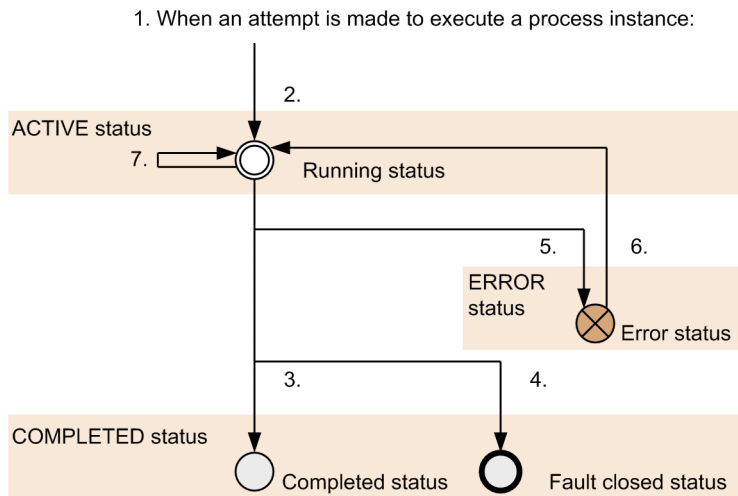
The following table describes the attribute information of process instances.

Table 3–2: Attribute information of process instances






Attribute	Description
Identifier	Identifier for identifying individual process instances. The identifier is unique within the same business process definition. (The version might be different in some cases.)
Business process definition name	Definition name of the business process to which the target process instance belongs. This attribute corresponds to the service name used during invocation from the service requester.
Business process definition version	Definition version of the business process to which the target process instance belongs.
Status	Status of the target process instance
Start time	Start time (GMT) of the target process instance Note: The time when the process instance transits to ACTIVE status
End time	End time (GMT) of the target process instance Note: The time when the process instance transits to COMPLETED status

The following figure shows the status transition of a process instance.

Figure 3–14: Status transition of a process instance



Legend:

-  : Running process instance
-  : Completed process instance
-  : Process instance in fault status
-  : Process instance in error status
-  : Status transition of process instance

1. When you try to execute a process instance
2. When you try to execute an activity belonging to the process instance
3. When the global scope belonging to the process instance has been completed and the process instance terminates normally
4. When a fault occurs during execution of the process, and the process instance terminates in a fault because the fault could not be caught even in the global scope
5. When an exception occurs in the invoke service activity or scope activity, and the process instance is interrupted with an error
6. When you re-execute a process instance in which an error has occurred
7. When you re-execute a process instance that is already running

The following table describes the status of a process instance shown in *Figure 3-14 Status transition of a process instance*.

Table 3–3: Attribute information of process instances

Status		Description
ACTIVE status	Executing	The process instance is being executed.
ERROR status	Error	An exception occurred in the invoke service activity or scope activity, and the process instance is interrupted with an error.
COMPLETED status	Completed	The process's global scope has been completed (process instance is in the status of completed).
	Faulted	A fault occurred, and the process instance terminates in a fault because the fault could not be caught even in the global scope.

3.2.4 Activity

Activity is a business process component and represents the processing structure. The flow of the business process is defined by linking multiple activities.

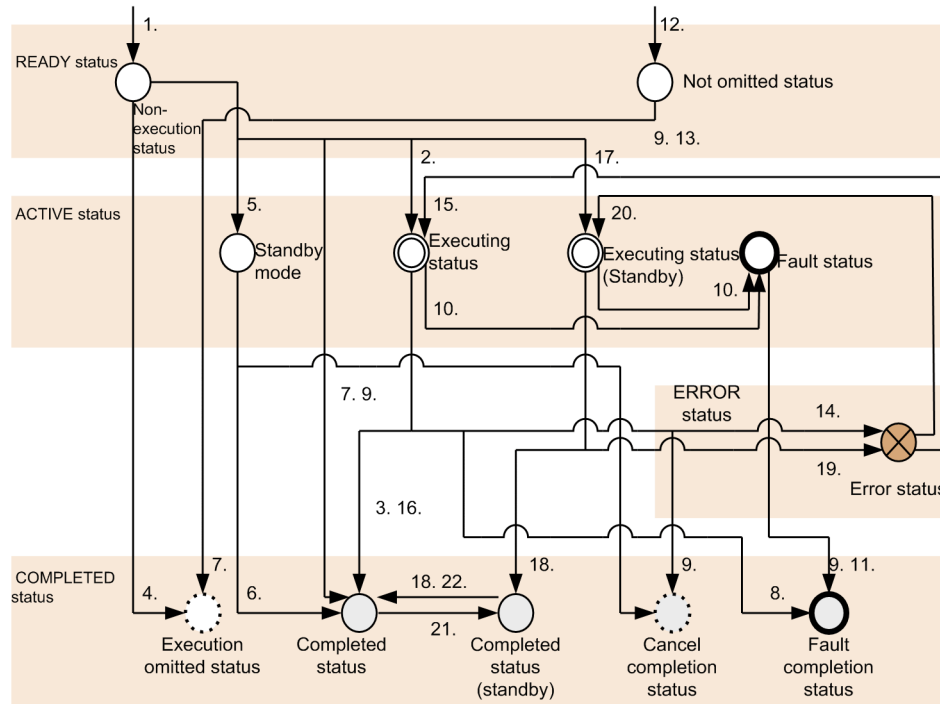
Define activity by deploying and linking activities in the Define Business Process Window in the development environment. The following table describes the attribute information of all activities:

Table 3–4: Attribute information of activity

Attribute	Description
Process instance identifier	Process instance identifier to which the target activity belongs.
Activity number	Number to identify activity. Corresponds to the index of activity list.
Business process definition name	Definition name of the business process to which the target activity belongs. Corresponds to the service name when invoking from a service requester.
Business process definition version	Definition version of the business process to which the target activity belongs.
Activity definition name	Activity definition name of the target activity.
Activity type	Activity type of the target activity.
Status	Status of the target activity.
Start time	Start time (GMT) of the target activity. Note Time when the activity transits to ACTIVE status
End time	End time (GMT) of the target activity. Note Time when activity transits to COMPLETED status
Standby cancellation time	Standby cancellation time (GMT) for target activity. Use it for wait activity.

The following figure shows the status transition of the activity:

Figure 3–15: Status transition of activity



Legend:

- Activity instance that is being executed
- :Non-completed activity instances, other than those being executed
- : Completed activity instance
- : Activity instances that became non-available in-between the process
- : Activity instance of fault status
- : Process instance of error status
- : Activity instance status transition

1. When an instance is generated for a new activity.
2. When you try to execute the activity instance.
3. When the execution of the activity instance is complete.
4. When the activity instance was executed but the execution was completed without the instance reaching the executing status.
5. When you try to execute the receive activity instance other than the reception for which a request is received from the service requester.
6. When a receive activity instance in waiting status receives a request from the service requester.
7. When target link is activated in false.
8. When a fault occurs at the time of executing the activity instance.
9. When a fault occurs and cancellation of activity execution is requested.
10. When a fault occurs and activity instance to be caught and handled with fault handler is executing.
11. When a fault occurs and the execution of activity instance handled in the catch is completed.
12. When you omit an enclosed activity in case a structured activity is omitted.
13. When you omit an activity in the unomission status.
14. When there is an exception in the invoke service activity process, and the activity instance process is aborted due to error.

15. When you perform re-execution of an activity instance that is an invoke service activity in error status.
16. When you perform re-execution of a wait activity instance which is in executing status.
17. When you try to execute a scope activity instance that controls transaction.
18. When the execution of a scope activity instance that controls the transactions is complete, transits to ("Completed (wait)", and later transits to "Completed".
19. When an exception occurs in a scope activity instance that controls the transactions and, the activity instance process is aborted due to error.
20. When you perform re-execution of a scope activity instance that controls transactions that are in an error status.
21. When you perform invocation after the execution of the reply activity in an asynchronous EJB.
22. When invocation is performed after the execution of the reply activity by an asynchronous EJB.

The following table lists and describes the status of the activity as shown in "Figure 3-15 Status transition of activity".

Table 3–5: Status of activity

Status		Description
READY status	Unexecution	Status in which the activity instance is not executed and is likely to be executed in the future.
	Unomission	Status in which the activity instance is not executed and is likely to be omitted in the future.
ACTIVE status	Executing	Status in which the activity instance is being executed.
	Executingwait	Status in which rollback is performed, and scope activity instance can be re-executed.
	Awaiting	Status in which the activity instance is received.
	Faulting	Activity instance is executing an activity caught by the faultHandler.
ERROR status	Error	An exception occurs in the invoke service activity or scope activity process and, the activity instance process is aborted due to error.
COMPLETED status	Completed	Completed status in which the activity instance is executed normally.
	Completedwait	Status in which the activity instance is normally executed. Instance of activity is likely to be re-executed.
	Canceled	Status in which a fault occurs, and execution of the activity instance is canceled.
	Faulted	Status in which a fault occurs during the execution of the activity instance.
	Omitted	Status in which the execution of the activity instance is omitted.

For details on all activities, see "5.6 Defining activities" in "Service Platform Basic Development Guide".

3.2.5 Types and roles of variables

In a business process, the values included in a message and the values used in the business can be maintained as variables. Variables include global variables and local variables, which are defined in the Define Business Process window in the development environment. The following table describes the roles of variables.

Table 3–6: Roles of variables

Variable	Description
Global variable	A global variable that is in the single process instance can be referenced from anywhere. Also, because the value of the variable is recorded in the database even when the process has been completed, the

Variable	Description
Global variable	progress status of the entire business process can be analyzed. The variable name is unique within a process.
Local variable	A local variable can be referenced within the declared scope. When the declared scope terminates, the value of the variable is also deleted from the database. Therefore, local variables can be used to temporarily reference values such as condition judgment during execution of the process. The variable name is unique within a scope. The same variable name can be declared for different scopes. However, when referencing a variable, the variable declared in the innermost scope at the reference location is referenced.

For details about the variable definitions, see *5.5.1 Defining variables* in the *Service Platform Basic Development Guide*.

3.2.6 Role of a correlation set

A correlation set is a character string used to uniquely identify the messages sent by service requesters. The XML type is the only message type that can be used in correlation sets.

If the HCSC server contains multiple process instances that can receive messages from service requesters for a single business process, the HCSC server uses the value of the correlation set included in the message to identify the process instances.

The service requester creates a request message that includes the correlation set, and requests invocation of the service components (invocation of the business process) to the HCSC server. The correlation set is also included in the response from the business process, and enables the user to identify the result for any sent message. Correlation sets are defined in the Define Business Process window in the development environment.

Note that a new correlation set can be specified and a message can be sent even when the business process invokes a service component or receives a response from the service component.

For details about identifying process instances based on the correlation set, see *5.5.2 Defining correlation sets* in the *Service Platform Basic Development Guide*.

3.3 Business processes that are to be made persistent and that are not to be made persistent

Business processes include those for which the execution status and execution log is made persistent in the database, and those for which these are not made persistent. To *make persistent* implies that the execution status and execution log of a process is recorded in the database.

The differences between these business processes are described in the following subsections.

3.3.1 Business processes that are to be made persistent

Execution status and execution logs of a business process to be made persistent are recorded, allowing you to easily follow process execution progress, and re-execute the process if an error occurs. You can specify a persistent business process when adding the business process in the development environment.

When you perform the following operations, set the business processes to be persistent:

Managing process instance execution logs

- Collecting process instance execution logs by using commands
- Collecting process instance execution logs by using windows in the operating environment

Re-executing process instances

- Re-executing process instances by using commands
- Re-executing process instances by using windows in the operating environment
- Re-executing process instances by using a service requester (SOAP communication or Session Bean)

For details about how to add a new business process, see *5.2.1 Adding a business process* in the *Service Platform Basic Development Guide*.

Important note

For persistent business processes, you cannot use the specified list repetition method of the while activity.

3.3.2 Business processes that are not to be made persistent

If business processes are not made persistent, the execution status and execution logs are not recorded. Therefore, specify the non-persistent setting for business processes that are required to exhibit high performance. You can specify a non-persistent business process when adding a business process in the development environment.

Note that some processes can be performed only in a persistent business process (not in a non-persistent business process) due to restrictions. You can define a non-persistent business process if the following conditions are met:

1. The business process does not include an invoke service activity that invokes services in an asynchronous service adapter.
2. The business process does not include a wait activity.
3. The business process does not include multiple receive activities.
4. Process instances are not re-executed. (If an error occurs, the service invocation request (business process invocation) is re-executed from the service requester.)

For details about how to add a new business process, see *5.2.1 Adding a business process* in the *Service Platform Basic Development Guide*.

3.4 Transaction of a business process

For a persistent business process, output the status of an activity to database when executing the business process.

This sub-section describes the timing to determine the status of an activity (start transaction and commit transaction).

The following table describes timings for the start and commit of transactions:

Table 3–7: Timings for the start and commit of transactions

Process	Timing
Start transaction	When a message is received with the receive activity.
	Immediately after committing the transaction before a synchronous communication model sends a message with the invoke service activity. Immediately after a transaction is committed before sending a message in the invoke service activity in case of a synchronous communication model.
	Furthermore, immediately after committing the transaction before sending a message by the invoke service activity, when OFF is specified for business process status compatibility (bp-status-compatible in the HCSC server setup definition), and irrespective of the invoke service activity status compatibility (timing is same for both synchronous and asynchronous communication model).
	Immediately after committing the transaction at the time of completion of the invoke service activity process, when business process status compatibility (bp-status-compatible in the HCSC server setup definition) is specified as OFF and invoke service activity status compatibility (bp-invoke-status-compatible in HCSC server setup definition) is specified as ON.
	Furthermore, immediately after committing the transaction when an exception has occurred in the invoke service activity process, when OFF is specified for business process status compatibility and also for invoke service activity status compatibility.
	Just before executing the activity that occurs after processing the reply activity.
	At the time of re-executing the process depending on the process re-execution utility.
	Immediately after the timer is released with the interval or time limit specified for a wait activity.
	Immediately after committing the transaction when a wait activity process is complete.
	Immediately after committing the transaction at the start of a scope activity that controls transaction.
	Immediately after committing the transaction at the end of a scope activity that controls transaction.
Commit transaction	Just before responding with a message in reply activity.
	Just before sending a message in the invoke service activity in case of a synchronous communication model.
	Furthermore, just before sending a message with the invoke service activity, when OFF is specified for business process compatibility (bp-status-compatible in HCSC server setup definition) and irrespective of the specification for the invoke service activity status compatibility (timing is same for both synchronous and asynchronous communication model).
	When the process for the invoke service activity is completed, if OFF is specified for business process compatibility (bp-status-compatible in the HCSC server setup definition) and ON is specified for invoke service activity status compatibility.
	Furthermore, when an exception has occurred in the process for the invoke service activity, when OFF is specified for business process compatibility and OFF is specified for invoke service activity status compatibility.
	When the processing of the business process ends after executing all executable activities.
	Just before registering timer in TimerService for J2EE server of the interval or time limit specified in the wait activity.
	When the process for the wait activity is complete.
	When starting the scope activity that controls the transaction.
	When finishing the scope activity that controls the transaction.

The timings of the start and commit transaction differ depending on whether ON or OFF is specified for business process compatibility (bp-status-compatible of HCSC server setup definition) and invoke service activity status compatibility (bp-invoke-status-compatible in HCSC server setup definition).

Invoke service activity status compatibility is valid when the business process status compatibility is specified as OFF.

When you specify ON for the invoke service activity status compatibility, operation will be same as service platform version 08-70 or earlier.

The following are examples for respective start and commit transaction timings.

3.4.1 Transactions when ON is specified in compatibility with the business process status

This subsection describes the transactions when ON is specified in compatibility with the business process status (bp-status-compatible of the HCSC server setup definition).

Note that when ON is specified in compatibility with the business process status, the operation is the same as for the Cosminexus Service Platform version 08-00 and earlier versions.

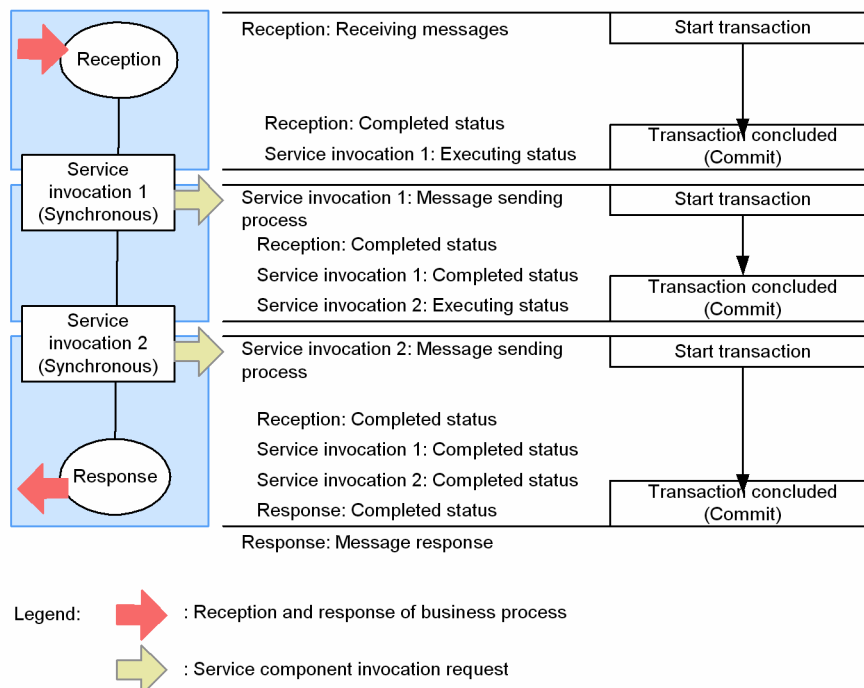
(1) When two synchronous invoke service activities are arranged

When the processing continues successfully

In such a case, the transaction is committed immediately before the message send processing of each invoke service activity, and a new transaction is started.

The following figure shows the transaction scope when the processing continues successfully during the synchronous service invocation:

Figure 3–16: Transaction scope when the processing continues successfully (Synchronous service invocation)

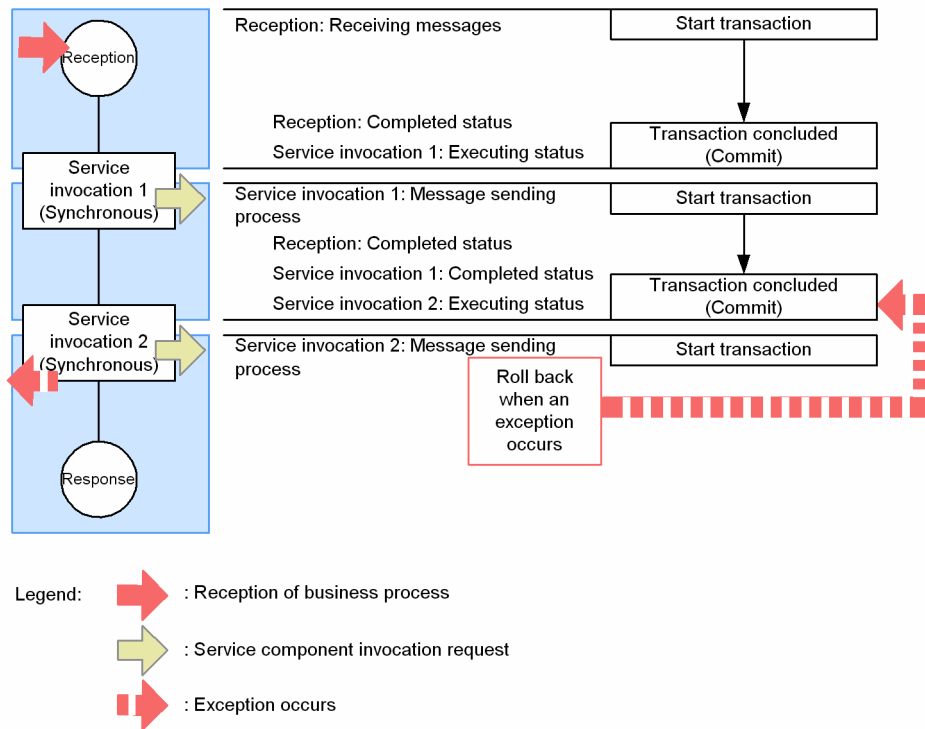


When an exception occurs

When the processing of Service invocation 2 results in an error because of an exception, a rollback occurs up to the point where the transaction was committed (Receive: Completed, Service invocation 1: Completed, Service invocation 2: Executing).

The following figure shows the transaction scope when an exception occurs during the synchronous service invocation:

Figure 3–17: Transaction scope when an exception occurs (Synchronous service invocation)



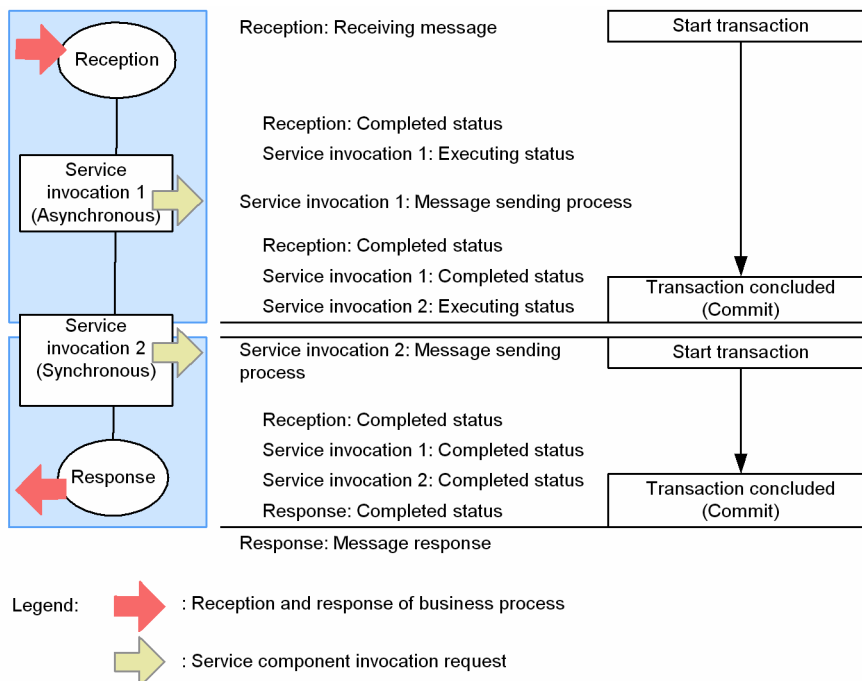
(2) When an asynchronous and a synchronous invoke service activity are arranged

When the processing continues successfully

In such a case, the transaction is not committed in the message send processing of the asynchronous invoke service activity, but is committed in the message send processing of the synchronous invoke service activity, and a new transaction is started.

The following figure shows the transaction scope when the processing continues successfully during the asynchronous and synchronous service invocation:

Figure 3–18: Transaction scope when the processing continues successfully (Asynchronous and synchronous service invocation)

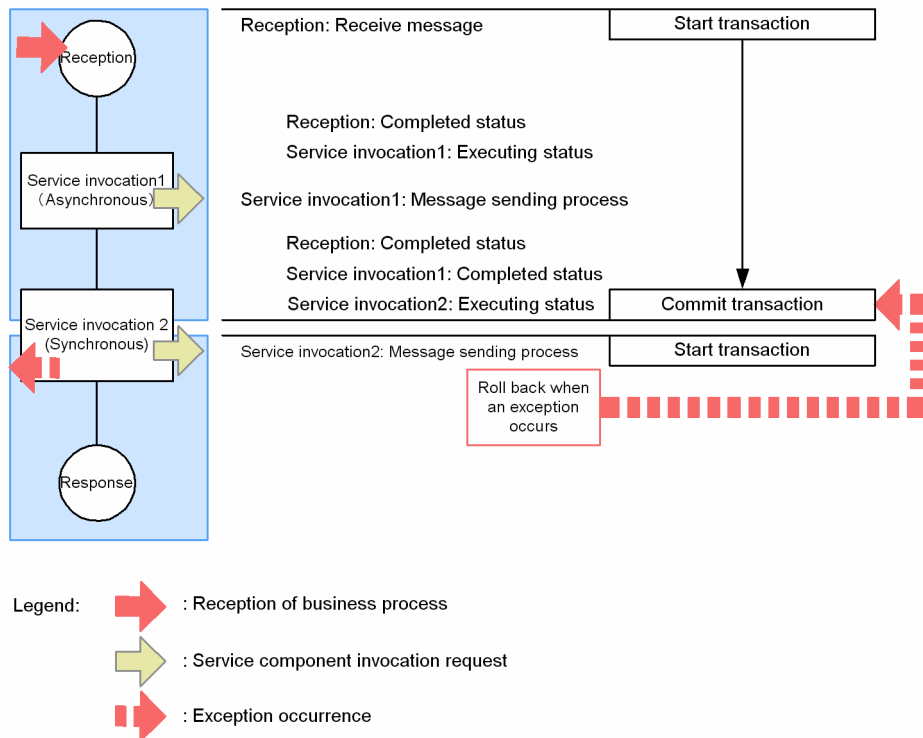


When an exception occurs (Asynchronous and synchronous service invocation)

When the processing of service invocation 2 results in an error because of an exception in a configuration where service invocation 1 is asynchronous and service invocation 2 is synchronous, a rollback occurs up to the point where the transaction was committed (Receive: Completed, Service invocation 1: Completed, Service invocation 2: Executing).

The following figure shows the transaction scope when an exception occurs during the asynchronous and synchronous service invocation:

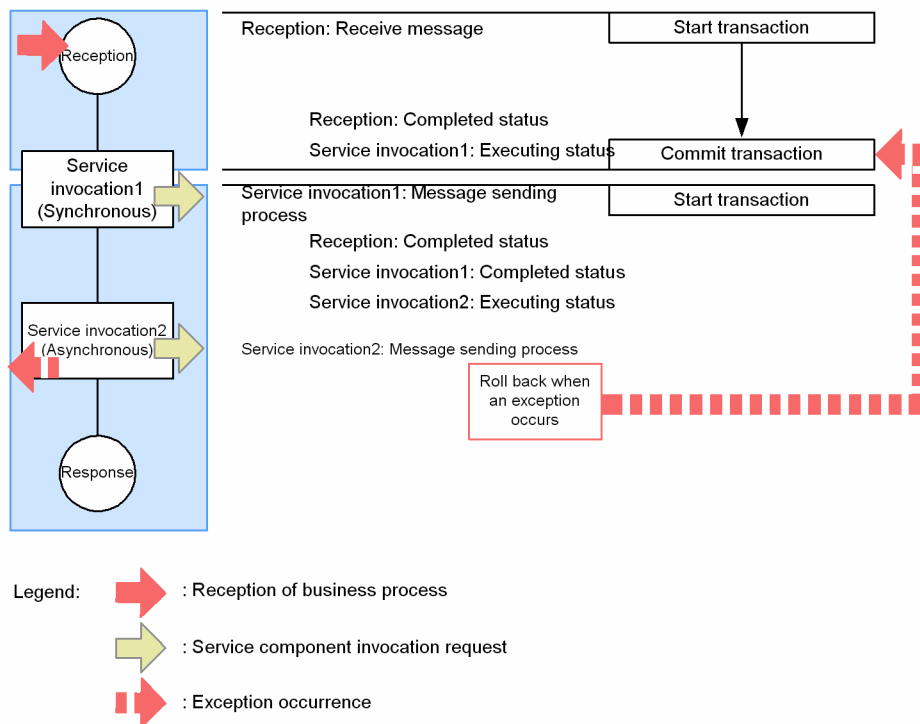
Figure 3–19: Transaction scope when an exception occurs (Asynchronous and synchronous service invocation)

**When an exception occurs (Synchronous and asynchronous service invocation)**

When the processing of service invocation 2 results in an error because of an exception in a configuration where service invocation 1 is synchronous and service invocation 2 is asynchronous, a rollback occurs up to the point where the transaction was committed (Receive: Completed, Service invocation 1: Completed, Service invocation 2: Executing).

The following figure shows the transaction scope when an exception occurs during the synchronous and asynchronous service invocation:

Figure 3–20: Transaction scope when an exception occurs (Synchronous and asynchronous service invocation)



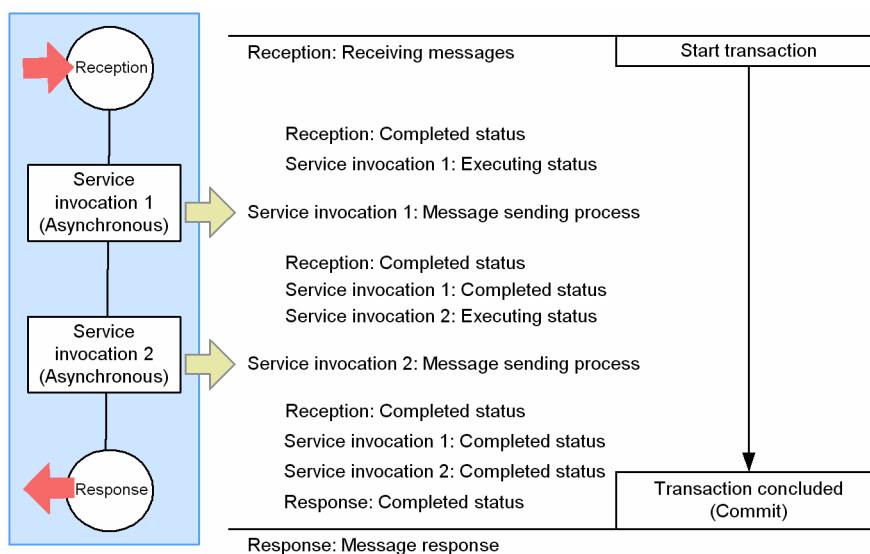
(3) When two asynchronous invoke service activities are arranged

When the processing continues successfully

In such a case, the transaction is not committed in the message send processing of the asynchronous invoke service activity. Therefore, the transaction is committed when the business process sends a response.

The following figure shows the transaction scope, when the processing continues successfully during the asynchronous service invocation:

Figure 3–21: Transaction scope when the processing continues successfully (Asynchronous service invocation)

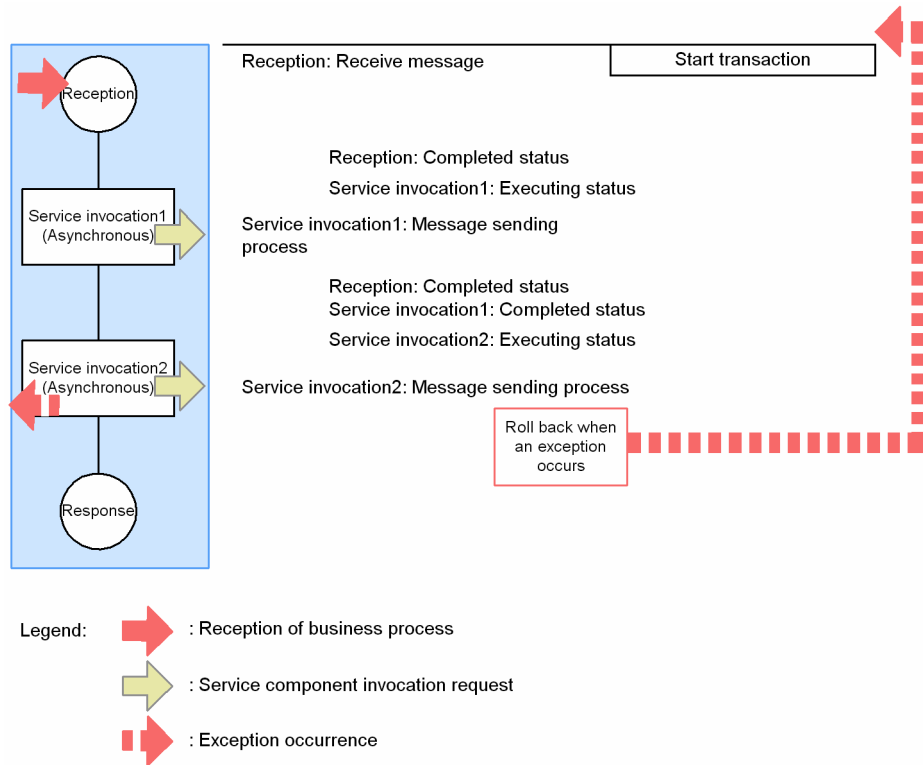


When an exception occurs

When the processing of service invocation 2 results in an error because of an exception, a rollback occurs up to the point where the transaction was committed (Receive: Unexecuted, Service invocation 1: Unexecuted, Service invocation 2: Unexecuted).

The following figure shows the transaction scope when an exception occurs during the asynchronous service invocation:

Figure 3–22: Transaction scope when an exception occurs (Asynchronous service invocation)



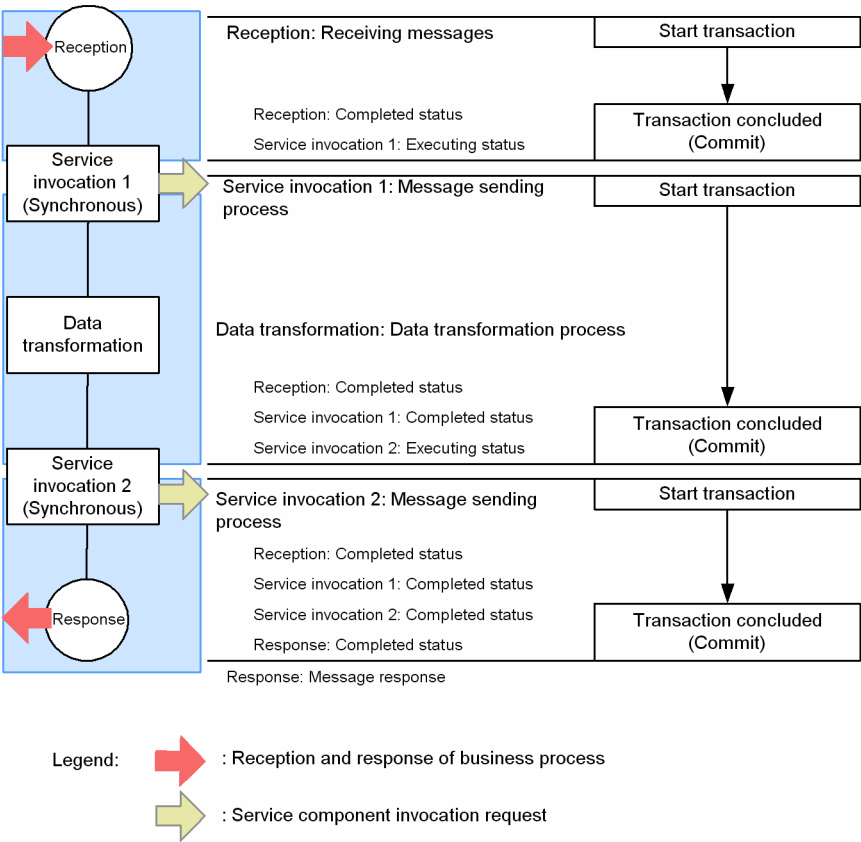
(4) When a data transformation activity exists after a synchronous invoke service activity

When the processing continues successfully

In such a case, the transaction is committed immediately before the message send processing of each invoke service activity, but the transaction is not committed in the data transformation activity. This holds true even for the activity other than the activity shown in *Table 3-7*.

The following figure shows the transaction scope when the processing continues successfully in the presence of the activities such as data transformation:

Figure 3–23: Transaction scope when the processing continues successfully (When activities such as data transformation exist)

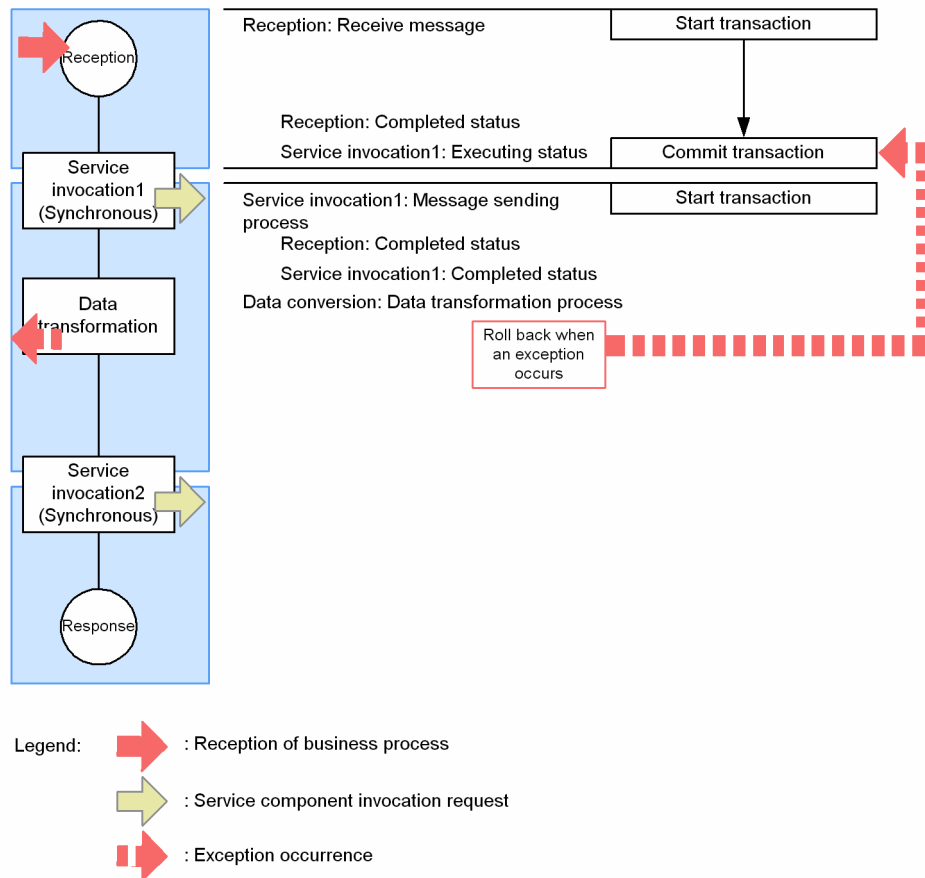


When an exception occurs (When activities such as data transformation exist after synchronous service invocation)

When an error occurs in the data transformation processing after service invocation 1 in a configuration where service invocation 1 is synchronous, a rollback occurs up to the point where the transaction was committed (Receive: Completed, Service invocation 1: Executing).

The following figure shows the transaction scope when an exception occurs in the presence of activities, such as the data transformation after the synchronous service invocation:

Figure 3–24: Transaction scope when an exception occurs (When activities such as the data transformation exist after synchronous service invocation)

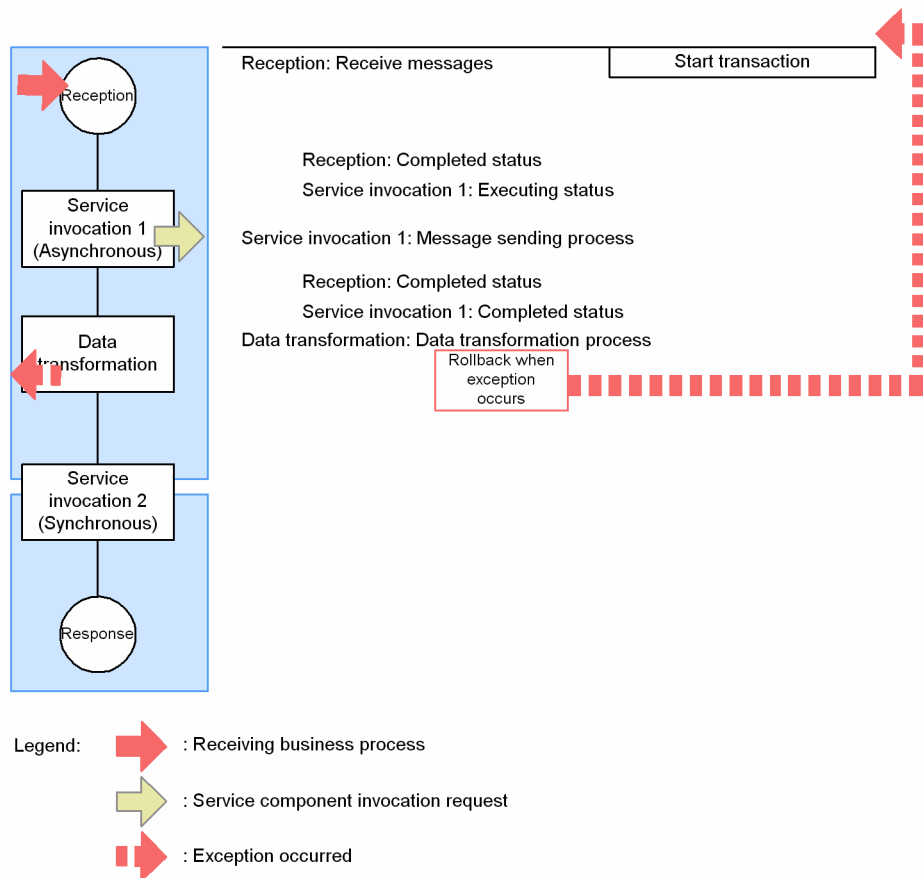


When an exception occurs (When activities such as data transformation exist after asynchronous service invocation)

When an error occurs in the data transformation processing after service invocation 1 in a configuration where service invocation 1 is asynchronous, a rollback occurs up to the point where the transaction was committed (Receive: Unexecuted, Service invocation 1: Unexecuted).

The following figure shows the transaction scope when an exception occurs in the presence of activities such as data transformation after asynchronous service invocation:

Figure 3–25: Transaction scope when an exception occurs (When activities such as data transformation exist after asynchronous service invocation)



3.4.2 Transactions when OFF is specified for compatibility with the business process status and ON is specified for compatibility with the invoke service activity status

This subsection describes transactions if you specify `OFF` for compatibility with the business process status (`bp-status-compatible`) and `ON` for compatibility with the invoke service activity status (`bp-invoke-status-compatible`) in the HCSC server setup definition.

If you specify `OFF` for compatibility with the business process status and `ON` for compatibility with the invoke service activity status, the transaction terminates when processing of the invoke service activity starts or finishes. If an exception occurs during execution of the invoke service activity of the business process, `Error` is set in the status of the process instance and the invoke service activity.

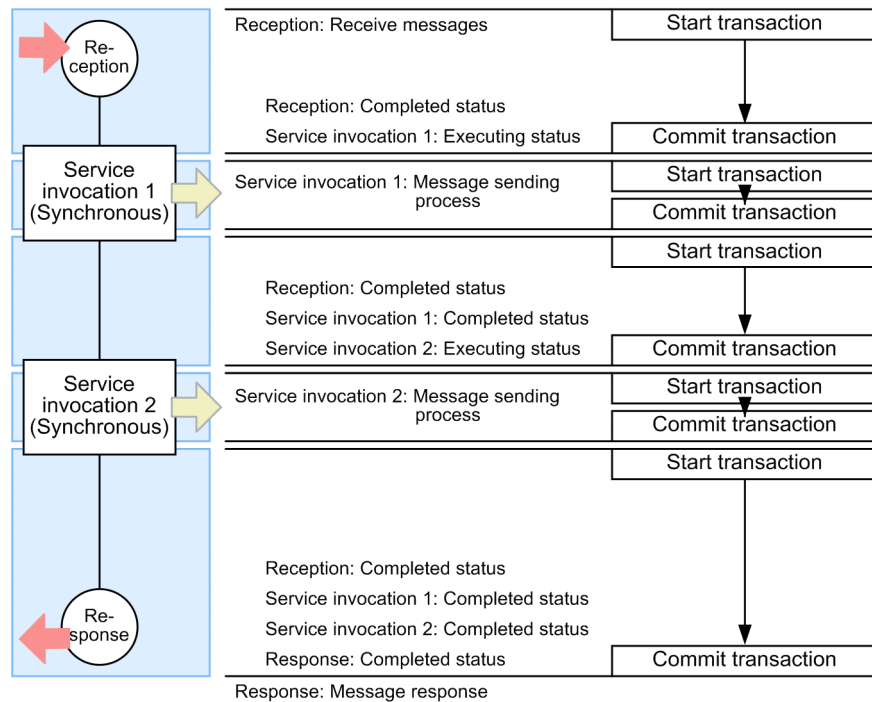
(1) If two synchronous invoke service activities are arranged



If processing continues successfully

The transaction is committed immediately before the message send process of each invoke service activity, and a new transaction starts. The transaction is also committed after the message send process finishes, and then a new transaction starts.

The following figure shows the transaction scope when processing continues successfully during the synchronous service invocation.

Figure 3–26: Transaction scope when processing continues successfully (synchronous service invocation)



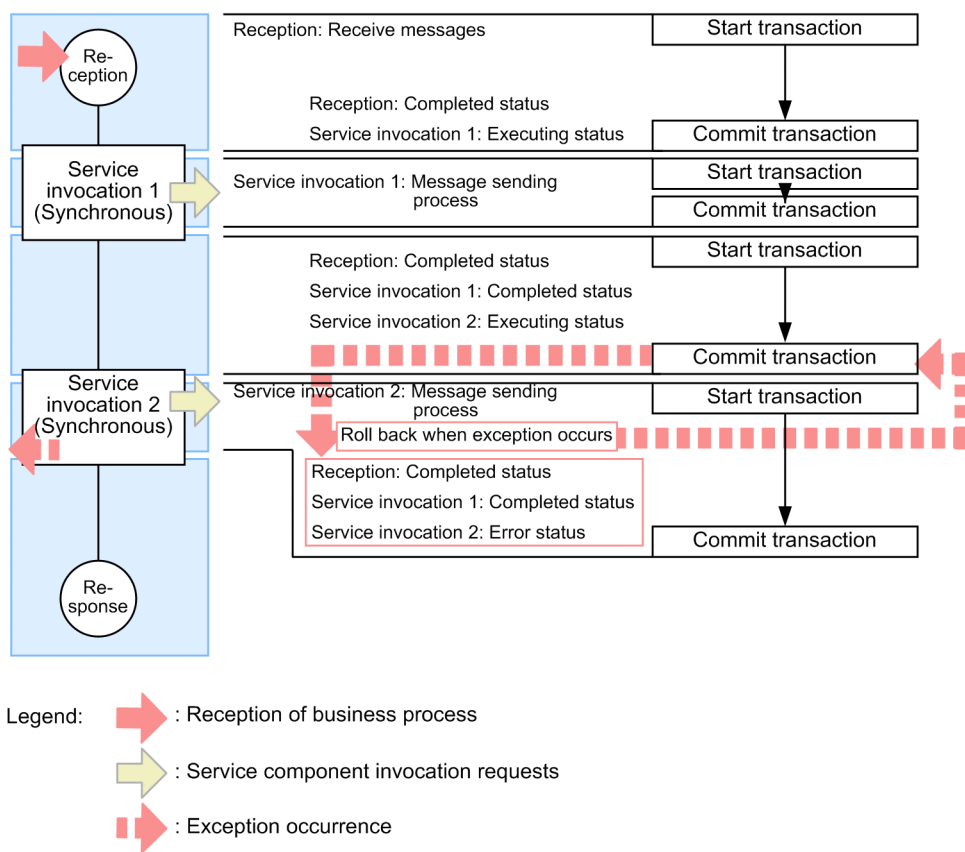
Legend:  : Business process reception and response
 : Service component invocation requests

If an exception occurs

If processing of service invocation 2 results in an error because of an exception, a rollback occurs up to the point where the transaction was committed (Reception: Completed, Service invocation 1: Completed, Service invocation 2: Executing). Then, `Error` is set in the status of the process instance and invoke service activity, and the transaction is committed.

The following figure shows the transaction scope if an exception occurs during the synchronous service invocation.

Figure 3–27: Transaction scope if an exception occurs (synchronous service invocation)



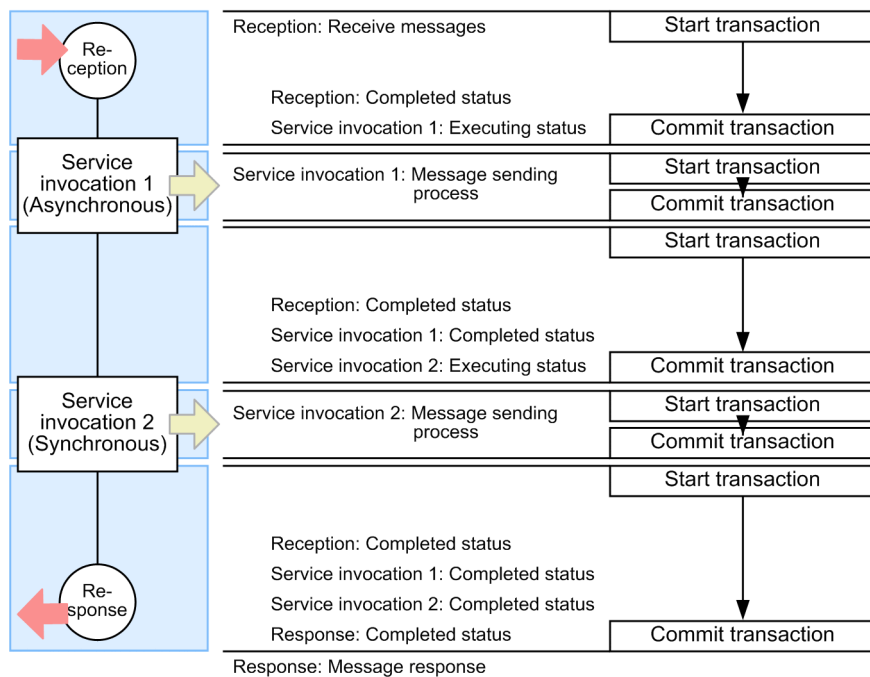
(2) If asynchronous and synchronous invoke service activities are arranged


If processing continues successfully


The transaction is committed immediately before the message send process of each invoke service activity, and a new transaction starts. The transaction is also committed after the message send process finishes, and then a new transaction starts.

The following figure shows the transaction scope when processing continues successfully during the asynchronous and synchronous service invocation.

Figure 3–28: Transaction scope when processing continues successfully (asynchronous and synchronous service invocation)



Legend:  : Business process reception and response

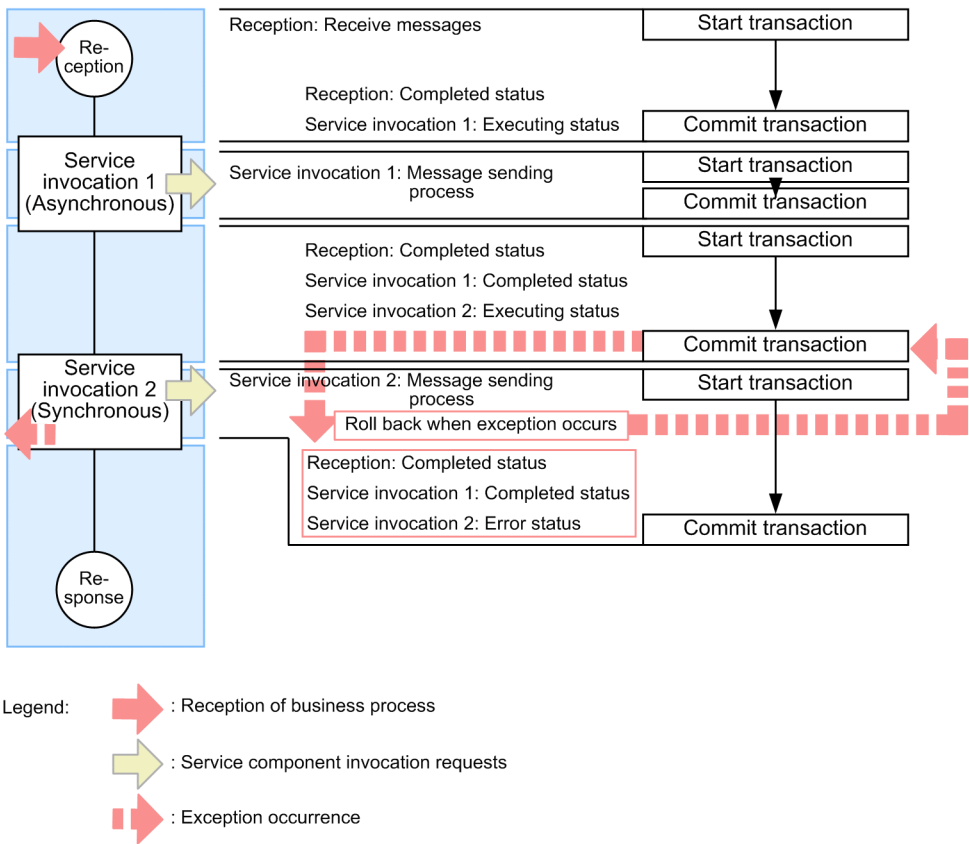
 : Service component invocation requests

If an exception occurs (asynchronous and synchronous service invocation)

If processing of service invocation 2 results in an error because of an exception, a rollback occurs up to the point where the transaction was committed (Reception: Completed, Service invocation 1: Completed, Service invocation 2: Executing). Then, `Error` is set in the status of the process instance and invoke service activity, and the transaction is committed.

The following figure shows the transaction scope if an exception occurs during asynchronous and synchronous service invocation.

Figure 3–29: Transaction scope if an exception occurs (asynchronous and synchronous service invocation)



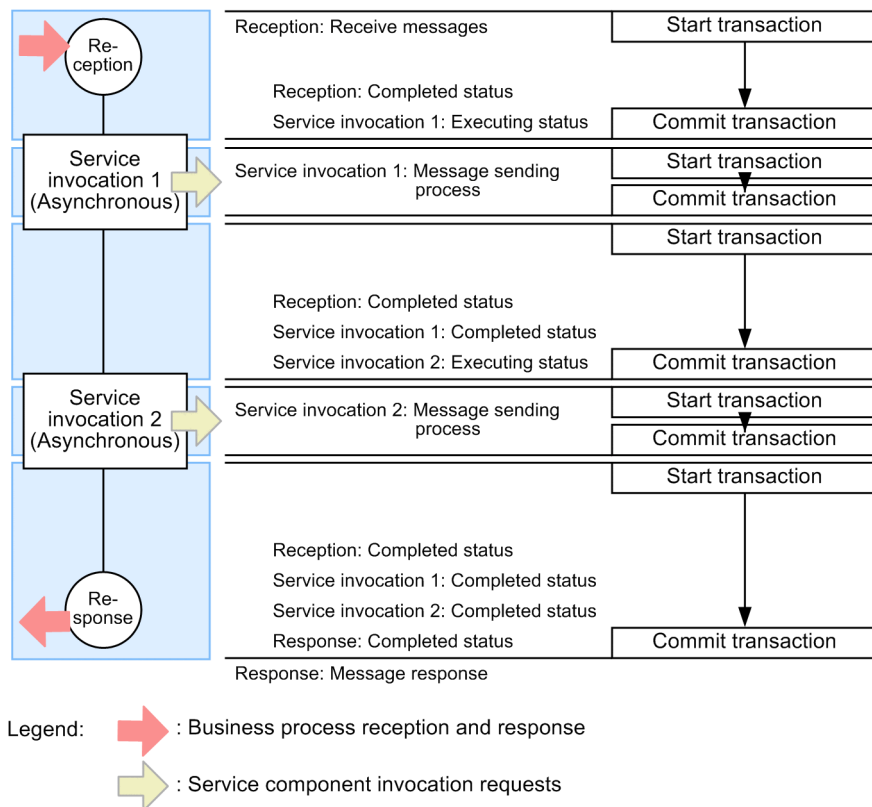
(3) If two asynchronous invoke service activities are arranged

If processing continues successfully

The transaction is committed immediately before the message send process of each invoke service activity, and a new transaction starts. The transaction is also committed after the message send process finishes, and then a new transaction starts.

The following figure shows the transaction scope when processing continues successfully during asynchronous service invocation.

Figure 3–30: Transaction scope when processing continues successfully (asynchronous service invocation)

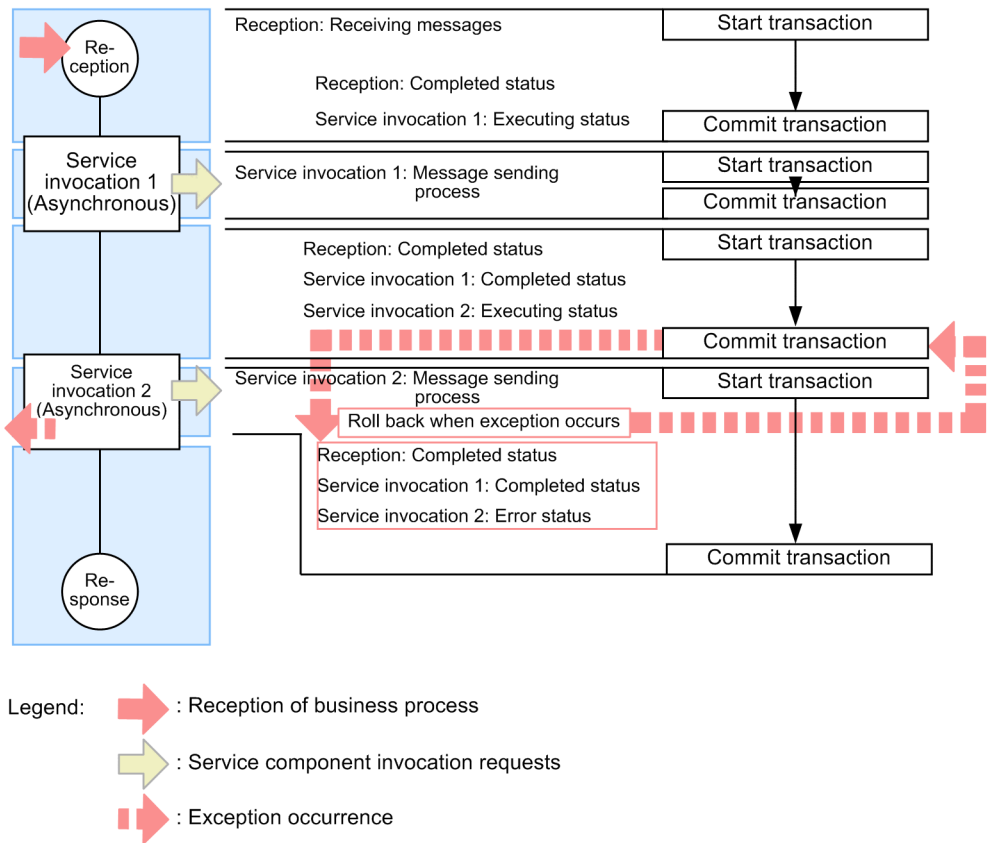


If an exception occurs

If processing of service invocation 2 results in an error because of an exception, a rollback occurs up to the point where the transaction was committed (Reception: Completed, Service invocation 1: Completed, Service invocation 2: Executing). Then, `Error` is set in the status of the process instance and invoke service activity, and the transaction is committed.

The following figure shows the transaction scope if an exception occurs during the asynchronous service invocation.

Figure 3–31: Transaction scope if an exception occurs (asynchronous service invocation)



(4) If a data transformation activity exists after a synchronous invoke service activity

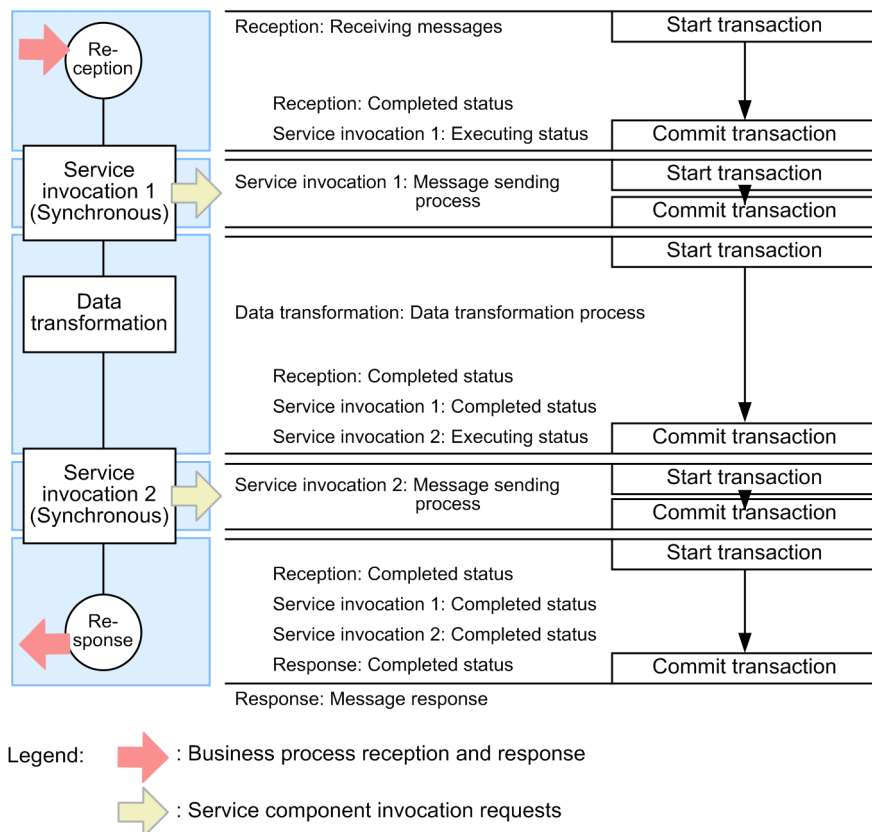
If processing continues successfully

The transaction is committed immediately before the message send process of each invoke service activity, and a new transaction starts. The transaction is also committed after the message send process finishes, and then a new transaction starts.

Note that the transaction is not committed in the data transformation activity. This applies even for an activity other than those shown in *Table 3-7 Timings for the start and commit of transactions*.

The following figure shows the transaction scope when processing continues normally in the presence of an activity such as data transformation.

Figure 3–32: Transaction scope when processing continues successfully (when activities such as data transformation exist)

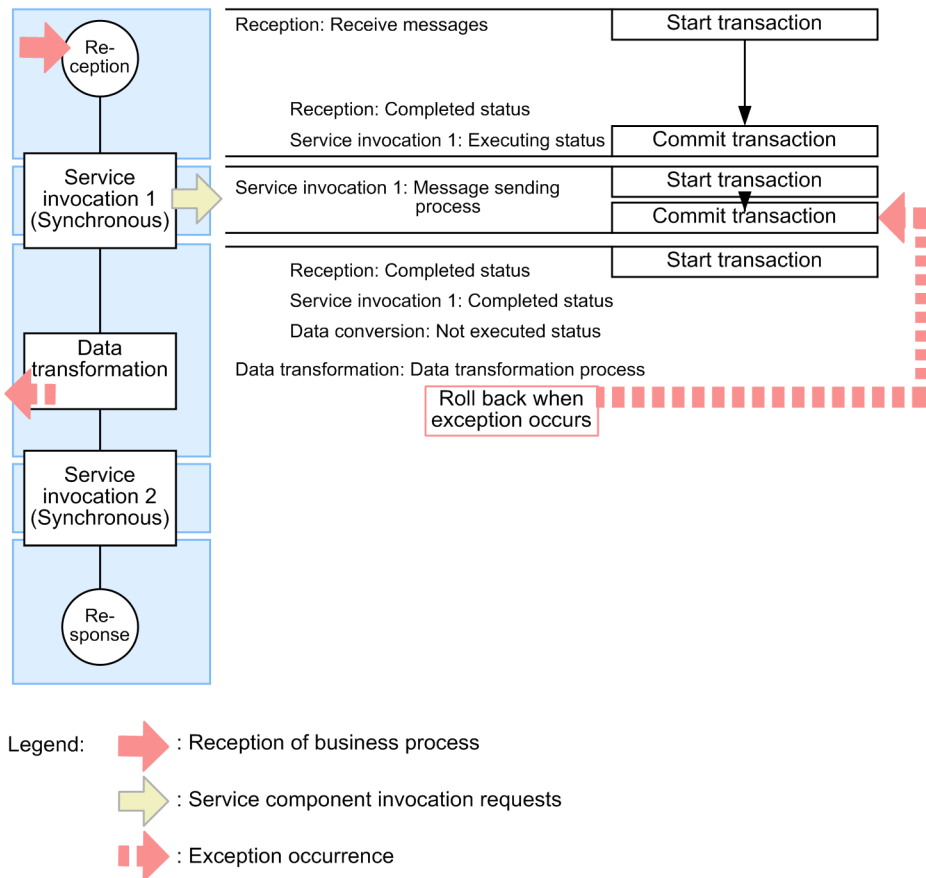


If an exception occurs (when activities such as data transformation exist)

If an error occurs in the data transformation process after service invocation 1, a rollback occurs up to the point where the transaction was committed (Reception: Completed, Service invocation 1: Completed, Data transformation: Not executed).

The following figure shows the transaction scope if an exception occurs in the presence of activities, such as data transformation.

Figure 3–33: Transaction scope if an exception occurs (when activities such as data transformation exist)



3.4.3 Transactions when OFF is specified in both compatibility with the business process status and compatibility with the invoke service activity status

This subsection describes transactions if you specify `OFF` for both compatibility with the business process status (`bp-status-compatible`) and compatibility with the invoke service activity status (`bp-invoke-status-compatible`) in the HCSC server setup definition.

The following describes the operation if you specify `OFF` for compatibility with the business process status and compatibility with the invoke service activity status.

- When processing of the invoke service activity starts, `Executing` is set in the status of the activity, and then the transaction terminates.
- When processing of the invoke service activity finishes, `Completed` is set in the status of the activity, and the transaction does not terminate.
- If an exception occurs during execution of the invoke service activity of the business process, `Error` is set in the status of the process instance and the invoke service activity, and then the transaction terminates.

Note that the operation is the same for synchronization and asynchronous communication models.

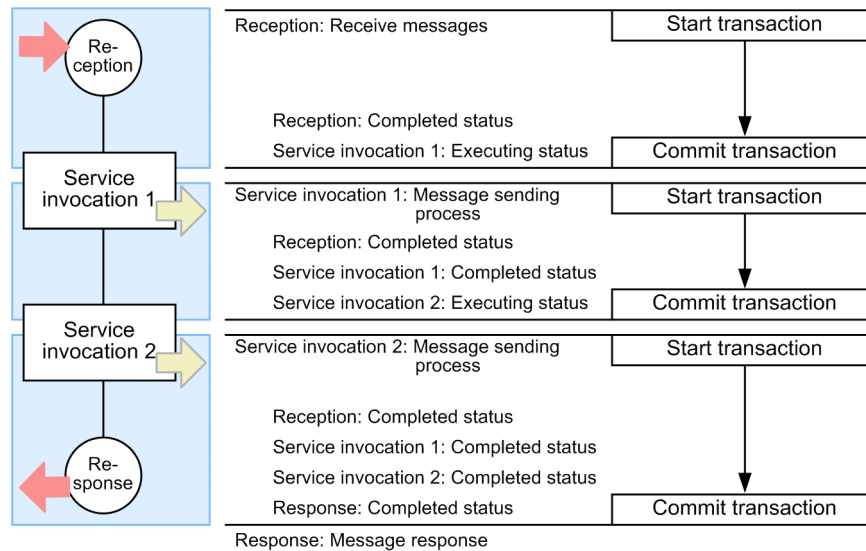
(1) If two invoke service activities are arranged



If processing continues successfully

The transaction is committed immediately before the message send process of each invoke service activity, and a new transaction starts.

The following figure shows the transaction scope when processing continues successfully during the service invocation.

Figure 3–34: Transaction scope when processing continues successfully



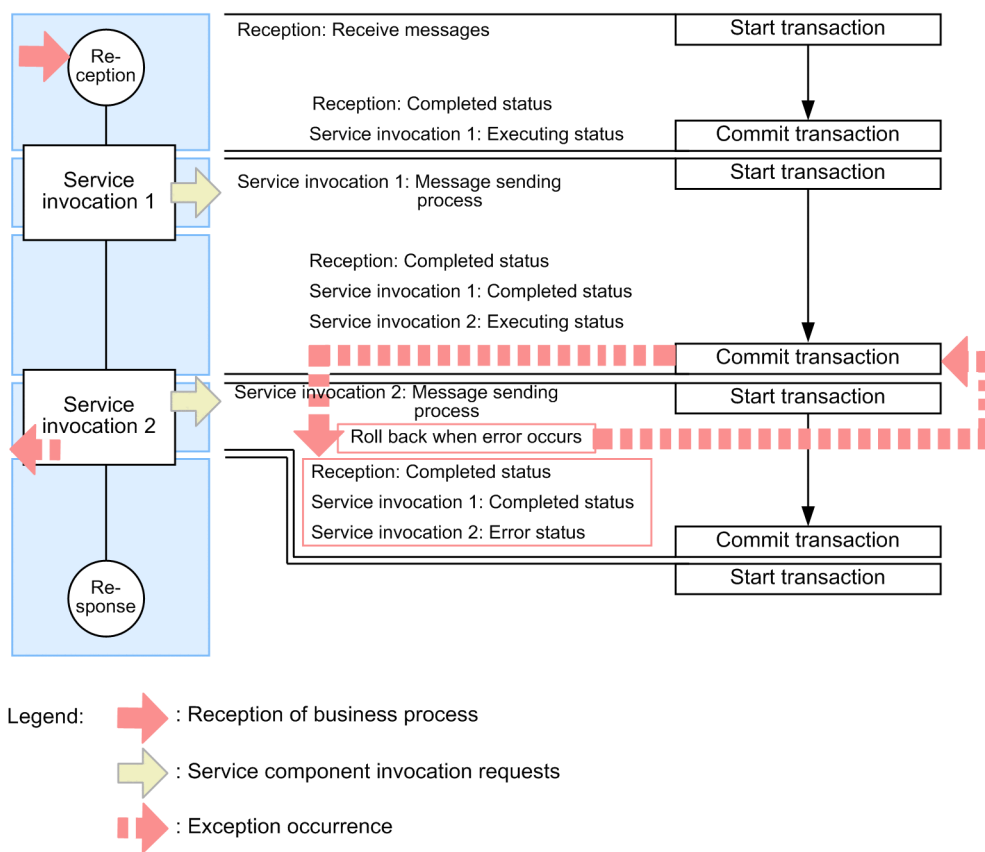
Legend:  : Business process reception and response
 : Service component invocation requests

If an exception occurs

If processing of service invocation 2 results in an error because of an exception, a rollback occurs up to the point where the transaction was committed (Reception: Completed, Service invocation 1: Completed, Service invocation 2: Executing). Then, `Error` is set in the status of the process instance and invoke service activity, and the transaction is committed.

The following figure shows the transaction scope if an exception occurs during the service invocation.

Figure 3–35: Transaction scope if an exception occurs



(2) If a data transformation activity exists after an invoke service activity

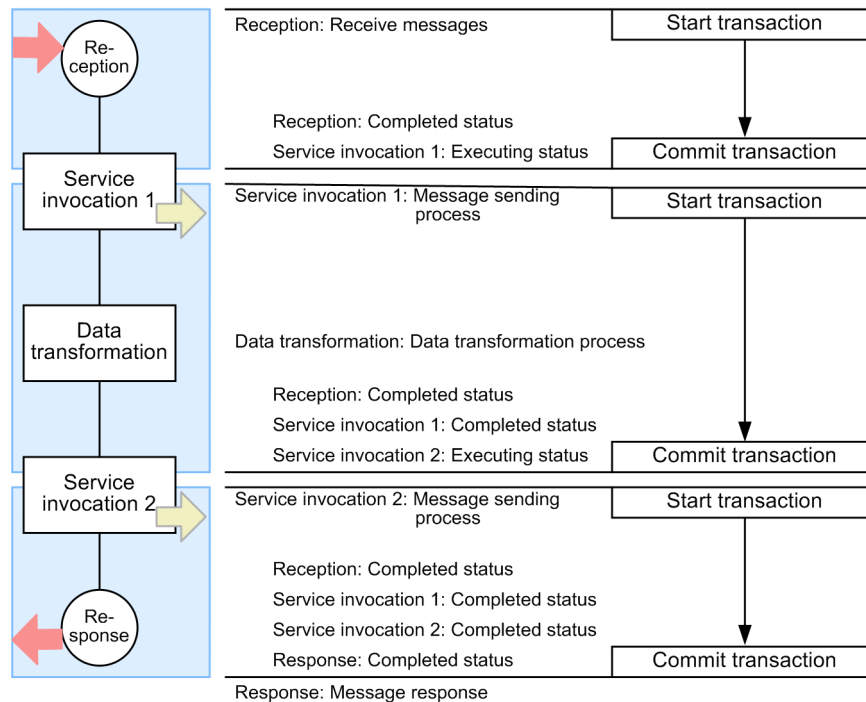
If processing continues successfully



The transaction is committed immediately before the message send process of each invoke service activity, and a new transaction starts.

Note that the transaction is not committed in the data transformation activity. This applies even for an activity other than those shown in *Table 3-7 Timings for the start and commit of transactions*.

The following figure shows the transaction scope when processing continues normally in the presence of an activity such as data transformation.

Figure 3–36: Transaction scope when processing continues successfully (when activities such as data transformation exist after service invocation)



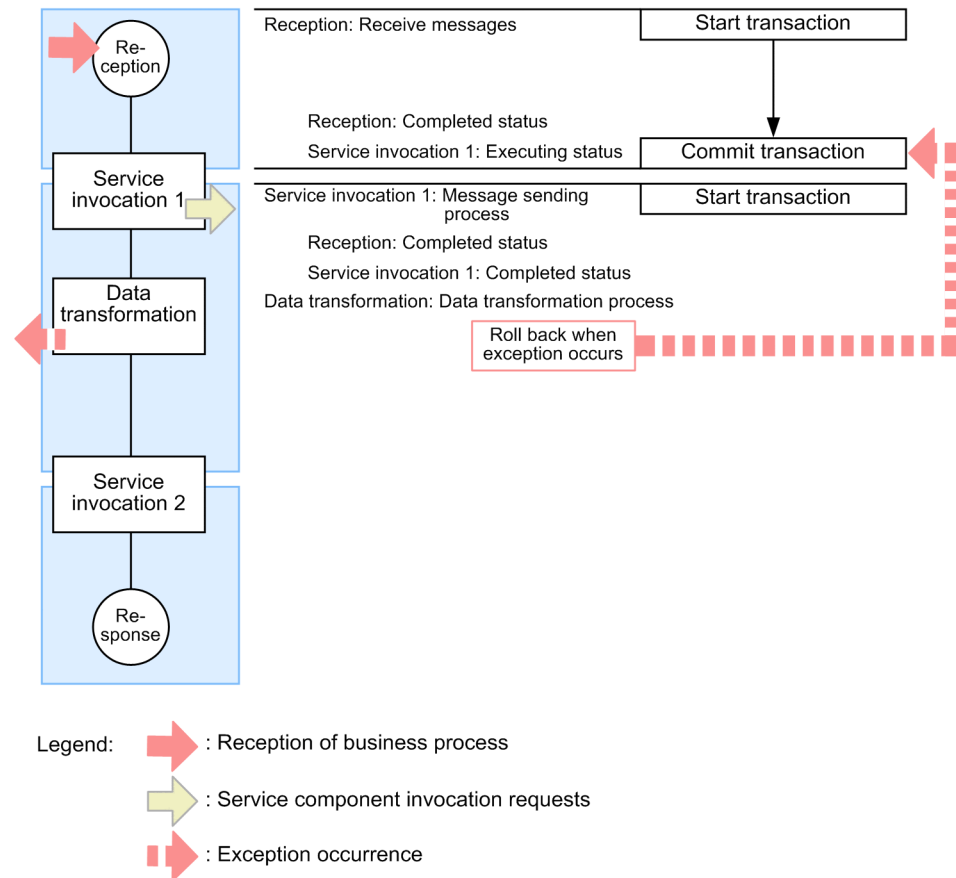
Legend:  : Business process reception and response
 : Service component invocation requests

If an exception occurs (when activities such as data transformation exist after service invocation)

If an error occurs in the data transformation process after service invocation 1, a rollback occurs up to the point where the transaction was committed (Reception: Completed, Service invocation 1: Executing).

The following figure shows the transaction scope if an exception occurs in the presence of an activity, such as data transformation, after service invocation.

Figure 3–37: Transaction scope if an exception occurs (when activities such as data transformation exist after service invocation)



3.4.4 Transactionsof the invoke service activity

The transactions for each invoke service activity and status transition is as follows:

- (1) When ON is specified for business process status compatibility (bp-status-compatible in the HCSC server setup definition file)

The following table shows transactions of the invoke service activity and status transition when ON is specified for business process status compatibility. Status transition is same irrespective of the settings of invoke service activity status compatibility (bp-invoke-status-compatible in the HCSC server setup definition file).

Table 3–8: Properties of transaction and status transition when ON is specified for business process status compatibility (bp-status-compatible in HCSC server setup definition file)

Invoke service activity operation	Specify ON for invoke service activity status compatibility		Specify OFF for invoke service activity status compatibility (Default)	
	Asynchronous	Synchronous	Asynchronous	Synchronous
Sending request message	N	Y(Executing) [#]	N	Y(Executing) [#]
Receiving response message (successfully completed)	N	N	N	N
Receiving response message (occurrence of a system exception)	N	N	N	N

Legend:

Y: Controls transaction.

N: Does not control transaction.

Note#

Committed status is mentioned within ().

For details on the HCSC server setup definition file, see "HCSC server setup definition file" in "Service Platform Reference Guide".

(2) When OFF is specified for business process status compatibility (bp-status-compatible in the HCSC server setup definition file)

The following table shows transactions of the invoke service activity and status transition when OFF is specified for business process status compatibility. Status transition differs depending on the settings of invoke service activity status compatibility (bp-invoke-status-compatible in the HCSC server setup definition file).

Table 3–9: Properties for transaction and status transition when OFF is specified for business process status compatibility (bp-status-compatible in the HCSC server setup definition file)

Invoke service activity operation	Specify ON for invoke service activity status compatibility		Specify OFF for service invoke service activity status compatibility (Default)	
	Asynchronous	Synchronous	Asynchronous	Synchronous
Sending request message	Y(Executing)#	Y(Executing)#	Y(Executing)#	Y(Executing)#
Receiving response message (successful completion)	Y(Completed)#	Y(Completed)#	N	N
Receiving response message (system exception generation)	Y(Error status)#	Y((Error status)#	Y(Error status)#	Y(Error status)#

Legend:

Y: Controls transaction.

N: Does not control transaction.

Note#

Committed status is mentioned within ().

If an error occurs in an activity after successfully executing the invoke service activity, message sending to the asynchronous service queue of the invoke service activity and updating the process of the database that invokes DB adapter is enabled.

For details on the HCSC server setup definition file, see "HCSC server setup definition file" in "Service Platform Reference Guide".

3.4.5 Transactions when you select the option to commit at start time and end time of scope

The scope activity controls transactions at the start and end time of scope, if you have selected "Commit only at the time of starting and ending this scope" in the [Scope activity] dialog.

This sub-section describes the transaction of scope activity that controls transaction:

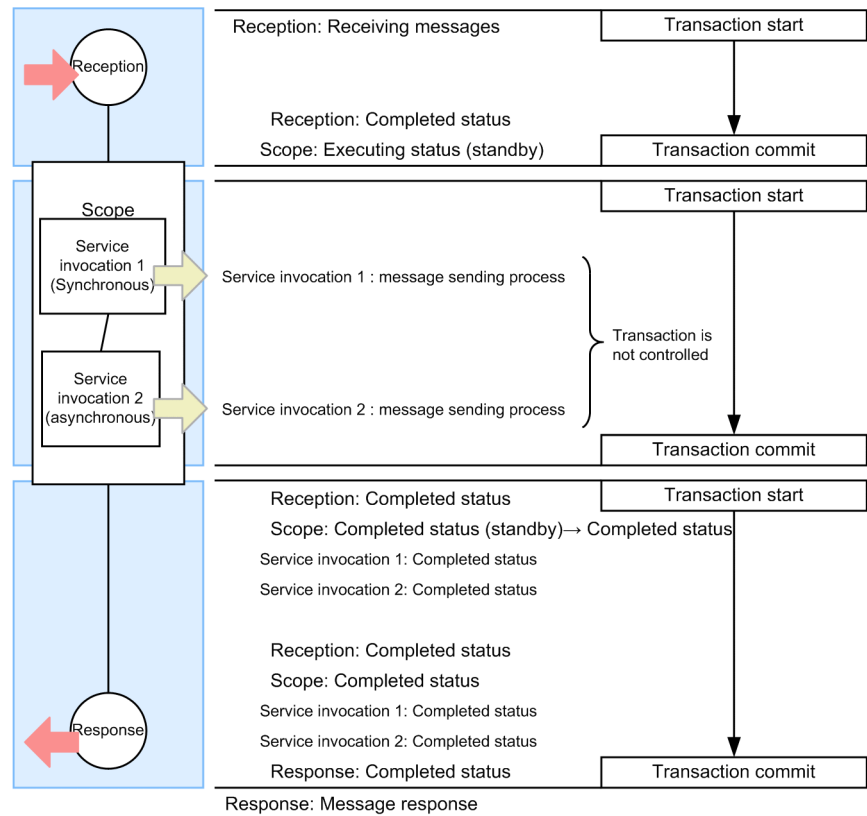
(1) When scope activity is completed successfully



In the start process of the scope activity, the scope activity status is changed to (Executingwait), the transaction is committed and, a new transaction is started. After that, the activity defined in the scope activity is executed. At this time, the transaction is not controlled by the activity defined in the scope activity.

In the end process of the scope activity, the scope activity status is changed to (Completedwait), transaction is committed and, a new transaction is started. After that, the status of the scope activity is changed to (Completed). At this time, there is status transition but the transaction is not committed.

The following figure shows the scope of transaction when the scope activity is completed successfully:

Figure 3–38: Scope of transaction when scope activity is completed successfully



Legend:  : Business process reception and response
 : Service component invocation request

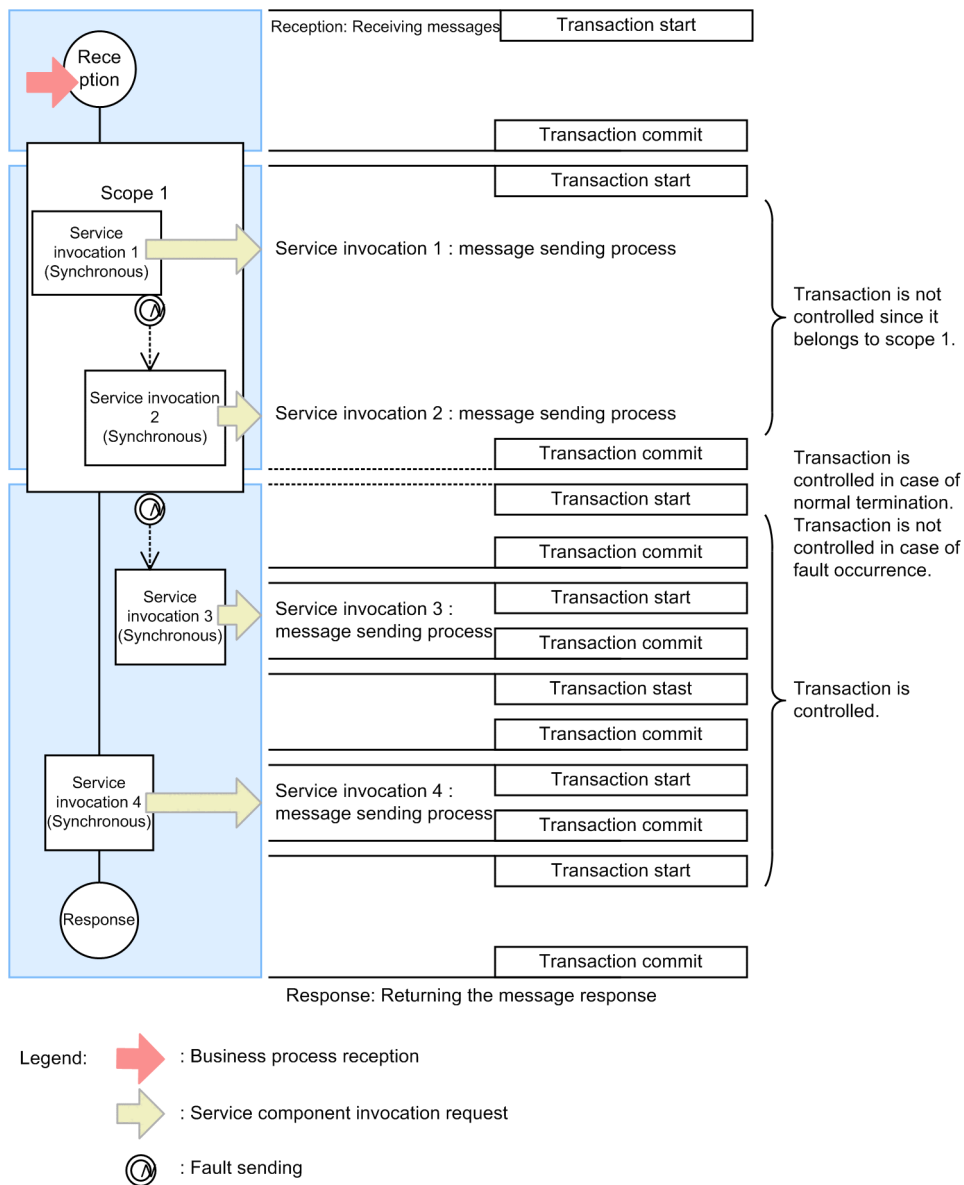
Reference note

Describes the scope of transaction control of the scope activity that controls transaction.

- Transaction is not controlled in the activity defined in the scope activity that controls transactions.
- Transaction is not controlled in the activity that processes faults of the scope activity that controls transactions.
- Transaction is not controlled in the activity that is defined after the scope activity that controls transactions.

The following figure shows the scope of transaction control of the scope activity that controls transactions:

Figure 3–39: Scope of transaction control of the scope activity that controls transactions

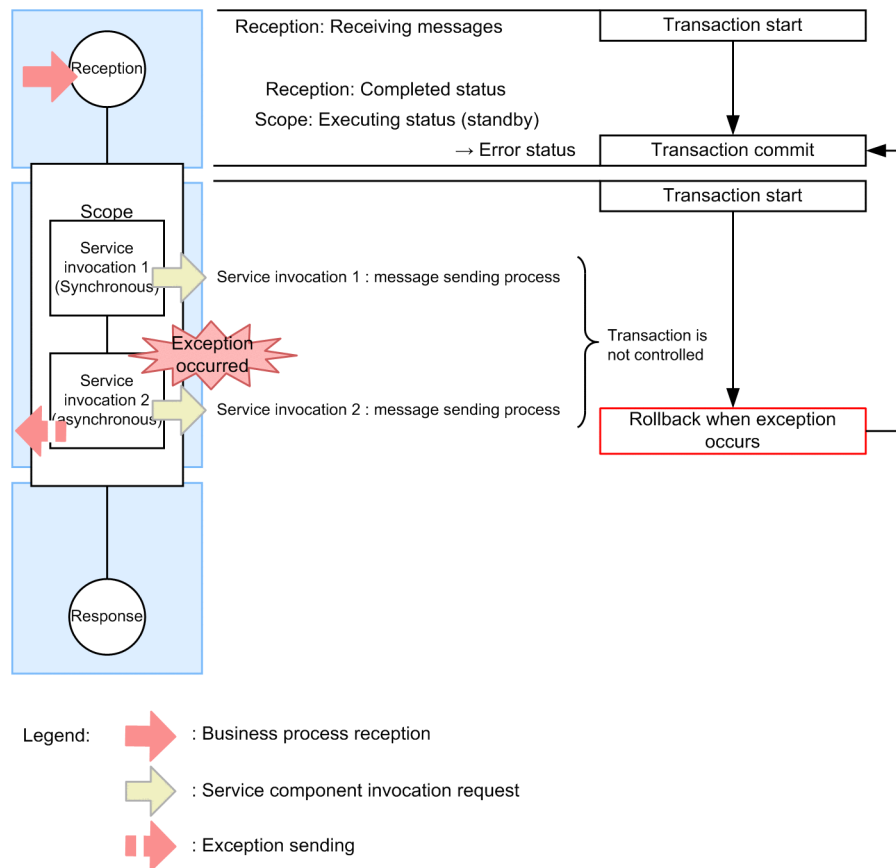


(2) When a system exception occurs in the scope activity

When a system exception occurs in the scope activity, the scope activity rolls back till the point where the transaction is committed (reception: rolls back till Completed, scope: (Executingwait)), and then sets error occurrence (Error) in the status of the scope, and commits transaction.

The following figure shows the scope of transaction when a system exception occurs in the scope activity:

Figure 3-40: Scope of the transaction when a system exception occurs in the scope activity

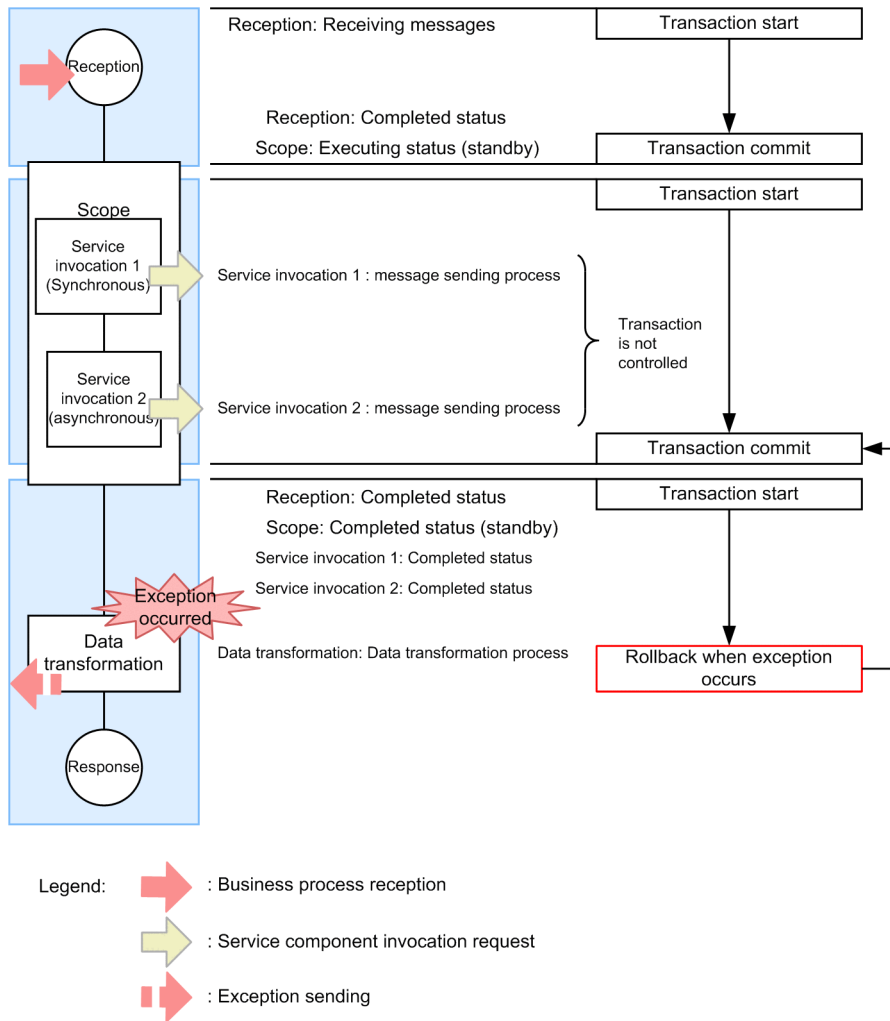


(3) When a system exception occurs in the activity after the scope activity

When a system exception occurs in the activity defined after the scope activity, the scope activity rolls back till transaction is committed (scope activity: (Completedwait)).

The following figure shows the scope of transactions when a system exception occurs in the activity defined after scope activity:

Figure 3–41: Scope of transactions when a system exception occurs in the activity defined after the scope activity



(4) When a fault occurs within the scope activity

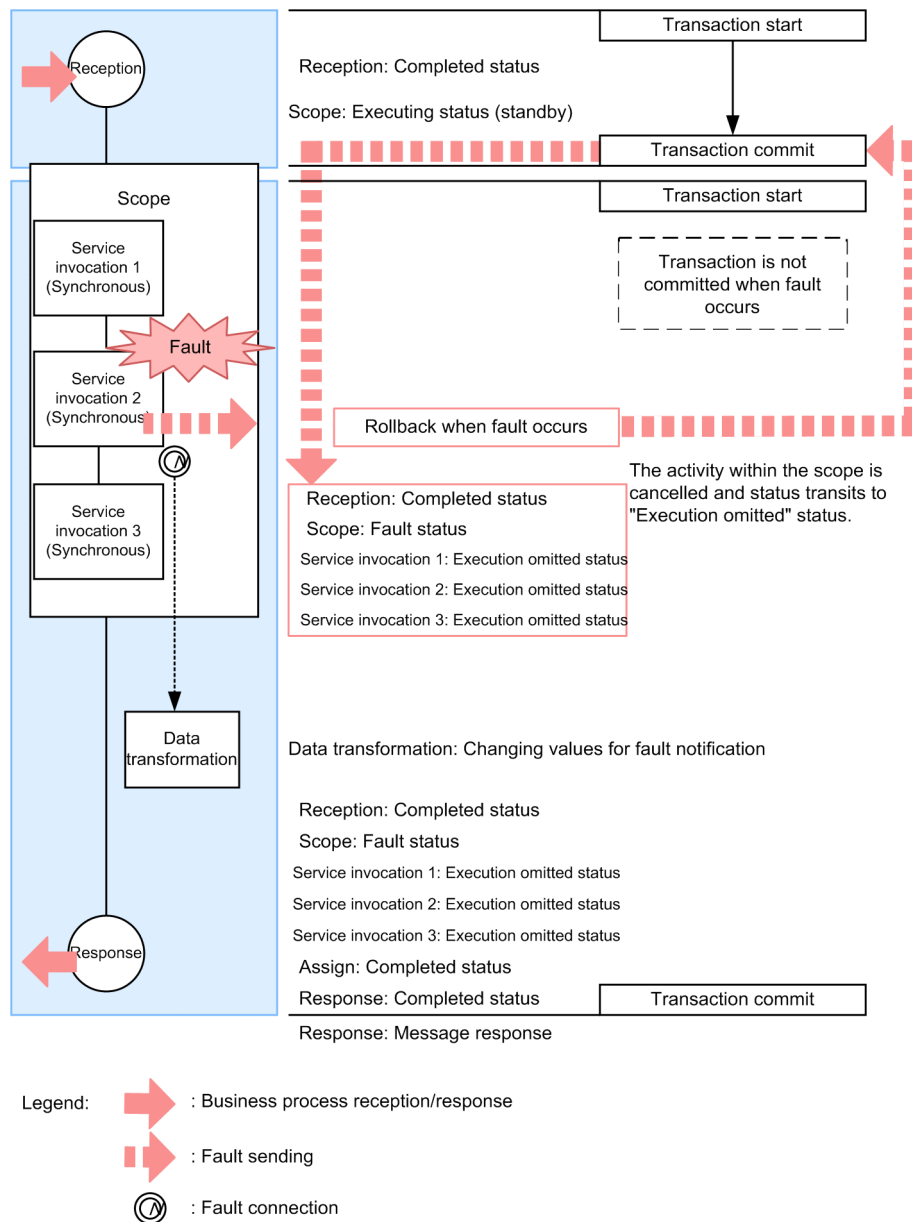
When a fault occurs in an activity that is defined in the scope activity that controls transactions and the fault is notified to the scope activity that controls transactions, the scope activity rolls back till the transaction is committed (receive: Completed, Scope: Executingwait).The process for activity defined in the scope activity is cancelled.

Based on the definition of the fault handler, if you execute the corresponding activity, the status of the scope activity becomes (Faulting). At this time, there is status transition, but the transaction is not committed.

At the time of notifying fault to the parent scope activity, the status of scope activity becomes (Faulted). Again, only status transition is performed, but the transaction is not committed. After notifying a fault to the parent scope activity, the operation is same as that of a scope activity that does not control transactions.

The following figure shows the scope of a transaction when a fault occurs in the scope activity:

Figure 3-42: Scope of transaction when a fault occurs in the scope activity

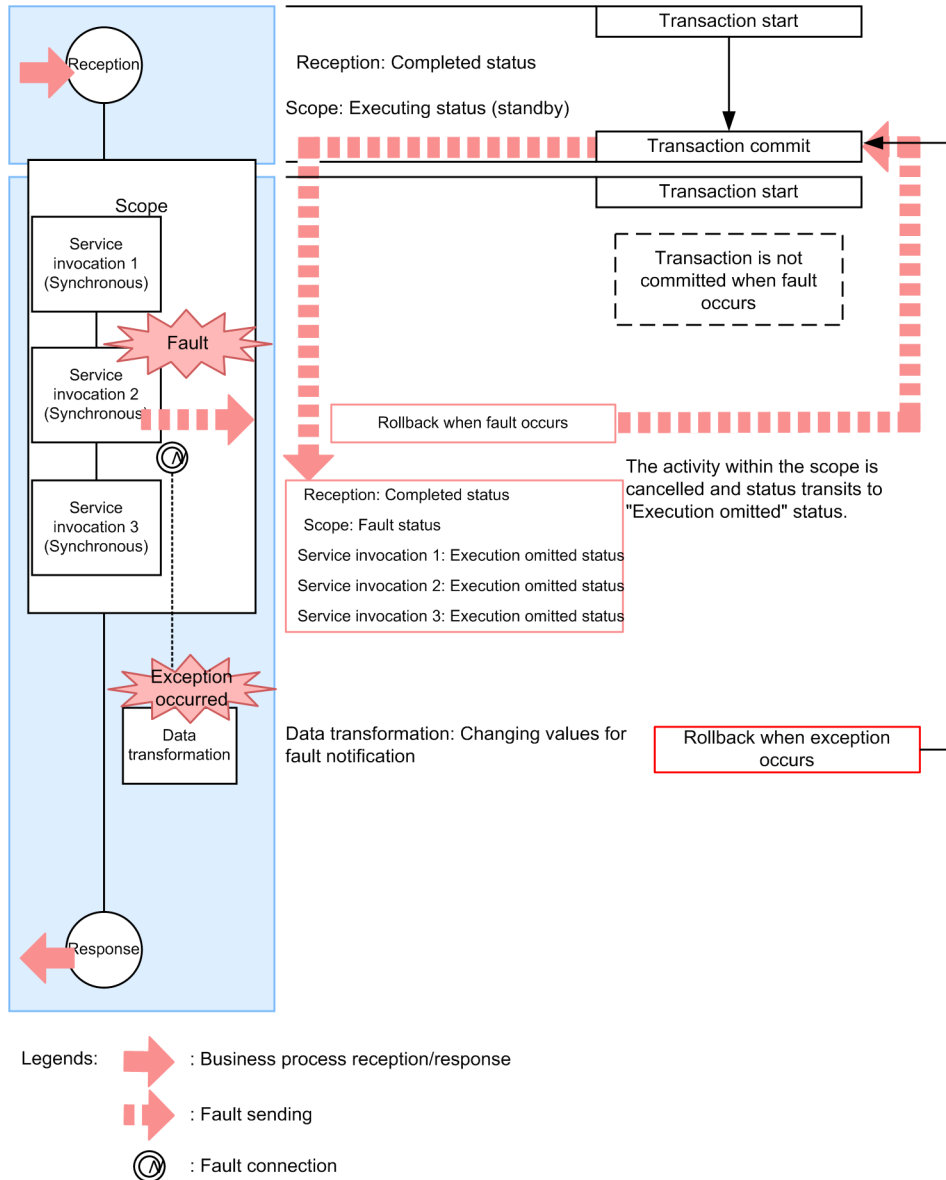


(5) When a system exception occurs after fault occurrence

Based on the definition of the fault handler, if you execute corresponding activity and if a system exception occurs, the scope activity rolls back till the executing status (Executingwait).

The following figure shows the scope of a transaction when a system exception occurs after a fault occurrence:

Figure 3–43: Scope of transaction when a system exception occurs after a fault occurrence



3.4.6 Transactions of the scope activity

The following table shows transactions of the scope activity and status transition:

Table 3–10: Transactions of the scope activity and status transition

Scope activity operation	Does not control transaction		Controls transaction	
	Status to transition to	Transaction control	Status to transition to	Transaction control
Start	Executing	N	(Executingwait)	Y
End	Completed	N	(Completedwait)	Y
			Completed	N
System exception occurrence	-	N	Error	Y

Scope activity operation	Does not control transaction		Controls transaction	
	Status to transition to	Transaction control	Status to transition to	Transaction control
Fault occurrence (processing connection)	Faulting	N	Faulting	N
Fault occurrence (Connection process complete)	Faulted	N	Faulted	N
Notify fault to parent	Faulted	N	Faulted	N
Omission	Omission	N	Omission	N
Unomission	Unomission	N	Unomission	N

Legend:

Y: Commits transaction.

N: Does not commit transaction.

-: Does not perform status transition to rollback.

3.4.7 Transactions of the reply activity

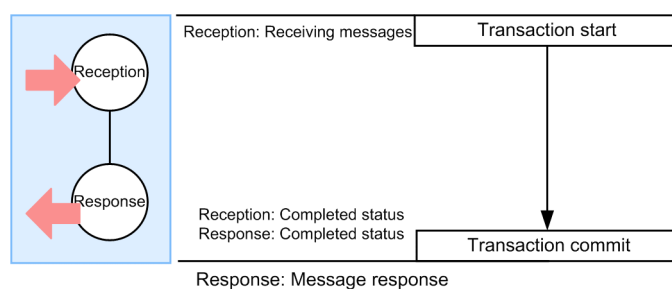
The transaction control of the reply activity is described here.


For details on how to asynchronously execute the activity defined after reply activity, see "3.7 Asynchronous execution of an activity that is defined after reply activity".

(1) Transaction control when an activity is not defined after the reply activity

When an activity is not defined after the reply activity, the status of the reply activity during the execution of the reply activity transitions to (Completed), and the transaction is committed just before responding with a message.

Figure 3–44: Transaction control when an activity is not defined after the reply activity



Legend:  : Business process reception/response

(2) Transaction control when an activity is defined after the reply activity

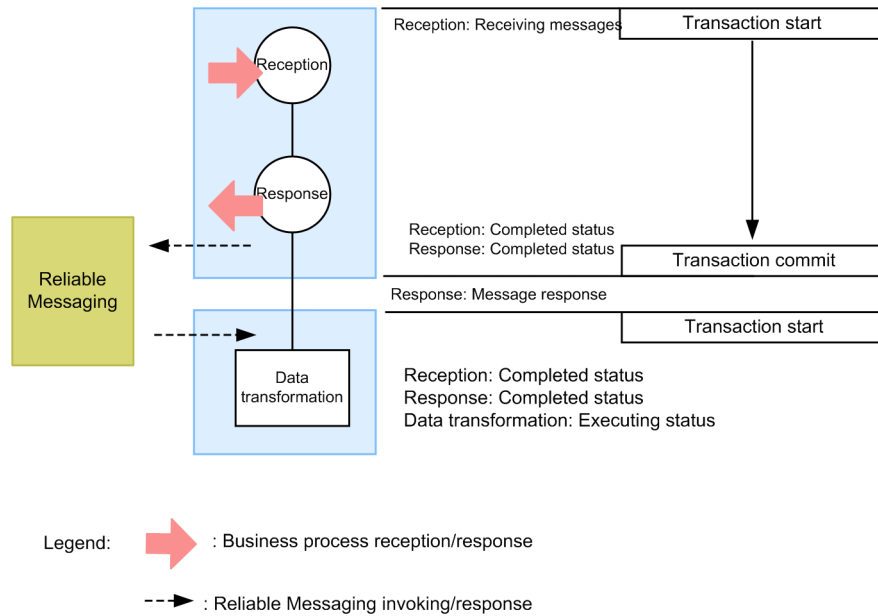
(a) When you use Reliable Messaging for executing the activity after the reply

When an activity is defined after the reply activity and you use Reliable Messaging for executing the activity after reply, transaction control is as follows:

1. When an activity is defined after the reply activity and you use Reliable Messaging for executing the activity after reply, the status of the reply activity during the execution of the reply activity transitions to (Completed).

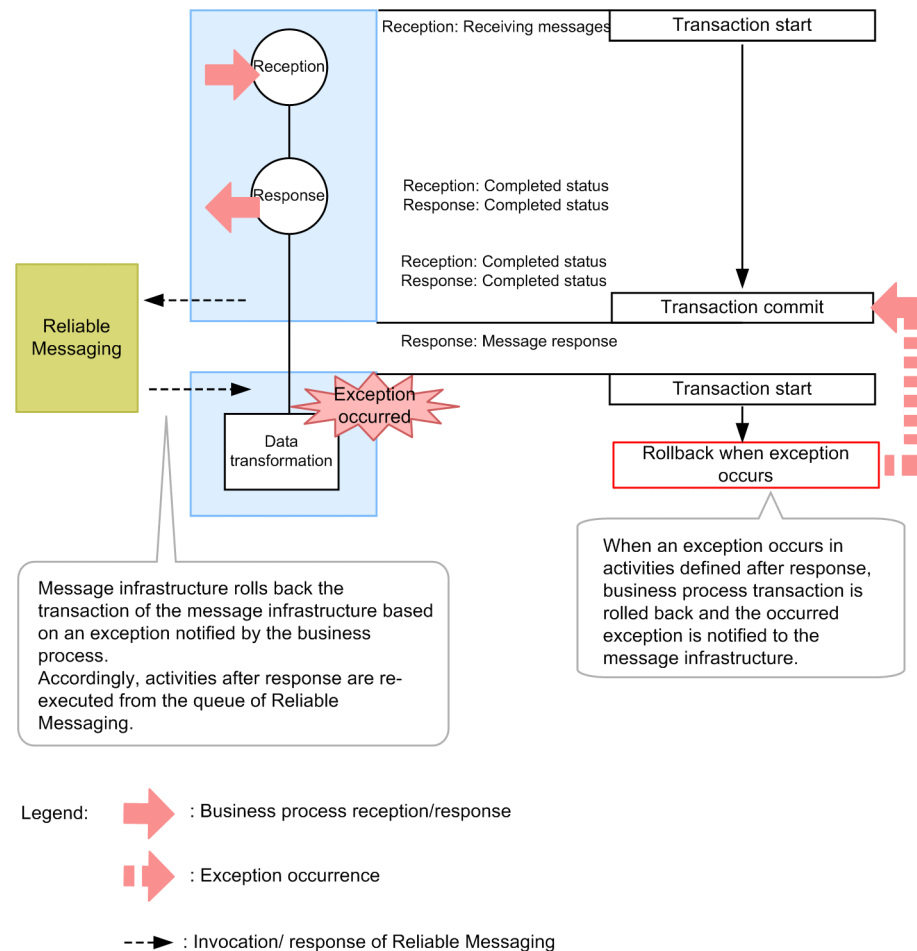
2. The request to process asynchronous execution of the activity defined after the reply activity is executed for Reliable Messaging, and the transaction is committed just before response message.
3. The activity defined after the reply activity is executed asynchronously from Reliable Messaging.
4. Transaction is started before executing the activity defined after the reply activity.

Figure 3–45: Transaction control when an activity is defined after the reply activity and Reliable Messaging is used for execution of the activity after reply



- If an exception occurs in the activity defined after the reply activity
When Reliable Messaging is used for executing an activity after reply and, if a system exception occurs in the activity defined after reply activity, transaction control is as follows:
 1. Rollback till the reply activity is (Completed).
 2. Depending on the rollback, the activity defined after the reply activity is executed again from Reliable Messaging.

Figure 3—46: Transaction control when Reliable Messaging is used to execute an activity after reply and there is an exception in the activity defined after reply activity



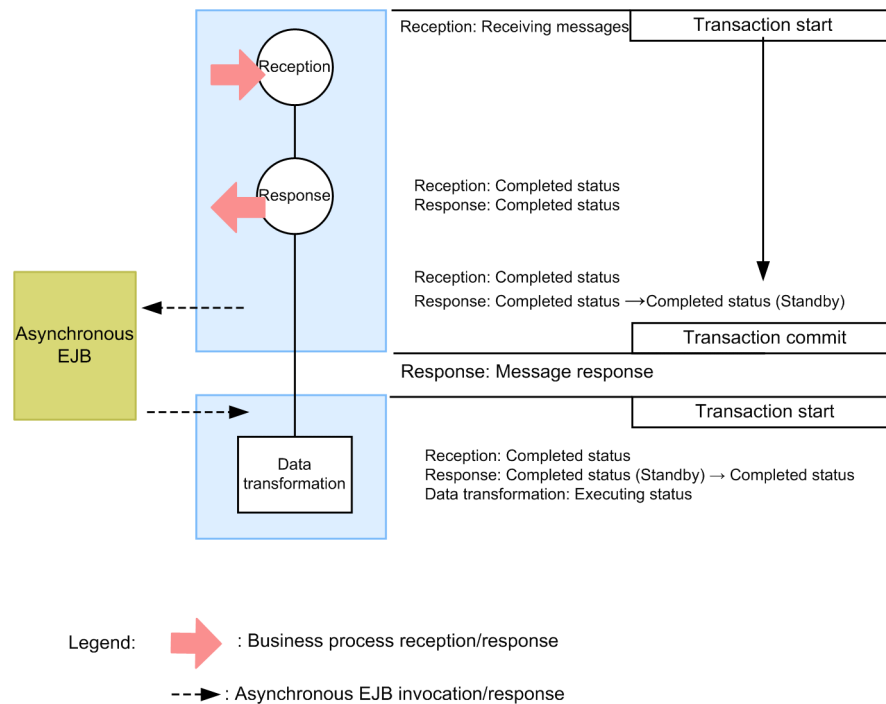
Whether a message can be resent from Reliable Messaging, depends on the value specified for maximum number of deliveries (RMMaxDeliveryNum). For details, see "Reliable Messaging".

(b) If asynchronous EJB is used for executing the activity after reply

When an activity is defined after the reply activity and you use asynchronous EJB to execute the activity after reply, transaction control is as follows:

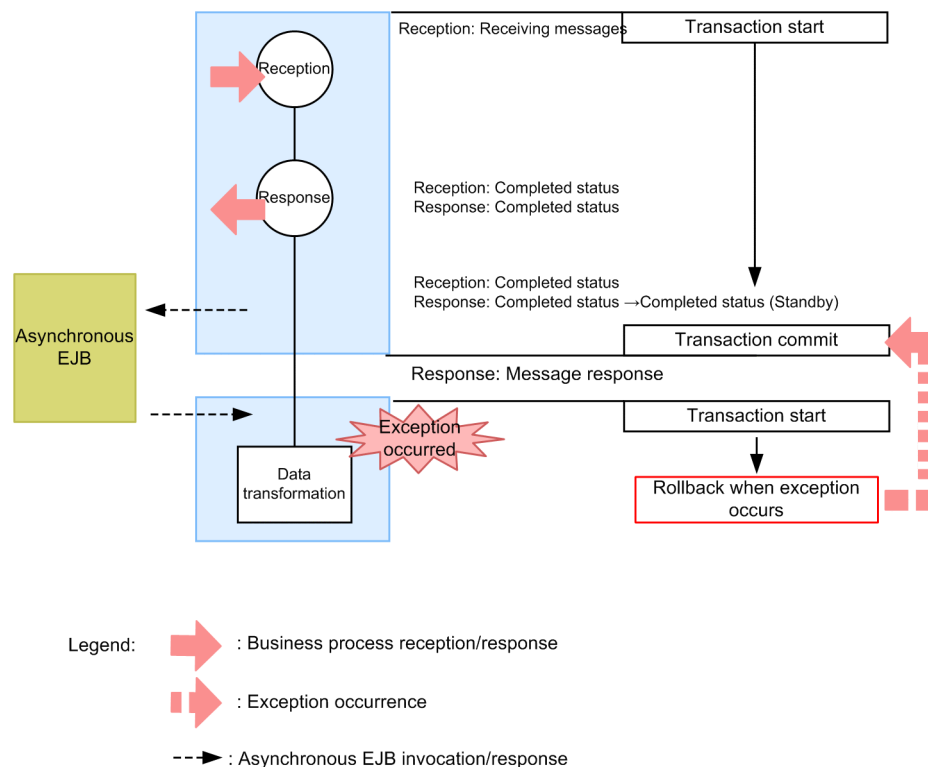
1. At the time of executing the reply activity, the status of reply activity transits to (Completed).
2. The request to process the asynchronous execution of the activity defined after reply is executed for asynchronous EJB, and the status of reply activity transits to (Completedwait).
3. Transaction is committed just before response message.
4. The activity defined after reply is executed asynchronously from asynchronous EJB and the status of reply activity transits to (Completed).
5. Transaction is started before executing the activity defined after reply.

Figure 3–47: Transaction control when an activity is defined after reply activity and asynchronous EJB is used for executing the activity after reply



- If an exception occurs in the activity defined after the reply activity
When asynchronous EJB is used to execute an activity after reply and an exception occurs in the activity defined after the reply activity, the status of the reply activity rolls back till (Completedwait). The activity defined after the reply activity is not executed again.

Figure 3–48: Transaction control when an asynchronous EJB is used to execute an activity after reply, and an exception occurs in the activity defined after the reply activity

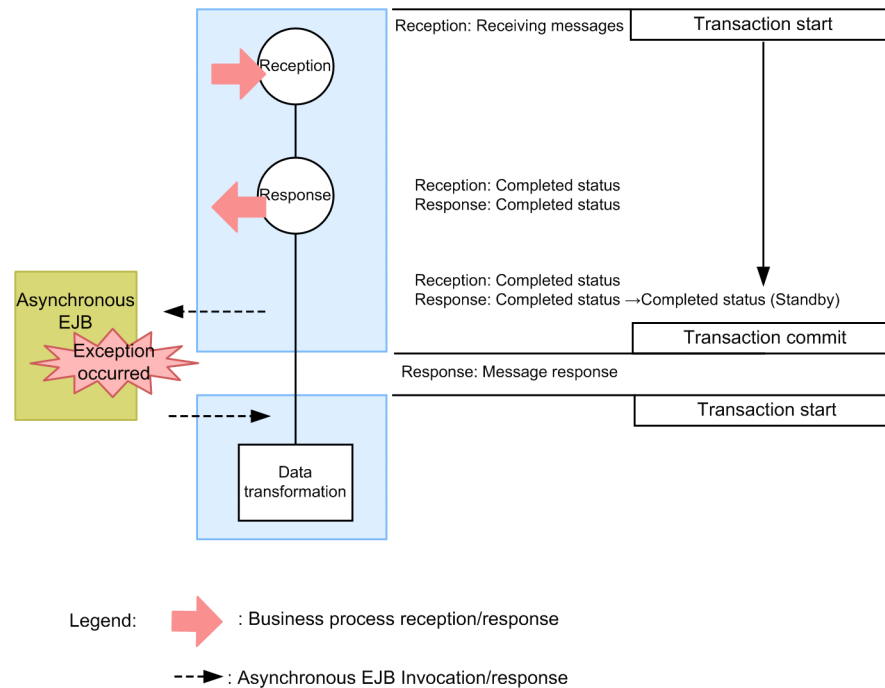


- Transaction control when an exception occurs at the time of executing asynchronous EJB

When asynchronous EJB is used for executing an activity after reply, and an exception occurs at the time of executing asynchronous EJB, the activity defined after the reply activity is not executed.

The status of the reply activity becomes (Completedwait).

Figure 3-49: Transaction control when asynchronous EJB is used to execute an activity after reply, and an exception occurs at the time of executing asynchronous EJB



(3) Transaction and status transition of the reply activity

The following lists the transaction and status transition of the reply activity.

Table 3-11: Transaction and status transition of the reply activity

Reply activity operation		When Reliable Messaging is used for executing the reply activity	When asynchronous EJB is used for executing the reply activity
Start	Transition to status	Completed	Completed
	Transaction control	N [#]	N [#]
Asynchronous execution request to Reliable Messaging	Transition to status	-	(Non-executable)
	Transaction control	N	
Asynchronous execution response from Reliable Messaging	Transition to status	-	(Non-executable)
	Transaction control	N	
Asynchronous execution request to asynchronous EJB	Transition to status	(Non-executable)	(Completedwait)
	Transaction control		N [#]
Asynchronous execution response from asynchronous EJB	Transition to status	(Non-executable)	Completed
	Transaction control		N [#]
Omission	Transition to status	Omission	Omission
	Transaction control	N	N

3. Business Process Functionality

Reply activity operation		When Reliable Messaging is used for executing the reply activity	When asynchronous EJB is used for executing the reply activity
Unomission	Transition to status	Unomission	Unomission
	Transaction control	N	N

Legend:

N: Transaction is not committed.

-: No transition.

Note#

Transaction is not committed at this point, but at the timing of a subsequent transaction control.

3.5 Re-executing a business process

During the execution of a business process, if the machine with an execution platform stops, or if invocation of service components fails, then a fault occurs and you can re-execute the interrupted process instance. Business process contents can be processed by re-executing the process instance.

The conditions for re-executing a process instance are as follows:

- When the HCSC server is started
- When a business process is started
- When synchronous (SessionBean) standard reception is started
- When asynchronous (MDB (WS-R)) standard reception is started in an environment that uses Reliable Messaging

Statuses to be re-executed are as follows:

Process status	Activity	
	Type	Status
Executing	Invoke service	Executing
Error	Invoke service	Error
Executing	Standby	Executing
Executing	Scope	(Executingwait)
Error	Scope	Error
Executing	Scope	(Completedwait)
Executing	Response	(Completedwait)

3.5.1 Re-executing business processes from windows in an operating environment

You can re-execute the canceled process instances one by one from the windows in the operating environment.

For details about how to re-execute a process instance from a window, see *6.1.5 Re-executing process instances* in the *Service Platform Setup and Operation Guide*.

3.5.2 Re-executing business processes with commands

You can re-execute the canceled process instances one by one by using commands.

You can also re-execute multiple process instances in a batch. You can re-execute multiple process instances on an HCSC server in a batch, or specify HCSC server names to re-execute process instances for each HCSC server. To perform batch re-execution of the process instance execution logs, you must use commands only.

For details about how to re-execute process instances by using commands, see *6.1.5 Re-executing process instances* in the *Service Platform Setup and Operation Guide*.

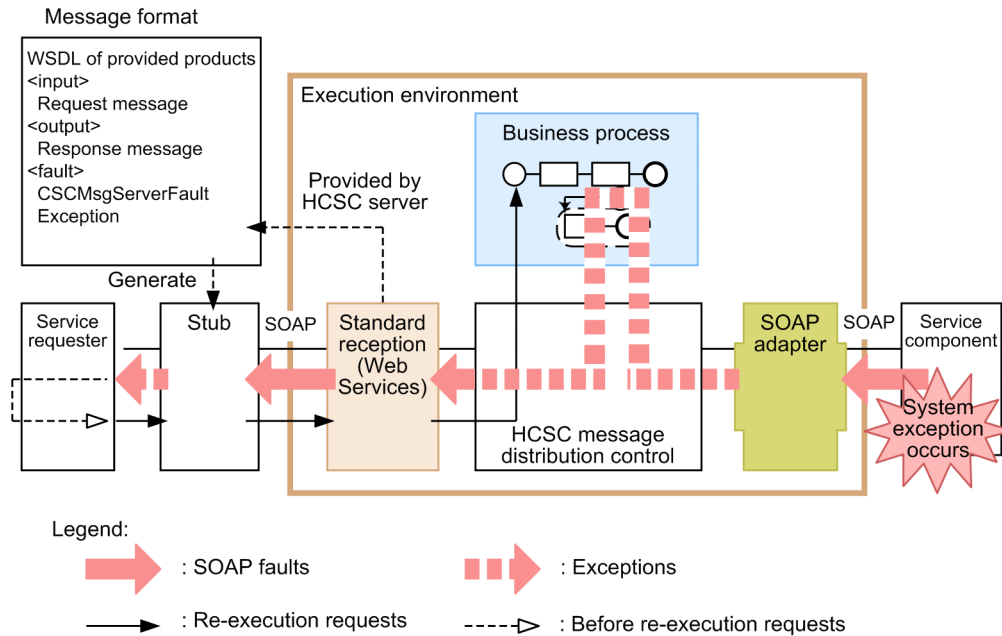
3.5.3 Re-executing business processes from service requesters (SOAP communication)

Process instances that were canceled due to an exception in the service component invocation process can be re-executed not only by the system administrator but also from a service requester.

To re-execute a process instance from a service requester, create a service requester that sends re-execution requests to synchronous (Web Services) standard reception.

If invocation of a service component results in an error and the business process instance ID is returned as error information, re-execute the process by specifying the business process instance ID. The following figure shows re-execution of a request sent from a service requester during SOAP communication.

Figure 3–50: Re-execution of a request sent from a service requester during SOAP communication



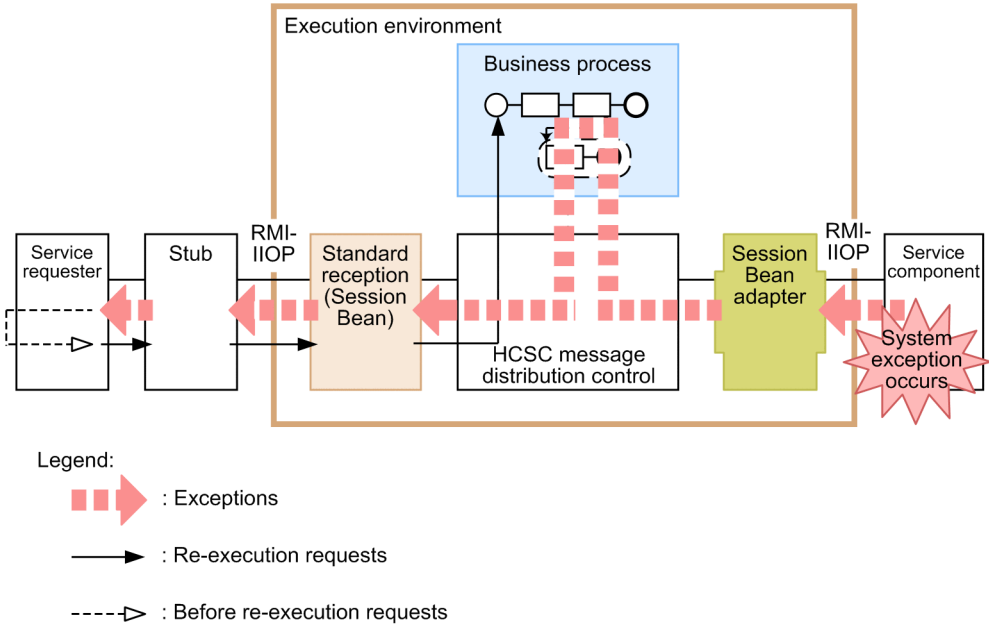
For details about creating a service requester that requests re-execution of a business process, see 8.2.9 *Creating a service requester for requesting the re-execution of business processes (Web Services and SOAP Communication Infrastructure)* in the *Service Platform Basic Development Guide*.

3.5.4 Re-executing business processes from service requesters (Session Bean)

You can also request re-execution of a business process from a service requester that sends requests to standard synchronous reception (Session Bean). To re-execute a business process from a service requester, create a service requester that sends requests to synchronous (Session Bean) standard reception.

If invocation of a service component results in an error and the business process instance ID is returned as an exception (error information), re-execute the process by specifying the business process instance ID. The following figure shows re-execution of a request sent from a service requester in a Session Bean.

Figure 3–51: Re-execution of a request sent from a service requester in a Session Bean

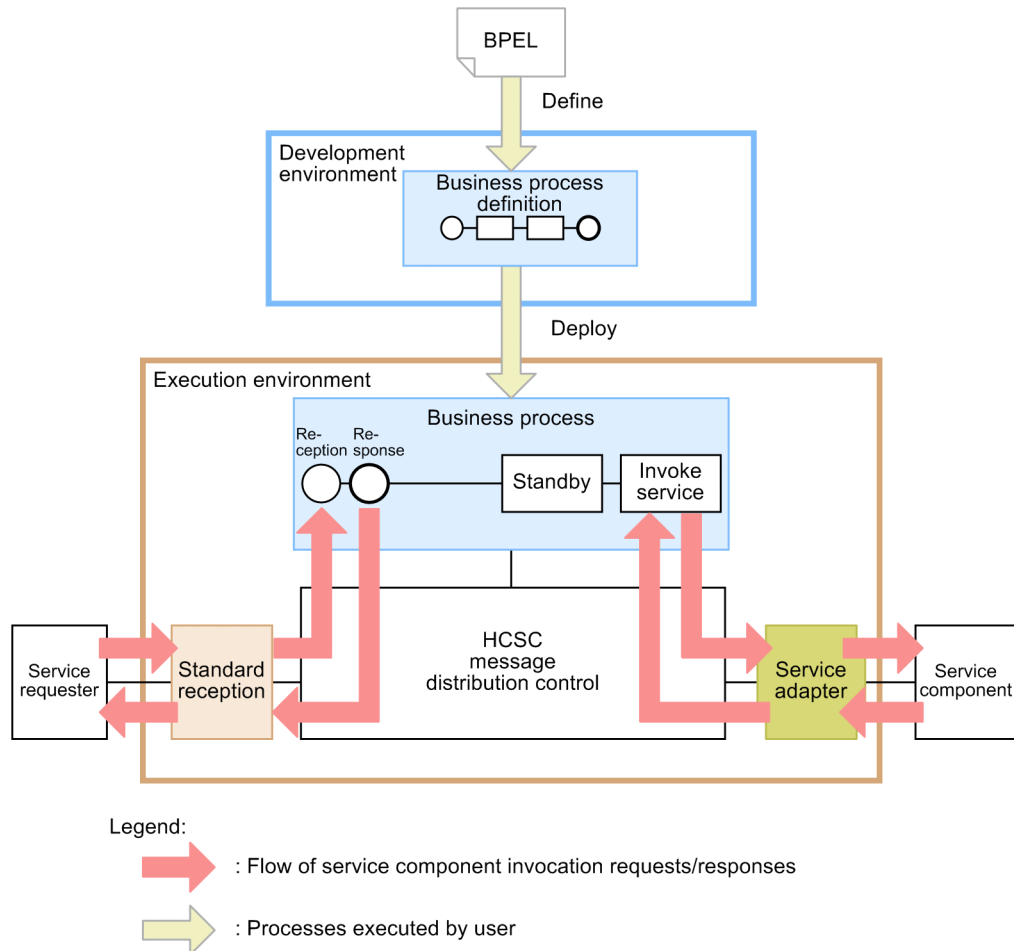


For details about creating a service requester, see 8.4.8 *Creating a service requester for requesting the re-execution of business processes (Session Bean)* in the *Service Platform Basic Development Guide*.

3.6 Standby processing using business processes

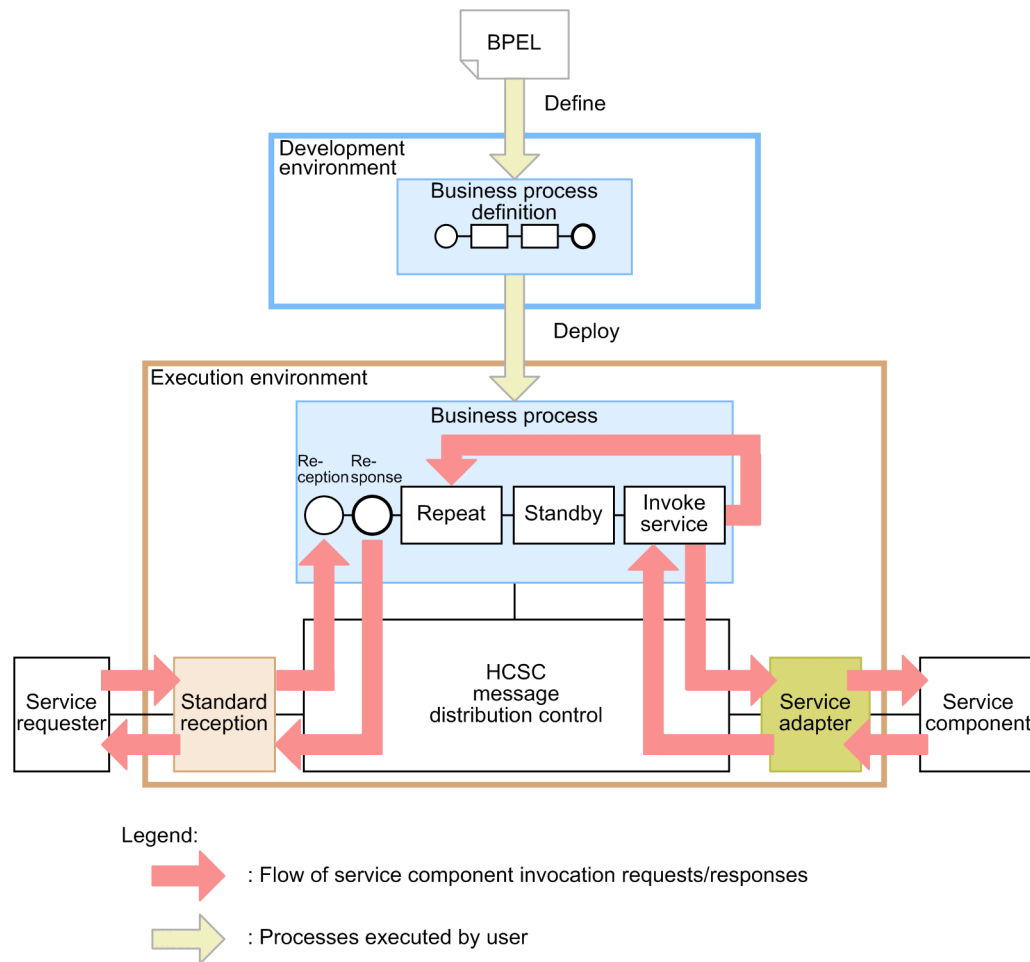
When executing a business process by defining a wait activity, you can delay the processing of the business process until a specific time period is reached. The following figure shows the standby processing of a business process by defining the wait activity.

Figure 3–52: Standby processing of a business process by defining the wait activity



By defining a wait activity within a while activity, you can implement processing executed at fixed intervals, such as daily processing. The following figure shows the standby processing of a business process when you define a wait activity in a while activity.

Figure 3–53: Standby processing of a business process when you define a wait activity in a while activity



The following describes the definition of a wait activity.

(1) Specifying the standby time in the wait activity

In the wait activity, specify either of the following as the standby time:

- Standby interval
- Expiration time for canceling the standby

In both cases, use the XPath expression to specify the standby time. Using the XPath expression allows you to dynamically specify the standby time because the operation is performed according to the time specified in the variables that flow within the business process. Instead of using the XPath expression, you can directly specify the standby interval or the standby expiration time. Use the XML Schema type `duration` to specify the standby interval. Use the XML Schema type `dateTime` to specify the standby expiration time. For details, see 5.6.10 *Standby activities* in the *Service Platform Basic Development Guide*.

(2) Conditions for executing a wait activity

Certain conditions must be met to execute standby activities. (For example, you cannot use the wait activity in a non-persistent business process, and you must set up the wait activity after the reply activity.) For details about the conditions for executing the wait activity, see 5.6.10 *Standby activities* in the *Service Platform Basic Development Guide*.

(3) Multiplicity associated with the wait activity

When considering performance during use of the wait activity, you must revise the multiplicity associated with the wait activity.

If standby processing is canceled within the time period specified in the wait activity for multiple process instances, check the maximum number of threads for calling back the timeout method (using `TimerService` of the J2EE server). If the maximum number of threads is smaller than the number of process instances, a bottleneck in performance might result. In such cases, specify the maximum number of threads for callback of the timeout method in the environment settings of the J2EE server.

In **Logical Server Environment Setup** of the management portal, specify the maximum number of threads for calling back the timeout method by using `TimerService`. Open the Preferences window of the EJB Container, and then change the value of **Maximum number of threads for callback of the timeout method**[#] in accordance with the number of instances of the business process. For details about the setup method, see *10.9.3 EJB container settings* in the *Cosminexus Application Server Management Portal Operation Guide*.

#

Corresponds to the `ejbserver.ejb.timerservice.maxCallbackThreads` key of the user property file for a J2EE server (`usrconf.properties`).

3.7 Asynchronous execution of an activity that is defined after reply activity

If you define any activity after the reply activity, you can execute that activity using Reliable Messaging or asynchronous EJB.

The function to be used for executing the activity after reply differs depending on the database and the existence of Reliable Messaging. The following table describes this concept as follows:

Table 3–12: Function to be used in execution of activity after reply

Reliable Messaging	Database	
	Yes	No
Yes	<ul style="list-style-type: none"> Reliable Messaging for persistent business process Asynchronous EJB[#] for non-persistent business process 	-
No	Asynchronous EJB [#]	Asynchronous EJB [#]

Legend:

-: Not applicable.

Note[#]

The following functions cannot be used when you use asynchronous EJB instead of Reliable Messaging for executing the activity after reply:

- Recovery process that uses the retry function of Reliable Messaging cannot be used.
For details on the recovery process, see "3.4.7 Transactions of the reply activity".
- The following receptions and adapter that presume Reliable Messaging cannot be used.
Standard reception (MDB (WS-R))
Standard reception (MDB (DB queue))
Standard adapter (MDB (WS-R))
Standard adapter (MDB (DB queue))

(1) Settings when you use asynchronous EJB for asynchronous execution

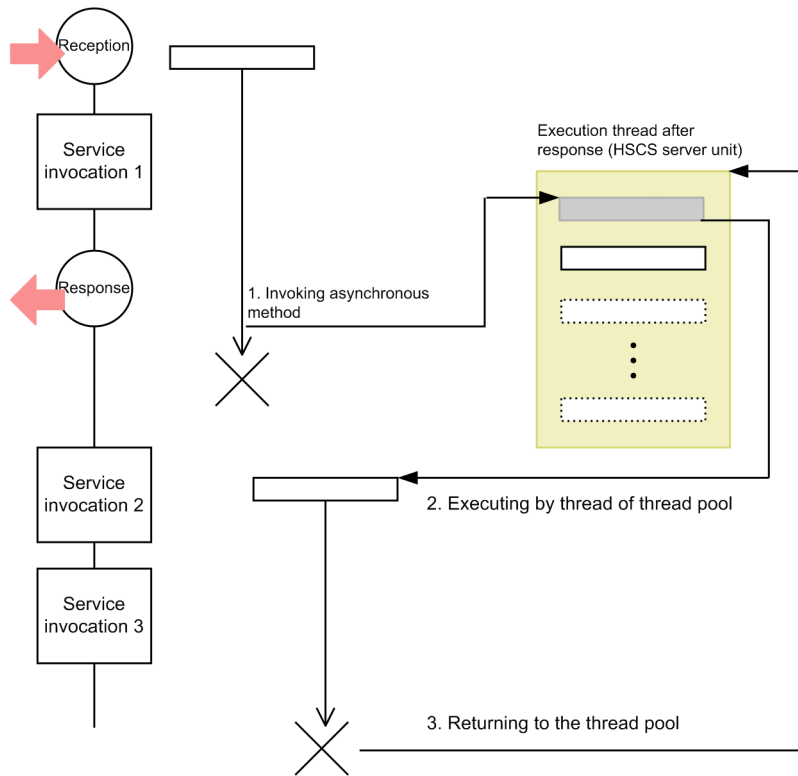
When you use asynchronous EJB, set the following properties in the HCSC server runtime definition file.


- `bp-reply-after-max-thread-pool-size`
Specifies the maximum number of threads that can be generated in a pool. The default value is 32.
- `bp-reply-after-min-thread-pool-size`
Specifies the number of unused threads that can be stored within the pool. The default value is 16.
- `bp-reply-after-thread-pool-keep-alive`
Specifies the duration (unit: seconds) for which threads are stored till the end of threads, when the number of unused threads exceeds the number specified in `min-thread-pool-size`. The default value is 60.


For the settings of the HCSC server runtime definition file, see "HCSC server runtime definition file" in "Service Platform Reference Guide".


The following is an overview of the execution of an activity after reply.

Figure 3–54: Overview of execution of an activity after reply



Legend:  : Business process reception/response

 : Minimum thread count

 : Maximum thread count

(2) How to re-execute if an exception occurs in the execution of asynchronous EJB

The following is the status of processes when an exception occurs in the activity defined after the reply activity, or when an exception occurs at the time of executing asynchronous EJB. The processes are re-executed in a batch:

- Process instance: Executing
- Reply activity: (Completedwait)
- Activity defined after the reply activity: Unexecution

Depending on the re-execution of process instance, you can execute the activity defined after the reply activity.

4

Functionality for Smooth System Operations

This chapter describes the functionality for smoothly performing operations of a system.

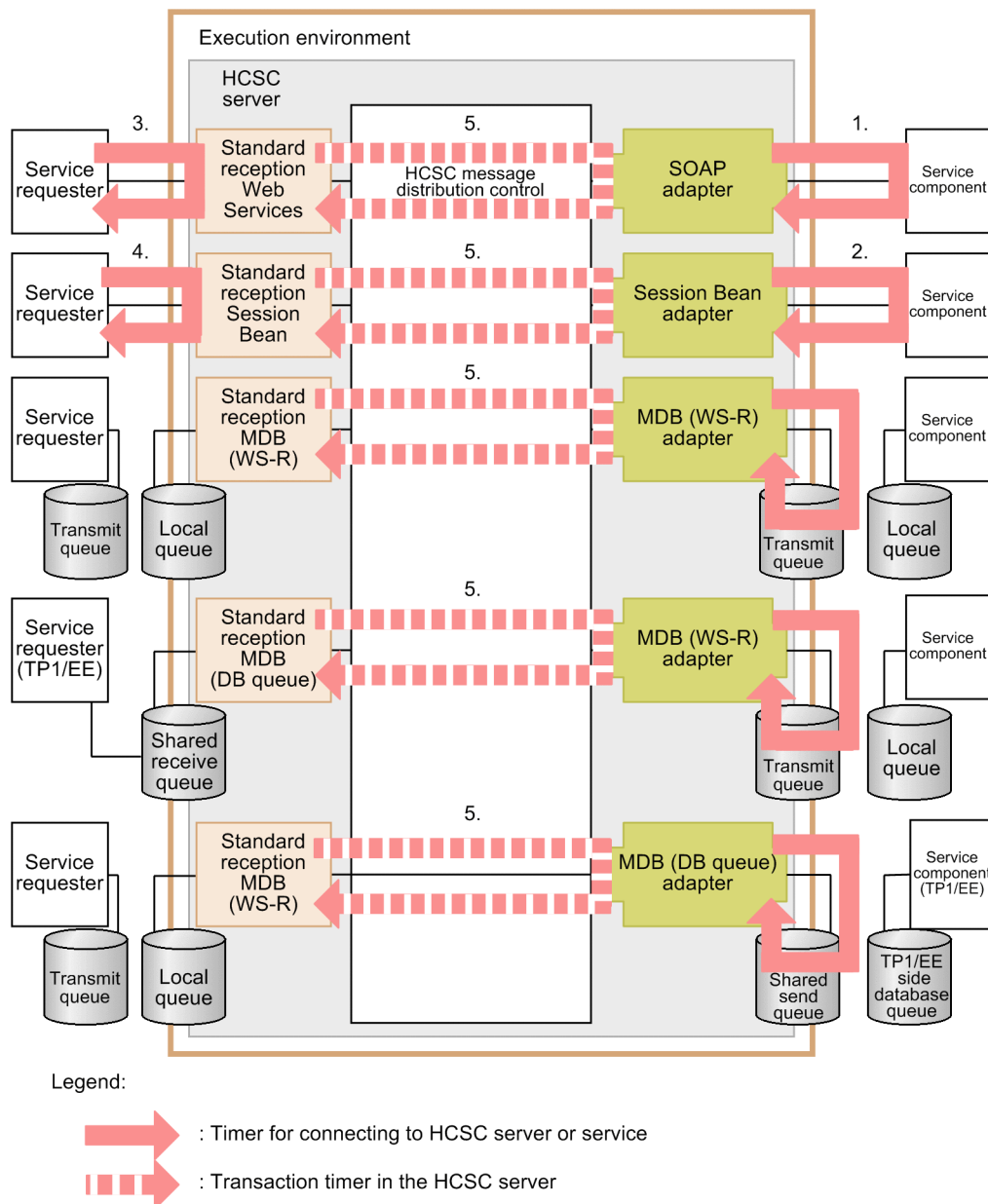
4.1 Setting up a timer for an HCSC server

For invoking a synchronous service component, you can set up a timeout value when no response is returned, and can avoid hang-ups caused when no response is returned. Also, the timer must be set up in consideration of the maximum time taken for actual processing in a service component.

4.1.1 Timers that can be set up on an HCSC server

The following figure shows the timers that can be set up on an HCSC server.

Figure 4-1: Timers that can be set up



Each number in Figure 4-1 *Timers that can be set up* indicates the following timers:

1. Timer (Web Services) used when connecting to service components
2. Timer (Session Bean) used when connecting to service components

3. Timer (Web Services) used when connecting to the HCSC server
4. Timer (Session Bean) used when connecting to the HCSC server
5. Timer for transactions within the HCSC server

The method for setting up a timer differs depending on the connection destination and the protocol.

(1) Timer (Web Services) used when connecting to a service component

When connecting to a service component (Web Services), you can set up timers used for writing, reading, and establishing a connection.

- Timer used for writing
Set up a timeout value for the `write` (send data) processing in the socket used in HTTP and HTTPS (TCP/IP communication). This timer is enabled when data sending after establishing a connection has not been completed.
- Timer used for reading
Set up a timeout value for the `read` (receive data) processing in the socket used in HTTP and HTTPS (TCP/IP communication). This timer is enabled when data reception after establishing a connection has not been completed.
- Timer used for establishing a connection
Set up a timeout value for the `connect` (socket connection) processing in the socket used in HTTP and HTTPS (TCP/IP communication). This timer is enabled when there is no response for a request when a connection is established.

The methods for setting up a timeout value are different depending on whether you are using the SOAP Communication Infrastructure or the JAX-WS engine. The following describes the setup method for each case.

(a) Setting up the timeout value (SOAP Communication Infrastructure)

This subsection describes how to set up a timeout value when using the SOAP Communication Infrastructure. You can set up the timeout value for the entire HCSC server or separately for each service adapter.

- Setting up the timer for the entire HCSC server
Define the timeout value in the server definition file of the SOAP Communication Infrastructure for the machine running the HCSC server. For details about setting up the server definition file, see *10.2 Setting up the server definition files* in the *Application Server SOAP Application Development Guide*.

Table 4–1: Timer (Web Services) settings for the entire HCSC server (SOAP Communication Infrastructure)

Value	Key name	Default value (seconds)
Write timeout value for the socket of a server client	<code>c4web.application.identifier.socket_write_timeout</code>	60
Read timeout value for the socket of a server client	<code>c4web.application.identifier.socket_read_timeout</code>	300
Connection timeout value for the socket of a server client	<code>c4web.application.identifier.socket_connect_timeout</code>	60

- Setting up the timer separately for service adapters
To set up the timer in the development environment, specify the timeout value in the client definition file in the Service Adapter Settings window. For details about setting up the client definition file, see *10.3 Setting up the client definition files* in the *Application Server SOAP Application Development Guide*.

Table 4–2: Timer (Web Services) settings for each service adapter (SOAP Communication Infrastructure)

Value	Key name	Default value (seconds)
Write timeout value for the socket of a client	<code>c4web.application.socket_write_timeout</code>	60
Read timeout value for the socket of a client	<code>c4web.application.socket_read_timeout</code>	300
Connection timeout value for the socket of a client	<code>c4web.application.socket_connect_timeout</code>	60

(b) Changing the timeout value (SOAP Communication Infrastructure)

To define the timeout value in the operating environment for the service adapter that is already deployed on the HCSC server, use the `cscsvccctl` command to change the communication timeout value of the service component invocation. For details about how to change a timeout value, see 5.3.26 *Changing the communication timeout value for invoking a service component* in the *Service Platform Setup and Operation Guide*. For details about how to use the `cscsvccctl` command, see *cscsvccctl (Managing the service information)* in the *Service Platform Reference Guide*.

Table 4–3: Changing the set value of the timer (Web Services) (SOAP Communication Infrastructure)

Value	Key name
Write timeout value	<code>cluster-name.service-ID.WebService.c4web.application.socket_write_timeout</code>
Read timeout value	<code>cluster-name.service-ID.WebService.c4web.application.socket_read_timeout</code>
Connection timeout value	<code>cluster-name.service-ID.WebService.c4web.application.socket_connect_timeout</code>

(c) Setting up a timeout value (JAX-WS engine)

This subsection describes how to set up a timeout value when using the JAX-WS engine. You can set up the timeout value for the entire HCSC server or separately for each service adapter.

- Setting up the timer for the entire HCSC server

Define the timeout value in the common definition file of the JAX-WS engine of the machine running the HCSC server. For details about setting up the common definition file, see 10.1.2 *Settings in the common definition files* in the *Application Server Web Service Development Guide*.

Table 4–4: Timer (Web Services) settings for the entire HCSC server (JAX-WS engine)

Value	Key name	Default value (milliseconds)
Read timeout value for the socket of a server client	<code>com.cosminexus.jaxws.request.timeout</code>	300000
Connection timeout value for the socket of a server client	<code>com.cosminexus.jaxws.connect.timeout</code>	60000

- Setting up the timer separately for service adapters

To set up the timer in the development environment, specify the timeout value in the client definition file in the Service Adapter Settings window. For details about setting up the client definition file, see *Client Definition Files* in the *Service Platform Reference Guide*.

Table 4–5: Timer (Web Services) settings for each service adapter (JAX-WS engine)

Value	Key name	Default value (seconds)
Read timeout value for the socket of a client	<code>com.cosminexus.csc.request.timeout</code>	300
Connection timeout value for the socket of a client	<code>com.cosminexus.csc.connect.timeout</code>	60

(d) Changing the timeout value (JAX-WS engine)

To define the timeout value in the operating environment for the service adapter that is already deployed on the HCSC server, use the `cscsvctl` command to change the communication timeout value of the service component invocation. For details about how to change a timeout value, see *5.3.26 Changing the communication timeout value for invoking a service component* in the *Service Platform Setup and Operation Guide*. For details about how to use the `cscsvctl` command, see *cscsvctl (Managing the service information)* in the *Service Platform Reference Guide*.

Table 4–6: Changing the set value of the timer (Web Services) (JAX-WS engine)

Value	Key name
Read timeout value	<code>cluster-name.service-ID.WebService.com.cosminexus.csc.request.timeout</code>
Connection timeout value	<code>cluster-name.service-ID.WebService.com.cosminexus.csc.connect.timeout</code>

(2) Timer (Session Bean) used when connecting to a service component

(a) Setting up a timeout value

The timer (Session Bean) used when connecting to a service component can be set up for the entire HCSC server or separately for service adapters. The following describes the setup methods.

- Setting up the timer for the entire HCSC server

Set up the timeout value in the EJB container of Application Server on the machine running the HCSC server. For details about the timeout of the RMI-IIOP communication, see *2.11.5 RMI-IIOP communication timeout* in the *Application Server EJB Container Functionality Guide*. For details about parameters to be set up, see *2.4 usrconf.properties (User property file for Java applications)* in the *Application Server Definition Reference Guide*.

Table 4–7: Timer (Session Bean) settings for the entire HCSC server

Value	File to be set up	Key name	Default value (seconds)
Timeout value enabled for all RMI-IIOP communications	User property file for a J2EE server (<code>usrconf.properties</code>)	<code>ejbserver.rmi.request.timeout</code>	0 (A timeout will not occur.)
Timeout value for communication with the CORBA Naming Service component		<code>ejbserver.jndi.request.timeout</code>	0 (A timeout will not occur.)

- Setting up the timer separately for service adapters

To define the timer in the development environment, specify the timeout value in the file specified for the client definition file in the Service Adapter Settings window. For details about setting up the client definition file, see *10.3 Setting up the client definition files* in the *Application Server SOAP Application Development Guide*.

Table 4–8: Timer (Session Bean) settings for each service adapter

Value	Key name	Default value (seconds)
Invocation timeout value	<code>c4web.application.ejb_timeout</code>	0 (A timeout will not occur.)

(b) Changing the timeout value

To define the timeout value in the operating environment for the service adapter that is already deployed on the HCSC server, use the `cscsvccctl` command to change the communication timeout value of the service component invocation. For details about how to change a timeout value, see 5.3.26 *Changing the communication timeout value for invoking a service component in the Service Platform Setup and Operation Guide*. For details about the `cscsvccctl` command, see *cscsvccctl (Managing the service information)* in the *Service Platform Reference Guide*.

Table 4–9: Changing the set value of the timer (Session Bean)

Value	Key name
Invocation timeout value	<code>cluster-name.service-ID.SessionBean.c4web.application.ejb_timeout</code>

(3) Timer (Web Services) used when connecting to the HCSC server

The timer (Web Services) used when connecting to the HCSC server is set up on the service requester machine. You can set up the timer for the entire machine on which the service requester is running or separately for service requesters.

The setup method is described separately for the cases where you use the SOAP Communication Infrastructure and where you use JAX-WS engine.

(a) Setting up the timeout value (SOAP Communication Infrastructure)

This subsection describes how to set up a timeout value when using the SOAP Communication Infrastructure. You can set up the timeout value for the entire service requester machine or separately for each service requester.

- **Setting the timer for the entire service requester machine**
Define the timeout value in the server definition file of the SOAP Communication Infrastructure for the service requester machine. The settings are the same as for 4.1.1(1)(a) *Setting up the timeout value (SOAP Communication Infrastructure)*. For details about setting up the server definition file, see 10.2 *Setting up the server definition files* in the *Application Server SOAP Application Development Guide*.
- **Setting up the timer separately for each service requester**
To set up a timer separately for each service requester to be created, set up the timer in the client definition file specified when executing the service requester. The settings are the same as for 4.1.1(1)(a) *Setting up the timeout value (SOAP Communication Infrastructure)*. For details about setting up the client definition file, see 10.3 *Setting up the client definition files* in the *Application Server SOAP Application Development Guide*.

(b) Setting up the timeout value (JAX-WS engine)

This subsection describes how to set up a timeout value when using the JAX-WS engine. You can set up the timeout value for the entire service requester machine or separately for each service requester.

- **Setting the timer for the entire service requester machine**
Define the timeout value in the common definition file of the JAX-WS engine of the service requester machine. The settings are the same as for 4.1.1(1)(c) *Setting up a timeout value (JAX-WS engine)*. For details about setting up the common definition file, see 10.2 *Settings in the common definition files* in the *Application Server Web Service Development Guide*.
- **Setting up a timer separately for each service requester**

To set up a timer separately for each service requester to be created, set up the timer in the process-wise definition file. For details about the settings of the process-wise definition file, see *10.1.3 Setting up the process-wise definition files* in the *Web Service Development Guide*.

(4) Timer (Session Bean) used when connecting to the HCSC server

The timer (Session Bean) used when connecting to the HCSC server can be set up for the entire machine on which the service requester is running or separately for service requesters. The following describes the setup methods:

- Setting up a timer for the entire service requester machine
Set up the timeout value in the EJB container of Application Server on the service requester machine. The settings are the same as for (2) *Timer (Session Bean) used when connecting to a service component*. For details about the timeout of the RMI-IIOP communication, see *2.11.5 RMI-IIOP communication timeout* in the *Application Server EJB Container Functionality Guide*. For details about parameters to be set up, see *14.3 usrconf.properties (User property file for Java applications)* in the *Application Server Definition Reference Guide*.
- Setting up a timer separately for each service requester
For each service requester to be created, specify a timeout value for API communication from `create` to `remove` in APIs (methods of the `RequestTimeoutConfigFactory` class and `RequestTimeoutConfig` class) when developing a service requester application.

(5) Timer for transactions within the HCSC server

The timer for transactions within the HCSC server is set up for the entire HCSC server. Set up the timeout value in the EJB container of Application Server on the machine running the HCSC server. For details about parameters to be set up, see *2.4 usrconf.properties (User property file for Java applications)* in the *Application Server Definition Reference Guide*.

Table 4–10: Timer settings for transactions within the HCSC server

Value	File to be set up	Key name	Default value (seconds)
Transaction timeout value of transactions started on the J2EE server	User property file for a J2EE server (<code>usrconf.properties</code>)	<code>ejbserver.jta.TransactionManager.defaultTimeout</code>	180

4.1.2 Changing the timeout value of service components

A communication timeout value is defined in the development environment. For SOAP adapters and Session Bean adapters, the communication timeout value for establishing a connection to service components can be changed in the operating environment. To change the timeout value of a service component in the operating environment, use the `cscsvctl` command. Note that the service target adapter must be in an undeployed state.

For details about how to change a timeout value, see *5.3.26 Changing the communication timeout value for invoking a service component* in the *Service Platform Setup and Operation Guide*. For details about how to use the `cscsvctl` command, see *cscsvctl (Managing the service information)* in the *Service Platform Reference Guide*.

4.1.3 Precautions to be taken when setting up a timer

(1) Relationship between the timer used when connecting to a service component and the timer used when connecting to the HCSC server

Specify a higher value for the timer used when connecting to the HCSC server as compared to the timer used when connecting to a service component.

Even when the time is required normally for invoking service components and returning responses, the timer used when connecting to the HCSC server will be timeout first, and the processing will not complete successfully.

When invoking a service component after invoking a business process, you must set up the timer used when connecting to the HCSC server after considering the fact that multiple service components will be invoked.

(2) Relationship between the timer used when connecting to a service component and the timer for transactions

Specify a higher value in the timer for transactions as compared to the timer used when connecting to a service component.

Even when the time is required normally for invoking service components and returning responses, if the transaction times out, it rolls back.

4.2 Changing the connection destination of service components

The connection destination of service components is defined in the development environment. For the following service adapters, the connection destination for establishing a connection to a service component can be changed in the operating environment:

- SOAP adapter
- Session Bean adapter
- MDB (WS-R) adapter
- MDB (DB queue) adapter

To change the connection destination of a service component in the operating environment, use the `cscsvctl` command. Note that the service target adapter must be in an undeployed state.

For details about how to change the connection destination, see *5.3.24 Changing the connection destination of a service component* in the *Service Platform Setup and Operation Guide*. For details about how to use the `cscsvctl` command, see *cscsvctl (Managing the service information)* in the *Service Platform Reference Guide*.

4.3 Catching a fault by using the SOAP fault operation definition file

This section describes how to handle SOAP faults of service adapters for Web Services, and the notes for handling a SOAP fault as a fault.

(1) Handling SOAP faults

For a SOAP adapter, if a SOAP fault returned from a service component meets both of the following conditions, the SOAP fault is handled as a fault:

- The SOAP fault is defined in the WSDL of the service component.
- The fault name is specified in `faultCode`.

Such a fault is called a *user-defined exception*. Because a user-defined exception is recognized even in a business process, conditional processing can be defined for the process after a fault occurrence.

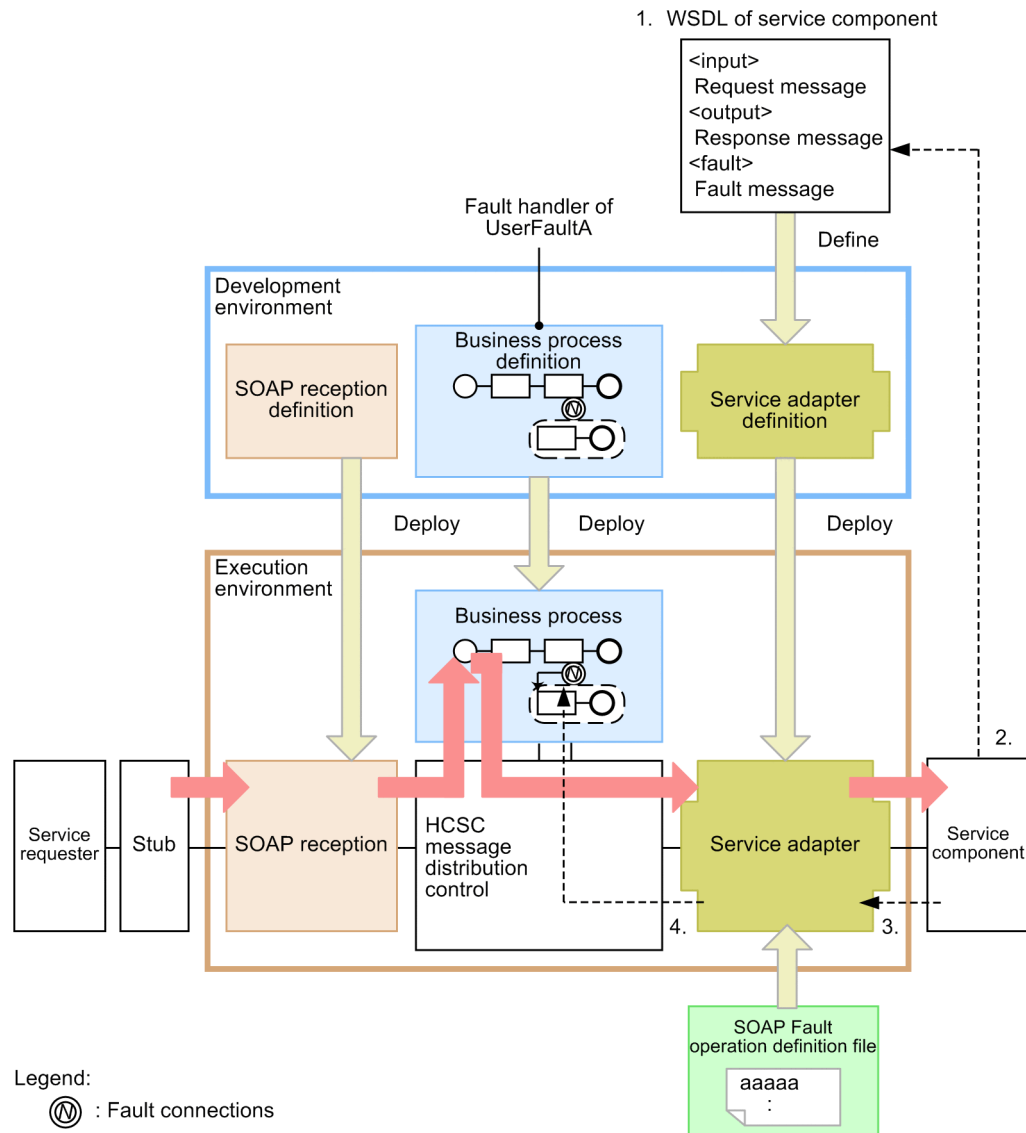
Depending on the type of service component, a SOAP fault that is not defined in the WSDL, or a SOAP fault with any value specified in `faultCode`, might be returned. If you want to handle such a SOAP fault as a fault, you can use the SOAP fault operating definition file to handle all the SOAP faults returned from service components as faults.

In the SOAP fault operation definition file, define the name space (URI of `targetNamespace`) declared in the WSDL of the service component. For details about the SOAP fault operation definition file, see the *SOAP fault Operation definition file* in the *Service Platform Reference Guide*.

For individual service adapters, interfaces of service components are defined in the WSDL. Whether a SOAP fault is handled as a fault is determined based on whether the name space defined in the WSDL matches the contents defined in the SOAP fault operation definition file.

The following figure shows how a SOAP fault is handled when the SOAP fault operation definition file is used.

Figure 4-2: Handling a SOAP fault when the SOAP fault operation definition file is used



1. Define the WSDL of the service component.
2. If the SOAP fault returned from the service component is defined in the WSDL of the service component, it will be handled as a fault.
3. If the returned SOAP fault is not defined in the service component's WSDL or is defined in a name space different from the WSDL definition, the SOAP fault is checked with the SOAP fault operation definition file.
4. If the SOAP fault matches the contents defined in the SOAP fault operation definition file, the SOAP fault is returned (as a fault) to the business process.

(2) Notes on handling a SOAP fault as a fault

If all of the following conditions exist when a SOAP fault returns from a service component, the fault name is set in `faultCode` and the SOAP fault is handled as a fault:

- The SOAP Communication Infrastructure must be used at the service component side.
- The `wsdl:part` element that defines the fault message must always reference the `element` element within the `xsd:schema` element by using the `element` attribute.

- The WSDL fault name (name attribute value of the `wsdl:fault` element) and the element name[#] indicating the exception type `complexType` defined in the schema (within `wsdl:types`) must be the same, including the name space.

When using a WSDL that does not meet the above conditions, if you want to handle SOAP faults returned from service components as faults, define the SOAP faults as user-defined exceptions in the SOAP fault operation definition file.

#

The element name is an element specified by the `wsdl:part` element (that is referenced by `wsdl:fault`) by using the `element` attribute. Specify an element that can be specified by tracing the WSDL in the following order:

1. `wsdl:fault`
2. `wsdl:message` (This element might not be traced in some cases.)
3. `wsdl:part`
4. `xsd:element`

Note that you can generate a WSDL file with the `style` attribute of the operation set to `rpc` by using the `Java2WSDL` command, which is a Service Platform development command. In this case, the generated WSDL file will be referenced with the `type` attribute, rather than with the `element` attribute. Therefore, do not use the generated WSDL file as is, but first modify and regenerate the WSDL file, and then use it. For details about how to modify and regenerate a WSDL file, see *4.3.2 Creating a service component message (for a Web service)* in the *Service Platform Basic Development Guide*.

When using faults with Service Platform, we recommend that you use a `document` style WSDL.

4.4 Checking the operation status from an application

4.4.1 Overview of checking the operation status

If you use Web Services (SOAP communication) from outside a service requester, and send a request for checking the operation status with the service name as a key, you can acquire the information of a service adapter deployed on the HCSC server.

Note that the operation status is not checked by checking the status of service components, but by checking the status of the service adapters deployed on the HCSC server.

4.4.2 Flow of checking the operation status

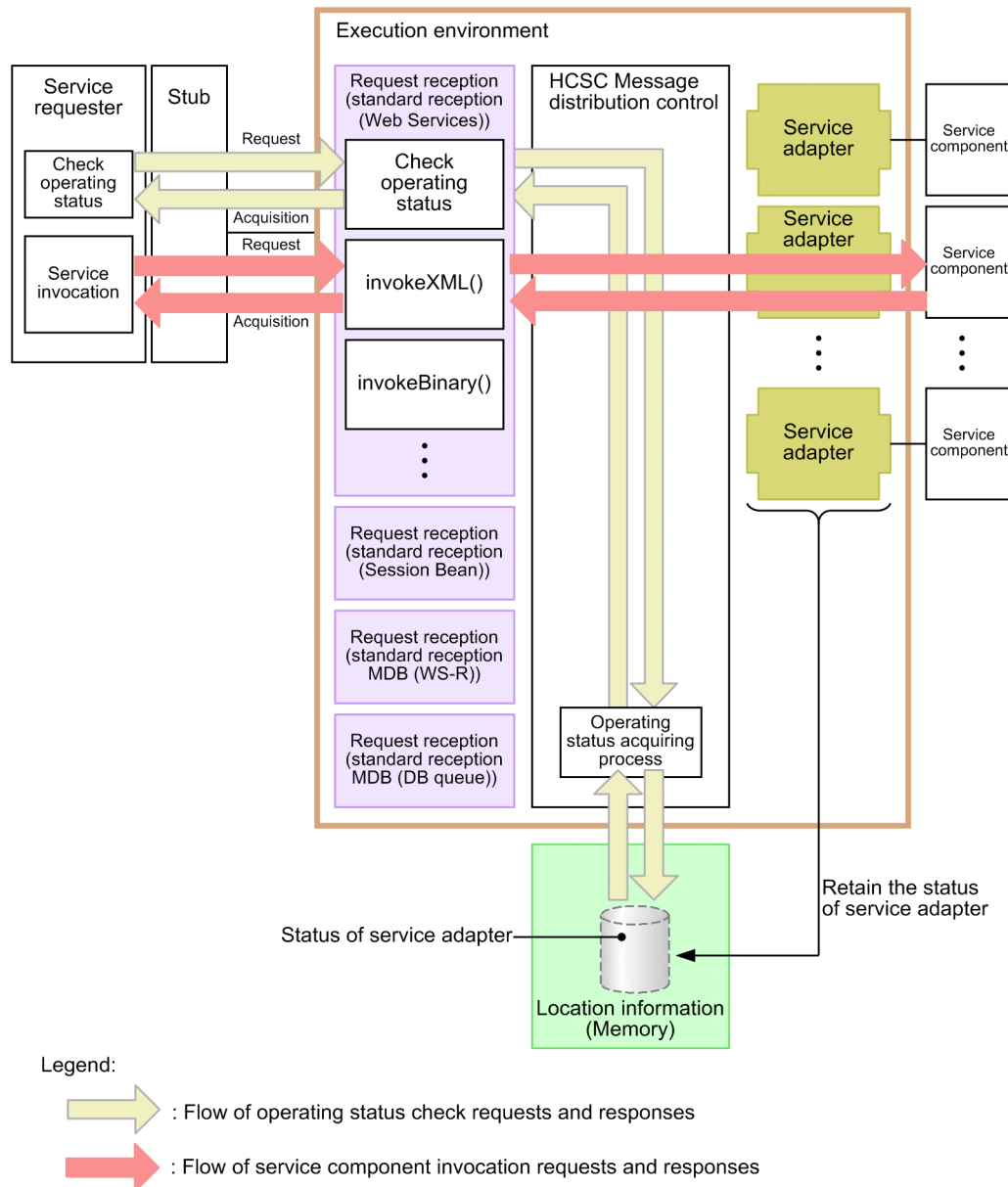
The methods (operations of request reception) for delivering messages from a service requester to the request reception (standard reception (Web Services)) include `invokeXML` and `invokeBinary`. Among these methods, you can use the `getServiceInfo` operation for checking the service adapter operation status.

When you send a request for checking the service adapter operation status from the service requester, the HCSC server searches the location information (information on the memory) of the service adapter corresponding to the specified service name. The HCSC server then sends the status as a response.

Note that you can send a request for checking the operation status only when the request reception of the HCSC server is running.

The following figure shows the flow of checking the operation status.

Figure 4-3: Flow of checking the operation status



4.4.3 Precautions on checking the operation status

(1) About synchronization between the status of service components and the status of service adapters

The functionality for checking the operation status is used to check the status of the service adapters deployed on the HCSC server and not to check the status of service components. Therefore, when checking the operation status, service components and service adapters must start and stop in synchronization. If a service adapter is running but a service component has stopped, a request for service component invocation causes an error.

(2) About differences in the results of checking the operation status and the actual status

Even if you confirm that the service adapter is running during the operation status check, if the service adapter stops immediately after the check, an error occurs when you send a request for service component invocation. Therefore,

you must consider that even if the running status of the service adapter is confirmed in the service requester, an error might return in response to a request for service component invocation if the service adapter has stopped.

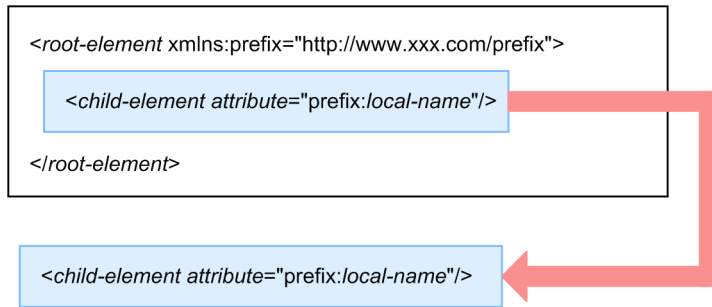
(3) About the status of request reception

If the request reception of the HCSC server is not running, an error occurs when you check the operation status of the service adapter in the service requester. Therefore, you must consider that an error might return in response to a request for checking the operation status of the service adapter from the service requester if the request reception has stopped.

4.5 Complementing name space prefixes specified in attribute values of XML messages

You can use the `QName` type as the type of attribute values of the XML schema in the user messages flowing within an HCSC server. In this case, if you output a child element extracted as a sub tree from the XML document as shown in the following figure, the name space declaration of `xmlns:prefix` will be omitted.

Figure 4–4: Omission of the name space declaration during acquisition of a sub tree



If the name space declaration is omitted, the following problems might occur:

- An attempt is made to catch faults by using a fault connection in the business process, but the faults from the service components cannot be caught.
- An error occurs during validation when the data validation functionality is used (ON is specified in the `telegram-validation` property of the HCSC server runtime definition file).
- When a service requester or service component receives an XML message in which the name space declaration is omitted, an error occurs in the service requester or service component.

For prefixes of child elements, the name space is resolved when the sub tree is acquired.

To correct such problems, specify ON in the `xmltelegram-namespace-complement` property of the HCSC server runtime definition file. However, if you specify ON, note that the message size becomes longer than when OFF is specified.

For details about the HCSC server runtime definition file, see *HCSC Server Runtime definition file* in the *Service Platform Reference Guide*.

4.6 Defining message format of any format

In development environment, by setting any format in format definition file of a message, you can perform service component request and response without considering message format.

Note that, reception and adapter in which you can handle data of any format are as follows:

- User-defined reception (Custom reception)
- Custom adapter

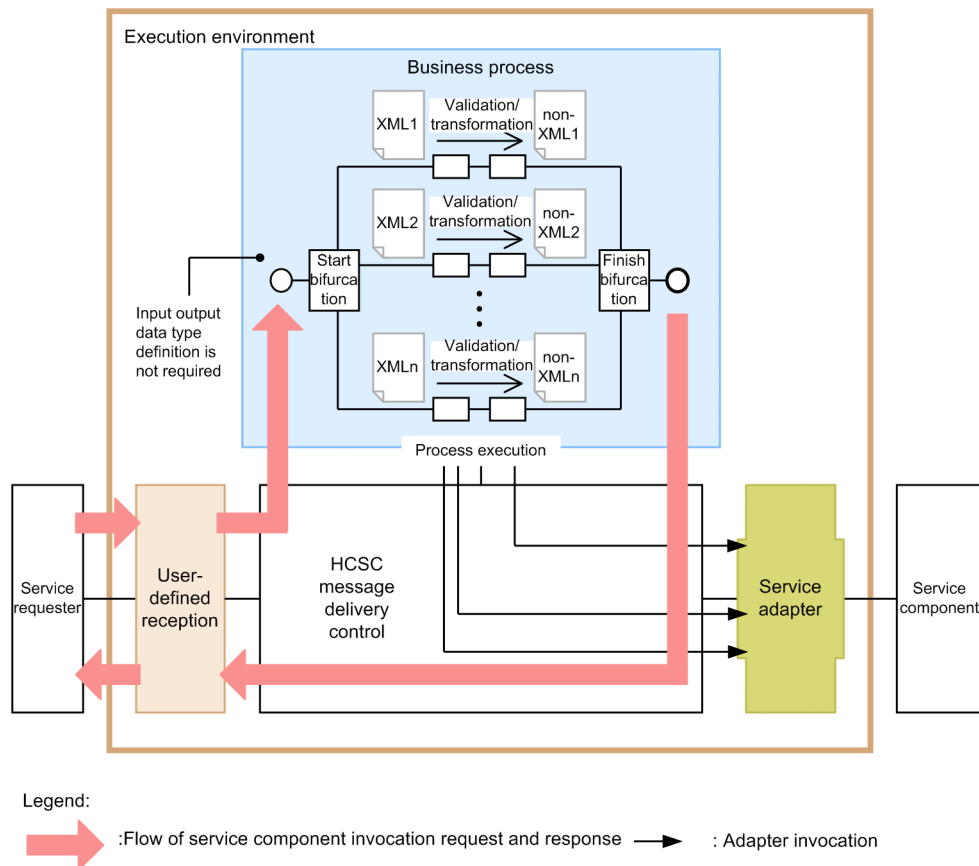
For details on combination of message types that does not transform the data, see "4.6.2 Combination of variables (Message type) when data is not transformed".

4.6.1 Flow of process when any format is set

When you set any format in the message format definition file, you need not consider data type of the request message and the response message.

The following figure shows the flow of process when you set any format in the message format definition file:

Figure 4–5: Flow of process when any format is set in the message format definition file

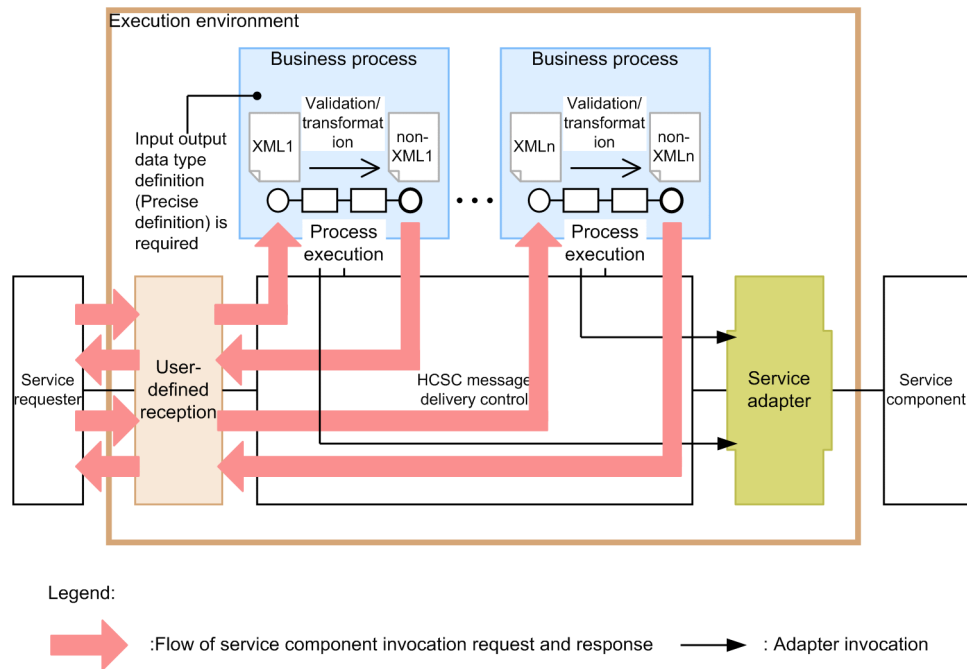


Reference note

When you do not set any format in the message format definition file, you must properly decide the data type of the request message and the response message. Also, you must create components (reception, business process, and service adapter) for each message format in the development environment.

The following figure shows flow of the process when any format is not set in the message format definition file:

Figure 4–6: Flow of the process when any format is not set in the message format definition file



4.6.2 Combination of variables (Message type) when data is not transformed

You can perform the operation that does not transform data by setting any format. The prerequisites when data is not transformed are as follows:

- Message type (XML, non-XML, any) of invoke service activity and data type which is set in components (Reception created in development environment, business process, and general terms for service adapter) invoked from invoke service activity are matching.
- When you set any format in invoke service activity of a business process, component to be invoked is defined with any format in setting screen of development environment.

Combination of message types when data is not transformed in reception and service adapter is as follows:

Table 4–11: Combination of message types when data is not transformed in reception

Component type	Type or method of data	Variable (Message type)					
		XML		non-XML		any	
		Request	Response	Request	Response	Request	Response
Standard reception (Web service)	invokeXML	Y	Y	A	A	A	A
	invokeBinary	A	A	Y	Y	Y	Y
Standard reception (SessionBean)	invokeXML	Y	Y	A	A	A	A
	invokeBinary	A	A	Y	Y	Y	Y
Standard reception (MDB (WS-R))	onMessage	Y	Y	Y	Y	Y	Y
Standard reception (MDB (Database queue))	onMessage	Y	Y	Y	Y	Y	Y

Component type	Type or method of data		Variable (Message type)					
			XML		non-XML		any	
			Requ est	Resp onse	Requ est	Resp onse	Requ est	Resp onse
User-defined reception (SOAP reception)			Y	Y	A	A	A	A
User-defined reception (Custom reception)	invokeXML	XML	Y	Y	A	A	A	A
		non-XML	A	A	A	A	A	A
		any	Y	Y	Y	Y	Y	Y
	invokeBinary	XML	A	A	A	A	A	A
		non-XML	A	A	Y	Y	A	A
		any	Y	Y	Y	Y	Y	Y
User-defined reception (TP1 and RPC reception)	XML		A	A	A	A	A	A
	non-XML		A	A	Y	Y	Y	Y
	any		Y	Y	Y	Y	Y	Y
User-defined reception (FTP reception)	XML		Y	Y	A	A	A	A
	non-XML		A	A	A	A	A	A
	any		A	A	A	A	A	A
User-defined reception (HTTP reception)	XML		Y	Y	A	A	A	A
	non-XML		A	A	Y	Y	Y	Y
	any		A	A	Y	Y	Y	Y
User-defined reception (Message Queue reception)	XML		Y	Y	A	A	Y	Y
	non-XML		A	A	Y	Y	Y	Y
	any		Y	Y	Y	Y	Y	Y

Legend:

Y: Supporting.

A: Not supporting.

Table 4–12: Combination of message types when data is not transformed in service adapter

Component type	Data type	Variable (Message type)					
		XML		non-XML		any	
		Request	Response	Request	Response	Request	Response
SOAP adapter	XML	Y	Y	A	A	A	A
	non-XML	-	-	-	-	-	-
	any	-	-	-	-	-	-
SessionBean adapter	XML	Y	Y	A	A	A	A
	non-XML	-	-	-	-	-	-
	any	-	-	-	-	-	-

4. Functionality for Smooth System Operations

Component type	Data type	Variable (Message type)					
		XML		non-XML		any	
		Request	Response	Request	Response	Request	Response
MDB (WS-R) adapter	XML	Y	-	A	-	A	-
	non-XML	A	-	Y	-	A	-
	any	-	-	-	-	-	-
MDB (Database queue) adapter	XML	Y	-	A	-	A	-
	non-XML	A	-	Y	-	A	-
	any	-	-	-	-	-	-
DB adapter	XML	Y	Y	A	A	A	A
	non-XML	A	A	A	A	A	A
	any	A	A	A	A	A	A
TP1 adapter	XML	A	A	A	A	A	A
	non-XML	A	A	Y	Y	A	A
	any	A	A	A	A	Y	Y
FTP adapter	XML	Y	Y	A	A	A	A
	non-XML	A	A	A	A	A	A
	any	A	A	A	A	A	A
File adapter	XML	Y	Y	A	A	A	A
	non-XML	A	A	Y	Y	A	A
	any	A	A	A	A	A	A
Object Access adapter	XML	Y	Y	A	A	A	A
	non-XML	A	A	A	A	A	A
	any	A	A	A	A	A	A
Message Queue adapter	XML	Y	Y	A	A	A	A
	non-XML	A	A	Y	Y	A	A
	any	A	A	A	A	A	A
File operations adapter	XML	Y	Y	A	A	A	A
	non-XML	A	A	A	A	A	A
	any	A	A	A	A	A	A
Mail adapter	XML	Y	Y	A	A	A	A
	non-XML	A	A	A	A	A	A
	any	A	A	A	A	A	A
HTTP adapter	XML	Y	Y	A	A	A	A
	non-XML	A	A	Y	Y	A	A
	any	A	A	A	A	Y	Y

Component type	Data type	Variable (Message type)					
		XML		non-XML		any	
		Request	Response	Request	Response	Request	Response
Custom adapter	XML	Y	Y	A	A	A	A
	non-XML	A	A	Y	Y	A	A
	any	A	A	A	A	Y	Y

Legend:

Y: Supporting.

A: Not supporting.

-: Cannot create.

The following table describes whether HCSC component wherein any format is set is supported or not when stand-alone service is executed.

Table 4–13: Possibility of supporting HCSC components when stand-alone service is executed

Component name	Method	Setting of format definition file	
		any	
		Request	Response
Standard reception (Web service)	invokeXML	A	A
	invokeBinary	Y	Y
Standard reception (SessionBean)	invokeXML	A	A
	invokeBinary	Y	Y
Standard reception (MDB (WS-R))	onMessage	Y	Y
Standard reception (MDB (Database queue))	onMessage	Y	Y

Legend:

Y: Supporting.

A: Not supporting.

4.7 General faults for converting system exceptions to faults

Even an activity that cannot generate a fault can send a fault by converting a system exception that occurred in the activity to a general fault and then sending it.

4.7.1 Overview of general faults for converting system exceptions to faults

The general fault message send functionality is used to convert a system exception that occurred in an activity to a fault message created in the common format and then send that message. This allows a fault to be sent from an activity that cannot generate a fault.

(1) Some activities can throw a converted fault message which is system exception occurred in the activities

The following activities are applicable:

- Data transformation activity
- Assign activity
- Switch activity

The following table describes the difference in operation when an exception occurs in an activity, based on different settings.

Table 4–14: Difference in operation when an exception occurs in an activity, based on different settings

Settings in the HCSC server runtime definition file	Operation when an exception occurs in an activity
<code>syserr-to-fault-convert=on</code>	Send a general fault message.
<code>syserr-to-fault-convert=off</code>	A system exception occurs.

(2) Validation activity

For the validation activity, you can set the `validate-fault-compatible` property to change the type of fault message that is to be sent if the result of validation processing is invalid.

The following table describes the difference in operations (based on different settings) when the result of validation processing is invalid.

Table 4–15: Difference in operation (based on different settings) when the result of validation processing is invalid

Settings in the HCSC server runtime definition file	Operation when the result of validation processing is invalid
<code>validate-fault-compatible=on</code>	Send the fault message specific to the validation activity. [#]
<code>validate-fault-compatible=off</code>	Send a general fault message. [#]

#

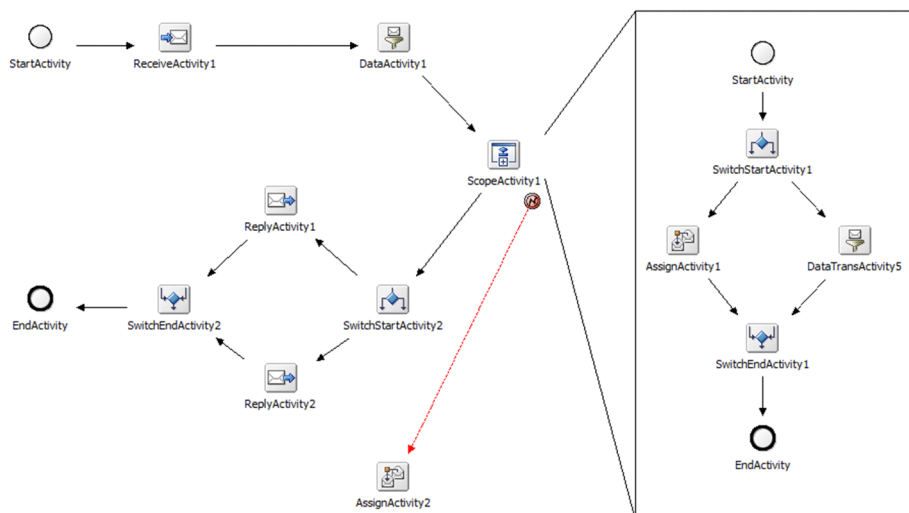
A fault message is sent only when `validation-activity` is set to ON in the HCSC server runtime definition file. If `validation-activity` is set to OFF, a fault message is not sent because validation processing itself is not performed.

4.7.2 Flow of identifying the location of a system exception by using a general fault

If a system exception occurs in an activity, or if the result of validation processing by the validation activity is invalid, the exception can be converted to a general fault message, and then sent. Based on the information in the fault message, you can reference the KDEC20087-W message (used to identify error locations) output in the message log to investigate the location in which an error occurred and the cause of the error.

The following shows an example of a business process definition that catches a general fault in the scope.

Figure 4-7: Example of a business process definition that catches a general fault in the scope



The following describes how to use a general fault to identify the location in which a system exception occurred.

1. In ScopeActivity 1, define an activity that sends general faults.
If an exception occurs in the activity, a general fault is sent.
2. The fault handler defined in ScopeActivity 1 catches the sent general fault (the fault message is stored in a variable).
3. Use AssignActivity 2 to assign root application information from the general fault message to the response message.
You can use a switch activity to separately define a response for normal processing and for a fault.
4. Use ReplyActivity 2 to notify the requester (user) of the root application information acquired from the general fault message in step 3.
5. Based on the root application information acquired in step 4, the user checks the message log and references the KDEC20087-W message used to identify error locations.

Reference note

The user can investigate the location in which an error occurred and the cause of the error by referencing the following information:

- Information about the location in which an error occurred in a business process and the cause of the error contained in the KDEC20087-W message used to identify error locations
- Error messages output around the KDEC20087-W message (used to identify error locations) in the message log

4.7.3 Mapping between general fault messages and message log

Based on the root application information set in a general fault message, you can map the general fault message and the message log by referencing the KDEEC20087-W message (used to identify error locations) output in the message log.

For details about mapping between general fault messages and the message log, see *4.7.2 Flow of identifying the location of a system exception by using a general fault*.

If a fault occurs in an activity, you can output a user message trace. You can use the user message trace function to obtain more detailed information.

For details about the user message trace, see *7.4.4 User message trace* in the *Service Platform Setup and Operation Guide*.

4.8 Changing value of user message wherein UOC is used

Use component-common UOC for referencing or editing the request message at user end. Also, you can reference or update I/O data of reception and service adapter by using component-common UOC

4.8.1 Overview of component-common UOC

In component-common UOC, you can perform following two operations:

- Creating instances of UOC class
Reading java class created by the user in system class loader and creating instances of UOC class.
- Invoking UOC class
In the component specified by user, invoke java class created by the user.

(1) Creating instances of UOC class

Instances of UOC class are created when HCSC server is started and maintained in the memory.

Flow of generating instances of UOC class is as follows:

1. Reading component-common UOC property file

Read component-common UOC property file; create data of Map type which sets invoker ID of UOC class, key information, invoked class path of UOC class as value, perform settings for process of creating jar file.

If reading of a property file fails, stop the process of starting an HCSC server after an exception log (KDEC11003-E) is output to message log.

2. Reading jar file for component-common UOC and creating an interface

Acquire class path from Map type data which is created by reading a property for component-common UOC, generate an instance of UOC class.

Store instances of generated UOC class in Map type data that sets a class instance as value and a class path as key.

When reading of class file fails, stop the process of starting HCSC server after outputting exception log (KDEC11002-E) to message log.

(2) Invoking UOC class

You can invoke UOC class in all component processes, since this class is invoked from common process of reception and common process of service adapter.

The following table describes process timing of component and existence of invocation of UOC class and process timing of a component:

Table 4–16: Process timing of a component and existence of invocation of UOC class

Type		At the time of invocation	At the time of start up	At the time of completion			At the time of termination
				Normal	Fault	System exception	
Standard reception		A	Y	Y	A	A	A
User-defined reception	In SOAP reception	A	Y	Y	A	A	A
	In reception other than SOAP	Y	Y	Y	A	A	Y
Business process		A	A	A	A	A	A
Service adapter		A	Y	Y	A	A	A

Legend:

Y: UOC class is invoked.

A: UOC class is not invoked.

(a) Input information to UOC class

Summarize the input information to UOC class in object of Map type and pass it.

Acquire data based on constant number (key) defined in the interface for UOC.

When you are creating UOC class, define constant number (key) in the implemented interface.

The defined constant numbers are as follows:

Table 4–17: Defined constant number

Constant number name (Key name)	Data type	Setting contents
COMPO_TYPE	int	1: Service adapter 2: Reception
COMPO_KIND	String	In reception <ul style="list-style-type: none"> StandardWebService: Standard reception (Web service) StandardSessionBean: Standard reception (SessionBean) StandardWSR: Standard reception (MDB (WS-R)) StandardDBQ: Standard reception (MDB (Database queue)) WebService: SOAP reception TP1 and RPC: TP1 and RPC reception FTP: FTP reception HTTP: HTTP reception MessageQueue: Message Queue reception Custom: Custom reception In service adapter <ul style="list-style-type: none"> WebService: SOAP adapter SessionBean: SessionBean adapter MDB_WSR: MDB (WS-R) adapter MDB_DBQ: MDB (Database queue) adapter DB: DB adapter TP1: TP1 adapter File: File adapter ObjectAccess: Object Access adapter MessageQueue: Message Queue adapter FTP: FTP adapter FileOperation: File operations adapter Mail: Mail adapter HTTP: HTTP adapter Custom: Custom adapter
COMPO_ID	String	Reception ID of reception or service ID of service adapter
TELEGRAM_DATA	CSCMsgTelegramManager	Request message or response message

Constant number name (Key name)	Data type	Setting contents
MONITOR_SEND_DATA	Map<String, Object>	Sent data storage Map
DESTINATION_DATA	CSCMsgDestinationData	Address information of service to be invoked [#]

Note[#]

You can acquire only in case of user-defined reception (excluding SOAP reception).

Contents of the message acquired from parameter Map are as follows:

Message

Pass request message and response message to UOC class in API (CSCMsgTelegramManager) for acquiring a message. Acquire a message by using "#getHeader" or "#getBody"

Note that since the message is passed in byte array, you must convert Document type in case of an XML message.

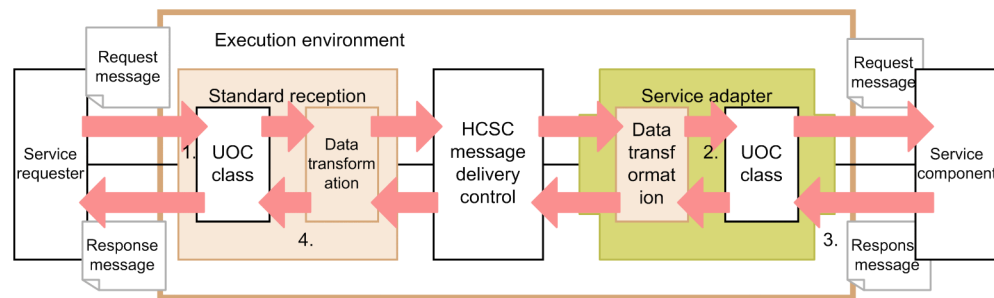
Address information

You can acquire address information of the service to be invoked in case of user-defined reception except for SOAP user-defined reception. Also, at the time of request reception of user-defined reception, use API for acquisition of address information of service. You can change the service to be invoked by setting any value.


(b) Timing of invoking UOC class

Timing of invoking UOC class and its relation with the message is as follows:

Figure 4-8: UOC Timing of invoking class and its relation with message



Legend:

 : Flow of service component invocation request and response

1. In reception, pass the request message from service requester to UOC class
2. In service adapter, pass the request message to be passed to a service component to UOC class.
3. In service adapter, pass the response message returning from service component to UOC class.
4. In reception, pass a response message from service adapter to UOC class.

Note

When address of service to be invoked is changed, there are cases when the address output to request trace differs from the real address.

(c) Output of performance analysis trace

If you use component-common UOC, performance analysis trace is definitely output. Timing of performance analysis trace is before and after invocation of UOC class.

Format which is output to performance analysis trace file is same as performance analysis trace of J2EE server. For details on performance analysis trace file, see "7.3 Acquiring performance analysis trace file wherein Management Server is used" in "Application Server Maintenance and Migration Guide".

4.8.2 Usage example of component-common UOC

An example of converting and updating value of request message in reception using component-common UOC is as follows:

The component used in this example is as follows:

- User-defined reception name: Sample user-defined reception 2
- Reception ID: rcp2

For details on association of component and component-common UOC class, see "Appendix H Componentcommon UOC "" in "Service Platform Basic Development Guide".

(1) Prerequisites

Operating environment which serves as a prerequisite is as follows:

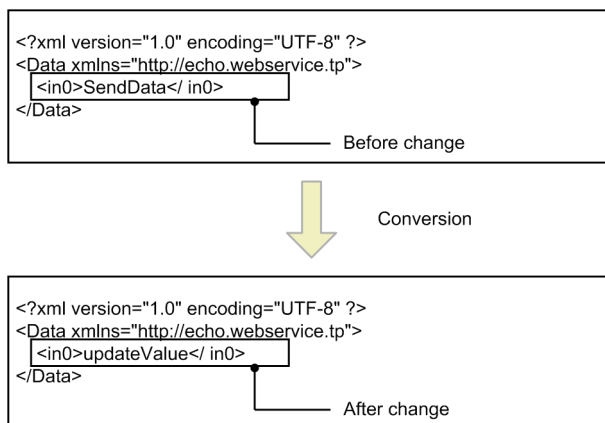
- J2EE server name: J2EEServer
- HCSC server name: HCSC
- SOAP mode: SOAP 1.1 mode
- Installation directory of the service platform: "C:\Program Files\Hitachi\Cosminexus"
- jar file storage directory: C:\uocjar
- Property storage directory: C:\uocconf

(2) Contents of process

In this example, convert a part of request message and it becomes a process called update.

Process contents of usage example are as follows:

Figure 4–9: Process contents of usage example



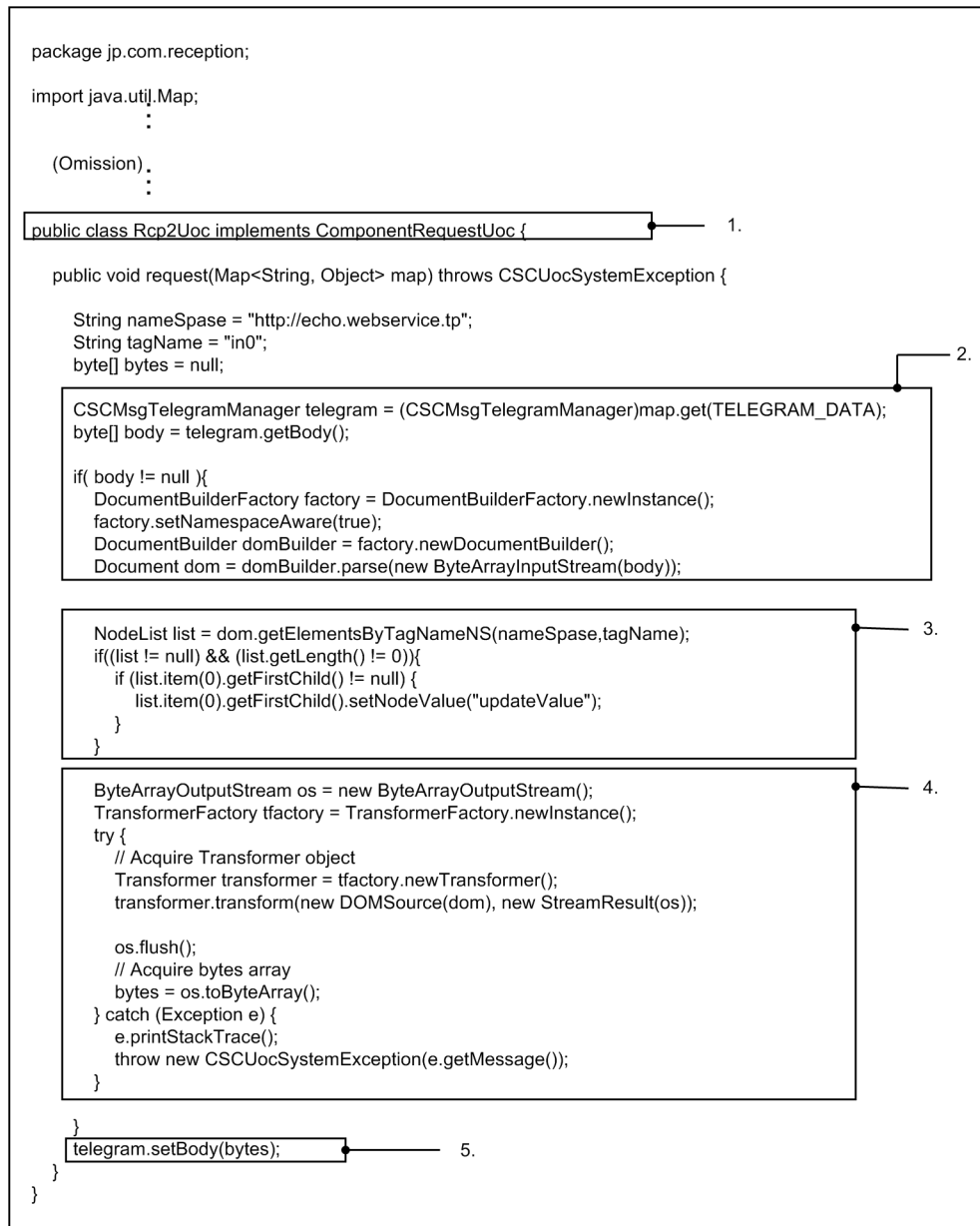
(3) Creating UOC class

When you are creating UOC class which changes data of a request message, you must combine the process in the following manner:

1. Implement an interface for UOC class which supports a request.
2. Acquire a message from CSCMsgTelegramManager object of a parameter.
3. Analyze the message and update.
Conversion of Document is required, since the message is acquired in byte array.
4. Convert the message in byte array.
5. Set the converted byte array in the method for updating messages of message acquisition API.

Example of creating UOC class is as follows:

Figure 4–10: Example of creating UOC class



Note#

Numbers within the figure correspond to steps from 1 to 5 of the process required for combination.

(4) Settings of an HCSC server

(a) Preparation of jar file

Create jar file using build tool such as Ant, store it in jar file storage directory (In this example, C:\uocjar)

(b) Preparation of property file

Contents of settings of the property file for component-common UOC are as follows:

```
rcp2=jp.com.reception.Rcp2Uoc
```

In HCSC server runtime definition file, add the definition in the following manner:

```
common-uoc-prop-path=C:\\uocconf
```

(c) Settings of option definition file (usrconf.cfg) for J2EE server

In usrconf.cfg file, set property value in the following manner:

```
add.class.path=C:\\uocjar\\uocClass2.jar
```

5

Performance Upgrade Function

This manual describes multiplicity of the service component invocation process, the cache function of XML messages, the pre-cache function of business processes, and the XML parser pool function of the data transformation process as a performance upgrade function.

5.1 Multiplicity of the service component invocation process

The requests for invoking service components from service requesters might be executed simultaneously. Therefore, you must set up multiplicity (simultaneously executable count) for the service component invocation process.

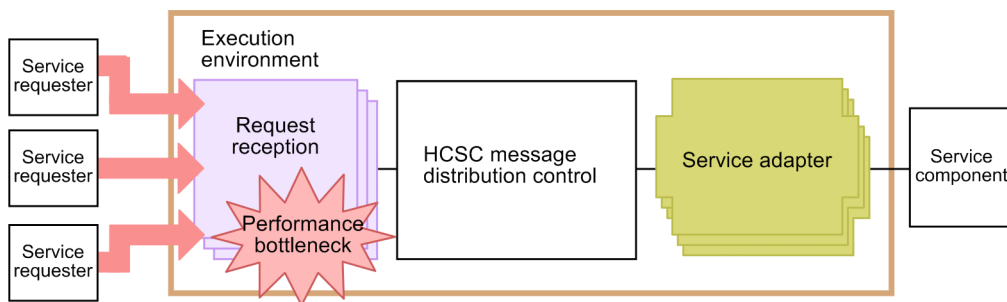
5.1.1 Multiplicity of the HCSC server

If the multiplicity is incorrectly set, a performance bottleneck might occur.

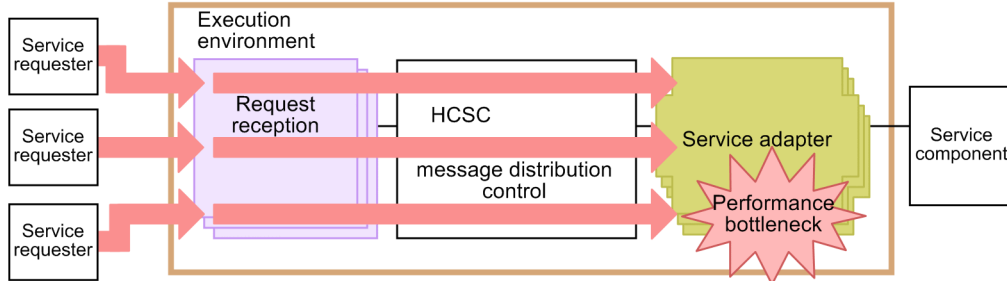
The following figure shows an example of a bottleneck caused by the multiplicity.

Figure 5–1: Example of a bottleneck caused by the multiplicity


- When multiplicity of request reception is less:



- When multiplicity of service adapter is less:



Legend:

 : Flow of service component invocation requests

As shown in *Figure 5-1 Example of a bottleneck caused by the multiplicity*, when a small value is set for the multiplicity for request reception (standard reception or user-defined reception), even if a large number of requests are received, they cannot be processed at the same time. For this reason, the number of processes that can be executed at the same time decreases. Furthermore, even when a higher value is set for the multiplicity for request reception, if the multiplicity defined by the service adapter is small, the service adapter will not be able to execute many processes at the same time. As a result, the service adapter causes a bottleneck and optimum performance will not be achieved.

(1) Multiplicity of the HCSC server

For synchronous (Web Services) standard reception, multiplicity is set in terms of the number of threads and the maximum number of simultaneous executions. For synchronous (Session Bean) standard reception, multiplicity is set in terms of the number of instances. For details about the multiplicity settings of the HCSC server, see *Table 5-1 Properties for setting the multiplicity of standard reception*, *Table 5-2 Property for setting the multiplicity of user-defined reception*, and *Table 5-3 Item for setting the multiplicity of a service adapter*.

Table 5–1: Properties for setting the multiplicity of standard reception

Definition file	Property	Description
HCSC server runtime definition file	<code>request-ejb.instance.minimum</code>	Minimum number of instances for synchronous (Session Bean) standard reception
	<code>request-ejb.instance.maximum</code>	Maximum number of instances for synchronous (Session Bean) standard reception
	<code>request-ejb.parallel.count</code>	Number of threads to be provided for CTM to call applications
	<code>request-soap.instance.minimum</code>	Minimum number of simultaneous executions for synchronous (Web Services or SOAP1.1) standard reception
	<code>request-soap.instance.maximum</code>	Maximum number of simultaneous executions for synchronous (Web Services or SOAP1.1) standard reception
	<code>request-soap1_2.instance.minimum</code>	Minimum number of simultaneous executions for synchronous (Web Services or SOAP1.2) standard reception
	<code>request-soap1_2.instance.maximum</code>	Maximum number of simultaneous executions for synchronous (Web Services or SOAP1.2) standard reception
	<code>request-jms.instance.maximum</code>	Maximum number of instances for asynchronous (MDB (WS-R)) standard reception

Table 5–2: Property for setting the multiplicity of user-defined reception

Definition file	Property	Description
User-defined reception runtime definition file	<code>user-defined-reception-soap.threads.maximum</code>	Maximum number of simultaneous executions

Table 5–3: Item for setting the multiplicity of a service adapter

Window	Description
Service Adapter Settings window in the development environment	Maximum number of instances

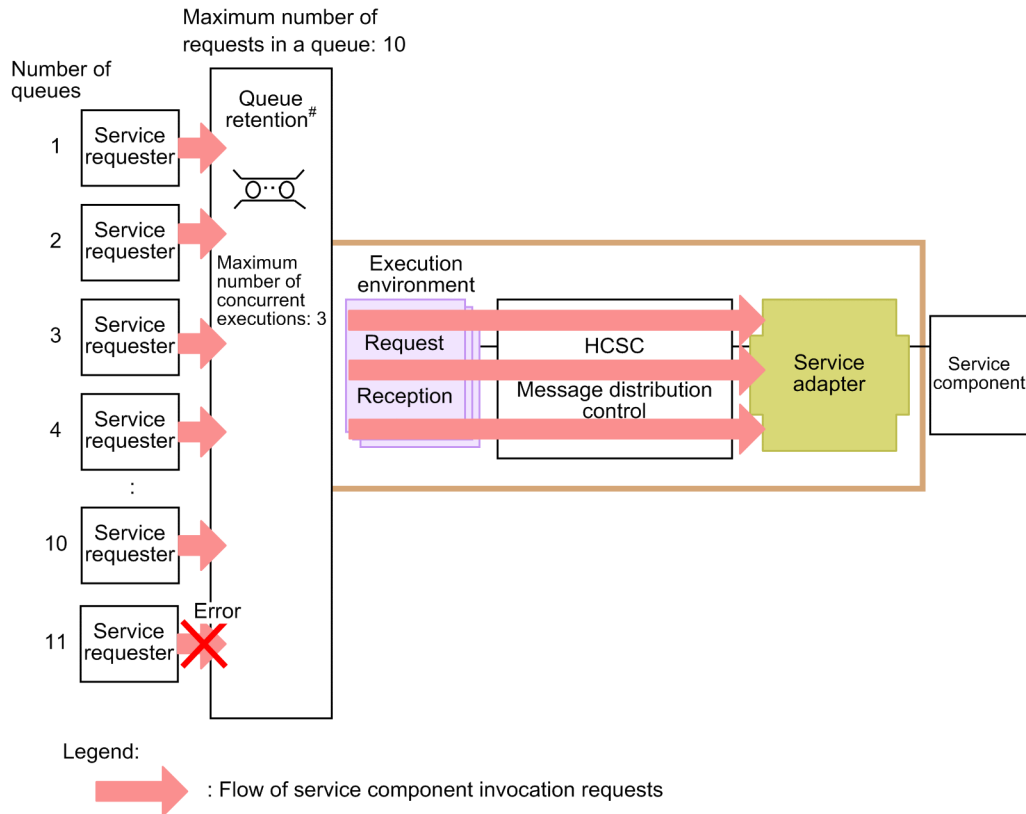
#

If the maximum number of instances is set to 0, no upper limit on the number of instances will be set.

(2) Multiplicity of Web Services (SOAP communication)

For Web Services (SOAP communication), in addition to the number of simultaneous executions, you can set a value for the maximum number of requests in a queue at the Web container level. If you specify the maximum number of simultaneous executions and the maximum number of requests in a queue, you can control the flow when receiving requests from the service requester. This is because you can make a process wait without executing the actual process. The following figure shows the relationship between the requests from a service requester and the maximum number of requests in a queue.

Figure 5–2: Relationship between the requests from a service requester and the maximum number of requests in a queue



#: Indicates messages in a queue waiting for execution in Web Services.

For example, *Figure 5-2 Relationship between the requests from a service requester and the maximum number of requests in a queue* indicates that the maximum number of simultaneous executions is set to 3 and the maximum number of requests in a queue is set to 10. In this case, of the requests that are received simultaneously, no more than three service component invocation requests can be processed simultaneously. If service component invocation requests exceeding the maximum number of simultaneous executions (four or more requests) are received, a number of requests not exceeding the maximum number of requests that can be held in the queue (10 requests) are placed in the pending state. Then, processes will be executed sequentially as soon as the previous process being executed ends.

When the number of requests in the queue reaches the upper limit (10 requests), new requests from the service requester cannot be accepted. Therefore, an error response will be sent to the service requester.

The maximum number of requests that can be held in the queue of the HCSC server is configured by using the properties of the definition files described in *Table 5-4 Properties for configuring the maximum number of requests in the queue for standard reception* and *Table 5-5 Properties for configuring the maximum number of requests in the queue for user-defined reception*.

Table 5–4: Properties for configuring the maximum number of requests in a queue for standard reception

Definition file	Property	Description
HCSC server runtime definition file	<code>request-soap.queue-size</code>	Size of the pending queue for synchronous (Web Services or SOAP1.1) standard reception
	<code>request-soap1_2.queue-size</code>	Size of the pending queue for synchronous (Web Services or SOAP1.2) standard reception

Table 5-5: Properties for configuring the maximum number of requests in a queue for user-defined reception

Definition file	Property	Description
User-defined reception runtime definition file	user-defined-reception-soap.queue-size	Size of the pending queue

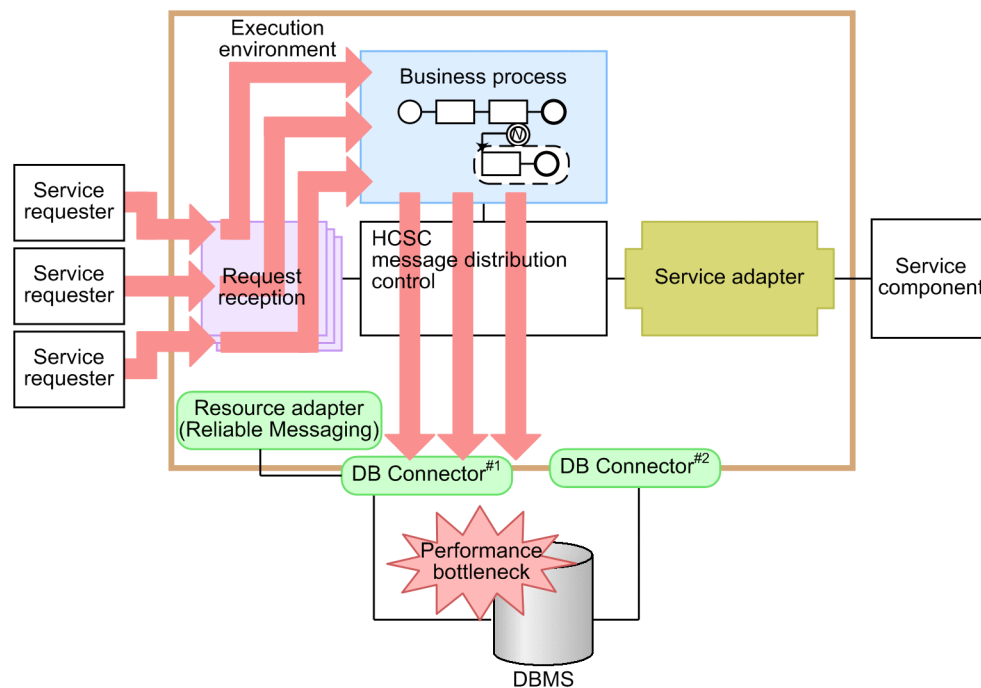
5.1.2 Multiplicity related to database access

When using a persistent business process (acquiring the execution log of process instances) or when using an asynchronous protocol, set the connection pool count in the DB Connector definition specified on the HCSC server.


(1) When using business processes that are to be made persistent


Of the two DB Connectors that are set up on the HCSC server, the DB Connector specified in the `dbcon-xadisdisplayname` property of the HCSC server setup definition file is used. If numerous requests are simultaneously sent from request reception via the business process, the HCSC server can not output all data to Database at one time because of performance bottleneck at DB connector. In such a case, a performance bottleneck occurs, causing performance degradation in the simultaneous execution of the business process. Therefore, you must set the multiplicity when collecting the execution logs of process instances. The following figure shows the occurrence of a bottleneck when a business process to be made persistent is used.

Figure 5-3: Occurrence of a bottleneck when a persistent business process is used



Legend:

 : Flow of service component invocation requests

 : Fault connections

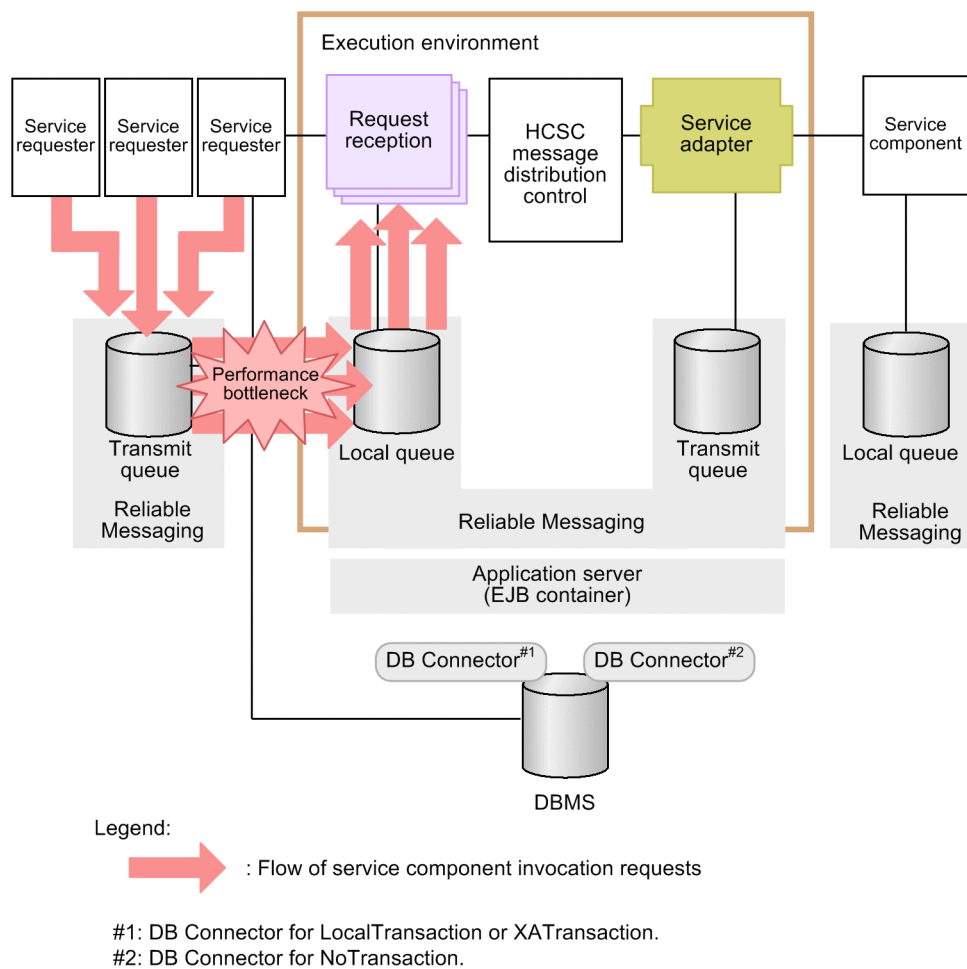
#1: DB Connector for LocalTransaction or XATransaction specified in the `dbcon-xadisdisplayname` property
 #2: DB Connector for NoTransaction or XATransaction specified in the `dbcon-noisdisplayname` property

(2) When using an asynchronous protocol

Of the two DB Connectors that are set up on the HCSC server, the DB Connector specified in the `dbconnectdisplayname` property of the HCSC server setup definition file is used to extract messages from a queue in asynchronous (MDB (WS-R) or MDB (DB Queue)) standard reception. This DB Connector is also used to send messages for an asynchronous (MDB (WS-R) or MDB (DB queue)) service adapter. Therefore, you must examine the connection pool count in addition to the multiplicity when the execution log of process instances is acquired, when the HCSC server extracts messages from the queue for request reception, and when the service adapter sends messages to the transmit queue.

If numerous requests are simultaneously received by a queue, the resource adapter of Reliable Messaging cannot extract the messages simultaneously, causing a performance bottleneck. Furthermore, even when the multiplicity of the service adapter is increased, if the number of simultaneous executions is small when the service adapter sends messages to a queue, a performance bottleneck also occurs. The following figure shows the occurrence of a bottleneck when an asynchronous protocol is used.

Figure 5-4: Occurrence of a bottleneck when an asynchronous protocol is used



(3) Setting the multiplicity related to database access

The multiplicity related to database access is set as shown in the following table.

Table 5-6: Properties for setting the multiplicity of a DB Connector

Definition file	Property	Description
Property file configured during setup of a DB Connector (LocalTransaction or XATransaction)	<code>MinPoolSize</code>	Minimum pool size
	<code>MaxPoolSize</code>	Maximum pool size

#1

Note that, when the maximum pool size has been increased, the number of simultaneous connections in the database must also be changed. (For HiRDB, this is `pd_max_users`, which is maximum number of connections of HiRDB.)

#2

For details about the maximum number of database connections used in the execution environment, see 3.1.2 *Setting up the software required for the execution environment* in the *Service Platform Setup and Operation Guide*.

Table 5–7: Properties for setting the multiplicity of Reliable Messaging

Definition file	Property	Description
Property file configured during setup of Reliable Messaging	MinPoolSize	Minimum pool size
	MaxPoolSize	Maximum pool size

5.1.3 Multiplicity related to Web Services (SOAP communication)

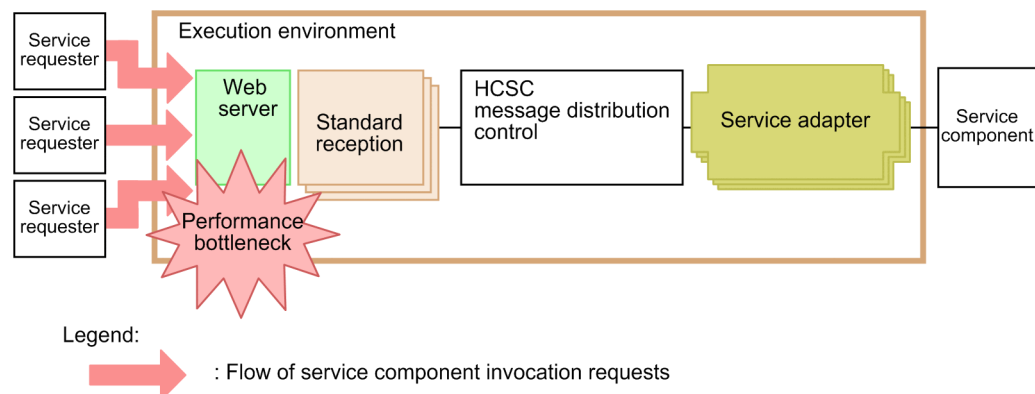
(1) Number of simultaneously executed threads

When you use Web Services (SOAP communication), the number of simultaneously executed threads for the Web server (HTTP server) set up in the execution environment is also related.

If there are multiple standard (Web Services) receptions and SOAP receptions, the total number of simultaneously executed threads specified on the Web server has priority over the sum of the number of simultaneously executed threads specified for each reception. Therefore, if the total number of simultaneously executed threads (on the Web server) is smaller than the sum of the number of simultaneously executed threads specified for each reception, a performance bottleneck might occur. This is because the numerous requests that are received at one time cannot be processed simultaneously.

The following figure shows the bottleneck when Web Services (SOAP communication) is used.

Figure 5–5: Bottleneck when Web Services (SOAP communication) is used



The multiplicity for Web Services (SOAP communication) is set as follows:

- For HTTP Server
Of the directives that can be specified in the configuration file, define the directive related to Web server performance.
For details, see 6.1.1 *List of directives* in the manual *HTTP Server User Guide*.
- For the in-process HTTP server
From the management portal, select **Logical Server Environment Setup, J2EE Server, Logical Server Names of J2EE Server**, and then **HTTP Server**. In the window that appears, specify the following information:
 - Number of simultaneously executed threads
 - Number of initial threads

For details, see *Chapter 5. In-process HTTP Server* in the *Application Server Web Container Functionality Guide*.

(2) Number of dedicated threads

Even if a sufficient number of simultaneously executed threads of the Web server (HTTP server) cannot be provided, you might want to perform the minimum number of executions in each request reception. In this case, define the number of dedicated threads.

With this definition, if requests exceeding the number of simultaneously executed threads of the Web server (HTTP server) are received in an individual request reception, threads for at least the specified number of dedicated threads are executed simultaneously.

If no requests are received in other request receptions, threads can be simultaneously executed up to the maximum number of simultaneous executions.

The number of dedicated threads in an HCSC server is set as shown in *Table 5-8 Properties for setting the number of dedicated threads for standard reception* and *Table 5-9 Property for setting the number of dedicated threads for user-defined reception*.

Table 5–8: Properties for setting the number of dedicated threads for standard reception

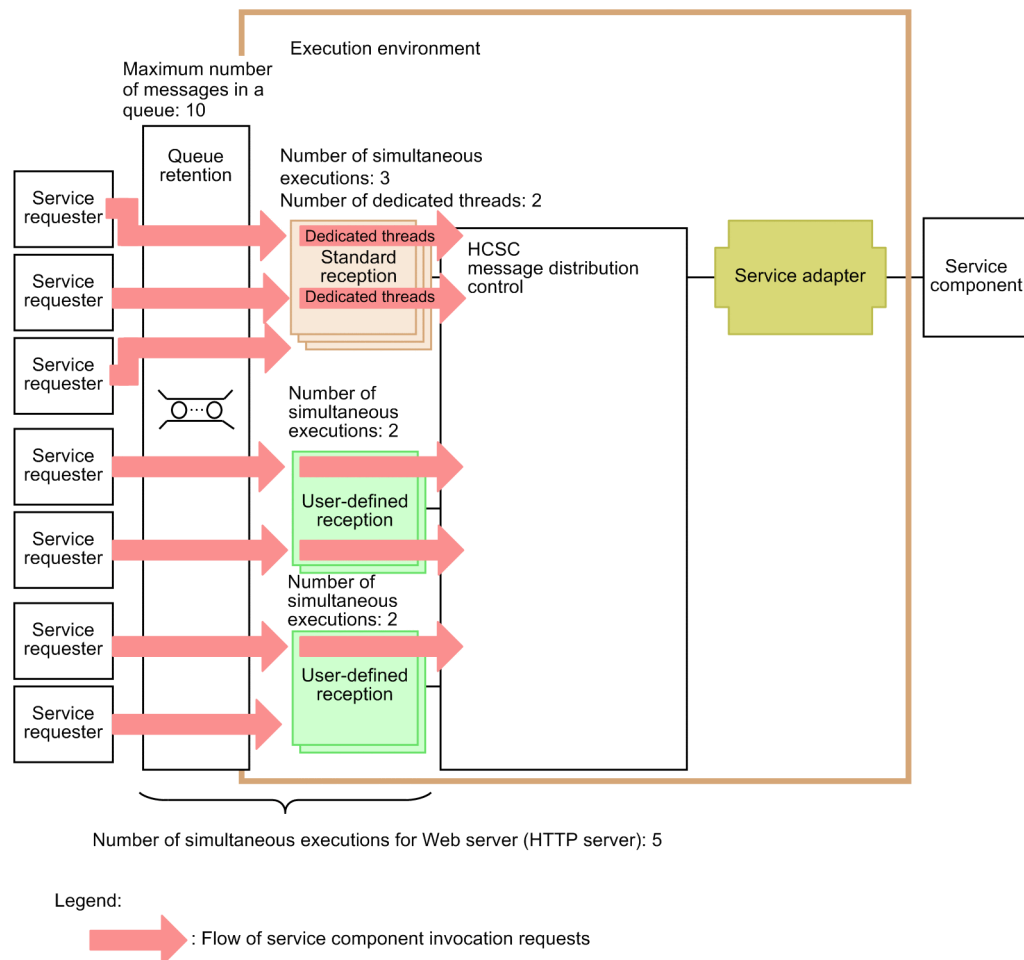
Definition file	Property	Description
HCSC server runtime definition file	<code>request-soap.exclusive.threads</code>	Number of dedicated threads for synchronous (Web Services or SOAP1.1) standard reception
	<code>request-soap1_2.exclusive.threads</code>	Number of dedicated threads for synchronous (Web Services or SOAP1.2) standard reception

Table 5–9: Property for setting the number of dedicated threads for user-defined reception

Definition file	Property	Description
User-defined reception runtime definition file	<code>user-defined-reception-soap.exclusive.threads</code>	Number of dedicated threads

The following figure shows the relationship between the maximum number of simultaneous executions and the number of dedicated threads.

Figure 5-6: Relationship between the maximum number of simultaneous executions and the number of dedicated threads



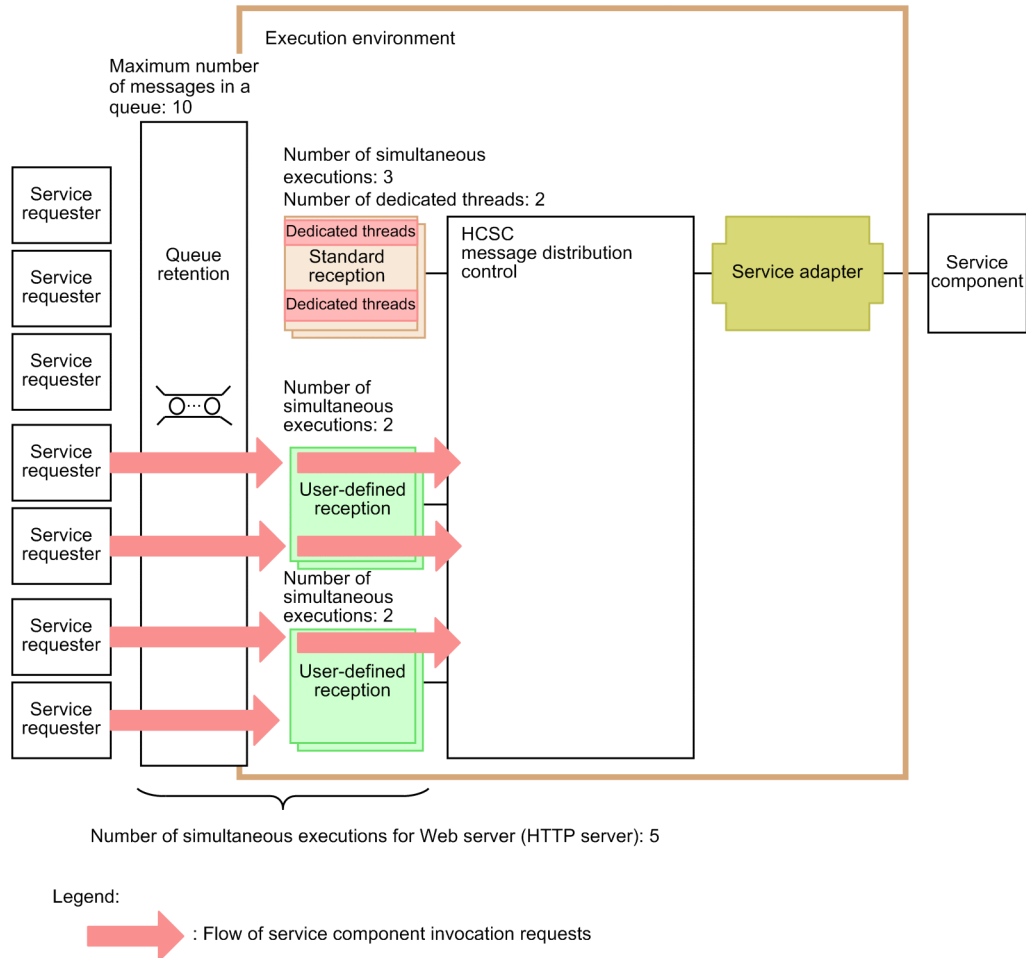
As shown in Figure 5-6, standard reception always occupies two threads when standard reception, two user-defined receptions, and the Web server (HTTP server) are set up as follows:

- Standard reception: The maximum number of requests in a queue is 10, the number of simultaneous executions is 3, and the number of dedicated threads is 2.
- User-defined reception: The number of simultaneous executions is 2.
- Web server (HTTP server): The number of simultaneous executions is 5.

Thus, if more than five requests (the number of simultaneous executions of the Web server (HTTP server)) are received, two threads will be used to execute the standard reception. The remaining three threads will be used to execute the user-defined receptions.

Suppose the maximum number of requests in a queue, the number of simultaneous executions of standard reception, the number of dedicated threads, and the number of simultaneous executions for the Web server (HTTP server) are set as shown in Figure 5-6 Relationship between the maximum number of simultaneous executions and the number of dedicated threads. In this case, even if no requests are received in the standard reception for which the number of dedicated threads is specified (see Figure 5-7 When no requests are received in reception for which the number of dedicated threads is specified), standard reception always occupies two threads. Furthermore, because the number of simultaneous executions is two for each user-defined reception, the total number of simultaneous executions must be four. However, because the number of simultaneous executions for the Web server (HTTP server) is five, only the remaining three threads can be used for the user-defined receptions. The number of simultaneous executions for other request receptions will not increase.

Figure 5–7: When no requests are received in the reception for which the number of dedicated threads is specified



5.1.4 Multiplicityrelated to XML analysis process

When you use multiple threads concurrently, you must generate instances per thread since the class used in the processing of the request message and the response message cannot be shared with multiple threads.

If you generate instances per analysis, the performance degrades, since the instance generation process takes time.

If you use the instance pool function, you can save the instances generated in the instance pool that is created on memory. You can reuse instances by sharing the instances saved in the instance pool across the entire system, thereby improving the processing efficiency.

To save instances generated in the instance pool, perform settings as per the following table:

Table 5–10: Properties used for setting the number of instances cached in an instance pool

Definition file	Property	Contents
HCSC server runtime definition file	xml-pooled-instance-minimum	Minimum number of instances of a class used in the process of request message and response message to be cached
	xml-pooled-instance-maximum	Maximum number of instances of a class used in the process of request message and response message to be cached

When you are reusing already created instances, define the minimum number of caches. If you define the minimum number of caches, the pool is initialized when you start the HCSC server. Generate the instance of the defined value and set it in the pool.

If you set the maximum number of caches, you can save the instance till it reaches the defined value. Even if you reduce the instances by reusing them, you need not perform depletion, since an instance is to be generated and set in the pool.

For details on the `xml-pooled-instance-minimum` property and the `xml-pooled-instance-maximum` property, see "HCSC server runtime definition file " in "Service Platform Reference Guide".

5.2 Cache functionality for XML messages

(1) Overview of the cache functionality for XML messages

If you use the cache functionality for XML messages, processing speed might be improved when XML messages that meet the following conditions are sent or received by the HCSC server:

- The order of the elements is the same.
- The order of the attributes is the same.
- The looping of the elements is the same.

Once an XML document has been parsed, its characteristics are maintained. As a result, parse processing speed can be improved for similar documents.

(2) Setting up the cache functionality for XML messages

To use the cache functionality for XML messages, set a value for the `xmltelegram-maxcache-num` property of the HCSC server runtime definition file.

For details about the `xmltelegram-maxcache-num` property, see *HCSC Server Runtime definition file* in the *Service Platform Reference Guide*.

5.3 Pre-cache function of a business process

Analyze the format definition and the transformation definition before processing a request in a business process by using the pre-cache function. You can cache the analysis result on the HCSC server. The request processing speed is improved since the cached definition is retained till the HCSC server or component stops.

Use the `cscprecache` command for using the pre-cache function. For details on the `cscprecache` command, see "cscprecache (Pre-cache of format definition and data transformation definition)" in "Service Platform Reference Guide".

5.4 XML parser pool function of the data transformation process

In the data transformation process, analyze XML with the XML parser of Java in the following manner:

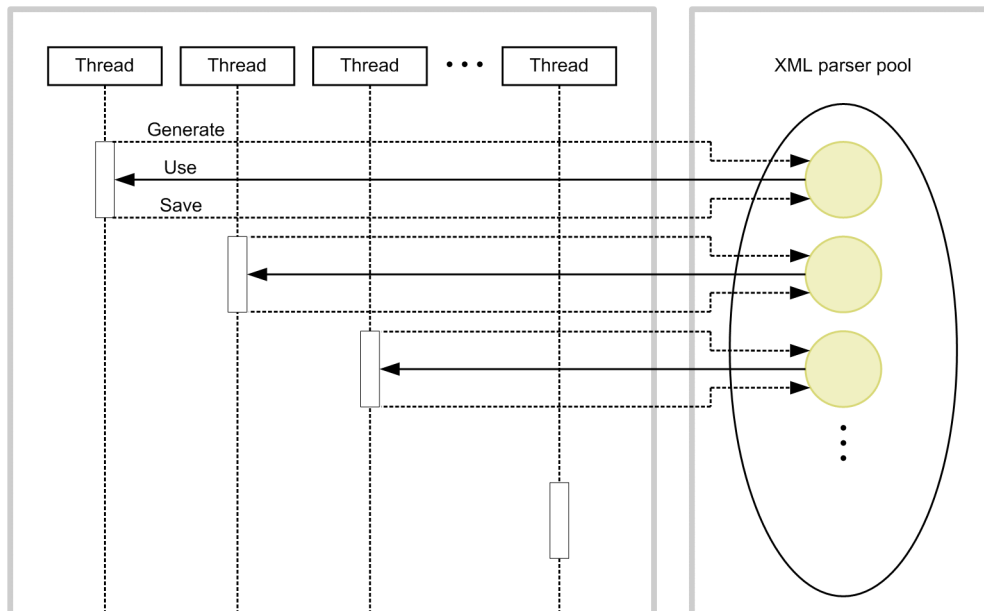
- DOM parser (javax.xml.DocumentBuilder class)
- SAX parser (org.xml.sax.XMLReader class)

When you are using multiple threads at the same time, XML parser is generated per thread since you cannot share instances of these XML parsers with multiple threads. If you generate XML parsers per analysis, the performance degrades, since the XML parser generation process takes time.


If you use the XML parser pool function, you can save the XML parser generated in the XML parser pool which is created on memory. You can reuse XML parsers by sharing the XML parsers saved in the XML parser pool across the entire system, thereby improving the processing efficiency.

The following figure shows the process flow of saving the generated XML parser in the XML parser pool which is created on memory:

Figure 5–8: Process flow of saving the generated XML parser in the XML parser pool which is created on memory

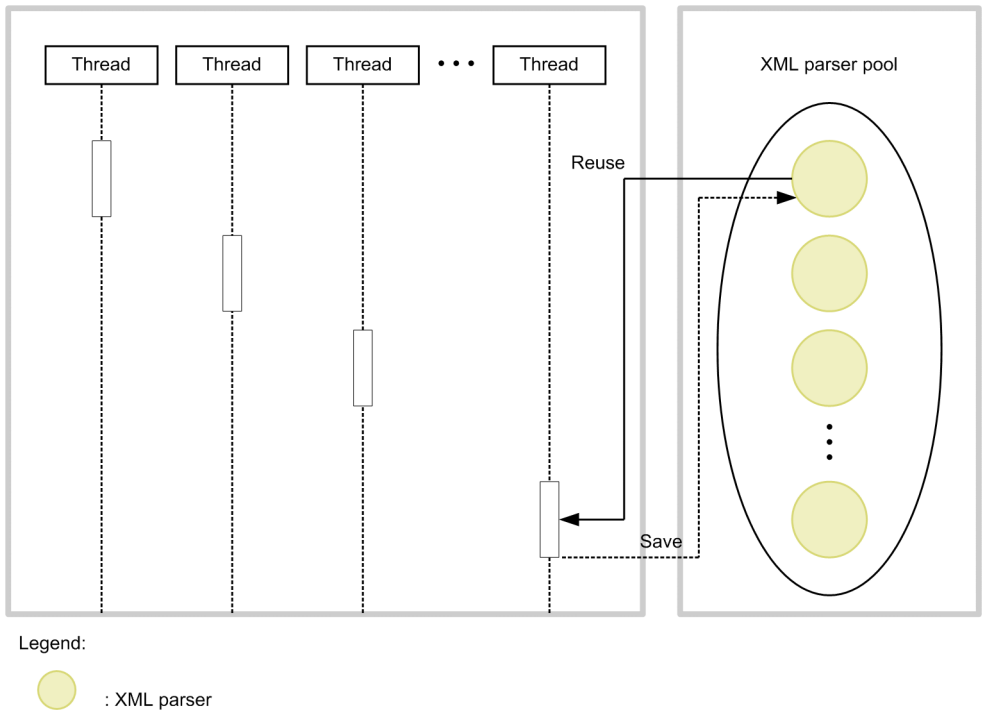


Legend:

 : XML parser

The following figure shows the process of reusing the XML parser saved in the XML parser pool:

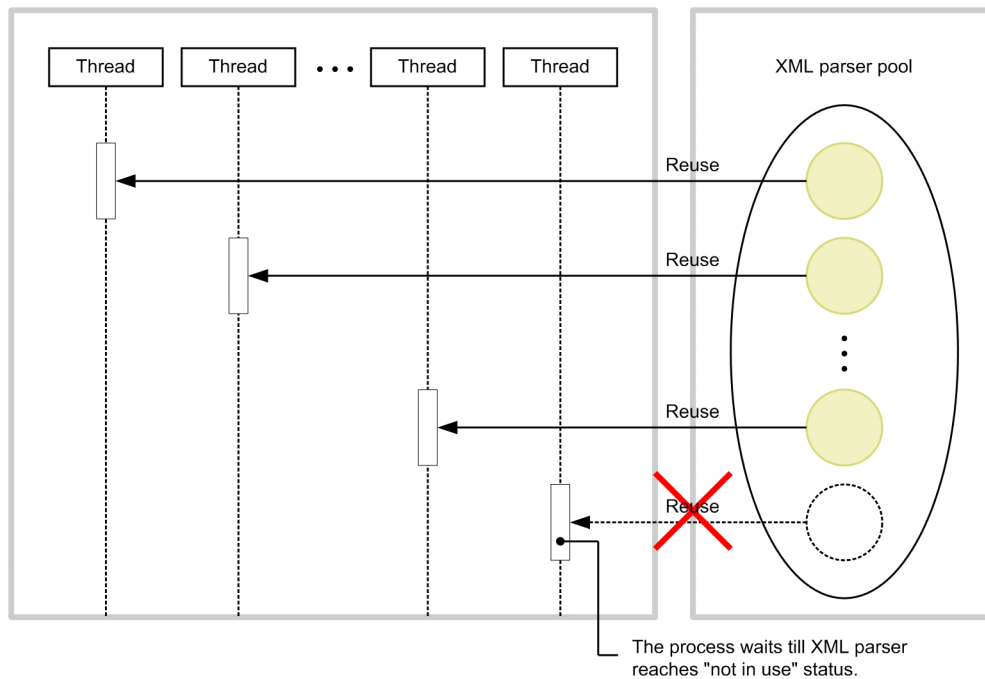
Figure 5-9: Process of reusing the XML parser saved in the XML parser pool




The maximum number of XML parsers that can be saved in the XML parser pool is 2,147,483,647 which is the maximum number of pools specified in option. When the number of threads that simultaneously use the XML parser exceed the maximum number of pools, stop the process till the XML parser is being used by other threads. When XML parser is no more in use, continue with the subsequent process.

The following figure shows the process to be performed when the numbers of threads simultaneously used by the XML parser exceed the maximum number of pools:

Figure 5–10: Process to be performed when the number of threads simultaneously using the XML parser exceeds the maximum number of pools



Legend:

 : XML parser

For the setup method for using the XML parser pool function, see "2.3.10 Using XML parser pool functionality option" in "Service Platform System Setup and Operation Guide".

6

Functionality for Failure Checking

This chapter describes the functionality for checking failures that occur in Cosminexus Service Platform.

6.1 Functionality for acquiring failure information, failure recovery, and failure analysis

6.1.1 Functionality for acquiring failure information

In the operating environment of Cosminexus Service Platform, the functionality that stores the trace information is provided for failures that occur in the operating and execution environments. The trace information is saved in the following files:

- Message log file
- Request trace file
- Performance analysis trace
- Maintenance information (exception log, maintenance log, and method trace)

You can check this trace information to learn about the time of failure occurrence, location, and causes.

Also, in windows of the development environment and the operating environment of the Cosminexus Service Platform, the messages, output because of the automatic checking being performed during the window operations, are displayed on the console view. You can check the contents of messages displayed on the console view, and then change all the definitions and settings for re-executing the operations.

6.1.2 Troubleshooting functionality

You can also check the information using commands, take a backup of files, and restore the environment for the failures that occur during the following operations:

- When setting up or unsetting up a system
- When executing an operation
- When invoking a service component

6.1.3 Failure analysis functionality

The Cosminexus Service Platform provides the *message tracking functionality* that assists in failure analysis by using the performance analysis trace file. *Message tracking* is a failure analysis tool that identifies the location of failure occurrence, and assists in differentiating and investigating the failure. When you start the message tracking, a special interactive prompt will appear. You execute a command in the displayed interactive prompt, and analyze the performance analysis trace file.

You can use the message tracking functionality anywhere on a machine on which uCosminexus Service Architect, uCosminexus Service Platform, or uCosminexus Operator for Service Platform is installed, if the performance analysis trace file is available when a failure occurs. Therefore, you can analyze the failure not only on the machine on which the erroneous HCSC server is running, but also in any remote environment other than the erroneous HCSC server.

6.2 Data validation functionality

6.2.1 Overview of the data validation functionality

For service adapters, you can verify that a request message and response message are valid for the defined format definition. If you use the data transformation functionality, you can also verify that the message before and after the transformation is valid for the defined format definition.

The data validation functionality verifies that the type of message sent during a request and response is valid for the type of format definition defined in the service adapter. If the message is not valid for the type of format, the data validation functionality returns an error before executing the service.

Because you can validate a message before executing a service in the test environment, you can prevent failures that would occur in the execution of the service.

Important note

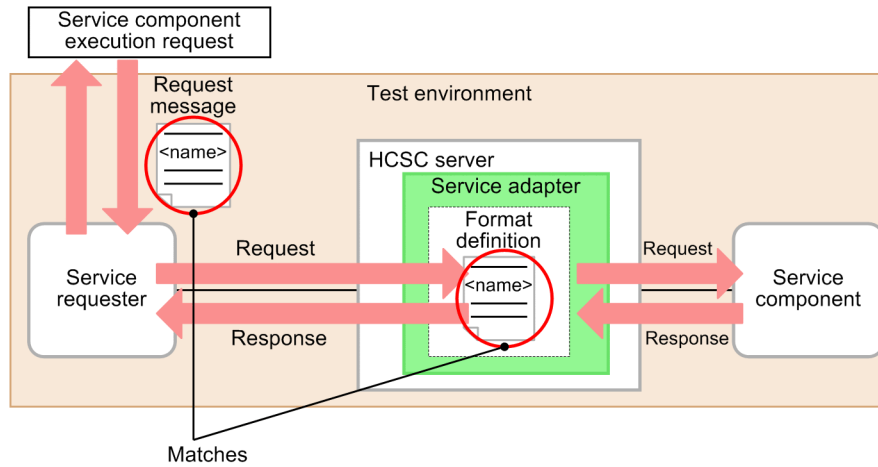
If the data validation functionality is enabled, the functionality checks whether the message received from a service requester or service matches the message format specified when the service adapter was defined. This check is not performed if the data validation functionality is disabled.

If the HCSC server receives a message that does not match the message format while the data validation functionality is disabled, the operation cannot be guaranteed.

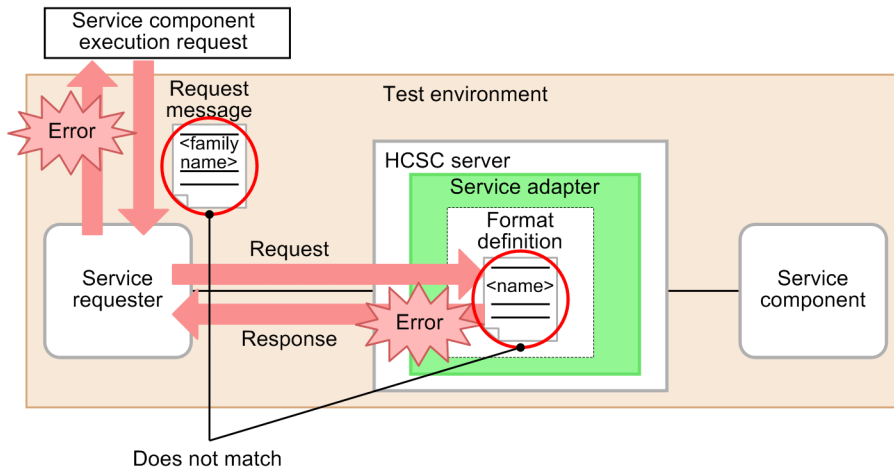
The following figure shows the flow of messages when the request messages and format definition match, and when they do not match.

Figure 6–1: Flow of messages when the request messages and format definition match, and when they do not match

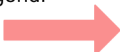
- When the request message matches the format definition:



- When the request message does not match the format definition:



Legend:

 : Flow of service component invocation requests

6.2.2 Setting up the data validation functionality

To use the data validation functionality, set a value in the `telegram-validation` property of the HCSC server runtime definition file.

For details about the `telegram-validation` property, see *HCSC Server Runtime definition file* in the *Service Platform Reference Guide*.

6.2.3 Scoping of data validation function

The following table describes scoping of data validation function: Note that, when the data validation function is enabled, the entire HCSC server becomes target for validation.

Table 6–1: Scoping of the data validation function

Type of service adapter	Message type	Request and response	Whether data transformation is performed or not in service adapter	Validation function is enabled or disabled	Validated message
SOAP adapter, SessionBean adapter, MDB (WS-R) adapter, MDB (Database queue) adapter	XML	Request	Data transformation is not performed.	Enable	Service component message
			Data transformation from the standard message? service component message is performed.	Enable	Standard message, service component message
		Response	Data transformation is not performed.	Enable	Service component message
			Data transformation from the service component message? standard message is performed.	Enable	Standard message, service component message
	Binary	Request	Data transformation is not performed.	Disable	-
			Data transformation other than requester? Standard message? service component message is performed.	Disable	-
		Response	Data transformation is not performed.	Disable	-
			Data transformation other than the service component message? Standard message? requester is performed.	Disable	-
Database adapter	XML	All	All	Disable	-

Legend:

?: Not applicable.

For details on a service adapter wherein the validation function of messages is enabled, see "7.7.15 Troubleshooting when there is a mistake in user message" in "Service Platform System Setup and Operation Guide."

6.3 User message trace functionality

6.3.1 Overview of the user message trace functionality

The *user message trace functionality* is used to output the contents of the messages flowing on the HCSC server to the user message trace file.

You can acquire the following items as user message traces:

- A service invocation request or response message received in the request reception
- A request or response message during the invocation of a business process
- A request or response message during the invocation of a service component from a service adapter
- A message before or after transformation when the data transformation is performed

By using the user message trace functionality, you can check that the messages flowing on the HCSC server are in accordance with the design. The smoothness of development will also improve because messages can be checked at each point (requester side and service side, BP activities, and data transformation).

6.3.2 Setting up the user message trace functionality

To use the user message trace functionality, set values in the following properties of the HCSC server runtime definition file:

- `telegramtrace` property
- `telegramtrace-filepath` property
- `telegramtrace-filesize` property
- `telegramtrace-filename` property
- `telegramtrace-trigger` property

For details about the properties, see *HCSC Server Runtime definition file* in the *Service Platform Reference Guide*.

If you specify the settings for acquiring a user message trace, the settings are enabled for the entire HCSC server.

6.3.3 Precautions on using the user message trace functionality

The user message trace functionality outputs the contents of a message to a file. Therefore, information contained in the message might be leaked, thus posing a security-related problem. To prevent such problems, do not use the user message trace functionality during normal operations, including actual operations.

In addition, be careful when handling the user message trace file that is output.

7

Functionality Used by Linking with Other Products

This chapter describes the operation management through linkage with JP1, databases that can be connected, and management of execution logs by linking with HiRDB.

7.1 Operation management through linkage with JP1

If you operate Service Platform linked with JP1, you can efficiently perform operations such as monitoring the overall business system and detecting problems. By automating the startup and termination of servers and applications, the efficiency of daily system operations can also be improved.

For details on system operations by linking with JP1, also see *12.2 Systems linked with JP1* in the *Application Server Operation, Monitoring, and Linkage Guide*.

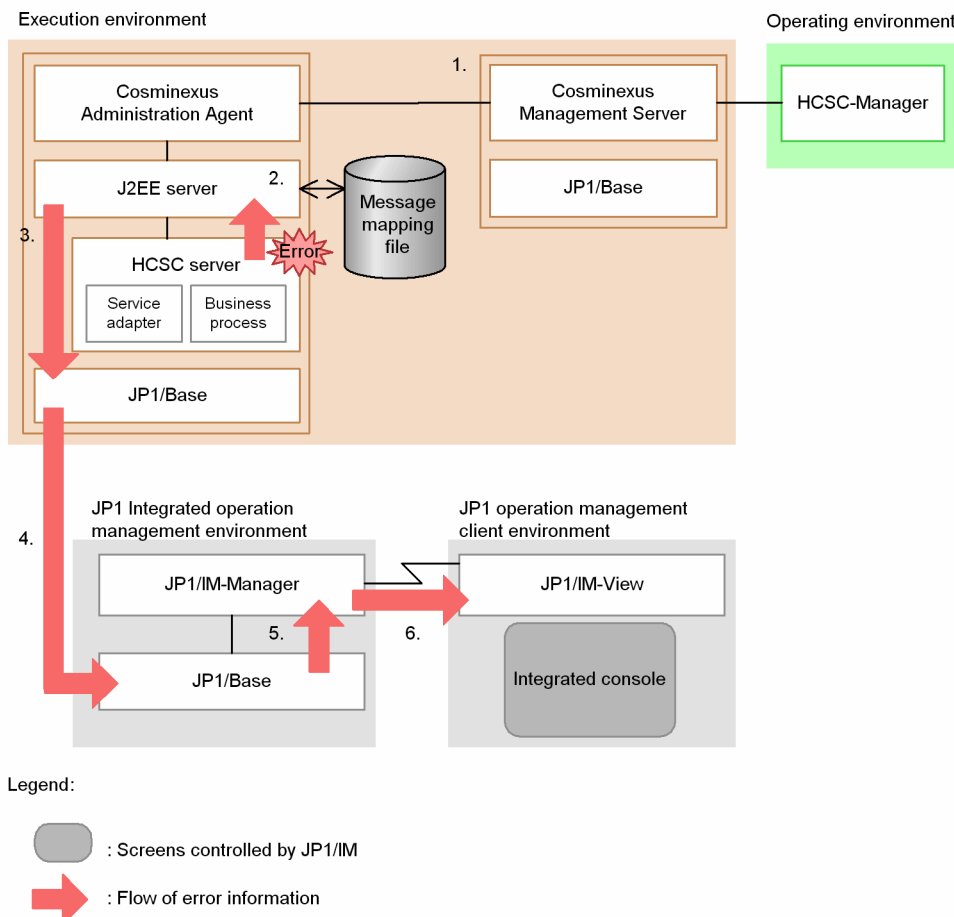
7.1.1 Detecting failures by linking with JP1/IM

With JP1 events, you can perform the focused management for failures that occur in systems. With the Cosminexus Service Platform, you can link a system with JP1/IM-Manager to detect failures, and can report failure messages to JP1 as JP1 events. In JP1/IM-View, you can also connect to the host of JP1/IM-Manager and display the JP1 events issued from the execution environment. You can customize the messages issued as JP1 events according to your environment.

If you in advance specify which message is to be reported for which failure, you can monitor the message log displayed in JP1/IM-View, and simplify the investigation of the type and cause of the failure.

An overview of failure monitoring by linking with JP1/IM is as follows:

Figure 7–1: Overview of failure monitoring by linking with JP1/IM



1. In the Cosminexus Management Server, you specify the settings for issuing JP1 events to be managed in JP1/IM.
2. A message, for a failure that occurs in the execution environment, converts to a JP1 event.
3. A message, for a failure that occurs in the system, will be issued as a JP1 event using JP1/Base.

4. JP1 events are transferred to the JP1 integrated operation management environment.
5. JP1/IM-Manager acquires the JP1 events transferred to the JP1 integrated operation management environment.
6. You can use JP1/IM-View from the integrated console, connect to JP1/IM-Manager, and then check the details of JP1 events.

7.1.2 Monitoring processes through JP1

This subsection describes the recommended processes when you monitor service platform systems by using JP1.

(1) Processes recommended for monitoring

The process for each logical server at the time of HCSC server startup is monitored by Manager. If failures such as process down or hang-up occur, the logical server restarts automatically. Therefore, it is recommended that you monitor the following processes at the time of monitoring service platform systems by using JP1:

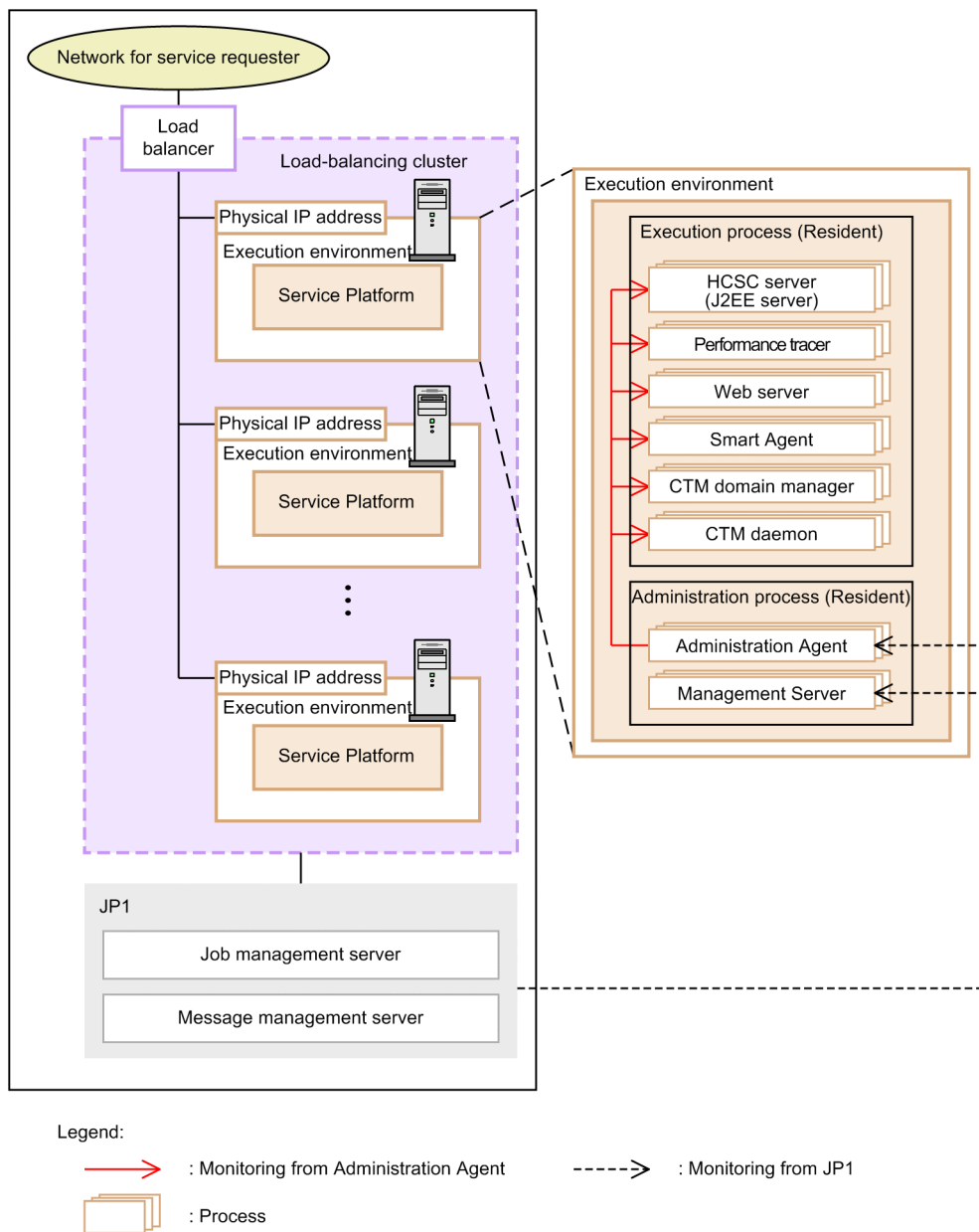
- Management Server
- Administration Agent

Tip

Among all processes of each logical server, if you monitor a process other than the processes recommended above by using JP1, you will perform double monitoring of the process.

The following figure shows the process configuration at the time of HCSC server startup:

Figure 7-2: Process configuration at the time of HCSC server startup



The following table lists names of the processes recommended for JP1 monitoring:

Table 7-1: Names of the processes recommended for monitoring from JP1

Process recommended for monitoring	Process name	
	For Windows	For UNIX
Management Server	mngsvr.exe	cjstartsv [#]
Administration Agent	adminagentsv.exe	adminagent

Note#

Monitors with the command line name (equivalent to ps -ef). An execution example is as follows:

/opt/Cosminexus/CC/server/bin/cjstartsv cosmi_m

"cosmi_m" is the default server name. When you change to a server name other than "cosmi_m", you must also change the command line name that performs the monitoring.

Also, when you use the database, it is recommended that you monitor the processes of the used database with JP1. For these process names, see the manual of each database.

(2) Points to be considered when monitoring the process

When you monitor number of processes with upper and lower limits, if you monitor with 1 as the upper and lower limit, the cjstartsv process and the adminagent process temporarily become bigger than 1, and the fault is detected. As recommended by Manager, perform monitoring without any upper limit, since Management Server and Administration Agent have a functionality to suppress multiple startups.

7.2 Databases that can be connected

This subsection describes databases that can be connected from the service platform, and JDBC drivers used for establishing a connection. The JDBC drivers that you use for establishing the connection differ according to the database. Furthermore, databases that can be connected and JDBC drivers to be used differ according to the functionalities that you use.

The following table describes the correspondence between databases that can be connected from the service platform, JDBC drivers, and the functionalities used for connection:

Table 7–2: Databases that can be connected from the service platform

Database	JDBC driver	Messaging function and business process function		DB adapter function	
		Local transaction	Global transaction	Local transaction	Global transaction
HiRDB Version 9	HiRDB Type4 JDBC Driver	A	A	A	A
HiRDB Version 8	HiRDB Type4 JDBC Driver	Y	Y	Y	Y
Oracle 11g R2 11.2	Oracle JDBC Thin Driver	Y	Y	Y	Y
Oracle 11g R1 11.1	Oracle JDBC Thin Driver	Y	Y	Y	Y
Oracle 12c 2.1	Oracle JDBC Thin Driver	Y	Y	Y	Y

Legend:

A: Can be used. Use is recommended.

Y: Can be used.

7.3 Management of execution logs by linking with HiRDB

If you manage execution logs by linking with HiRDB, you can improve efficiency when searching for and deleting execution logs.

7.3.1 Partitioning storage of execution logs

If HiRDB is used as the database to store a very large process instance execution log, you can use partitioning storage for the execution log to efficiently search and delete log data.

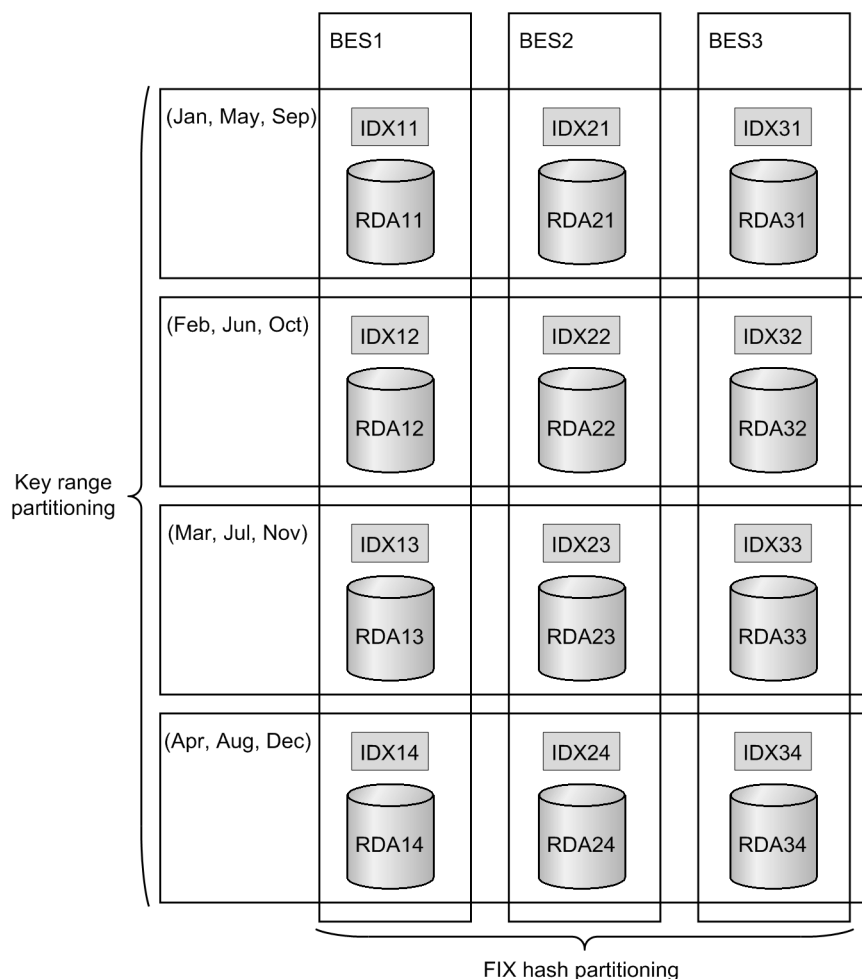
For Service Platform, you can use key range partitioning with boundary values specified using date data. You can also use matrix partitioning, in which partitioned data is further partitioned by FIX hash partitioning using process instance identifiers.

To use matrix partitioning, HiRDB Advanced High Availability is required.

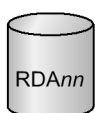
For details about key range partitioning and FIX hash partitioning, see the explanation of the types of table row partitioning in the *HiRDB Installation and Design Guide*.

The following figure shows an example of matrix partitioning using a combination of key range partitioning with boundary values (date data) specified and process instance identifiers.

Figure 7–3: Example of matrix partitioning using a combination of key range partitioning with boundary values (date data) specified and process instance identifiers



Legend: **IDXnn** : RDAREA for index



RDA_{nn} : RDAREA for data

(mm,mm,mm): Key range partitioning key

BESn: Back-end server name

(1) Key range partitioning with boundary values specified using date data

For Service Platform, the table column name (`SplitKey`) related to the management of process instance execution logs can be used as the key range partitioning key.

By dividing the execution log and storing it in RDAREAs in a row-partitioned table on the database server, you can manipulate log data in units of RDAREAs. Therefore, the larger the execution log, the more efficient deletions become.

For details about an example of deleting execution log data in units of RDAREAs, see *6.1.4 Deleting process instance execution logs* in the *Service Platform Setup and Operation Guide*.

(2) FIX hash partitioning using process instance identifiers

For Service Platform, the table column name (`ProcessID`) related to the management of process instance execution logs can be used as the FIX hash partitioning key.

If the partitioning key is specified for the search condition, only the RDAREAs expected to have the target data will be searched. This increases search efficiency.

Important note

If the partitioning key is not specified for the search condition, all RDAREAs will be searched. As a result, due to communication between servers, the searches might not be performed as efficiently as when partitioning is used.

The SQL statement that is issued during request processing differs depending on the business process contents. To check details about the SQL statement that is issued, use the SQL trace functionality. For details about the SQL trace functionality, see the *HiRDB UAP Development Guide*.

(3) Notes on dividing the execution log

Note the following when dividing the execution log for process instances:

- You cannot migrate the execution log from an environment in which an HCSC server has already been set up.
- To delete the execution log data in units of RDAREAs, you must stop reception on all HCSC servers.
- Enhance the duplication check of correlation sets. To enhance the duplication check of correlation sets, specify `ON` for the `db-tbl-split-corcheck-use` property in the HCSC server setup definition file. If `OFF` is specified, requests might not be processed correctly. For details about the `db-tbl-split-corcheck-use` property, see *HCSC Server Runtime definition file* in the *Service Platform Reference Guide*.
- If the same value is specified for correlation sets, all of the requests might result in errors. Specify a unique value for each correlation set.

8

Functionality to Transfer Files by Integrating with FTP (FTP Integration)

This chapter describes the functionality for transferring files by integrating with FTP (FTP integration).

8.1 Overview of FTP integration

The functionality that uses FTP (File Transfer Protocol) to relay the transfer of files between various systems is called **FTP integration**.

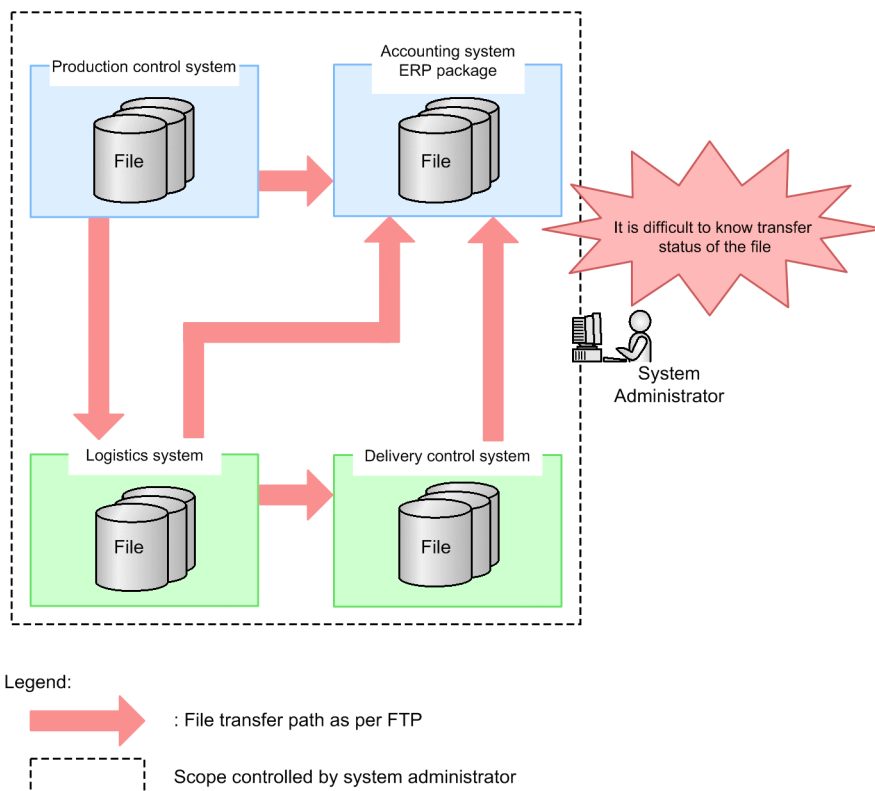
This section describes the purpose and features of FTP integration.

(1) Purpose of FTP integration

In the existing systems that use FTP, the systems that transfer files are integrated individually. Therefore, it is very difficult for a System Administrator to manage the transfer status for the entire system (such as the transfer route and data types).

The following figure shows an example of how transfer status is managed by FTP in an existing system:

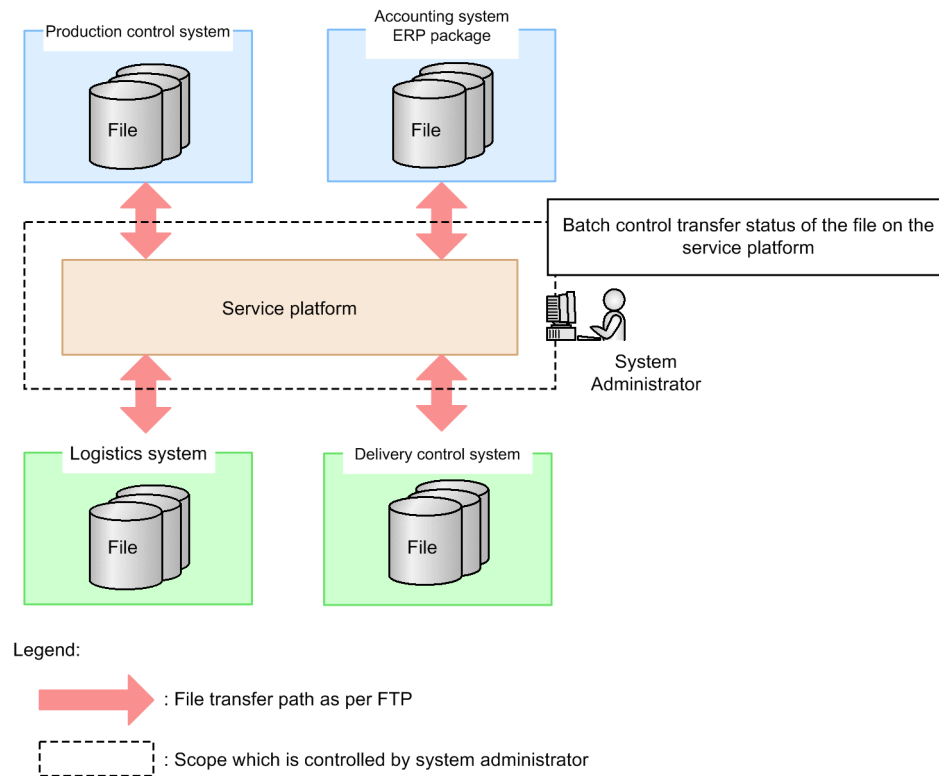
Figure 8–1: Example of an existing system



By using FTP integration, the transfer between systems that was being managed individually in the past, can now be managed in a batch on Service Platform.

The following figure shows an example of how the transfer status is managed by FTP in a system that uses Service Platform:

Figure 8–2: Example of a system using Service Platform



Managing the status of transfer between systems in a batch also has the advantage of reducing the modification costs because it is easy to understand the extent of the impact on the entire system when system is modified.

(2) Features of FTP integration

FTP integration includes the following features:

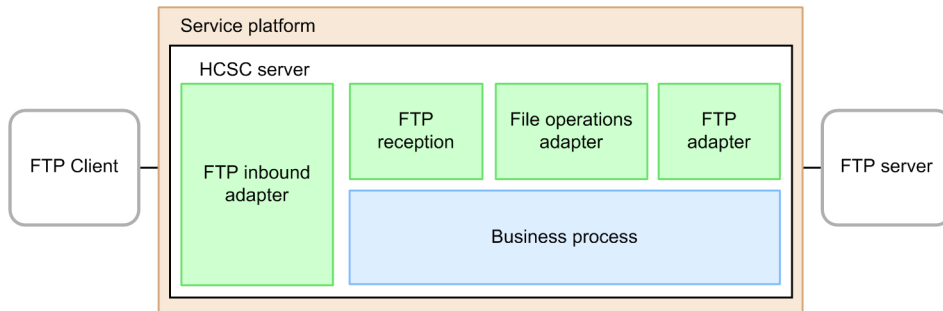
- You can manage the status of transfer between systems by using FTP.
By using the user-defined Java classes of a business process, you can obtain the following information, required to understand the transfer status, as log:
 - Route of transfer between systems
 - Transfer start and end time, and the time required for transfer
 - Transfer frequency
 - Amount of data transferred
- You can integrate your system with various systems that are using FTP.
You can relay the sending and receiving of files by connecting to an existing system. If the file formats and handled character codes differ between the systems, you can handle this by using the layout and character code conversion functionality. You do not require additional software for the FTP client and FTP server you want to connect to.

8.2 System configuration for using FTP integration

In file transfer by FTP, the FTP client requests transfer to the FTP server, and uploads or downloads files. In this manual, the system that requests the transfer is called an **FTP client**, and the system that receives the transfer request is called an **FTP server**.

The following figure shows the system configuration in the FTP server, FTP client, and Service Platform for using FTP integration:

Figure 8–3: System configuration for using FTP integration



The elements configuring the system are as follows:

FTP inbound adapter

An FTP inbound adapter is a resource adapter that is used for operations such as processing the requests from the FTP client, authenticating the FTP client, and processing file data transfer. The requests from an FTP client are sent to an FTP reception via an FTP inbound adapter.

! Important note

Connect the FTP client to the FTP inbound adapter with IPv4.

FTP reception

An FTP reception is an interface that is used to receive the requests from an FTP client. The FTP reception sends and receives files to and from an FTP client and business processes.

Business process

A business process defines activities for invoking a file operations adapter and FTP adapter.

File operations adapter

A file operations adapter is an adapter used for file operations such as file layout conversion, character code conversion, replication, and deletion.

FTP adapter

An FTP adapter is an adapter used for sending and receiving files by connecting to an FTP server.

8.3 Examples of file transfer patterns using FTP integration

The file transfer patterns that you can implement by using FTP integration are as follows:

- Transfer of files from an FTP client to an FTP server
- Transfer of files from an FTP server to an FTP client
- Transfer of files from an FTP client to another FTP client
- Obtaining a list of FTP server files

You can implement each file transfer pattern by combining the business processes, FTP receptions, FTP adapters, and file operations adapters.

This section describes the features of each file transfer pattern and the procedure of file transfer.

8.3.1 Transfer of files from an FTP client to an FTP server

In this transfer pattern, you use the STOR or APPE transfer commands to send files from an FTP client to an FTP server.

This transfer pattern includes the following two types:

To use synchronous business processes

This transfer pattern returns a response to the FTP client after completing the transfer of files to the FTP server.

In this transfer pattern, you can synchronize the state of processing in the FTP client and FTP server as is the case when FTP integration is not used.

To use asynchronous business processes

This transfer pattern returns a response to the FTP client and then sends the files to the FTP server.

You can use this transfer pattern in the following cases because the response time seen from the FTP client is shortened:

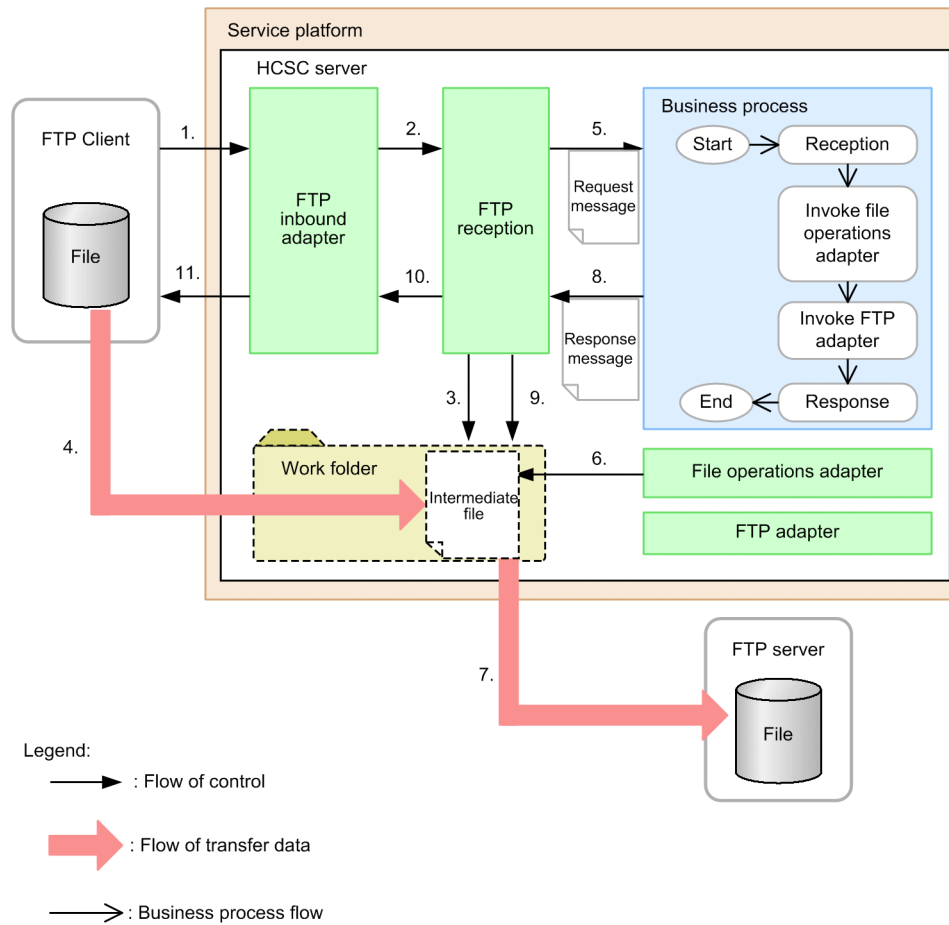
- To multicast the same information from one point to multiple points
- To send large-sized files that exceed the non-communication monitoring time of the FTP client

The respective file transfer procedure are as follows:

(1) Procedure of file transfer using synchronous business processes

The following figure shows the procedure of file transfer:

Figure 8–4: Procedure of file transfer from an FTP client to an FTP server (When synchronous business processes are used)



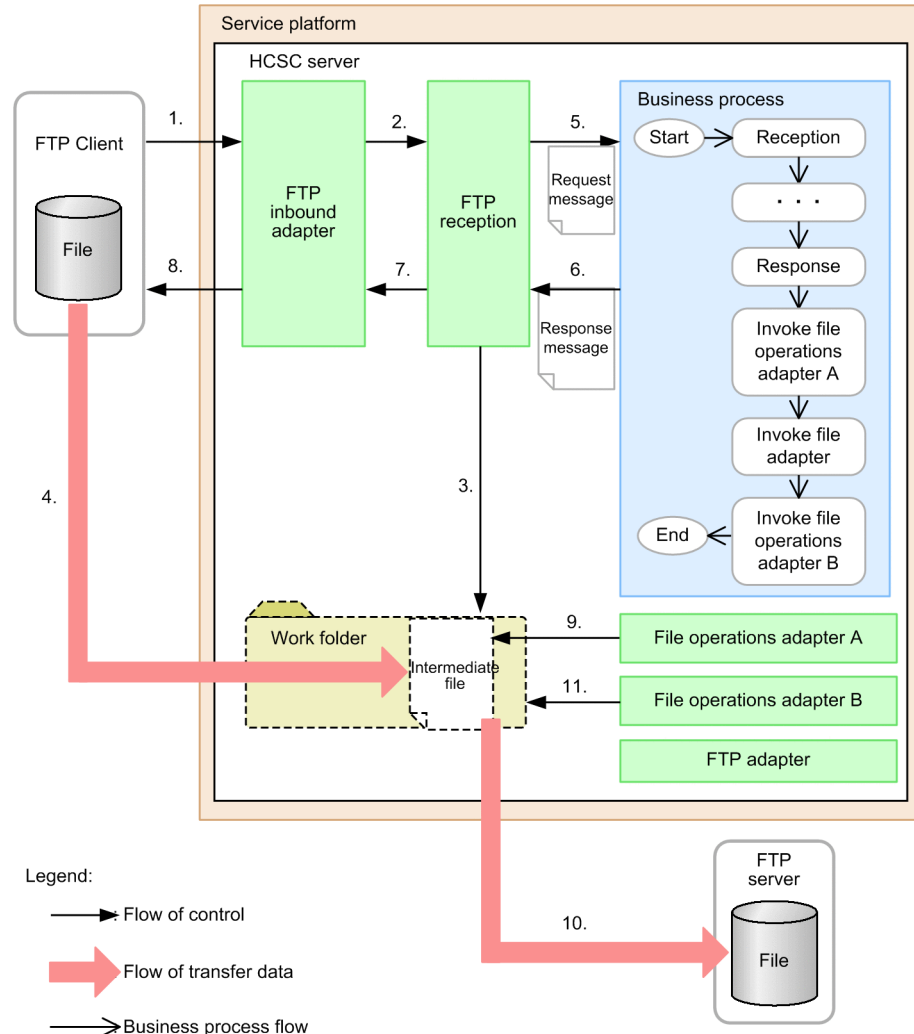
The following is a description of the flow of control, data transfer, and business processes in the figure. The following numbers correspond to the numbers in the figure:

1. The FTP client executes a transfer command (STOR command or APPE command).
2. The FTP inbound adapter that receives the transfer command invokes the FTP reception.
3. The FTP reception creates a work folder (a temporary folder for handling files on the HCSC server).
4. The file data transferred from the FTP client is output to the work folder as an intermediate file.
5. The FTP reception invokes a business process. At this time, the information received from the FTP client, such as the transfer command and name of the intermediate file, is stored in a request message and passed to the business process.
6. The file operations adapter is invoked from the business process, and then the file operations adapter reads the intermediate file from the work folder, and performs processing such as layout conversion and code conversion.
7. The FTP adapter is invoked from the business process, and then the FTP adapter transfers the data of the intermediate file to the FTP server.
8. A reply activity is executed with the business process, and then a response message is passed to the FTP reception.
9. The FTP reception deletes the work folder and the intermediate file in the work folder.
10. The FTP reception returns a response to the FTP inbound adapter.
11. The FTP inbound adapter that receives the response returns the execution result of the transfer command to the FTP client.

(2) Procedure of file transfer using asynchronous business processes

The following figure shows the procedure of file transfer:

Figure 8–5: Procedure of file transfer from an FTP client to an FTP server (When asynchronous business processes are used)



The following is a description of the flow of control, data transfer, and business processes in the figure. The following numbers correspond to the numbers in the figure:

1. The FTP client executes a transfer command (STOR command or APPE command).
2. The FTP inbound adapter that receives the transfer command invokes the FTP reception.
3. The FTP reception creates a work folder.
4. The file data transferred from the FTP client is output to the work folder as an intermediate file.
5. The FTP reception invokes a business process. At this time, the information received from the FTP client, such as the transfer command and name of the intermediate file, is stored in a request message and passed to the business process.
6. The reply activity is executed with the business process, and then a response message is passed to the FTP reception.
7. The FTP reception returns a response to the FTP inbound adapter.
8. The FTP inbound adapter that receives the response returns the execution result of the transfer command to the FTP client.

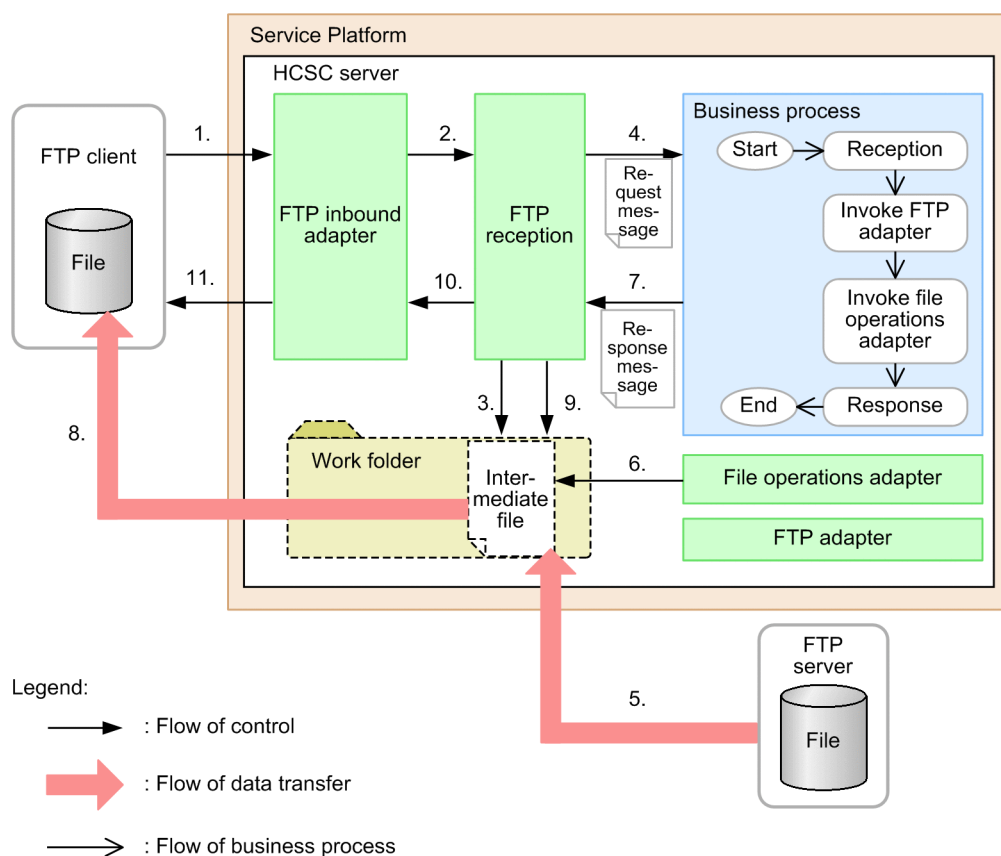
9. The file operations adapter A is invoked from the business process, then the file operations adapter A reads the intermediate file from the work folder, and performs processing such as layout conversion and code conversion.
10. The FTP adapter is invoked from the business process, and then the FTP adapter transfers the data of the intermediate file to the FTP server.
11. The file operations adapter B is invoked from the business process, and then the file operations adapter B deletes the work folder and the intermediate file in the work folder before the business process ends.

8.3.2 Transfer of files from an FTP server to an FTP client

In this transfer pattern, you use the `RETR` transfer command from an FTP client to send files from an FTP server to the FTP client.

The following figure shows the procedure for transferring a file.

Figure 8–6: Procedure for transferring files from an FTP server to an FTP client



The following describes the control flow, data transfer flow, and business process flow in the figure. The following numbers correspond to the numbers in the figure.

1. The FTP client executes a transfer command (`RETR` command).
2. The FTP inbound adapter that receives the transfer command invokes the FTP reception.
3. The FTP reception creates a work folder.
4. The FTP reception invokes a business process. At this time, the information received from the FTP client, such as the transfer command, is stored in a request message and passed to the business process.
5. The FTP adapter is invoked from the business process. Then, the FTP adapter obtains a file from the FTP server, and then outputs it to the work folder as an intermediate file.
6. The file operations adapter is invoked from the business process. Then, the file operations adapter reads the intermediate file from the work folder, and then performs processing such as converting the layout and code.

7. A reply activity is executed with the business process, and then a response message containing intermediate file information is passed to the FTP reception.
8. The intermediate file data indicated in the response message is transferred from the work folder to the FTP client.
9. The FTP reception deletes the work folder and the intermediate file in the work folder.
10. The FTP reception returns a response to the FTP inbound adapter.
11. The FTP inbound adapter that receives the response returns the execution result of the transfer command to the FTP client.

8.3.3 Transfer of files from an FTP client to another FTP client

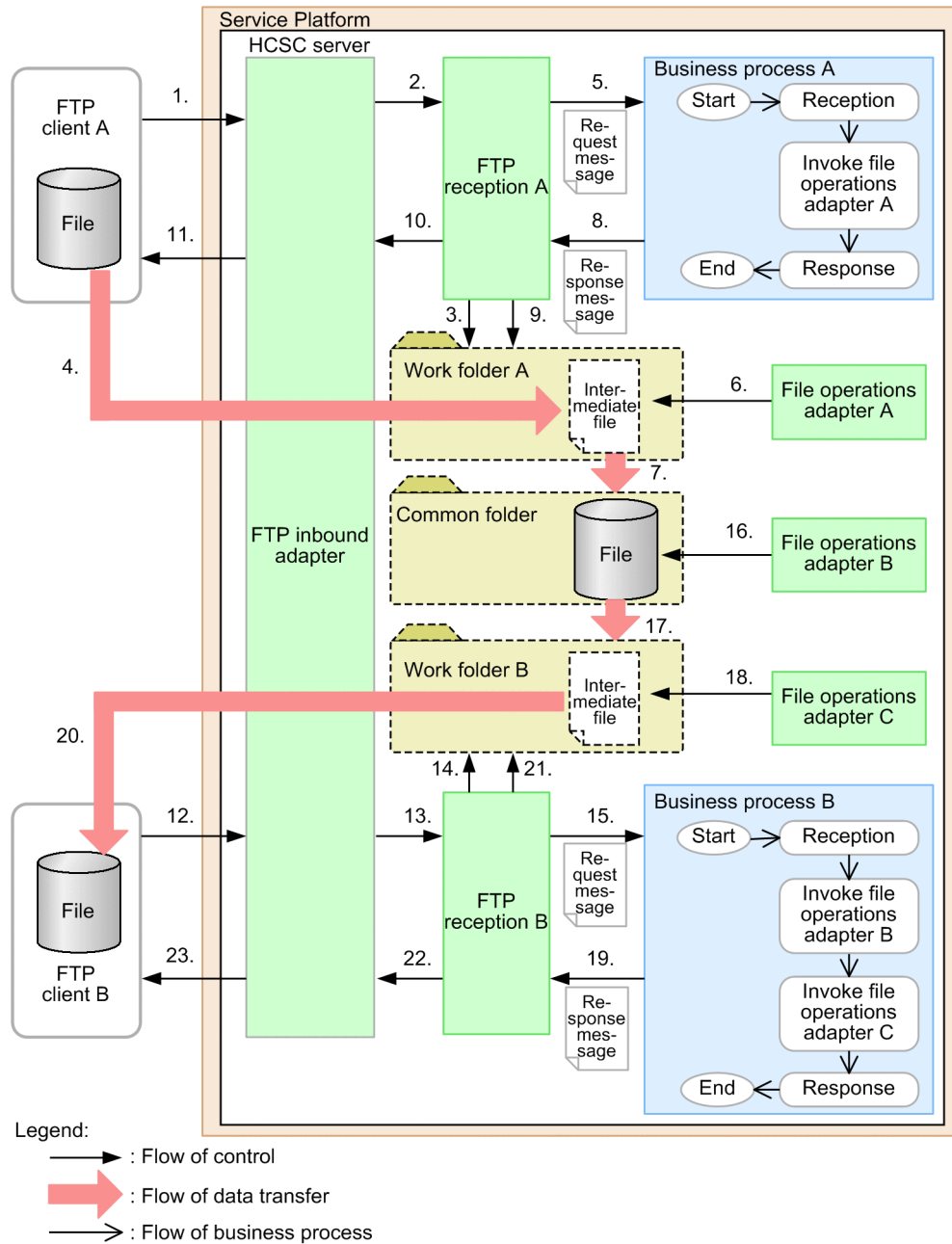
This subsection describes a transfer pattern to send and receive files between FTP clients in a one-to-one configuration or one-to-many configuration.

In this transfer pattern, the file sending system and receiving system can access the HCSC server at any time. Therefore, this transfer pattern is useful for transferring files between systems with different operation time zones, such as overseas sites.

The following figure shows the procedure for transferring a file.

In this figure, FTP client A executes the `STOR` or `APPE` command to send a file to the HCSC server (numbers 1 to 11 in the figure). Then, FTP client B executes the `RETR` command to obtain the file from the HCSC server (numbers 12 to 23 in the figure)

Figure 8–7: Procedure for transferring a file from an FTP client to another FTP client



The following describes the control flow, data transfer flow, and business process flow in the figure. The following numbers correspond to the numbers in the figure.

1. FTP client A executes a transfer command (STOR or APPE command).
2. The FTP inbound adapter that receives the transfer command invokes FTP reception A.
3. FTP reception A creates work folder A.
4. The file data transferred from FTP client A is output to work folder A as an intermediate file.
5. FTP reception A invokes business process A. At this time, the information received from FTP client A, such as the transfer command and intermediate file name, is stored in a request message, and then passed to business process A.
6. File operations adapter A is invoked from business process A. Then file operations adapter A accesses the intermediate file in work folder A.

7. File operations adapter A replicates the intermediate file in work folder A to a common folder (a folder created by the user).
8. A reply activity is executed with business process A, and then a response message is passed to FTP reception A.
9. FTP reception A deletes work folder A and the intermediate file in that folder.
10. FTP reception A returns a response to the FTP inbound adapter.
11. The FTP inbound adapter that receives the response returns the execution result of the transfer command to FTP client A.
12. FTP client B executes a transfer command (RETR command).
13. The FTP inbound adapter that receives the transfer command invokes FTP reception B.
14. FTP reception B creates work folder B.
15. FTP reception B invokes business process B. At this time, the information received from FTP client B, such as the transfer command, is stored in a request message and then passed to business process B.
16. File operations adapter B is invoked from business process B. Then, file operations adapter B accesses the file in the common folder.
17. File operations adapter B replicates the file in the common folder to work folder B.
18. File operations adapter C is invoked from business process B. Then, file operations adapter C reads the intermediate file from work folder B, and then performs processing such as converting the layout and code.
19. A reply activity is executed with business process B, and then a response message containing the intermediate file information is passed to FTP reception B.
20. The intermediate file data indicated in the response message is transferred from the FTP inbound adapter to FTP client B.
21. FTP reception B deletes work folder B and the intermediate file in that folder.
22. FTP reception B returns a response to the FTP inbound adapter.
23. The FTP inbound adapter that receives the response returns the execution result of the transfer command to FTP client B.

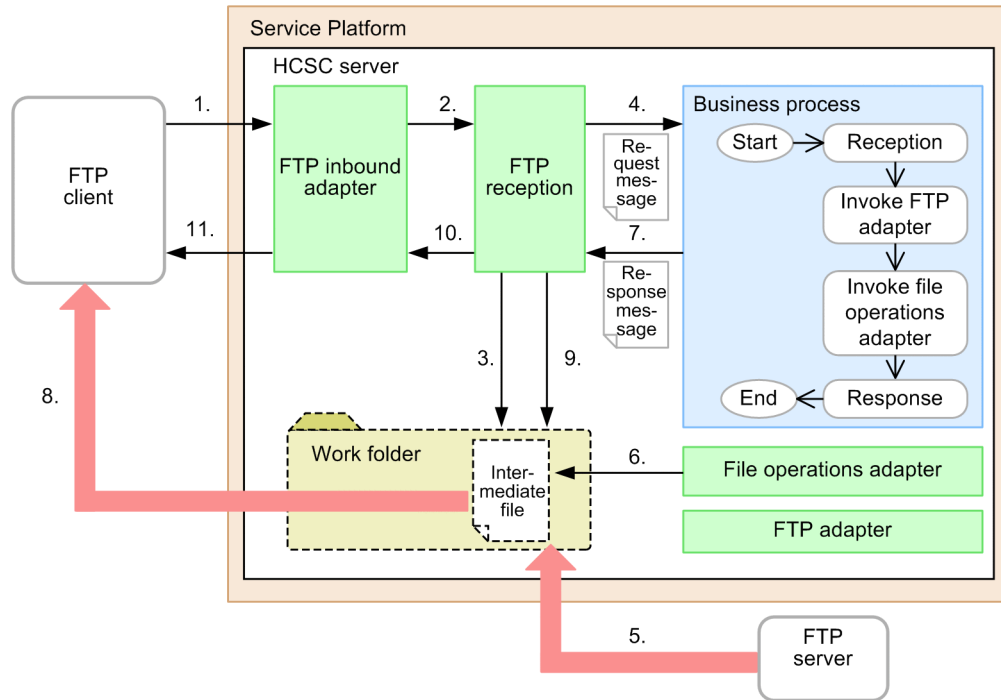
8.3.4 Obtaining a file list from an FTP server

In this transfer pattern, you use a list command from an FTP client or service requester to obtain a file list from an FTP server.

(1) When outputting a file list to a work folder

The following describes an example in which a file list obtained from an FTP server is output to a work folder and then sent to an FTP client.

Figure 8–8: Procedure for obtaining a file list from the FTP server (when output to a work folder)



Legend:

- : Flow of control
- ➡ : List data flow
- ➞ : Flow of business process

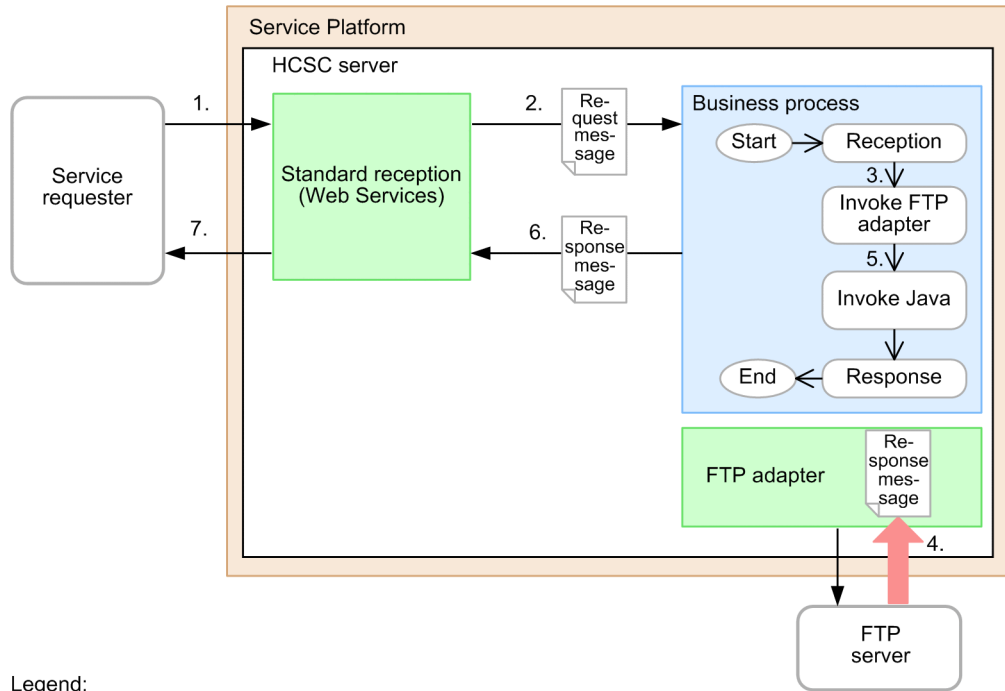
The following describes the control flow, list data flow, and business process flow in the figure. The following numbers correspond to the numbers in the figure.

1. The FTP client executes a list command.
2. The FTP inbound adapter that receives the list command invokes the FTP reception.
3. The FTP reception creates a work folder.
4. The FTP reception invokes a business process. At this time, the information received from the FTP client, such as the FTP command, is stored in a request message and passed to the business process.
5. The FTP adapter is invoked from the business process. Then, the FTP adapter obtains a file list from the FTP server, and then outputs it to the work folder as an intermediate file.
6. The file operations adapter is invoked from the business process. Then, the file operations adapter reads the intermediate file from the work folder, and then performs processing such as converting the layout and code.
7. A reply activity is executed with the business process, and then a response message containing intermediate file information is passed to the FTP reception.
8. The intermediate file data indicated in the response message is transferred from the work folder to the FTP client.
9. The FTP reception deletes the work folder and the intermediate file in the work folder.
10. The FTP reception returns a response to the FTP inbound adapter.
11. The FTP inbound adapter that receives the response returns the execution result of the list command to the FTP client.

(2) When outputting a file list to a response message

The following describes an example in which a file list obtained from an FTP server is output to a response message and then processed by a business process.

Figure 8–9: Procedure for obtaining a file list from the FTP server (when output to a response message)



Legend:

- : Flow of control
- ➡ : List data flow
- ➞ : Flow of business process

The following describes the control flow, list data flow, and business process flow in the figure. The following numbers correspond to the numbers in the figure.

1. The service requester sends a request for a list command.
2. Standard reception (Web Services) invokes a business process. At this time, the information received from the service requester, such as the FTP command, is stored in a request message and passed to the business process.
3. The FTP adapter is invoked from the business process.
4. The FTP adapter obtains a file list from the FTP server, and then stores it in memory and in a response message.
5. The list data in the response message of the FTP adapter is obtained by using a request message for the invoke java activity, and then analyzed.
6. The business process returns a response message to standard reception (Web Services).
7. Standard reception (Web Services) returns a response to the service requester.

8.4 FTP reception functionality

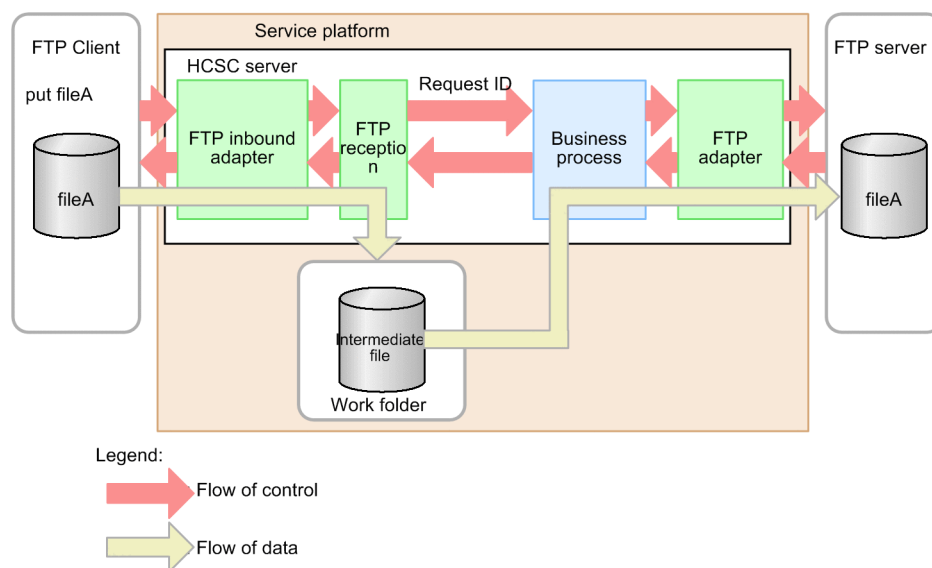
This section provides an overview of the purpose of an FTP reception, and using an FTP reception to send and receive files.

An FTP reception is an interface that invokes the business processes by receiving a connection from the FTP client via the FTP inbound adapter, and which mediates the file data and list information that is sent and received between the FTP client and business processes.

By performing operations in integration with the file operations adapter and FTP adapter that is invoked from the business process, you can transfer files between the FTP client and FTP server via the business process, and obtain the list information of files.

The following figure shows the procedure of FTP integration using an FTP reception, taking the example of transferring files to the FTP server:

Figure 8–10: Procedure of FTP integration using an FTP reception



8.4.1 Operations and FTP commands supported in an FTP reception

In an FTP reception, the unit of operations for a file is called an operation.

The following types of operations are supported in an FTP reception:

PUT operation

This operation transfers files from an FTP client to an FTP server.

GET operation

This operation transfers files from an FTP server to an FTP client.

GETINFO operation

This operation obtains the file list information of an FTP server and sends this information to the FTP client.

These operations are invoked from the FTP inbound adapter when an FTP client invokes an FTP command for the FTP inbound adapter.

! Important note

- Do not define multiple FTP receptions specifying the GETINFO operation for one FTP inbound adapter.
- When you associate multiple FTP receptions for one business process, select the same operation in all these FTP receptions. The operation is not guaranteed if you define multiple FTP receptions with different operations for one business process.

(1) Types of operations and FTP commands supported in an FTP reception

The following table describes the FTP commands supported in an FTP reception:

Table 8–1: FTP commands supported in an FTP reception

FTP command	Usage	Operation of the invoked FTP reception
STOR	To accumulate the file contents in a server-side file. If a file already exists on the server, the existing file content is overwritten.	PUT operation
APPE	To accumulate the file contents in a server-side file. If a file already exists on the server, the transferred data is added to the existing file.	
RETR	Obtains the replication of a file existing on the server-side.	GET operation
LIST	Obtains the list of file information existing on the server-side.	GETINFO operation
NLST	Obtains the list of names of files existing on the server-side.	
SITE CSCTHR	Sets up the FTP commands to be executed in a business process before and after the transfer commands and before and after the list commands. If you execute this command before the FTP client executes the transfer commands or list commands, the information specified with this command along with the information of the transfer commands or list commands executed thereafter, is passed to the business process.	--

Legend:

--: No operation is invoked.

(2) Execution format and points to be considered on the FTP commands

The execution format and points to be considered on the FTP commands are as follows:

Execution format

```
<FTP command> /<Reception ID of FTP reception>/<Remote file path>
```

Points to be considered

- Make sure that you specify a forward slash (/) as the sign specified at the beginning of the reception ID of an FTP reception and the separation character specified for the remote file path. For details on the format of the reception ID of an FTP reception, see "8.4.4 Generating the request ID".
- For the restrictions on the specification of arguments for the transfer commands, follow the FTP protocol specifications.

8.4.2 Communication model and business processes that can be used with FTP receptions

This subsection describes the types of communication models and business processes that can be used with FTP receptions.

(1) Communication model that can be used with FTP receptions

The communication model that can be used with an FTP reception is synchronous communication.

(2) Business processes that can be used with FTP receptions

The two types of business processes that can be associated with an FTP reception are synchronous and asynchronous business processes. The following table describes the features of each business process:

Table 8–2: Features of the business processes

Business process type	Features	Remarks
Synchronous	A business process invoked from an FTP reception, but which returns a response after the target processing is complete. No activity is executed after a response is returned.	You can use this business process regardless of the operation type and file integration method.
Asynchronous	A business process invoked from an FTP reception, but which returns a response before the target processing is complete. Activities are executed after a response is returned.	From among the business processes, this processing is executed asynchronously (processing executed after returning a response to the FTP reception), and you cannot execute the processing related to the GET operation and GETINFO operation.

8.4.3 Transactions when FTP reception is used

The FTP inbound adapter does not contain information that is carried over into the subsequent FTP reception processing using transactions. Therefore, a new transaction, which is different from the transaction for the FTP inbound adapter, is started in the FTP reception. However, a new transaction, which is further different from the FTP reception, is started in the business process.

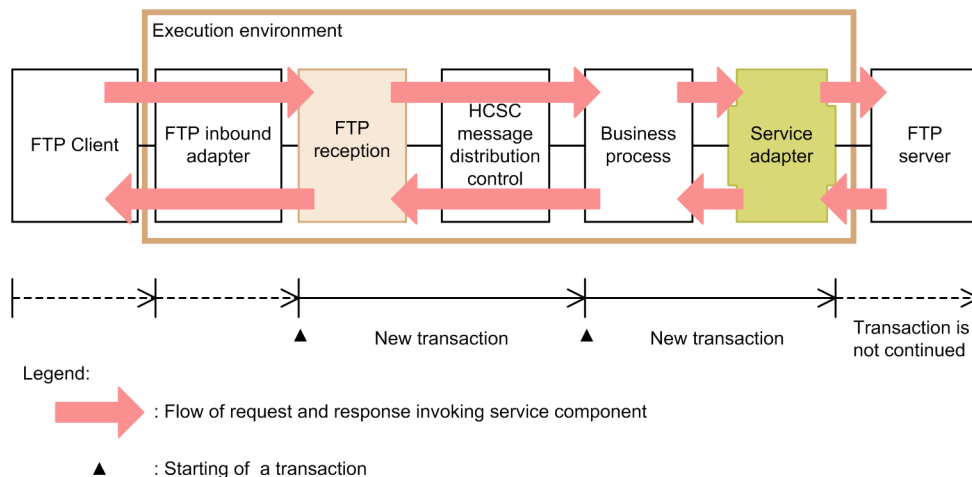
The new transaction start timing differs when the FTP reception timeout is monitored and the timeout is not monitored.

For details on the monitoring of FTP reception timeout, see "8.4.9 Monitoring a timeout".

(1) FTP reception transactions (When the timeout is not monitored)

The following figure shows the FTP reception transactions when a business process is invoked without monitoring the FTP reception timeout:

Figure 8–11: FTP reception transactions (When the timeout is not monitored)

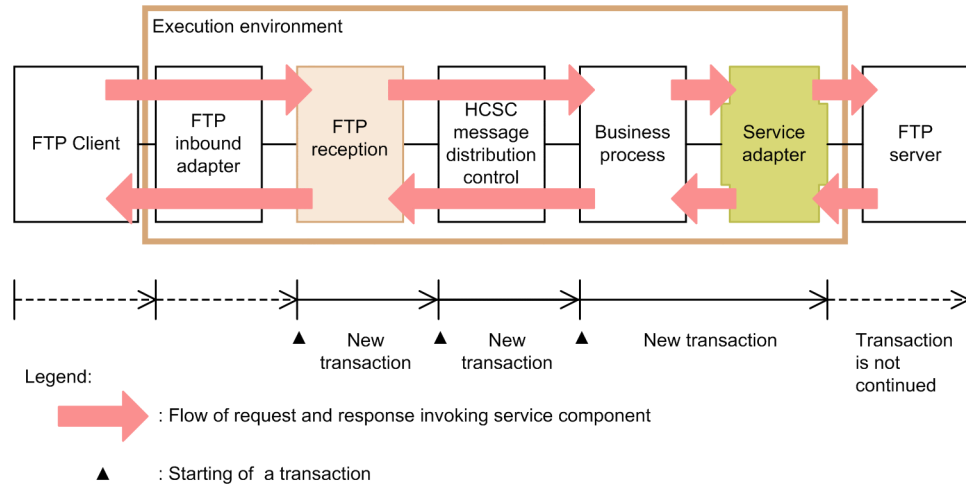


Note that when files are transferred to the server by using an FTP adapter in the business process, and if an error occurs in the processing of the business process after the transfer of files to the server is complete, the transaction is rolled back, and an error is returned to the FTP client. However, the status of the server-side files remains as is after transfer, and does not return to the pre-transfer status.

(2) FTP reception transactions (When the timeout is monitored)

When the FTP reception timeout is monitored, the business process is invoked with a different thread, so a new transaction starts in HCSC message delivery control. The following figure shows the FTP reception transactions when a business process is invoked with FTP reception timeout monitoring:

Figure 8–12: FTP reception transactions (When the timeout is monitored)



The transaction started in the HCSC message delivery control is monitored with the transaction timeout value for the J2EE server. For details on the transaction timeout settings, see "8.6.4 Setting up a transaction timeout" in "Application Server System Design Guide".

It is recommended that when you set up the timeout value for an FTP reception; specify a value that is greater than the transaction timeout value for the J2EE server.

8.4.4 Generating the request ID

When the FTP reception receives a request from the FTP client, the FTP reception generates a **request ID** for each request. The business process is executed by specifying the generated request ID.

The request ID is generated in the following format:

```
<Reception ID><IP address of FTP client><Date><Time><Serial number><Partition key><HCSC server name>
```

The following table describes the items in the request ID:

Table 8–3: Items in the request ID

Item	Number of digits	Content
<Reception ID>	8 digits	This is the reception ID specified in the FTP reception settings. If the reception ID is less than 8 digits, "-" (hyphens) are specified to the right side of the reception ID for the missing digits.
<IP address of FTP client>	12 digits	This is the IP address of FTP client that sends the request. If each byte of the IP address is less than 3 digits, "0" is specified to the left side of each byte for the missing digits.
<Date>	8 digits	The date on which the FTP reception received the request is set up in the yyyyymmdd format. <ul style="list-style-type: none"> yyyy: Year mm: Month dd: Day

Item	Number of digits	Content
<Time>	9 digits	The time at which the FTP reception received the request is set up in the hhmmssSSS format. <ul style="list-style-type: none"> • hh: Hours • mm: Minutes • ss: Seconds • SSS: Milliseconds
<Serial number>	3 digits	An integer value is set up sequentially from "000" to make the folder name unique. If <Date><Time><Serial number> is duplicated with an existing request ID, the serial number moves up by one. If the values are duplicated even after specifying an integer value from "001" to "999", change the <Date> and <Time>, and set the serial number to "000".
<Partition key>	1 digit	This is a partition key indicating that the HCSC server name has been added. A single-byte forward slash (/) is used.
<HCSC server name>	Maximum 8 digits	This is the name of the HCSC server deployed by the FTP reception. If the HCSC server name is less than 8 digits, the shortfall is not supplemented.

8.4.5 Creating the work folder and intermediate files

To transfer the file data between business processes, the FTP reception creates an intermediate file to temporarily store the file data and list information sent from the FTP client. The intermediate file is stored in a work folder that is generated by the FTP reception for each request.

Set up the directory for creating the work folder in the work-folder property of the HCSC server runtime definition file. For details, see "HCSC server runtime definition file" in "Service Platform Reference Guide".

Note that the work folder and the intermediate files stored in the work folder are deleted when the following conditions are fulfilled:

- The value of the `urecp-ftp.work-dir.auto-delete` property in the FTP reception definition file is "true".
- The FTP reception invokes a business process, and from the invoked business process, the FTP reception receives a response message wherein the value of the `<success>` tag is true.
- The transfer of data (file contents and list data) between the FTP reception and FTP inbound adapter terminated normally, or the value of `<reply-code>` in the response message of the GETINFO operation is a value from 400 to 599.

If the above conditions are not fulfilled, or if the process terminates abnormally and the business process is re-executed with a command such as the `cscpireexec` command (re-execution of the process instance), the work folder and intermediate files in the work folder are not deleted automatically. If the FTP reception cannot delete the work folder, delete the work folder and intermediate files as follows:

If the value of the `urecp-ftp.work-dir.auto-delete` property in the FTP reception definition file is "false"

Invoke the file operations adapter in the final processing executed by the business process and then delete the work folder and intermediate files, or reference the work folder with the `cscfswls` command, check whether you want to delete that work folder, and then delete the work folder with the `cscfswrm` command.

If the value of the `urecp-ftp.work-dir.auto-delete` property in the FTP reception definition file is "true"

Reference the work folder with the `cscfswls` command, check whether you want to delete that work folder, and then delete the work folder with the `cscfswrm` command.

For details, see "`cscfswls` (Referencing the work folder)" or "`cscfswrm` (Deleting the work folder)" in "Service Platform Reference Guide".

8.4.6 Business process access control

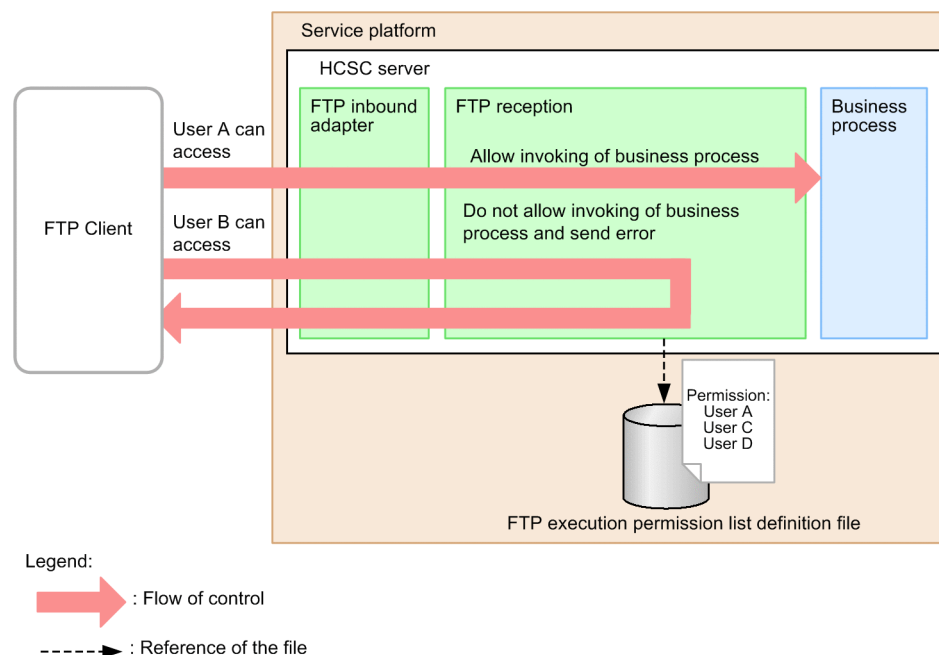
This subsection describes business process access control and management of FTP commands executed before and after transfer as the FTP reception functionality.

By setting up users that are allowed or prohibited access to the business processes, the FTP reception can control the access to the business processes for each user.

You can specify the business process access control settings in the FTP execution permission list definition file. Based on the settings in the FTP execution permission list definition file, the FTP reception decides whether to allow business process access to the users who are accessing the business process from an FTP client. The FTP reception invokes the business process only when the access is allowed, and returns an error without invoking the business process when the access is denied.

The following figure shows the procedure of business process access control:

Figure 8–13: Procedure of business process access control



In this figure, a business process is invoked for the request from user A who is allowed access in the FTP execution permission list definition file, but an error is returned without invoking a business process for the request from user B who is denied access in the FTP execution permission list definition file.

The following table describes the contents set up in the FTP execution permission list definition file:

Table 8–4: Contents set up in the FTP execution permission list definition file

Content	Specification method	Default value
Users who are allowed to access the associated business processes by the FTP reception	Specify the user name in the urecp-ftp.permission.allow property. To specify multiple users, separate the user names with a comma (.). If you specify an asterisk (*), access is allowed to all the users.	*
Users who are denied to access the associated business processes by the FTP reception	Specify the user name in the urecp-ftp.permission.deny property. To specify multiple users, separate the user names with a comma (.).	None

Note

If the same users are specified in both, the settings that deny access have a higher priority.

Note that you can store the FTP execution permission list definition file in any folder with any file name, but you must specify the absolute path of the FTP execution permission list definition file in the FTP reception configuration file. If

the absolute path of the FTP execution permission list definition file is not specified in the FTP reception configuration file, all the users are allowed to access the business process.

Furthermore, if the contents of the FTP execution permission list definition file are changed, you must restart the FTP reception that is using the changed FTP permission list definition file.

For details, see "FTP execution permission list definition file " or "FTP reception configuration file " in "Service Platform Reference Guide".

8.4.7 Managing FTP commands executed before and after transfer and list

By executing the SITE CSCTHR command, you can set up the FTP commands to be executed in a business process before executing the transfer commands, after executing the transfer commands, before executing the list commands, or after executing the list commands. If the FTP client executes the SITE CSCTHR command before executing the transfer commands or list commands, the information specified in the SITE CSCTHR command is passed to the business process along with the information of the FTP commands executed thereafter.

The execution format of the SITE CSCTHR command is as follows:

Execution format

```
SITE CSCTHR {-B|-A} <FTP command>[;<FTP command> ...]
```

Apart from the FTP commands, to specify commands that are independently supported by the FTP server, specify the SITE command in the <FTP command> part as follows, and then specify the command in the argument of the SITE command:

```
SITE CSCTHR {-B|-A} SITE <Commands independently supported by the FTP server>
```

Furthermore, by specifying -B or -A in the option, you can set up whether to execute the specified FTP command before executing the transfer command/ list command, or whether to execute the specified FTP command after executing the transfer command/ list command.

Operations when the -B option is used

The FTP command specified in the "<FTP command>[;<FTP command> ...]" part is set up in the <ftp-commands-before> tag of the request message of the FTP reception. An FTP command specified in this format is executed before the transfer command or list command is executed. If the SITE CSCTHR command is executed multiple times in this format, the "<FTP command>[;<FTP command> ...]" part is added to the <ftp-commands-before> tag, linked with semicolons (;), in the sequence in which the SITE CSCTHR commands are executed.

Operations when the -A option is used

The FTP command specified in the "<FTP command>[;<FTP command> ...]" part is set up in the <ftp-commands-after> tag of the request message of the FTP reception. An FTP command specified in this format is executed after the transfer command or list command is executed. If the SITE CSCTHR command is executed multiple times in this format, the "<FTP command>[;<FTP command> ...]" part is added to the <ftp-commands-after> tag, linked with semicolons (;), in the sequence in which the SITE CSCTHR commands are executed.

You can specify and manage the commands, which can be executed in the business process before and after executing the transfer commands and before and after executing the list commands, in the FTP command permission list definition file (for FTP receptions). The FTP commands that are not specified in the FTP command permission list definition file (for FTP receptions) are not added to the <ftp-commands-before> tag, or <ftp-commands-after> tag of the request message of the FTP reception even if such FTP commands are included in the SITE CSCTHR commands. In such a case, the business process is not invoked, so an error occurs.

Note that you can store the FTP command permission list definition file (for FTP receptions) in any folder with any file name, but you must specify the absolute path of the FTP command permission list definition file (for FTP receptions) in the FTP reception configuration file. If the absolute path of the FTP command permission list definition file (for FTP receptions) is not specified in the FTP reception configuration file, all the FTP commands are prohibited execution in the business process.

If you change the contents of the FTP command permission list definition file (for FTP receptions), you must restart the FTP reception that is using the changed FTP command permission list definition file (for FTP receptions).

For defined contents that you can edit in the FTP command permission list definition file (for FTP receptions), see "FTP command permission list definition file " in "Service Platform Reference Guide".

8.4.8 Executing the LIST and NLST commands by specifying list command options

If argument options for the LIST or NLST command are defined in the list command options definition file in advance, you can execute a list command by specifying argument options.

The following shows the execution format of commands with argument options specified.

- For the LIST command

```
LIST [ <option>][ <path>]
```

- For the NLST command

```
NLST [ <option>][ <path>]
```

For details about the list command options definition file, see *List command options definition file* in the *Service Platform Reference Guide*.

The list command options definition file can be stored in any path using any file name. Note that the absolute path of the list command options definition file must be specified in the FTP reception configuration file.

For details about the FTP reception configuration file, see *FTP reception configuration file* in the *Service Platform Reference Guide*.

If argument options are specified in the list command options definition file, when an FTP client executes a list command, information specified in the list command arguments is separated into <option> and <path> as defined in this definition file. Then, each piece of information is set in <getinfo-option> and <getinfo-path> in the request message of the GETINFO operation.

If no argument option is specified in the list command options definition file, all information specified in the list command arguments is set in <getinfo-path> as <path> information. This also applies if a path to the command options definition file is not specified in the FTP reception configuration file.

If you change the settings of the list command options definition file, you need to restart the FTP reception associated with this file to enable the changes.

8.4.9 Monitoring a timeout

By specifying the timeout value in urecp-ftp.request-timeout in the FTP reception configuration file, you can monitor the processing time from the passing of the request message to the business process until the return of the response message. If the response does not return even after the lapse of the specified time, a timeout exception occurs as an error.

Note that by default, the FTP reception timeout is not monitored.

When you monitor the timeout in the FTP reception, threads are generated for each FTP reception in the HCSC message delivery control. Among the threads used for timeout monitoring, you can specify the number of threads, which are generated and are resident (number of resident threads) when the reception starts, in the urecp-ftp.resident-thread.count property of the FTP reception configuration file.

If the FTP reception receives tasks exceeding the number of resident threads, additional threads are temporarily generated for processing the tasks. The additionally generated threads stand by for 300 seconds after the tasks are processed, and are destroyed if the next task does not arrive.

You can specify the maximum number of threads, generated for monitoring the FTP reception timeout (number of resident threads + number of additionally generated threads), in the urecp-ftp.timer-thread.maximum property of the FTP reception configuration file. Note that if you specify the same value for the number of resident threads and maximum number of threads, additional threads are not generated.

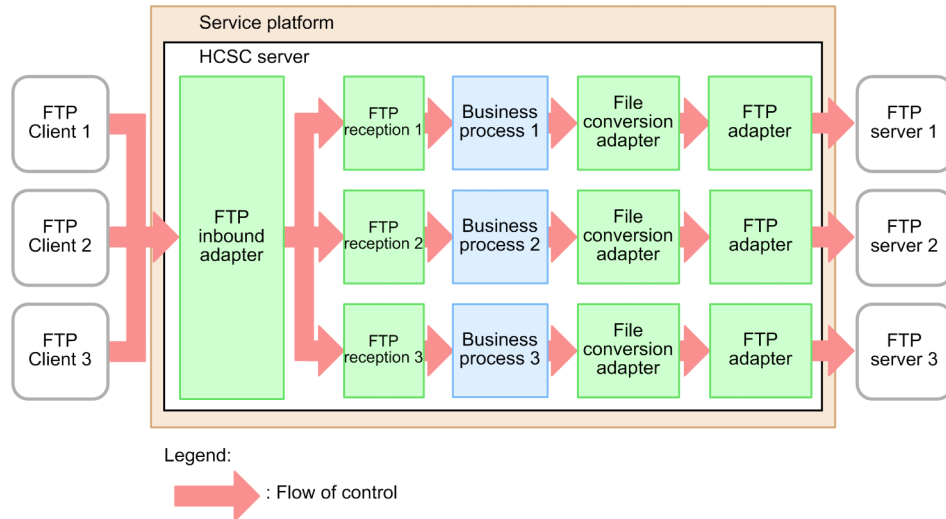
For details on the properties you can specify in the FTP reception configuration file, see "FTP reception configuration file " in "Service Platform Reference Guide".

8.5 FTP inbound adapter functionality

An FTP inbound adapter receives transfer requests from the FTP client and distributes the requests to an FTP reception associated with a business process.

The following figure shows the location of an FTP inbound adapter in FTP integration:

Figure 8–14: Location of FTP inbound adapter in FTP integration



This section describes the following functionality supported by the FTP inbound adapter:

- Checking the size after sending and receiving files
- Timeout monitoring
- Security-related functionality

For details on the security-related functionality of the FTP inbound adapter and how to set up the functionality, see "Appendix H. Encryption and authentication by secure protocols (FTP inbound adapter)" in "Service Platform Reception and Adapter Definition Guide".

8.5.1 Checking the file size after sending and receiving files

The FTP inbound adapter compares the file size before and after transfers by linking with JP1/File Transmission Server/FTP. The `SITE FSIZE` command executed from an FTP client checks whether a file was corrupted during transfer processing, thus ensuring the reliability of files that are sent and received.

For details about JP1/File Transmission Server/FTP, see the manual *JP1/File Transmission Server*.

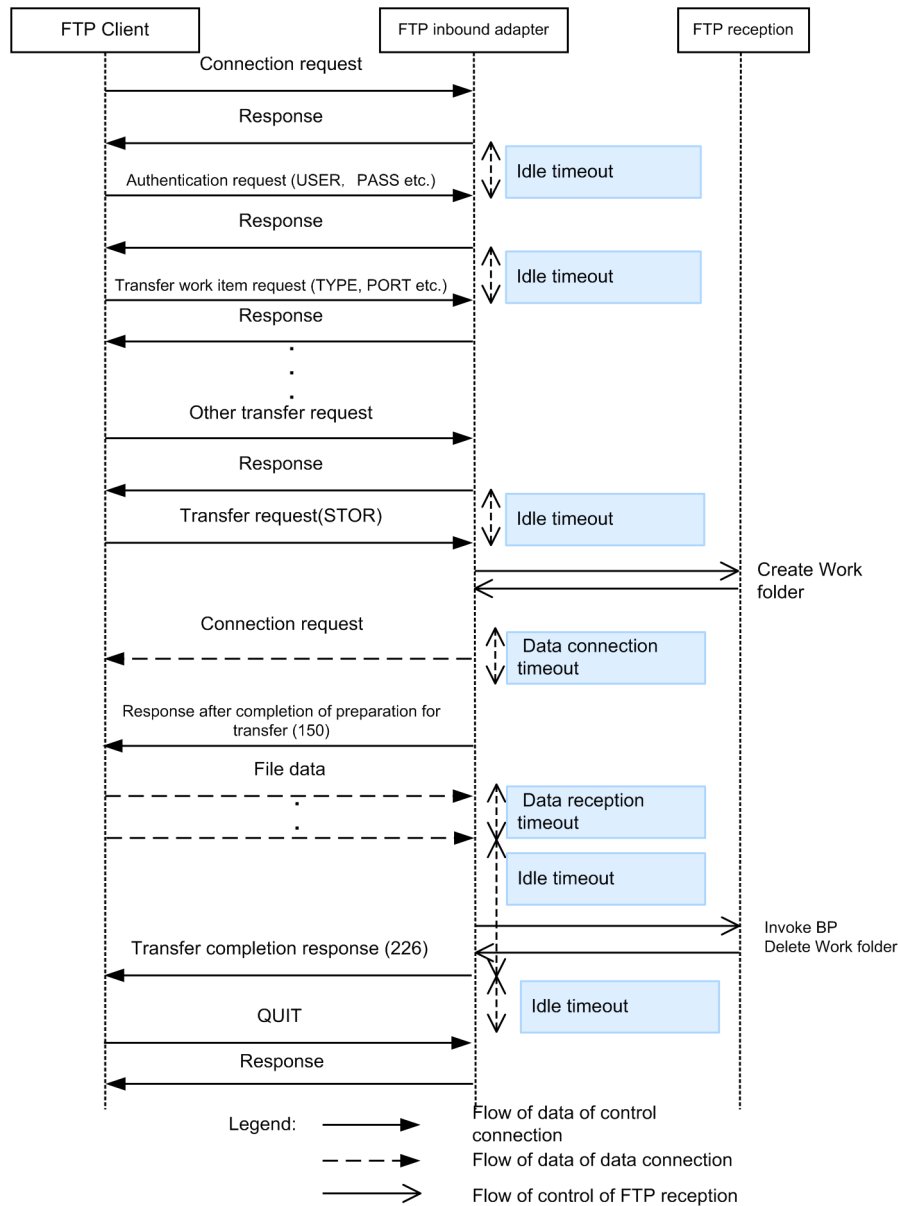
8.5.2 Monitoring a timeout

A timeout is detected in an FTP inbound adapter if the communication processing does not respond due to an FTP client or network error during communication between the FTP inbound adapter and FTP client, and during the distribution of requests to the FTP reception, and if the file operation processing subsequent to FTP reception takes time.

If a timeout is detected, a message is output and the connection is closed to prevent a decline in the processing performance of the system.

The following figure shows the points at which the timeout is monitored in the FTP inbound adapter, taking the example of file transfer using the Active mode and STOR:

Figure 8-15: Points at which the timeout is monitored in the FTP inbound adapter (Example of file transfer using the Active mode and STOR)



(1) Operations when a timeout occurs

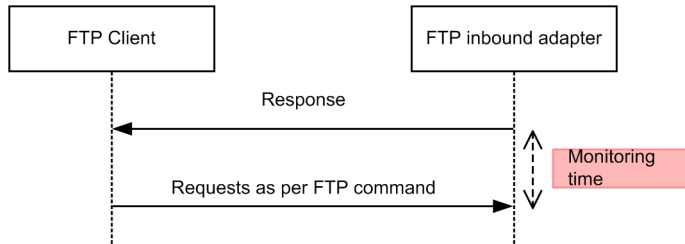
This subsection describes the operations when a timeout occurs and the points to be considered for each point.

(a) Idle timeout

Monitors the time from the return of a response from the FTP inbound adapter until the next FTP command is sent from the FTP client, for a control connection.

The following figure shows the procedure of monitoring idle timeout:

Figure 8–16: Procedure of monitoring idle timeout



The operations when the timeout occurs and points to be considered are as follows:

Operations when the timeout occurs

- A message is output.
- The control connection that timed out is disconnected.

Points to be considered

The range of idle timeout monitoring during the sending and receiving of data in a data connection is as follows:

1. Time from the receipt of transfer requests from the FTP client, transition of control to the FTP reception, and until the FTP inbound adapter completes the connection establishment request
2. Time from completion of sending and receiving of file data (or list data) with the FTP client, transition of control to the FTP reception, and until the FTP inbound adapter returns a transfer completed response

If the total time in 1 and 2 above exceeds the idle timeout monitoring value, an idle timeout occurs. However, the idle timeout occurs after the sending and receiving of data ends (after the transfer completed response is sent).

Note that idle timeout is not monitored for the time from the completion of connection establishment request, until the completion of sending and receiving of the file data (or list data).

The following figures show the idle timeout during data sending, and idle timeout during data receiving in a data connection:

Figure 8–17: Idle timeout during data sending

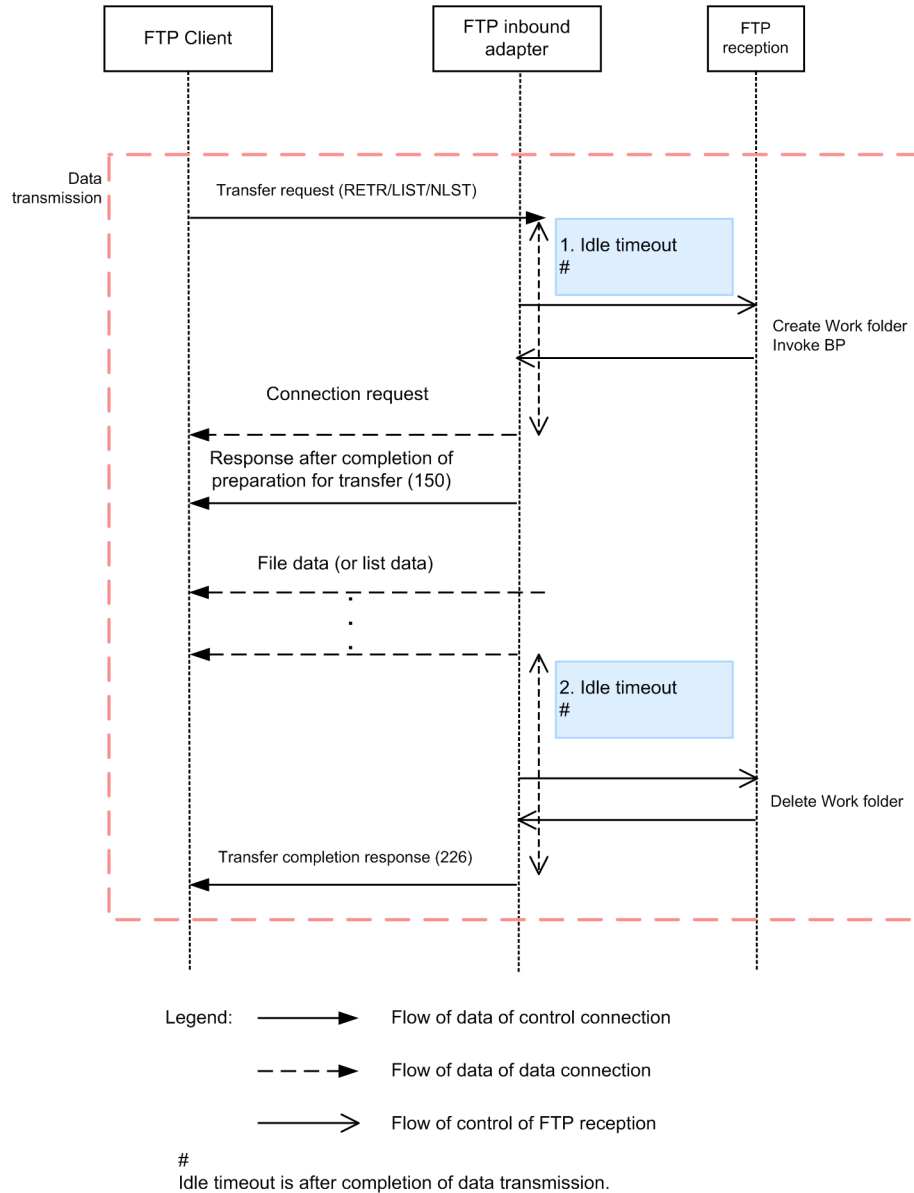
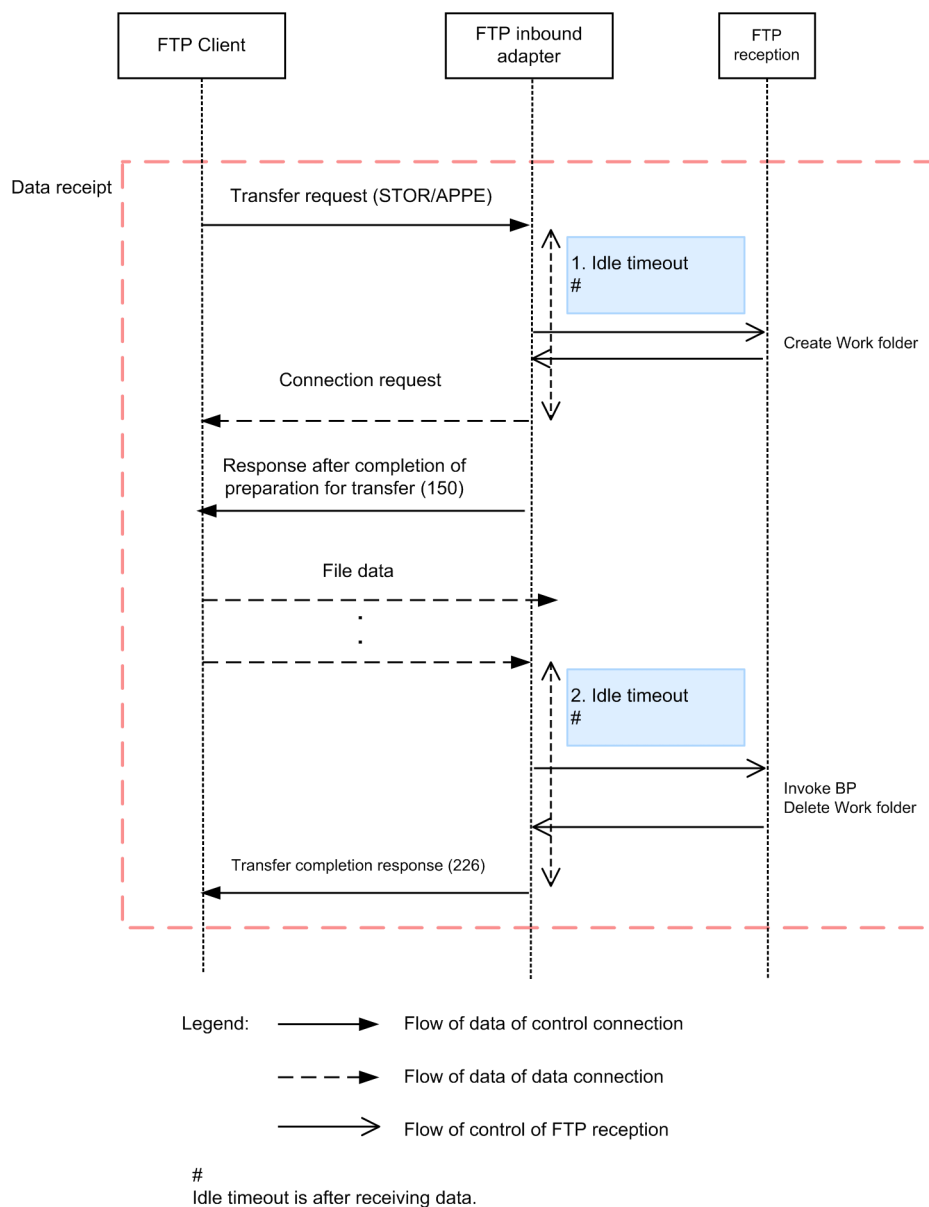


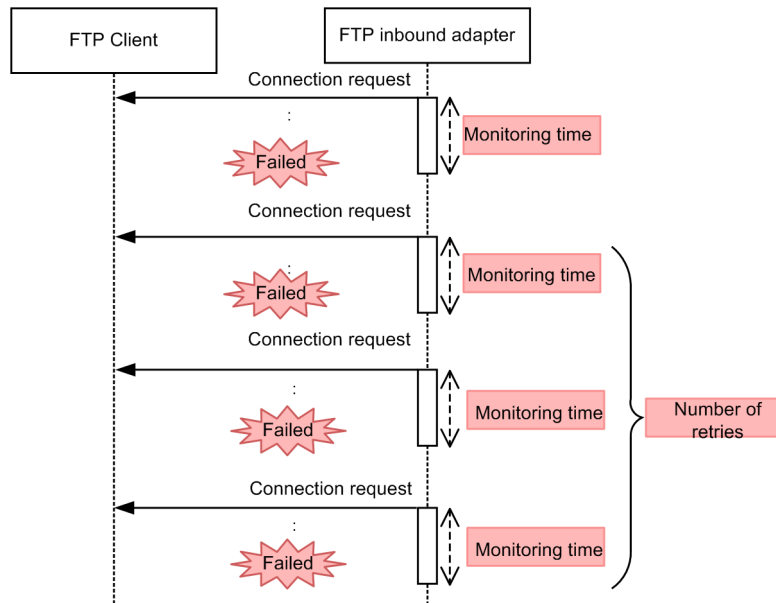
Figure 8–18: Idle timeout during data receiving



(b) Timeout in establishing a data connection

If the connection method of data connection is the Active mode, the time is monitored from the start to the completion of the establishment of data connection to the FTP client. Specify the monitoring time with the connection monitoring time and retry count. The following figure shows the procedure of monitoring the timeout in establishing a data connection:

Figure 8–19: Procedure of monitoring the timeout in establishing a data connection



The operations when the timeout occurs and points to be considered are as follows:

Operations when the timeout occurs

- A message is output.
- The response code "425 can't open data connection." is returned with the control connection.

Points to be considered

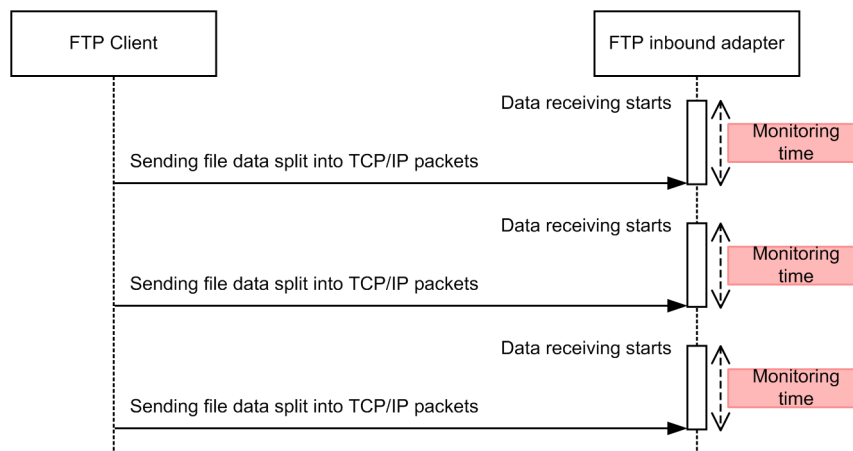
If the monitoring time for establishing connection set up in TCP/IP is shorter than the specified monitoring time, a timeout occurs in the monitoring time set up in TCP/IP. Also, even if a timeout occurs in the monitoring time set up in TCP/IP, a connection is retried only for the specified retry count.

(c) Timeout in data reception

The time is monitored from when the file data divided into TCP/IP packets can be received from the FTP client, until the file data divided into TCP/IP packets is sent.

The following figure shows the procedure of monitoring timeout in data reception:

Figure 8–20: Procedure of monitoring timeout in data reception



The operations when the timeout occurs are as follows:

Operations when the timeout occurs

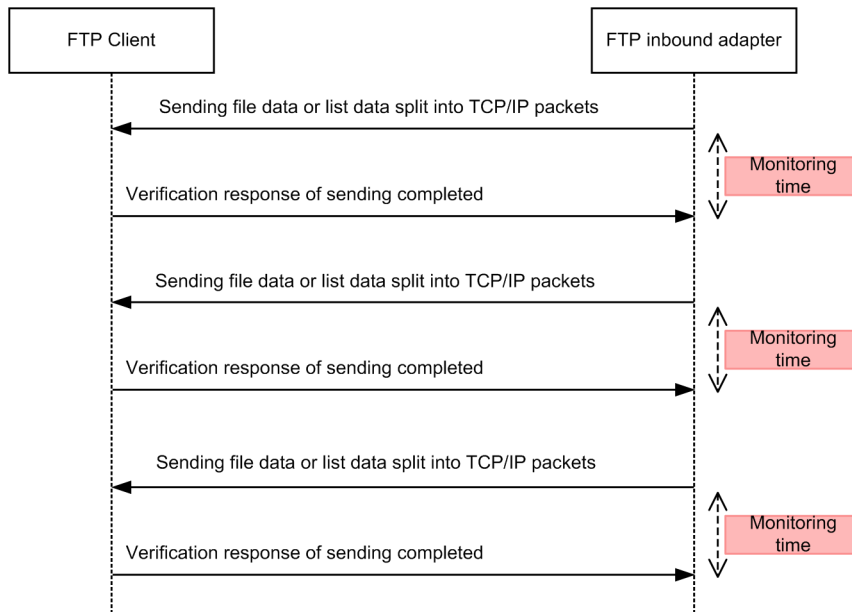
- A message is output.
- The data connection where the timeout occurred is disconnected.
- The response code "551 <File path>: Error on output file." is returned with the control connection.
- The exception "FileTransferFailedException" is reported to the FTP reception.

(d) Timeout in data transmission

The time is monitored from the start of transmission of the file data or list data divided into TCP/IP packets to the FTP client, until a confirmation response is received for the completion of transmission of the file data or list data divided into TCP/IP packets.

The following figure shows the procedure of monitoring the timeout in data transmission:

Figure 8–21: Procedure of monitoring the timeout in data transmission



The operations when the timeout occurs are as follows:

Operations when the timeout occurs

- A message is output.
- The data connection where the timeout occurred is disconnected.
- The following response codes are returned with the control connection:
 - In the case of file data transfer
"551 <File path>: Error on input file." is returned.
 - In the case of list data transfer
"551 Requested action aborted." is returned.
- The exception "FileTransferFailedException" is reported to the FTP reception.

(2) Setting method

Set up the following items in the HITACHI Connector Property file that is obtained with the cjgetrarprop command:

- nioListener_idleTimeout (Idle timeout)
- nioListener_dataConnection_active_connectRetryCount (Retry count for establishing data connection)
- nioListener_dataConnection_active_connectRetryInterval (Retry interval for establishing data connection)
- nioListener_dataConnection_idleTimeout (Timeout in data transmission and data reception)

For details on the setting method, see "3.2.3 Setting up the FTP inbound adapter" in "Service Platform System Setup and Operation Guide".

8.6 FTP adapter functionality

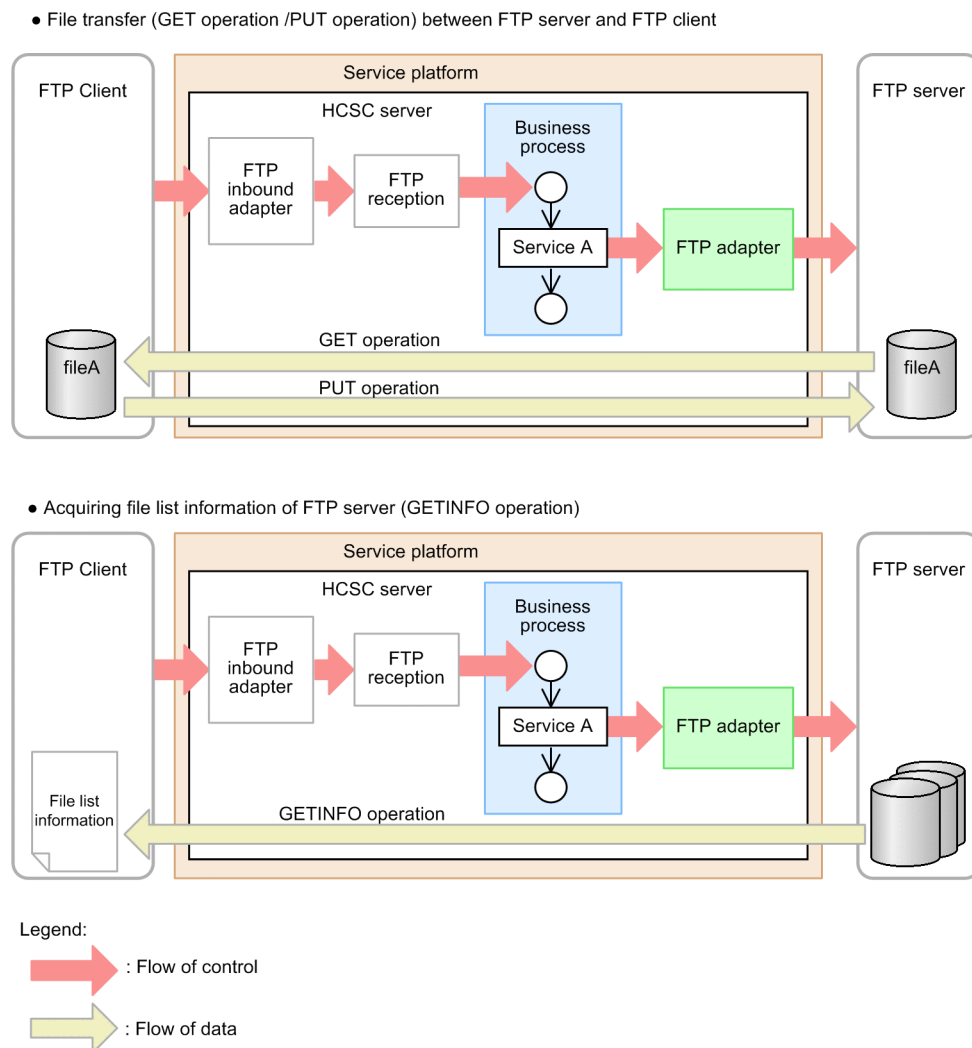
This section describes the role of an FTP adapter, and the procedure of sending and receiving files using an FTP adapter.

An FTP adapter is a service adapter that is used for sending and receiving files between an FTP server and Service Platform.

In a system using FTP integration, an FTP adapter is required for each FTP server you want to connect to, in order to send and receive files between the FTP client and FTP server.

The following figure shows the procedure of sending and receiving files using an FTP adapter:

Figure 8–22: Procedure of sending and receiving files using an FTP adapter



The following is a description of the flow of control and data in the figure:

When transferring files from an FTP server to an FTP client (GET operation)

If you execute the file acquisition command from the FTP client, the command execution request is delivered to the FTP reception via the FTP inbound adapter. Thereafter, the business process is started and the FTP adapter is invoked from the business process.

The FTP adapter, which receives the execution request from the business process, connects to the FTP server and sends the file to the FTP client.

When transferring files from an FTP client to an FTP server (PUT operation)

If you execute the file transmission command from the FTP client, the command execution request is delivered to the FTP adapter in the same way as in the GET operation. Thereafter, the FTP adapter connects to the FTP server, and sends the files from the FTP client to the FTP server.

When obtaining the file list information of the FTP server and sending the information to the FTP client (GETINFO operation)

If you execute the information acquisition command (LIST command or NLST command) from the FTP client, the list of information on the files existing on the FTP server or the list of file names is obtained in accordance with the executed command.

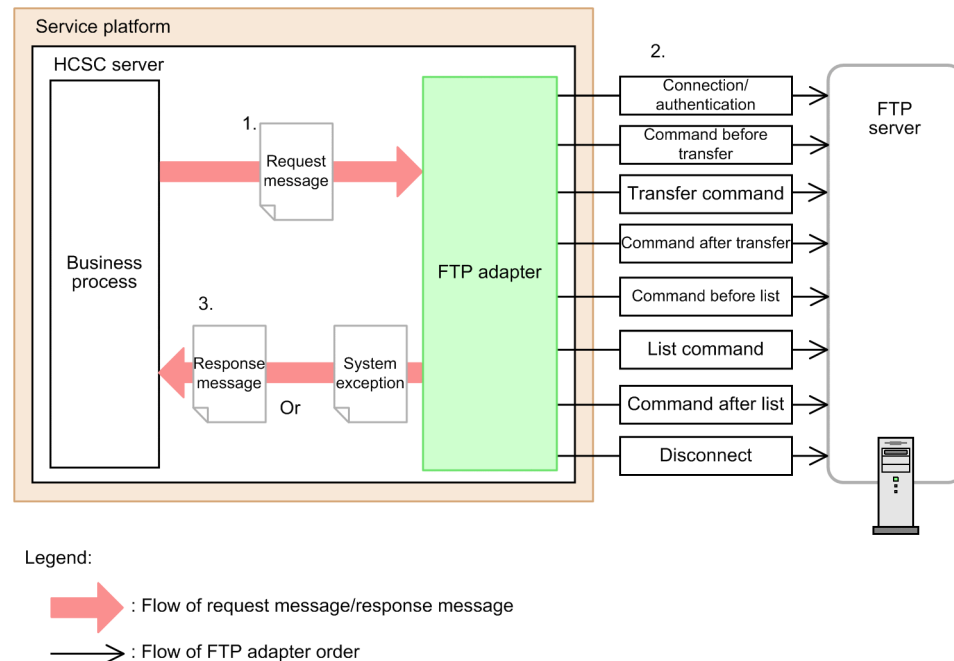
For details on the procedure of sending and receiving files in the entire FTP integration system, see "8.3 Examples of file transfer patterns using FTP integration".

8.6.1 Flow of FTP adapter processing

This subsection describes the flow of processing during the FTP adapter-based sending and receiving of files.

The following figure shows the flow of FTP adapter processing when execution requests are received from a business process:

Figure 8–23: Flow of FTP adapter processing



1. The FTP adapter is invoked on receiving an execution request from the business process. At this time, the file transfer information is stored in the request message and passed to the FTP adapter.
2. The FTP adapter that receives the request message executes processing such as connection or authentication, transfer commands, list commands, and disconnection for the FTP server.
3. After the invocation of the FTP server is complete, the FTP adapter sets the results in a response message, and returns the message to the business process.
If an error occurs, the FTP adapter returns a system exception instead of the response message.

8.6.2 Executing the FTP commands

This subsection describes the types, execution conditions, and timer management of the FTP commands executed by the FTP adapter, as the FTP adapter functionality.

After receiving an execution request from the business process, the FTP adapter executes FTP commands, such as authentication or connection, and transfer commands, for the FTP server in accordance with the conditions.

The following table describes the types and execution conditions of the FTP commands executed by the FTP adapter:

Table 8–5: Types of FTP commands and executability of the commands

Classification	FTP command	Executability of the command
Connection or authentication	(connect processing)	M
	AUTH	A ^{#1}
	USER	M
	PASS	M
	ACCT	A ^{#2}
Protection levels	PBSZ	A ^{#3}
	PROT	A ^{#3}
Transfer parameters	TYPE	A ^{#2}
	MODE	A ^{#2}
	PASV, PORT	Y ^{#4}
Checking file size	SITE FSIZE	A ^{#5}
Commands before and after transfer, and commands before and after list ^{#6}	MDTM, SIZE, STAT, DELE, MKD, XMKD, RMD, XRMD, PWD, XPWD, CDUP, XCUP, CWD, XCWD, NOOP, SITE, RNFR, RNTD	A ^{#7}
Transfer or information acquisition	STOR, APPE, RETR, LIST, NLST	Y ^{#2}
Disconnect	QUIT	M

Legend:

M: Must be executed. Invoke

Y: Any one is executed.

A: Depends on the conditions.

#1

Executed when FTPS is used, and the mode is the Explicit mode. For details on the FTPS-related settings, see "Appendix H.1 Secure connection using FTPS (FTP adapter)" in "Service Platform Reception and Adapter Definition Guide".

#2

The command is executed depending on the specified request message format. For details on the execution conditions, see "3.3.10 Defining an FTP adapter" in "Service Platform Reception and Adapter Definition Guide".

#3

Executed when FTPS is used.

#4

Depends on the settings in the ftpadp.pasvport property of the FTP-adapter execution-environment property file. For details on the FTP-adapter execution -environment property file, see "FTP-adapter execution-environment property file" in "Service Platform Reference Guide".

#5

For details on the execution conditions of the SITE FSIZE command, see "8.6.4 Checking the file size after sending and receiving files".

#6

This command is executed before or after executing the file transfer commands, or before or after executing the list commands.

#7

These commands are executed when execution is specified in the <ftp-commands-before> tag and <ftp-commands-after> tag of the request message format.

! Important note

The specifications of the FTP commands executed by the FTP adapter comply with RFC959, RFC2228, and RFC3659. However, there are differences in functionality in the RFC specifications and FTP adapter specifications for the following commands:

- TYPE
You cannot specify EBCDIC for the TYPE command of the FTP adapter.
- MODE
You cannot specify the block mode for the MODE command of the FTP adapter.
- LIST
You can specify options in the command arguments. However, the destination FTP server must support the specification of options.
- NLST
You can specify options in the command arguments. However, the destination FTP server must support the specification of options.

8.6.3 Folder specification used for sending and receiving files

In the PUT operation and GET operation, files are sent and received by using the work folder managed by the FTP reception in the HCSC server, or the common folder that is used in common in a cluster configuration. In the GETINFO operation, apart from the work folder and common folder, you can also output the file list to the response message.

You can specify the output destination with the FTP adapter request message. The following tables describe the output destination types and specification methods for each operation:

- In the PUT operation and GET operation

Output destination	Specification method
Common folder	Specify common="true" in the attribute of the <local-folder> tag of the request message.
Work folder	Specify common="false" in the attribute of the <local-folder> tag of the request message.

- In the GETINFO operation

Output destination	Specification method
Common folder	Specify common="true" in the attribute of the <local-folder> tag of the request message.
Work folder	Specify common="false" in the attribute of the <local-folder> tag of the request message, and specify the <request-id> tag.
Response message	Omit the <local-folder> tag and <request-id> tag of the request message.

For the contents of the request message format of the FTP adapter, see "3.3.10 Defining an FTP adapter" in "Service Platform Reception and Adapter Definition Guide".

8.6.4 Checking the file size after sending and receiving files

This subsection describes the functionality for checking the file size after sending and receiving files.

(1) Overview of file size checking

If an FTP server uses JP1/File Transmission Server/FTP, you can confirm that the size of files sent and received between the FTP server and an FTP adapter is correct on both the sending side and receiving side.

If a file was corrupted during a transfer but the response code from the FTP adapter indicates that the transfer ended normally, the user cannot see that the file is corrupted. In this case, the FTP adapter compares the file size at the transfer source with the file size at the destination. If the file sizes are different, the FTP adapter returns a system exception (transfer error) to the business process.

Note that if a file is transferred in ASCII mode between OSs for which the number of bytes in the linefeed code is different, the file size changes because the linefeed code is transformed automatically. In this case, however, no error occurs.

For details about the functionality for checking the file size by using JP1/File Transmission Server/FTP, see the manual *JP1/File Transmission Server*.

(2) Setting method

To enable file size checking, you must specify `TRUE` for the `ftpadp.file-size-check` property in the FTP-adapter runtime-environment property file.

For details about the FTP-adapter runtime-environment property file, see *FTP-adapter runtime-environment property file* in the *Service Platform Reference Guide*.

(3) Note

If `TRUE` is specified for the `ftpadp.file-size-check` property in the FTP-adapter runtime-environment property file when the FTP server does not use JP1/File Transmission Server/FTP, the FTP adapter returns an exception and terminates request processing.

8.6.5 Timer management

This subsection describes the timer management functionality of the FTP adapter, and the method of setting up the timeout value.

In the FTP adapter, you can set up the following timeout:

1. Timeout in establishing a control connection
2. FTP command reply timeout
3. Timeout in obtaining the lock
4. Timeout in establishing a data connection
5. Timeout in data transfer or acquisition
6. Listen timeout

You can specify each timeout value in the FTP-adapter execution-environment property file.

For details on the FTP-adapter execution-environment property file, see "FTP-adapter execution-environment property file" in "Service Platform Reference Guide".

(1) Timeout in establishing a control connection

The execution of commands between the FTP adapter and FTP server and its response are communicated by establishing a control connection.

If there is a request to establish a control connection from the FTP adapter, and the connection cannot be established because of errors such as other party busy, the connection processing is retried for the specified count at the specified interval. When the connection processing is retried, the retry status is output to the maintenance log.

The following table describes the parameters to be set up:

Table 8–6: Settings for timeout in establishing a control connection

Value	Key name	Default value
Retry count for establishing a control connection (times)	<code>ftpadp.control-con.retry.count</code>	5
Retry interval for establishing a control connection (seconds)	<code>ftpadp.control-con.retry.interval</code>	2
Timeout value for establishing one control connection (seconds)	<code>ftpadp.control-con.connect.timeout</code>	10

The timeout value is obtained with the following formula using the values specified for these parameters:

(Timeout value for establishing one control connection + Retry interval for establishing a control connection) x Retry count for establishing a control connection

(2) FTP command reply timeout

If there is no response even after the specified time elapses after issuing an FTP command from the FTP adapter, an exception is thrown. By setting up the FTP command reply timeout, you can set up the time until the exception is thrown.

The following table describes the parameters to be set up:

Table 8–7: Settings for FTP command reply timeout

Value	Key name	Default value (seconds)
FTP command reply timeout value	ftpadp.ftp-command-reply.timeout	60

(3) Timeout in obtaining the lock

When you transfer a file from the common folder to the FTP server with the PUT operation, a shared lock is set up for the file in the common folder.

Furthermore, when you obtain the file from the common folder with the GET operation, a dedicated lock is set up for the file in the common folder.

When the shared lock or dedicated lock is being set up and if the settings fail due to a conflict with other processes, the lock acquisition processing is retried for the specified count at the specified interval. The timeout value is obtained by adding the product of the retry interval and retry count to the file access time for obtaining each lock.

The following table describes the parameters to be set up:

Table 8–8: Settings for timeout in obtaining the lock

Value	Key name	Default value
Shared lock retry count (times)	ftpadp.read-lock.retry.count	0
Shared lock retry interval (seconds)	ftpadp.read-lock.retry.interval	1
Dedicated lock retry count (times)	ftpadp.write-lock.retry.count	0
Dedicated lock retry interval (seconds)	ftpadp.write-lock.retry.interval	1

When the file is output to the common folder with the GETINFO operation, and if a file with the same name already exists, the file is overwritten. If the file with the same name is being used in another process, the processing is retried at the timeout value specified in the dedicated lock retry count and dedicated lock retry interval.

(4) Timeout in establishing a data connection

The transfer of data between the FTP adapter and FTP server is communicated by establishing a data connection.

In the case of the PASV connection mode, if there is a request to establish a data connection and the connection cannot be established due to errors such as other party busy, the connection processing is retried for the specified count at the specified interval. When the connection processing is retried, the retry status is output to the maintenance log.

In the case of the PORT connection mode, the timeout in establishing a data connection depends on the FTP server specifications; therefore, you cannot specify the timeout in the FTP adapter.

The following table describes the parameters to be set up:

Table 8–9: Settings for timeout in establishing a data connection

Value	Key name	Default value
Retry count for establishing a data connection in the PASV mode (times)	ftpadp.data-con.retry.count	5

Value	Key name	Default value
Retry interval for establishing a data connection in the PASV mode (seconds)	ftpadp.data-con.retry.interval	2
Timeout value for establishing one data connection (seconds)	ftpadp.data-con.connect.timeout	10

The timeout value is obtained with the following formula using the values specified for these parameters:

(Timeout value for establishing one data connection + Retry interval for establishing a data connection in the PASV mode) x Retry count for establishing a data connection in the PASV mode

(5) Timeout in data transfer or acquisition

If data transfer or list data acquisition is not complete even after the specified time elapses after executing a data transfer command or list command from the FTP adapter, an exception is thrown. By setting up data transfer timeout and data acquisition timeout, you can specify the time until the exception is thrown.

When this exception occurs, the local files and remote files are handled as follows:

- Local files in which the RETR command, LIST command, or NLST command is being executed
The FTP adapter is deleted.
- Remote files in which the STOR command or APPE command is being executed
Depends on the FTP server specifications.

The following table describes the parameters to be set up:

Table 8–10: Settings for timeout in data transfer or acquisition

Value	Key name	Default value (seconds)
Data transfer timeout value	ftpadp.data-con-put.timeout	60
Data acquisition timeout value	ftpadp.data-con-get.timeout	60

(6) Listen timeout

When the connection mode is the PORT mode, after the FTP adapter receives the response of the PORT command, if the server does not make a data connection establishment request in the Listen state in the given period of time, an exception is thrown. By setting up the Listen timeout, you can specify the time until the exception is thrown.

The following table describes the parameters to be set up:

Table 8–11: Settings for Listen timeout

Value	Key name	Default value (seconds)
Listen timeout value	ftpadp.listen.timeout	60

8.7 File operations adapter functionality

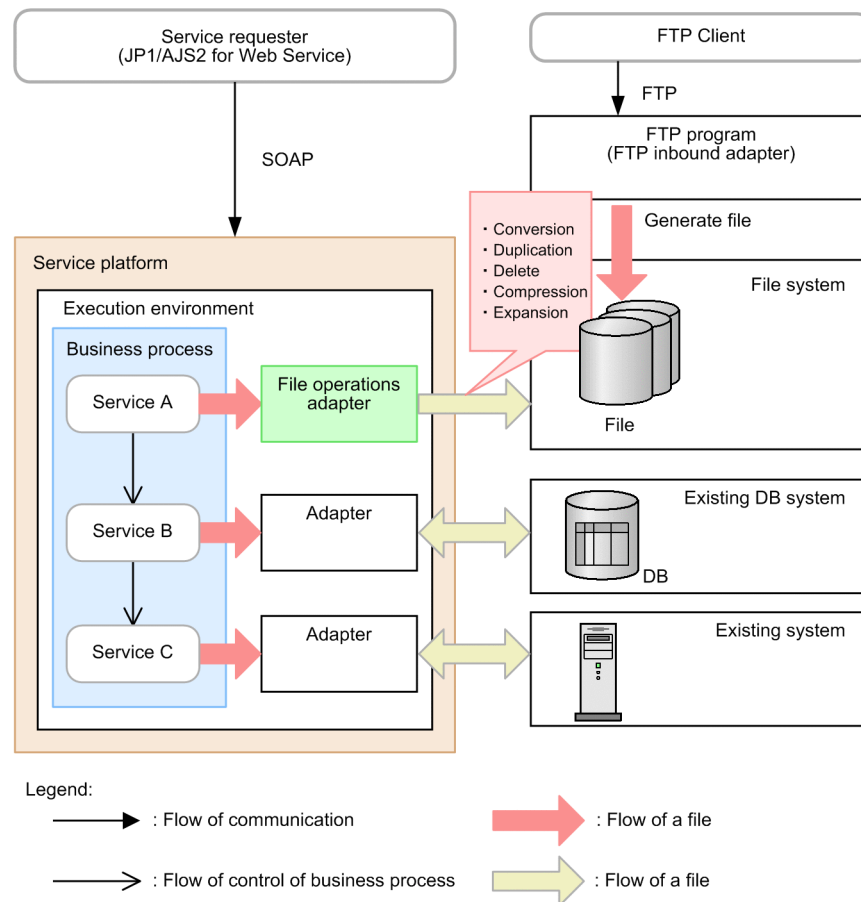
This section describes the configuration of the file operations adapter and the supported operations.

The file operations adapter is an adapter that is used for performing file operations such as layout conversion, character code conversion, replication, deletion, compression, and un-compression for files generated by using programs such as the FTP program. In the file operations adapter, the unit of file operations is called as operation.

When you use FTP integration, you must define a file operations adapter for each operation by using the FTP reception and FTP adapter.

The following figure shows the configuration of a file operations adapter:

Figure 8–24: Configuration of a file operations adapter



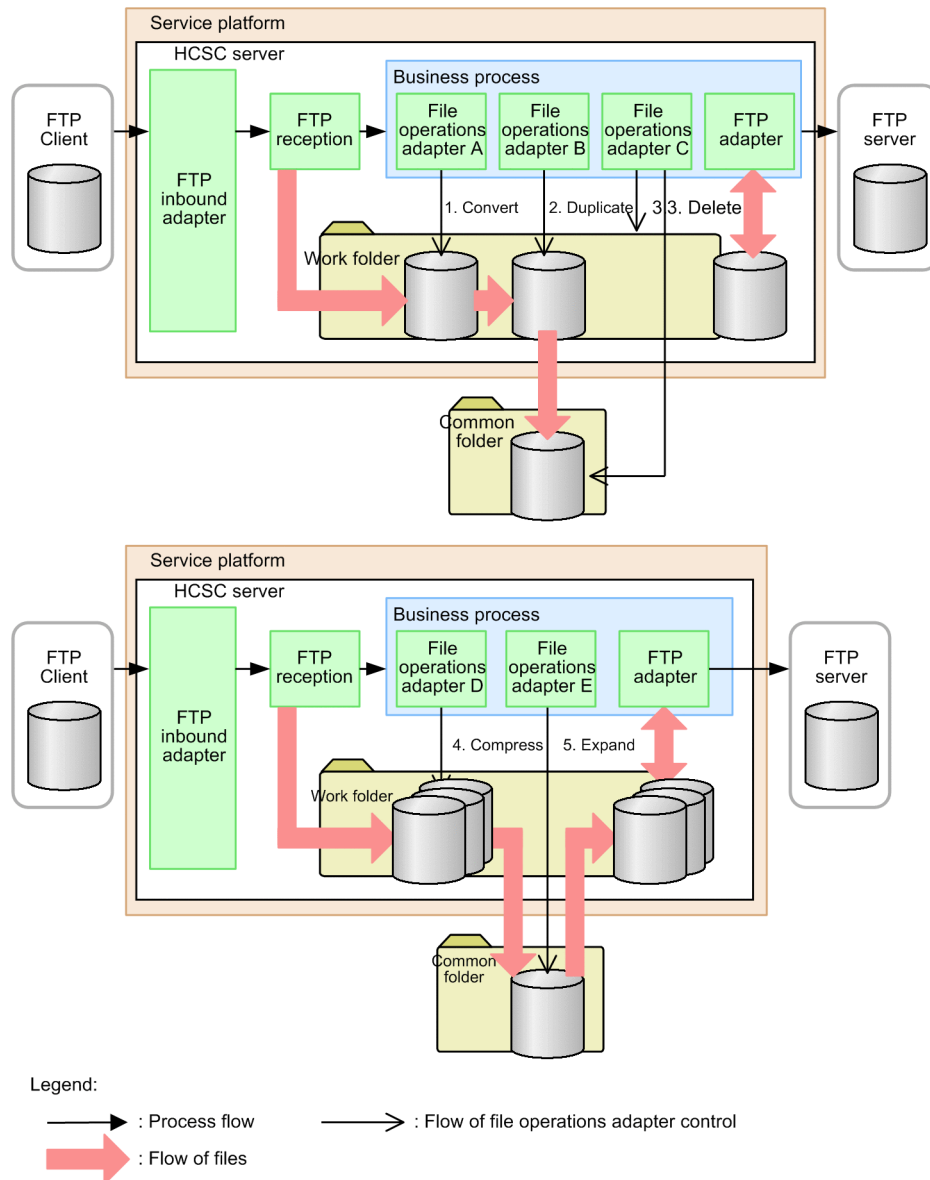
8.7.1 Operations supported by the file operations adapter

The operations supported by the file operations adapter include the following types:

- File conversion operation
- File replication operation
- File or folder deletion operation
- File compression operation
- File un-compression operation

The following figure shows the functioning of each operation:

Figure 8–25: Functioning of the file operations adapter



The following is a description of the points 1 to 5 in the figure. The following numbers correspond to the numbers in the figure:

1. The file operations adapter A (file conversion operation) reads the conversion source input file from the work folder (or common folder), and writes the post-conversion output file to the work folder (or common folder).
2. The file operations adapter B (file replication operation) reads the input file from the work folder (or common folder), and replicates the output file in the work folder (or common folder).
3. The file operations adapter C (file or folder deletion operation) deletes the files in the common folder, or the work folder.
4. The file operations adapter D (file compression operation) reads the compression source input file from the work folder (or common folder), and writes the post-compression output file in the work folder (or common folder).
5. The file operations adapter E (file un-compression operation) reads the un-compression source input file from the work folder (or common folder), and writes the output file after un-compression in the work folder (or common folder).

To use the file operations adapter to convert or delete files, you must define a file operations adapter for each operation.

You can also combine and use the file operations adapters of the respective operations.

8.7.2 Request and response messages for each operation

The message format of the request and response messages used in the file operations adapter differs for each operation.

For details on the message format of the request and response messages for each operation, see "3.3.11 Defining file operations adapters" in "Service Platform Reception and Adapter Definition Guide".

8.7.3 File conversion operation

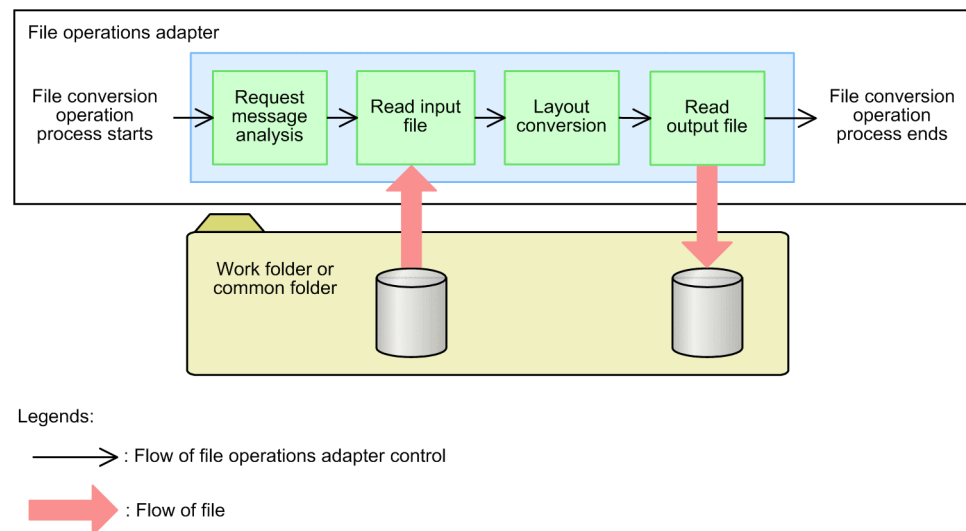
This subsection describes the functionality and file formats supported in the file conversion operation.

(1) Overview of functionality of the file conversion operation

The file conversion operation reads the conversion source input file from the work folder or common folder, and writes the post conversion output file to the work folder or common folder.

The following figure shows an overview of the processing of the file conversion operation:

Figure 8–26: Processing of the file conversion operation



(2) Supported I/O file formats

The following table describes the formats of the input and output files that are supported in the file conversion operation:

Table 8–12: File formats supported in the file conversion operation

No.	Processing method	Input file format		Output file format	Support
1	Batch processing method	Binary format		Binary format	Y
2				XML format	Y
3		XML format		Binary format	Y
4				XML format	Y
5	Partition processing method	Binary format	Fixed length format	Binary format	Y

No.	Processing method	Input file format		Output file format	Support
6	Partition processing method	Binary format	Fixed length format	XML format	Y
7			Length tag format	Binary format	N
8				XML format	N
9			linefeed separator format	Binary format	Y
10				XML format	Y
11			Other than above	Binary format	N
12				XML format	N
13		XML format		Binary format	Y
14				XML format	Y

Legend:

Y: Supported.

N: Not supported.

The differences between the fixed length, length tag, and linefeed separator formats are as follows:

- **Fixed length format**
In this format, you determine the byte length of the records in advance.
- **Length tag format**
This format includes numeric information indicating the length or number of data elements.
- **linefeed separator format**
In this format, the occurrence of a linefeed code (LF (0x0a) or CR+LF (0x0d 0x0a)) is considered as a record separator.

You can convert the layout when you convert the file. Also, if the I/O format is the binary format, you can also assign or delete a linefeed code and convert the character code.

Reference note

If the input file contains even 1-byte data, file conversion is executed (even if the data only contains null, linefeed character, or EOF character, file conversion is executed).

(3) Batch processing method

In the batch processing method, the input files are read and converted in a batch. After conversion, the data is written to the output file in a batch.

The target data format, and the contents to be converted in the batch processing method are as follows:

- **Target data format**
This is the range that can be defined in one binary format definition or XML schema.
- **Contents to be converted**
This is the range that can be defined in one data conversion definition file (extension: xsl).

! Important note

You cannot convert a file where the size of the input and output files is 500 MB or more. The operations are not guaranteed if you convert a file of 500 MB or more.

(4) Partition processing method

The reading of the input file, conversion, writing to the output file is processed for each record of the input file. Repetitive processing is executed until the end of the input file.

For examples of definition when a file is converted from the binary format to XML format, and from the XML format to binary format with the partition processing method, see "Appendix E Example for defining file operations adapter " in "Service Platform Reception and Adapter Definition Guide".

(a) Types of judgment methods for each record

The judgment method for each record varies depending on the input file format.

In the binary format

The judgment for each record includes the following methods:

- **Fixed length method**

This method specifies the byte length of the record in advance. Use this method when the input file is in the fixed length format.

- **Linefeed separator method**

This method considers the occurrence of the linefeed code (LF (0x0a) or CR+LF (0x0d 0x0a)) as a record separator. Use this method when the input file is in the linefeed separator format. For details on the linefeed separator method, see the description in "(d) Partitioning the records (when the input file is in the binary format)".

If the linefeed separator method is selected in the partition processing method, regardless of the binary format definition, the separation of records is determined on the occurrence of the linefeed code (LF (0x0a) or CR+LF (0x0d 0x0a)), so when the linefeed byte pattern of the character code existing in the conversion source file is not a linefeed code (examples of character code: UTF-16_BIG, UTF-16_LITTLE, KEIS+EBCDIC, KEIS+EBCDIK, and KEIS), you cannot perform normal processing in the partition processing method of the linefeed separator method.

Note that if you select CUSTOM, the processing depends on the data to be processed.

In the XML format

Specify the elements to be handled as records in the file operations adapter definition file.

Important note

The sizes of the input file and output file are not limited. However, a file, where the size of one record in the input file and output file is 500 MB or more, cannot be converted regardless of the input file format. The operations are not guaranteed if you convert a file where the size of one record is 500 MB or more. Also, if the size of one element configuring a record exceeds 500 MB or more, the operation is not guaranteed regardless of the input file format.

(b) Target data formats

The data formats of the input file subject to the partition processing method are the binary format definition and XML schema. The target data structure must be a structure in which the header record part, data record part, and trailer record part occur in this sequence.

The data record part is a data having a structure wherein the decided element sequence is continuously repeated. This structure is the same as the database table structure, the decided element sequence (lines in the binary format, and data beneath specific elements in the XML format) is called a record, and this is an image that is repeated n times. There are no restrictions on the number of elements in a record and the number of records.

The header record part and trailer record part are complex content elements containing any element, and the records are not repeated. Furthermore, you can omit these parts.

The following figures show the images of the data file structure and data record part:

Figure 8–27: Data file structure

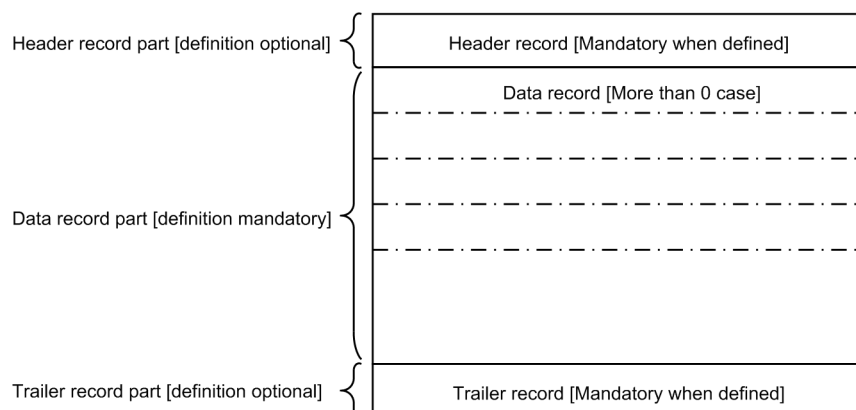


Figure 8–28: Image of the data record part

The diagram shows a table representing the data record part. The table has 6 rows and 4 columns. The first row is labeled with element names and lengths: A(5), B(10), C(4), and D(15). A bracket above the first four columns is labeled "Data record". A bracket to the right of the first four columns is labeled "n Record". An arrow points from the word "Element" to the first column. Ellipses (...) are shown to the right of the first row, indicating repetition.

A(5)	B(10)	C(4)	D(15)	...

Legend:

... : Repetition omitted

The data format corresponds to the binary format (fixed length format and linefeed separator format), and XML format. The input and output formats must have the structure of the above formats.

The following figures show the examples of data in the binary format:

Figure 8–29: Example of data in the fixed length format

The table shows a fixed-length record format. The first row is labeled "H E A D E R" and the last row is labeled "T R A I L E R". The data rows contain hexadecimal values. A bracket above the table indicates "Length of 1 record is fixed".

Length of 1 record is fixed															
H	E	A	D	E	R										
0	0	0	0	1	a	b	c								
0	0	0	0	2	d	e	f								
0	0	0	0	3	g	h	i								
0	0	0	0	4	j	k	l	m	n						
0	0	0	0	5	o	p	q								
T	R	A	I	L	E	R									

Figure 8–30: Example of data in the linefeed separator format

Separate 1 record by linefeed

The table shows a linefeed separator format. The first row is labeled "H E A D E R" and the last row is labeled "T R A I L E R". The data rows contain hexadecimal values, with a linefeed character (LF) at the end of each record. A bracket above the table indicates "Separate 1 record by linefeed".

Separate 1 record by linefeed															
H	E	A	D	E	R										
0	0	0	0	1	a	b	c								
0	0	0	0	2	d	e	f								
0	0	0	0	3	g	h	i								
0	0	0	0	4	j	k	l	m	n						
0	0	0	0	5	o	p	q								
T	R	A	I	L	E	R									

(c) Files required for the partition processing method

The binary format definition, XML schema, and data transformation definition that must be defined in the partition processing method are as follows:

Binary format definition and XML schema

The required definitions vary depending on the format of the input file and output file.

If the input file or output file is in the binary format

The file formats on the input side and output side must be defined, so use the binary format definition editor to create the required formats from among the following formats:

- Format of the header record part on the input side
- Format of the data record part on the input side
- Format of the trailer record part on the input side
- Format of the header record part on the output side
- Format of the data record part on the output side
- Format of the trailer record part on the output side

If the input file or output file is in the XML format

Create the XML schema for the input side and output side. Unlike the binary format, you can define the header record, data record, and trailer record in one XML schema. The XML schema is required for creating the data transformation definition, but is not used for executing the transformation operation.

Data transformation definition

You must define the data transformation between the formats on the input side and output side, so use the mapping definition editor to create the following data transformation definitions:

- Data transformation definition for the header record part
- Data transformation definition for the data record part
- Data transformation definition for the trailer record part

Note that if the output side is the XML format, you can perform mapping by limiting the items beneath the complex content element that specifies the mapping target. If the header record, data record, and trailer record are defined in one XML schema, you can generate the data transformation definition for each record part by limiting the items beneath the complex content element that forms the root element of each record and performing mapping. For details, see "6.4.8 Restricting mapping range " in "Service Platform Basic Development Guide".

(d) Partitioning the records (when the input file is in the binary format)

Judgment method for records

You can define record demarcation by selecting from fixed length format or linefeed separator format.

Specify the type of the fixed length format or linefeed separator format with the following parameter in the file operations adapter definition file:

- `csc.adapter.fileOperation.transform.format`

For the contents, see "File operations adapter definition file" in "Service Platform Reference Guide".

- For the fixed length method

The record demarcation is the position specified in the following parameters of the file operations adapter definition file:

- `csc.adapter.fileOperation.transform.header.fixedSize`
- `csc.adapter.fileOperation.transform.data.fixedSize`
- `csc.adapter.fileOperation.transform.trailer.fixedSize`

- For the linefeed separator method

The record demarcation is the position where the linefeed code (LF (0x0a) or CR+LF (0x0d 0x0a)) occurs in the input file. The end of file is also handled as the record demarcation. Note that if the file end occurs immediately after a linefeed, this is handled as one record separator along with the last linefeed.

Location and length of the record part

In the partition processing method, you must determine which record is currently being read from among the header record part, data record part, and trailer record part. The methods for determining the position and length of the record part for the fixed length format and linefeed separator format are as follows:

- In the fixed length method

The following table describes the determination for the position and length of the record part:

Table 8–13: Record types depending on the record presence specification (Fixed length method)

Record presence specification	Presence of header record part	Present	Present	Absent	Absent
	Presence of trailer record part	Present	Absent	Present	Absent
Position and length of the record part	Header record part	Top of file to Header record length	Top of file to Header record length	--	--
	Data record part	Header record part End position to Data record length * Number of data records	Header record part End position to Data record length * Number of data records	Top of file to Data record length * Number of data records	Top of file to Data record length * Number of data records
	Trailer record part	Data record part End position to Trailer record length	--	Data record part End position to Trailer record length	--

Legend:

--: Not applicable.

Obtain the number of data records with the following formula:

Data record part size =

Data file size - (Header record length + Trailer record length)

Number of records = Size of data record part / Data record length (rounded off after the decimal point)

If there is a remainder in the calculation results of the number of records, a message (KDEC80045-W) is output and then the processing continues. In this case, the possibilities are as follows, so check and eliminate the cause:

EOD exists at the end of the input file.

There is an error in the input file.

There is an error in the settings for the following items in the file operations adapter definition file:

csc.adapter.fileOperation.transform.headerRecord

csc.adapter.fileOperation.transform.trailerRecord

csc.adapter.fileOperation.transform.header.fixedSize

csc.adapter.fileOperation.transform.data.fixedSize

csc.adapter.fileOperation.transform.trailer.fixedSize

- In the linefeed separator method

The following table describes the determination for the position and length of the record part:

Table 8–14: Record types depending on the record presence specification (Linefeed separator method)

Record presence specification	Presence of header record part	Present	Present	Absent	Absent
	Presence of trailer record part	Present	Absent	Present	Absent
Position and length of the record part	Header record part	First line	First line	--	--
	Data record part	First line onwards to Just before last line	First line onwards to Last line	First line to Just before last line	First line to Last line
	Trailer record part	Last line	--	Last line	--

Legend:

--: Not applicable.

If the message (KDEC80047-E) is output and the processing is terminated, the possibilities are as follows, so check and eliminate the cause:

The number of records in the input file is not sufficient.

There is an error in the settings for the following items in the file operations adapter definition file:

`csc.adapter.fileOperation.transform.headerRecord`

`csc.adapter.fileOperation.transform.trailerRecord`

Table 8–15: Error determining pattern

Header record part	Trailer record part	Number of lines to be converted in the file
Yes	Yes	1 line or less
Yes	No	0 lines or less
No	Yes	0 lines or less

(e) Partition of records (When the input file is in the XML format)

To specify the record demarcation of the header record, data record, and trailer record for the input XML file, set up the path of the elements to be read in the file operations adapter definition file.

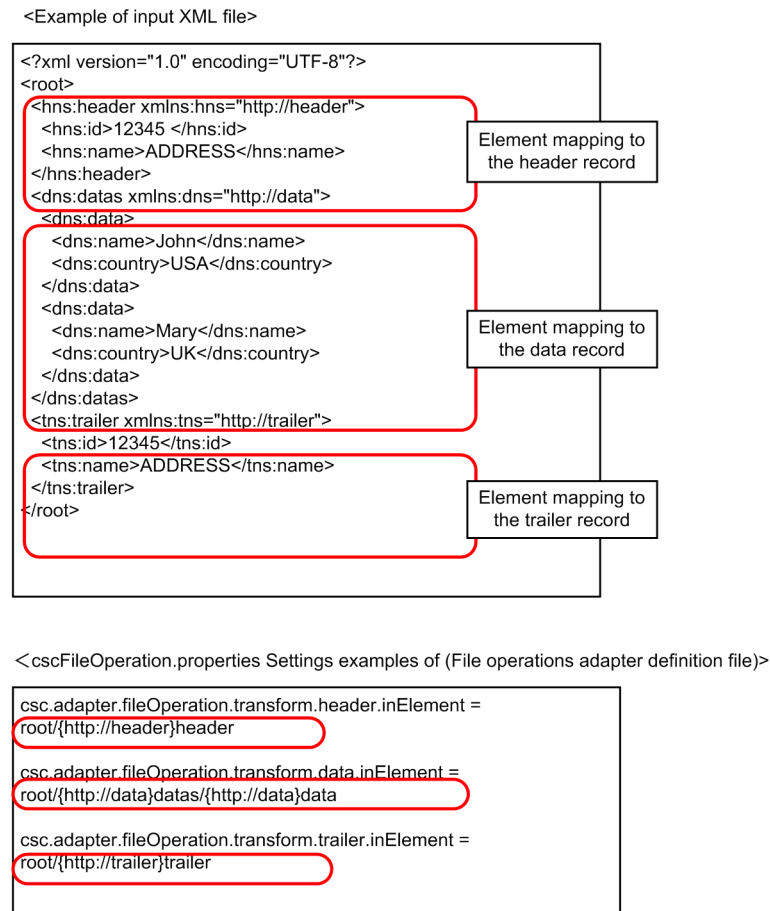
The following table describes the items and contents set up in the file operations adapter definition file:

Table 8–16: Settings in the file operations adapter definition file (Partition input in the input XML file)

Set items	Set contents
<code>csc.adapter.fileOperation.transform.header.inElement</code>	Specify the path of the elements corresponding to the header record in the input XML file. If a header record is not used, this item need not be specified (ignored even if specified).
<code>csc.adapter.fileOperation.transform.data.inElement</code>	Specify the path of the elements corresponding to the data record in the input XML file.
<code>csc.adapter.fileOperation.transform.trailer.inElement</code>	Specify the path of the elements corresponding to the trailer record in the input XML file. If a trailer record is not used, this item need not be specified (ignored even if specified).

The following figure shows the example settings of the input XML file and file operations adapter definition file:

Figure 8–31: Example settings of the input XML file and file operations adapter definition file



(f) Merging of records (When the output file is in the XML format)

In the partition processing method, a output template XML file is used for the input file in order to merge the conversion results of each record into one XML file.

The contents of the output template XML file must be formatted according to the output XML schema created in "(b) Target data formats". If the file is not an integral-form XML document, an error occurs during conversion.

The following table describes the specifications for the output template XML file:

Table 8–17: Specifications for the output template XML file

Item	Specifications
File name	File name specified in the csc.adapter.fileOperation.transform.outputTemplateXmlFile property of the file operations adapter definition file
Character encoding	UTF-8

Specific elements in the output template XML file are replaced in the records (output data), to create the output XML file.

Specify the elements replaced in the records in the file operations adapter definition file. The following table describes the items and contents set up in the file operations adapter definition file:

Table 8–18: Settings in the file operations adapter definition file (Merging the output XML file)

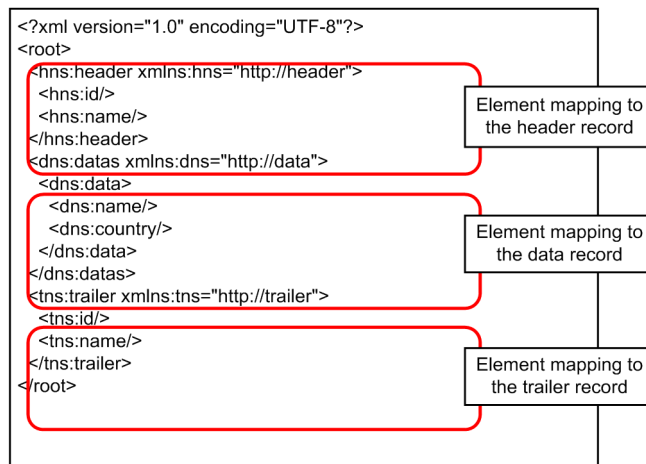
Set items	Set contents
csc.adapter.fileOperation.transform.header.outElement	Specify the path of the elements corresponding to the header record in the output XML file.

Set items	Set contents
csc.adapter.fileOperation.transform.header.outElement	If a header record is not used, this item need not be specified (ignored even if specified).
csc.adapter.fileOperation.transform.data.outElement	Specify the path of the elements corresponding to the data record in the output XML file.
csc.adapter.fileOperation.transform.trailer.outElement	Specify the path of the elements corresponding to the trailer record in the output XML file. If a trailer record is not used, this item need not be specified (ignored even if specified).

The following figure shows the example settings of the output template XML file, and file operations adapter definition file:

Figure 8–32: Example settings of the output template XML file and file operations adapter definition file

<Example of the template XML file for output>



<cscFileOperation.properties Settings examples of (File operations adapter definition file)>

```

csc.adapter.fileOperation.transform.header.outElement =
root/{http://header}header

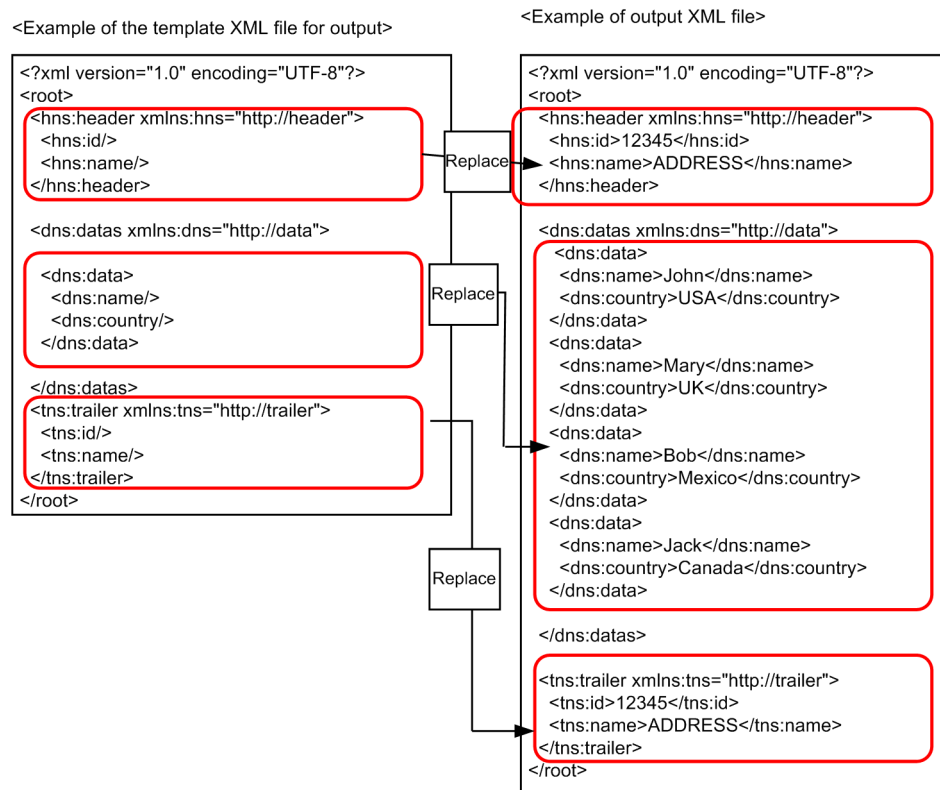
csc.adapter.fileOperation.transform.data.outElement =
root/{http://data}datas/{http://data}data

csc.adapter.fileOperation.transform.trailer.outElement =
root/{http://trailer}trailer

```

The following figure shows an example of converting the output template XML file into the output XML file. Note that the linefeed and spaces of the output XML file are displayed in this example, but in the actual output XML file, the linefeed and spaces between the elements are deleted.

Figure 8–33: Example of XML file output



The output XML file that uses the output template XML file is output in the format described in the following table:

Table 8–19: Format of the output XML file

Item	Specifications
Character encoding	UTF-8
Spaces between the elements	Deleted.
Linefeeds between the elements	Deleted.
Namespace declaration	<ul style="list-style-type: none"> The Namespace declaration of the output template XML file is inherited. Un-used Namespace declaration is not deleted. The missing Namespace declaration is supplemented with input XML declaration.
File contents	XML data is output according to the output XML schema. The compliance with the output XML schema is not validated.

! Important note

- If you specify character encoding that is not supported by XML Processor, an error occurs during parsing. For the character encoding supported by XML Processor, see "1.3.2 Character codes that can be processed" in "Application Server XML Processor User Guide".
- If elements not corresponding to any header, data, and trailer records are defined in the output template XML file, the elements are output as along with the value. Even if there are spaces before and after the value, the spaces are not removed.
- In the data record, the element occurring first, and the sibling elements occurring in continuation are handled as one data record group. After the data record group is terminated, there is an error if a data record occurs again.
- When the contents of the output template XML file are not formatted according to the output XML schema created in "(b) Target data formats", even if you specify the correct value existing in the output template XML file for the element replaced in the record, the record conversion results are not replaced and the processing is terminated. In this case, a runtime error does not occur. Therefore, even if the settings specify that the input data be output with the data

transformation definition, if the record conversion results are not output to the output file, review and, if necessary revise, the contents of the output XML schema and output template XML file.

(g) Simultaneous conversion of multiple data records

If the partition processing method is selected in the file conversion operation, you can collectively convert multiple data records as one element. You can perform the conversion whether the input file is in the binary format or XML format.

The following figures show examples where the input file is in the binary format, the total number of data records is 6, and two data records are converted as one element:

Figure 8–34: Example of converting two data records as one element (in the fixed length format)

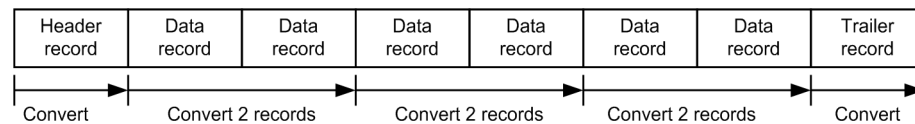
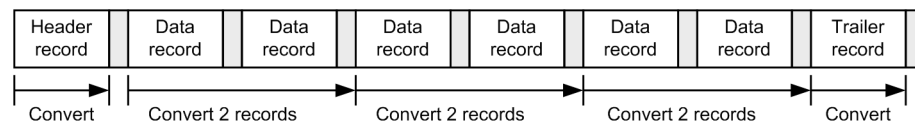



Figure 8–35: Example of converting two data records as one element (in the linefeed separator format)



Legend:  : Linefeed (LF or CR+LF)

Reference note

In the linefeed separator format, the conversion includes the linefeed of the data records.

Specify the settings for one element (number of records converted simultaneously) in the following parameter of the file operations adapter definition file:

- `csc.adapter.fileOperation.transform.data.batchSize`

For the contents of the parameter, see "File operations adapter definition file" in "Service Platform Reference Guide".

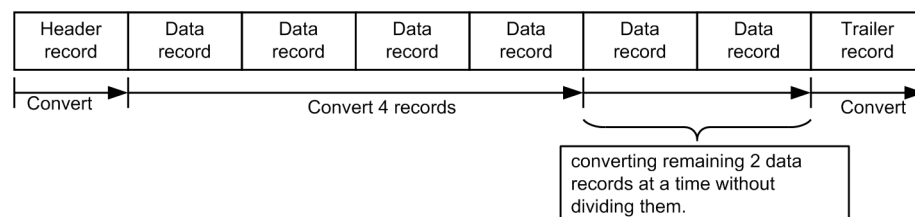
! Important note

If you convert multiple data records simultaneously, you improve the conversion efficiency, but if you enlarge one element, the processing performance might decline. Using 500 MB as a guide for the upper limit of data size per element, appropriately specify the number of records to be converted simultaneously.

If the total number of data records is indivisible by the value specified in the parameter, convert the remaining records at one time.

The following figure shows an example of conversion where the total number of data records is 6, and four data records are converted as one element:

Figure 8–36: Example of conversion when the records are indivisible by the value specified for the element



(h) Simultaneously converting multiple header records

When all the following conditions are fulfilled, you can collectively convert multiple header records as one element:

- The data type of the input file is a binary format.
- The partition processing method is specified.
- The separator format is specified in the record judgment method.
- The header record exists.

You can specify the number of header records to be converted in the `csc.adapter.fileOperation.transform.header.batchSize` property of the file operations adapter definition file.

In the following conditions, an error might occur during conversion:

- Many header records in the data
If the number of header records in the input data is more than the value specified in the `csc.adapter.fileOperation.transform.header.batchSize` property of the file operations adapter definition file, some of the header records are considered as data records, and a conversion error might occur.
- Few header records in the data
If the number of header records in the input data is less than the value specified in the `csc.adapter.fileOperation.transform.header.batchSize` property of the file operations adapter definition file, some of the data records are considered as header records, and a conversion error might occur.
- Total number of records in the input data is less than the specified records
If the total number of records in the input data is less than the value specified in the `csc.adapter.fileOperation.transform.header.batchSize` property of the file operations adapter definition file, an error (KDEC80047-E) occurs.

(5) How to specify the file formats to be converted

Specify the format of the files to be read or written in the file conversion operation by first creating a binary format definition file on the binary format definition screen, and then specifying the file format as a self-defined file of the file operations adapter.

For details on the binary format definition screen, see "1.2.1 Binary Format Definition Window" in "Service Platform Reference Guide".

For details on creating the binary format definition file, see "4.4 Creating Message Formats (Binary Format Definition File)" in "Service Platform Basic Development Guide".

For details on specifying the self-defined file of the file operations adapter, see "File operations adapter definition file" in "Service Platform Reference Guide".

(6) How to specify the data transformation definition

Specify the data transformation definition file (xsl file) to be converted from the pre-created input file to the output file, as the self-defined file of the file operations adapter.

To create a mapping definition file with the mapping definition editor for the development environment, use the data transformation definition to create the xsl file. For details on specifying the self-defined file of the file operations adapter, see "File operations adapter definition file" in "Service Platform Reference Guide".

(7) File conversion using character code conversion UOC

In the file conversion operation, you can use the character code conversion UOC to convert files.

When you use the character code conversion UOC, you must set up the following files for the self-defined file on the Service Adapter Settings window (details):

1. To use the same self-defined file for the conversion of the input file and output file
 - `csc_owncodeconvert.properties`
2. To use different self-defined files for the conversion of the input file and output file
 - `csc_owncodeconvert_in.properties`
 - `csc_owncodeconvert_out.properties`

If you specify the file described in 1 at the same time as the file described in 2, the file described in 1 is given priority and the file described in 2 is ignored.

For details on how to specify the settings in the Service Adapter Settings window (details), see the locations describing the file conversion operation in "3.3.11 Defining file operations adapters" in "Service Platform Reception and Adapter Definition Guide".

For an overview of the character code conversion UOC and the implementation method, see "Appendix I Character code conversion using character code conversion UOC" in "Service Platform Basic Development Guide".

(8) Specifying a self-defined file in the partition processing method

The following table describes the self-defined files required for converting the partition processing method in the file conversion operation, and the list of properties of the file operations adapter definition file that must be specified. For details on the file operations adapter definition file, see "File operations adapter definition file" in "Service Platform Reference Guide".

Table 8–20: Self-defined files and properties of file operations adapter definition file that must be specified in the partition processing method

Files specified as self-defined file		Input file		Output file	
		Binary	XML	Binary	XML
XSLT file for header layout conversion		Y#1	Y#1	Y#1	Y#1
XSLT file for data layout conversion		Y	Y	Y	Y
XSLT file for trailer layout conversion		Y#2	Y#2	Y#2	Y#2
FDX file for input header		Y#1	--	--	--
FDX file for input data		Y	--	--	--
FDX file for input trailer		Y#2	--	--	--
XML schema file for input header		--	--	--	--
XML schema file for input data		--	--	--	--
XML schema file for input trailer		--	--	--	--
Properties specified in the file operations adapter definition file	csc.adapter.fileOperation.transform.input (input file type)	Y	Y	--	--
	csc.adapter.fileOperation.transform.output (output file type)	--	--	Y	Y
	csc.adapter.fileOperation.transform.headerRecord (use of header records)	Y#1	Y#1	Y#1	Y#1
	csc.adapter.fileOperation.transform.trailerRecord (use of trailer records)	Y#2	Y#2	Y#2	Y#2
	csc.adapter.fileOperation.transform.header.inElement (name of input header element)	--	Y#1	--	--
	csc.adapter.fileOperation.transform.data.inElement (name of input data element)	--	Y	--	--
	csc.adapter.fileOperation.transform.trailer.inElement (name of input trailer element)	--	Y#2	--	--
	csc.adapter.fileOperation.transform.header.outElement (name of output header element)	--	--	--	Y#1

Files specified as self-defined file		Input file		Output file	
		Binary	XML	Binary	XML
Properties specified in the file operations adapter definition file	csc.adapter.fileOperation.transform.data.outElement (name of output data element)	--	--	--	Y
	csc.adapter.fileOperation.transform.trailer.outElement (name of output trailer element)	--	--	--	Y ^{#2}
	csc.adapter.fileOperation.transform.outputTemplateXmlFile (file name of output template XML file)	--	--	--	Y
FDX file for output header		--	--	Y ^{#1}	--
FDX file for output data		--	--	Y	--
FDX file for output trailer		--	--	Y ^{#2}	--
XML schema file for output header		--	--	--	--
XML schema file for output data		--	--	--	--
XML schema file for output trailer		--	--	--	--
Output template XML file		--	--	--	Y
Input property for character code conversion UOC (common I/O file) csc_owncodeconvert.properties		Y ^{#3}	--	Y ^{#3}	--
Input property for character code conversion UOC (for converting the input file) csc_owncodeconvert_in.properties		Y ^{#3}	--	--	--
Input property for character code conversion UOC (for converting the output file) csc_owncodeconvert_out.properties		--	--	Y ^{#3}	--

Legend:

Y: Must be specified.

--: Need not be specified.

#1

Specify this file or property only when a header record is used.

#2

Specify this file or property only when a trailer record is used.

#3

Specify this property when the character code conversion UOC is used.

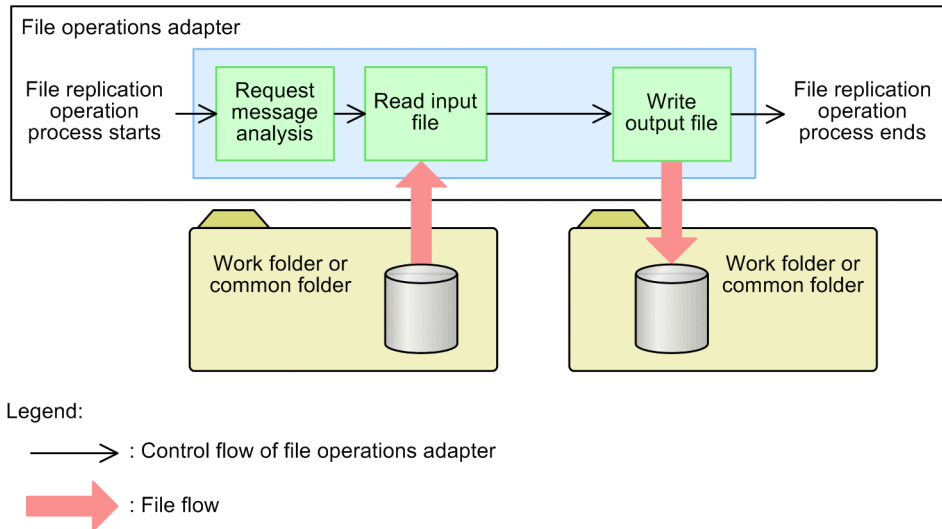
8.7.4 File replication operation

This subsection describes the functionality and file formats supported in the file replication operation.

(1) Overview of functionality of the file replication operation

The file replication operation reads the replication source input file from a work folder or common folder, and then writes the replicated output file to another work folder or common folder. The following figure shows an overview of processing of the file replication operation.

Figure 8–37: Processing of the file replication operation



(2) Supported file formats

The file replication operation can be used with any format of input and output files. All files that exist in a work folder or common folder can be replicated.

8.7.5 File or folder deletion operation

This subsection describes the functionality and file formats supported in the file or folder deletion operation.

(1) Overview of functionality of the file or folder deletion operation

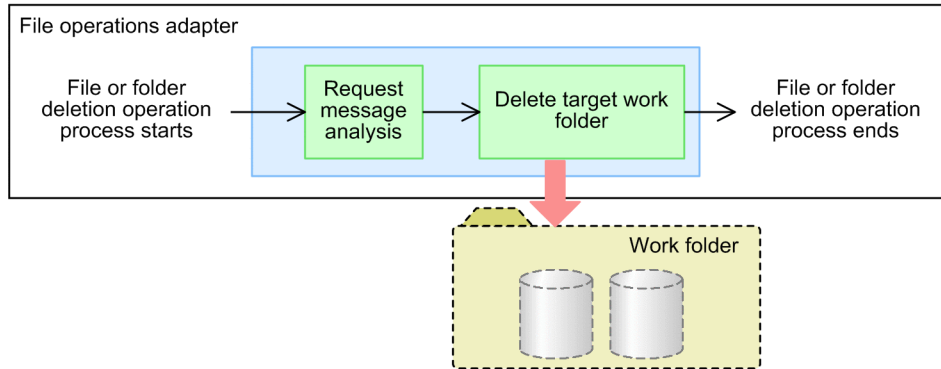
The file or folder deletion operation performs the following processing:

- Deleting a work folder
- Deleting a single file in a common folder

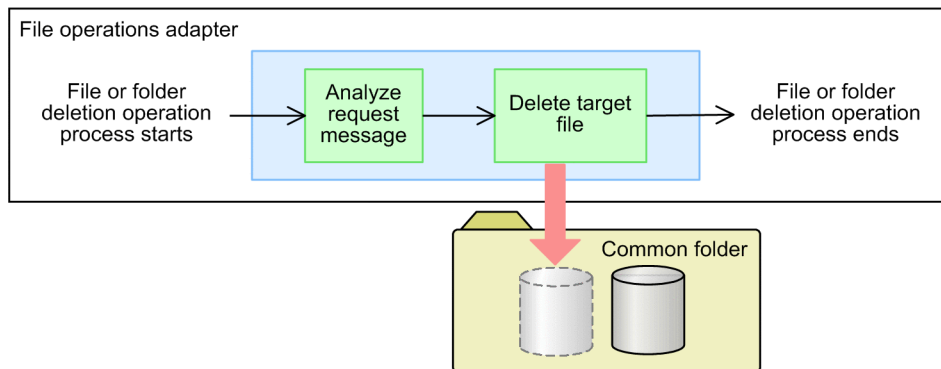
The following figure shows an overview of processing of the file or folder deletion operation.

Figure 8–38: Processing of the file or folder deletion operation

● When deleting a work folder:



● When deleting a single file in a common folder:



Legend:

→ : Control flow of file operations adapter

→ : File flow

[(File)] : File to be deleted

(2) Supported file formats

The file deletion operation can be used with any format of input and output files. Any file in a common folder can be deleted.

(3) Deleting a work folder

The folder deletion operation deletes the created work folder based on the request ID specified in the request message. All files and folders in the target work folder will be deleted.

8.7.6 File compression operation

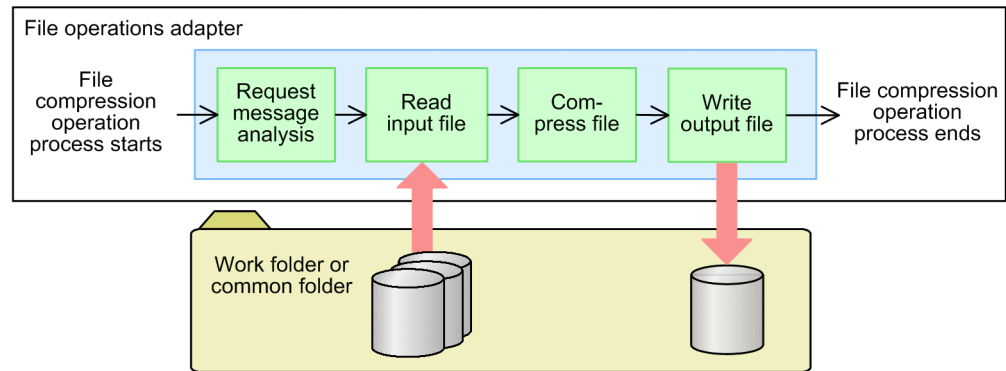
This subsection describes the functionality and file formats supported in the file compression operation.

(1) Overview of functionality of the file compression operation

The file compression operation reads the compression source input file from the work folder or common folder, and then writes the compressed output file to the work folder or common folder.

The following figure shows an overview of processing of the file compression operation.

Figure 8–39: Processing of the file compression operation



Legend:

→ : Control flow of file operations adapter

➡ : File flow

(2) Supported file formats

The table below shows the file formats supported in the file compression operation. Note that compression of directories is not supported. If a directory is specified as the input file, an error message (KDEC80035-E) is output and the file compression operation is canceled.

Table 8–21: File formats supported in the file compression operation

File format	Input file	Output file
ZIP format [#]	Y	Y
Other than the above	Y	N

Legend:

Y: Supported

N: Not supported

[#]

Only ZIP format files, compressed by using the DEFLATE compression algorithm, are supported.

The following shows the restrictions on input and output files used for the file compression operation.

Table 8–22: Restrictions on using the file compression operation

Item	Restriction
Input file name	<p>Only the following alphanumeric characters and symbols can be used:</p> <ul style="list-style-type: none"> Alphanumeric characters Alphabetic characters (a to z, A to Z) and numbers (0 to 9) can be used. Symbols The following symbols cannot be used: \\, /, : <p>If the file name contains a character that cannot be used, an error message (KDEC80022-E) is output and the file compression operation is canceled.</p>
Output file name	<p>The following symbols cannot be used: \\, /, :</p>

Item	Restriction
Output file name	If a file name contains a symbol that cannot be used, an error message (KDEC80022-E) is output and the file compression operation is canceled.
Output file entry name	The same restrictions as for the input file name apply.
Number of input files	You can specify from 1 to 999 input files. If this condition is not met, operation is not guaranteed. Multiple input files in a single input source folder can be compressed. However, multiple input source folders cannot be specified.
Input file size	The maximum input file size is 4 GB. If the file size exceeds the maximum, an error message (KDEC80105-E) is output and the file compression operation is canceled.
Output file size	The maximum output file size is 4 GB. If the file size exceeds the maximum, an error message (KDEC80106-E) is output and the file compression operation is canceled.

8.7.7 File expansion operation

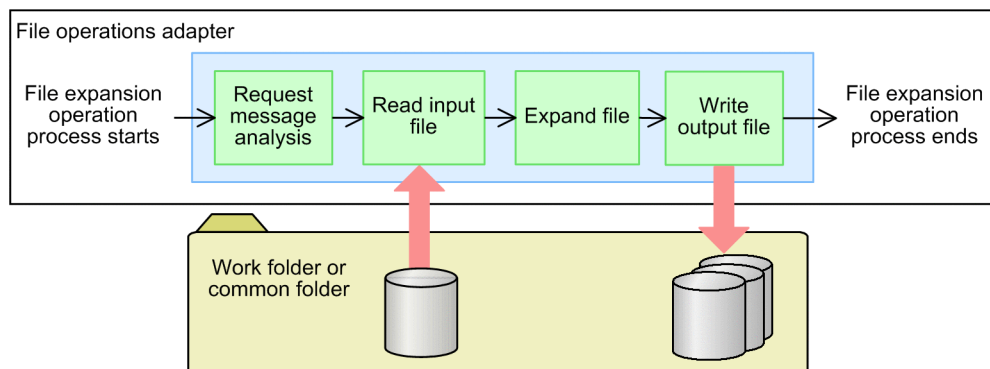
This subsection describes the functionality and file formats supported in the file expansion operation.

(1) Overview of functionality of the file expansion operation

The file expansion operation reads the expansion source input file from a work folder or common folder, and then writes the expanded output file to the work folder or common folder.

The following figure shows an overview of processing of the file expansion operation.

Figure 8–40: Processing of the file expansion operation



Legend:

→ : Control flow of file operations adapter

→ : File flow

(2) Supported file formats

The table below shows the file formats supported in the file expansion operation. Expansion of a ZIP file that contains a directory is not supported. If a ZIP file that contains a directory is specified as the input file, an error message (KDEC80123-E) is output and the file expansion operation is canceled.

Table 8–23: File formats supported in the file expansion operation

File format	Input file	Output file
ZIP format [#]	Y	Y
Other than the above	N	Y

Legend:

Y: Supported

N: Not supported

#

Only ZIP format files, compressed by using the DEFLATE compression algorithm, are supported.

The following shows the restrictions on input and output files used for the file expansion operation.

Table 8–24: Restrictions on using the file expansion operation

Item	Restriction
Input file name	<p>The following symbols cannot be used: \, /, :</p> <p>If a file name contains a symbol that cannot be used, an error message (KDEC80022-E) is output and the file expansion operation is canceled.</p>
Output file name	<p>Only the following alphanumeric characters and symbols can be used:</p> <ul style="list-style-type: none"> Alphanumeric characters Alphabetic characters (a to z, A to Z) and numbers (0 to 9) can be used. Symbols The following symbols cannot be used: \, /, : <p>If the file name contains a character that cannot be used, an error message (KDEC80123-E) is output and the file expansion operation is canceled.</p>
Output file entry name	<p>The same restrictions as for the input file name apply.</p> <p>The output file name is automatically assigned, and uses the prefix of the output file name defined in the request message.</p>
Number of output files	<p>The number of output files after expansion must be in the range of 1-999. If this condition is not met, an error message (KDEC80125-E) is output and the file expansion operation is canceled.</p> <p>Multiple output files can be expanded into a single folder. However, multiple output destination folders cannot be specified.</p>
Input file size	<p>The maximum input file size is 4 GB. If the file size exceeds the maximum, an error message (KDEC80126-E) is output and the file expansion operation is canceled.</p>
Output file size	<p>The maximum output file size is 4 GB. If the file size exceeds the maximum, an error message (KDEC80127-E) is output and the file compression operation is canceled.</p>

8.7.8 Managing files and folders used for each operation

You can use the file operations adapter to convert, replicate, and delete files by specifying work folders or common folders. Files can be input and output between work folders or between common folders. Files can also be input from a work folder and output to a common folder.

(1) Specification if files with the same name exist

For the following operations, you can specify whether to overwrite an existing file or output an error message if a file with the same name already exists in the output destination folder:

- File conversion operation
- File replication operation
- File compression operation
- File expansion operation

For details about the file operations adapter definition file, see *File operations adapter definition file* in the *Service Platform Reference Guide*.

(2) Setting up the lock acquisition timeout

If a file to be read is in a common folder, a shared lock for the file is acquired. If a file to be written or to be deleted is in a common folder, dedicated lock for the file is acquired.

If an attempt to acquire the lock fails, lock acquisition processing is retried (at the specified interval) up to the number of times specified for the retry count. The timeout value is obtained by adding the product of the retry interval and retry count to the file access time for acquiring each lock. To set up the timeout, specify the lock retry count and lock retry interval parameters in the file operations adapter runtime-environment property file. The following table describes the parameters to be specified.

Table 8–25: Lock acquisition timeout settings

No.	Value	Key name	Default value
1	Shared lock retry count	readLock.retry.count	0
2	Shared lock retry interval (seconds)	readLock.retry.interval	1
3	Dedicated lock retry count	writeLock.retry.count	0
4	Dedicated lock retry interval (seconds)	writeLock.retry.interval	1

For details about the file operations adapter runtime-environment property file, see *File operations adapter runtime-environment property file* in the *Service Platform Reference Guide*.

8.8 File access using the work folder and common folder

In FTP integration, you use the work folder and common folder to relay the transfer of files on Service Platform. This section describes the work folder and common folder.

8.8.1 File access using the work folder

The work folder is used to temporarily store the intermediate files managed Service Platform for each file transfer pattern.

The FTP reception generates a work folder for each request. For details on the work folder, see "[8.4.5 Creating the work folder and intermediate files](#)".

8.8.2 File access using the common folder

The common folder is used to store the commonly used files when multiple business processes reference a file, such as file transfer pattern from an FTP client to another FTP client.

(1) Creating the common folder

The user creates the common folder during system setup. You can use the created folder as the common folder by defining the folder as "name of common folder" in the HCSC server runtime definition file.

Note that you can create multiple common folders for one HCSC server.

(2) Accessing the common folder

The common folder can be accessed from all the business processes. However, the limitations are as follows:

- One common folder definition name must exist for one common folder.
- You can access the common folder only when you use the common folder definition name.
- You cannot access the subfolders in the common folder. Therefore, to access a subfolder, you must specify a common folder definition name for that folder.
- The reading and writing of files existing in the common folder is executed with the permission of the HCSC server Administrator.

(3) Defining the path corresponding to the common folder definition name

You can define multiple common folder definition names for the HCSC server.

For each common folder definition name, you must specify the full path indicating that common folder definition name. Code the common folder definition name in the common-folder-<common folder definition name> property of the HCSC server runtime definition.

For details on the HCSC server runtime definition file, see "HCSC server runtime definition file" in "Service Platform Reference Guide".

(4) Exclusive control of files in the common folder

To access the files in the common folder, obtain a file lock for both reading and writing.

(a) Lock types

In the access of files in the common folder, a shared lock is obtained for reading, and a dedicated lock is obtained for writing. The following table describes the possibility of locking a file when the current lock is a shared lock or dedicated lock. Note that another process indicates a process which is accessed by using the exclusive management APIs other than the locked process.

Table 8–26: Shared lock and dedicated lock

Current lock	Re-lock from another process/ thread	
	Shared lock	Dedicated lock
shared lock	Y	N
dedicated lock	N	N

Legend:

Y: Can be locked.

N: Cannot be locked.

(b) Parameters for obtaining locks

This subsection describes the following parameters that are used for obtaining the file lock:

Lock retry count

If the file lock has already been obtained by another process or thread, the file lock cannot be obtained. At this time, re-lock is tried only for the given lock retry count.

Lock retry interval

This is the time interval (seconds) from lock failure until the next lock acquisition attempt.

(5) Deleting the common folder and the files in the common folder

The user deletes the common folder. Delete the files in the common folder with the following methods:

- Delete operation of the file operations adapter
- Command for folder operations

For details on the method of using the delete operation of the file operations adapter, see "*8.7.1 Operations supported by the file operations adapter*".

For details on the method of using the commands for folder operations, see "*5.4.22 Deleting files in the common folder*" in "Service Platform System Setup and Operation Guide".

8.9 Scope of FTP integration

This section describes the FTP commands supported in FTP integration, the methods of specifying the FTP commands, and the points to be considered for using FTP integration.

8.9.1 Commands supported in FTP integration

The following table lists the commands supported in FTP integration.

Table 8–27: Supported FTP commands

Classification	Command	Option	Component				RFC
			FTP inbound adapter	FTP reception	Business process	FTP adapter	
Authentication	ACCT	--	N	--	--	Y	959
	USER	--	Y	Y	Y	Y	959
	PASS	--	Y	N	N	Y	959
Control	ABOR	--	N	--	--	N	959
	NOOP	--	Y	--	--	Y	959
	PASV	--	Y	--	--	Y	959
	PORT	--	Y	--	--	Y	959
	QUIT	--	Y	--	--	Y	959
	REIN	--	N	--	--	N	959
	REST	--	N	--	--	N	959
	EPRT	--	N	--	--	N	2428
	EPSV	--	N	--	--	N	2428
Session	CDUP	--	S	S	S	Y	959
	CWD	--	S	S	S	Y	959
	MODE	--	Y	--	--	Y	959
		S	Y	--	--	Y	--
		B	N	--	--	N	--
		C	N	--	--	N	--
	LPRT	--	N	--	--	N	1639
	LPSV	--	N	--	--	N	1639
	ADAT	--	N	--	--	N	2228
	AUTH	--	Y	--	--	Y	2228
	CCC	--	N	--	--	N	2228
	CONF	--	N	--	--	N	2228
	ENC	--	N	--	--	N	2228
	MIC	--	N	--	--	N	2228

8. Functionality to Transfer Files by Integrating with FTP (FTP Integration)

Classification	Command	Option	Component				RFC
			FTP inbound adapter	FTP reception	Business process	FTP adapter	
Session	PBSZ	--	Y	--	--	Y	2228
	PROT	--	Y	--	--	Y	--
	OPTS	--	Y	--	Y	Y	2389
File	TYPE	--	Y	Y	Y	Y	959
	LANG	--	N	--	--	N	2640
Transfer	APPE	--	Y	Y	Y	Y	959
	RETR	--	Y	Y	Y	Y	959
	STOU	--	N	--	--	N	959
	STRU	--	Y	Y	Y	Y	959
		F	Y	Y	Y	Y	--
		R#	Y	Y	Y	Y	--
		P	N	--	--	N	--
Update	ALLO	--	S	S	S	Y	959
	DELE	--	S	S	S	Y	959
	MKD	--	S	S	S	Y	959
	RMD	--	S	S	S	Y	959
	RNFR	--	S	S	S	Y	959
	RNTO	--	S	S	S	Y	959
	SMNT	--	S	S	S	Y	959
	STAT	--	S	S	S	Y	959
Reference	HELP	--	N	--	--	N	959
	LIST	--	Y	Y	Y	Y	959
	NLST	--	Y	Y	Y	Y	959
	PWD	--	S	S	S	Y	959
	SITE	--	Y	Y	Y	Y	959
	STOR	--	Y	Y	Y	Y	959
	SYST	--	S	S	S	S	959
	FEAT	--	N	--	--	N	2389
	MDTM	--	S	--	S	Y	3659
	MLSD	--	N	--	--	N	3659
	MLST	--	N	--	--	N	3659
	SIZE	--	S	--	S	Y	3659

Legend:

Y: Supported

S: Supported by the `SITE CSCTHR` command.

N: Not supported

--: Not applicable

#

If the `STRU R` command is sent from the FTP client, file conversion processing is not performed. In addition, data is not relayed to the FTP server. The operation is the same as for the `STRU F` command.

The following describes the classification of the FTP commands in the table.

(1) Authentication

Commands required for connecting sessions. These commands are used for access management in an HCSC server.

User names passed by the `USER` command must be inherited to a business process as attributes of files that are sent and received.

(2) Control

Commands relating to control such as connecting and releasing sessions. There is no need to inherit information to the FTP adapter via a business process.

(3) Session

These commands are used to set information that is effective within a session. There is no need to inherit information to the FTP adapter via a business process.

(4) File

These commands are used to set information for files that are sent and received. Information must be inherited to the FTP adapter via a business process.

(5) Transfer

These commands are used to transfer files.

When a transfer command is executed, a business process is started and information is inherited to the FTP adapter from the FTP inbound adapter via a business process.

(6) Update

These commands update files and directories on the FTP server.

These commands can be executed for batch processing before or after file transfer. An update command is specified in the `SITE CSCTHR` command from the FTP client.

(7) Reference

These commands are used for referencing information about files and directories on the FTP server, and then passing the information to the FTP client.

Like transfer commands, `LIST` commands inherits information to the FTP adapter from the FTP inbound adapter via a business process. Commands other than `LIST` commands are specified in the `SITE CSCTHR` command from the FTP client, and then the command results can be viewed on the business process.

8.9.2 Specification method for transfer commands

This subsection describes how to specify the transfer commands from the FTP client.

In the transfer commands, you specify the name of the file to be sent and received, and the reception ID of FTP reception for identifying the business process to be started.

The specification format and the contents specified for the transfer commands are as follows:

Specification format

/<Reception ID>/<Path on the server>/<File name>

Specified contents

/ (Forward slash)

This is the path separation character. This character is fixed regardless of the OS.

<Reception ID>

This is the reception ID of FTP reception. This ID identifies the business process to be started by the specified reception ID.

<Path on the server>

This is the path on the destination FTP server. You can omit this specification.

<File name>

This is the name of the file to be sent and received.

8.9.3 Points to be considered on using FTP integration

This subsection describes the points to be considered for using the FTP integration functionality.

(1) Range in which the FTP integration functionality can be used in combination with other functionality

The following table describes the range in which the functionality used for FTP integration can be used in combination with the other existing functionality:

Table 8–28: Range in which the FTP integration functionality can be used in combination with existing functionality

Adapters	Standard reception (synchronous or asynchronous)	SOAP reception	TP1/RPC reception	FTP reception
SOAP adapter	Y	Y	Y	A
SessionBean adapter	Y	Y	Y	A
MDB (WS-R) adapter	Y	Y	Y	A
MDB (DB queue) adapter	Y	Y	Y	A
DB adapter	Y	Y	Y	A
TP1 adapter	Y	Y	Y	A
File adapter	Y	Y	Y	A
Object Access adapter	Y	Y	Y	A
Message Queue adapter	Y	Y	Y	A
FTP adapter	A	A	A	Y
File operations adapter	A	A	A	Y
Mail adapter	Y	Y	Y	Y
Data transformation	Y	Y	Y	Y
Invoke java activity	Y	Y	Y	Y

Legend:

Y: The functionality can be combined and used.

A: The operations are not supported when the functionality are combined.

(2) Points to be considered related to the creation of business processes

- A business process must be created for each file transfer route.
- The business process must be created such that a response is returned to the FTP reception.
- You cannot process multiple files with one business process.
- You cannot use BPEL parallel processing (FLOW) in the FTP integration.

(3) Points to be considered on using FTP with server operations

To use the FTP for operating and maintaining a server on Service Platform, you must set up a separate FTP server. In this case, specify the design in such a way that the port number or IP address of both the FTP servers is unique.

(4) Points to be considered on the execution of FTP commands

If you use the ftp commands for Linux and perform file transfer requests for the FTP client, the following two types of messages are output in the message log of the FTP inbound adapter:

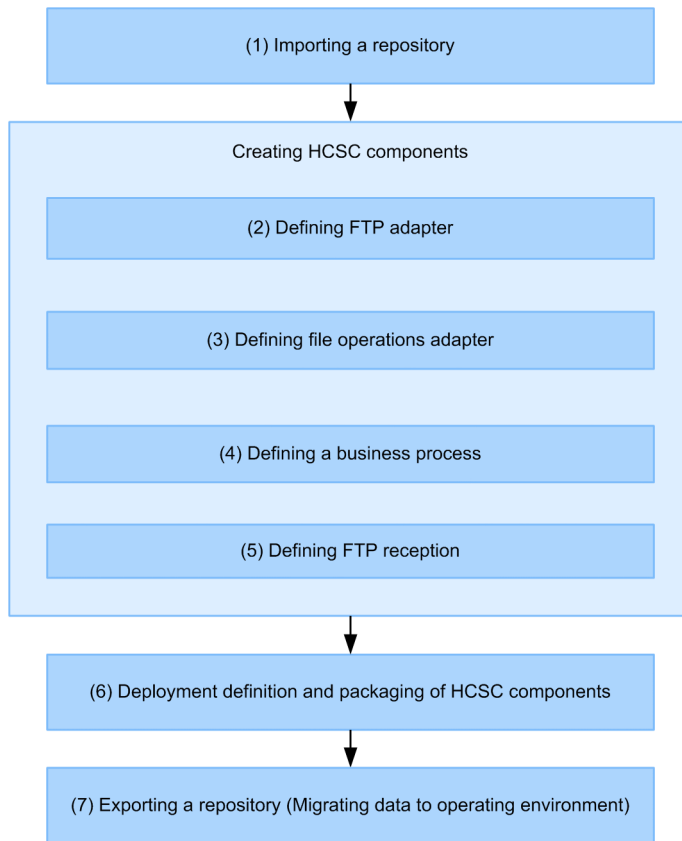
- A message indicating that the SYST command, which is the command of an unsupported FTP protocol, was received (KDEC07531-E)
- A message indicating that the error response was returned to the client (KDEC07542-E)

Furthermore, the response "502 Command SYST not implemented." is returned to the FTP client, but this does not affect the file transfer requests.

8.10 Flow of development in the FTP integration system

The following figure shows the flow of development in the FTP integration system. All the items in the figure are operations to be implemented in the development environment:

Figure 8–41: Flow of development in the FTP integration system



(1) Importing a repository

The repository exported in the operating environment is imported into the development environment.

For details on how to import a repository, see "3.2.3 Importing a repository" in "Service Platform Basic Development Guide".

(2) Defining an FTP adapter

Use the Service Adapter Settings window to define an FTP adapter. Define one FTP adapter for each FTP server to be connected to.

For details on defining an FTP adapter, see "3. Defining Adapters" in "Service Platform Reception and Adapter Definition Guide".

(3) Defining a file operations adapter

Use the Service Adapter Settings window to define a file operations adapter. Define one file operations adapter for each operation, such as conversion, replication, and deletion.

For details on defining a file operations adapter, see "3. Defining Adapters" in "Service Platform Reception and Adapter Definition Guide".

(4) Defining a business process

Use the Define Business Process window to define a business process. Define a business process for each file transfer route.

For details on defining a business process, see "5. Defining Business Processes" in "Service Platform Basic Development Guide".

(5) Defining an FTP reception

Use the User-defined Reception Settings window to define an FTP reception. Define one FTP reception for each business process (define multiple FTP receptions as and when required). Note that you cannot map one FTP reception to multiple business processes.

For an overview of defining an FTP reception, see "*8.4 FTP reception functionality*". For details on defining an FTP reception, see "2.4 Defining FTP reception" in "Service Platform Reception and Adapter Definition Guide".

(6) Packaging and deployment definition of HCSC components

The definition files related to the definitions of the specified HCSC components, and the files provided by the execution environment are together packaged in an EAR file. This EAR file also defines the information for the cluster (or single HCSC server) that deploys the HCSC components (deployment definition).

For details on the packaging and deployment definition, see "7. Packaging HCSC Components and Defining Deployment" in "Service Platform Basic Development Guide".

(7) Exporting the repository

You export the repository that stores the definition files required for deploying the HCSC components. The information exported from the development environment is imported into the operating environment and used.

For details on exporting a repository, see "3.2.2 Exporting repository" in "Service Platform Basic Development Guide".

For details on how to operate an FTP integration system after being imported into the operating environment, see "5.4 HCSC server operations (for FTP integration)" in "Service Platform System Setup and Operation Guide".

8.11 Design for troubleshooting

This section describes the policy for system error design that is provided for errors in FTP integration.

8.11.1 Overview of error design

The points to be considered in the error design for FTP integration are as follows:

(1) Error detection

The components in FTP integration are linked by synchronous invocation, so you can detect the occurrence of an error by the return from the invocation destination. However, the invocation destination might also become non-responsive. Therefore, you must design appropriate timer management during setup, and detect the occurrence of an error whereby the files cannot be transferred normally.

For details on timer management in FTP integration, see "*8.11.2 Timer design*".

(2) Error notification

When an error occurs, the occurrence of the error must be reliably reported to the FTP client that executed FTP integration and the system operator.

The FTP client that executed FTP integration is notified about the occurrence of the error with the return of the FTP command. The system operator is notified about the occurrence of the error with the output of a message. The message about error occurrence is output to a determined message, message log, or file. Therefore, by monitoring the error messages with the monitoring functionality such as JP1, you can notify the system operator of the occurrence of an error in a timely manner.

(3) Recovery processing

Design appropriate transaction processing to simplify the recovery processing for the FTP client and system operator after an error occurs.

For details on the transaction design for FTP integration, see "*8.11.3 Transaction design*".

(4) Investigation of cause

To recover from the error and to prevent a reoccurrence, you must identify and eliminate the cause of the error. Therefore, various error information items are obtained to identify the cause of the error.

For details on obtaining the error information for FTP integration, see "*8.11.4 Troubleshooting design*".

8.11.2 Timer design

When the invocation destination becomes un-responsive, and the processing time exceeds the assumption, timer management is implemented with the aim of detecting this state and taking appropriate action.

The following table describes the list of timers used in FTP integration:

Table 8–29: List of timers used in FTP integration

Component	Timer	Purpose of timer management	Operations in the case of timeout
FTP inbound adapter ^{#1}	Non-communication monitoring timer	This timer monitors the time from the command response until the next command is issued in a control connection.	The connection is closed and the FTP communication terminates with an error.
	Client connection awaiting timer	This timer monitors the time until the connection establishment request arrives from an FTP client, when the data connection method is Active mode.	The connection is closed and the FTP communication terminates with an error.

Component	Timer	Purpose of timer management	Operations in the case of timeout
FTP inbound adapter ^{#1}	Data reception monitoring timer	This timer monitors the time required for reading the data received at one time in a data connection.	The connection is closed and the FTP communication terminates with an error.
	Data transmission monitoring timer	This timer monitors the time required for writing the data transmitted at one time in a data connection.	The connection is closed and the FTP communication terminates with an error.
FTP reception ^{#2}	FTP reception monitoring timer	This timer monitors the time from the invocation of the FTP reception until the response. This is used as a precaution when the FTP reception invocation source detected the timeout previously.	The timeout is detected when the FTP reception returns a response, a system exception is returned, and the processing is terminated.
Business process ^{#3}	Service invocation monitoring timer	This timer monitors the time from service invocation until response. This is used as a precaution when the business process invocation source timed out previously.	The timeout is detected when the response is returned from the service, a system exception is returned, and the processing is terminated.
	Business process monitoring timer	This timer monitors the time of the business process operations (such as data transformation) from service invocation until response. This is used as a precaution when the business process invocation source timed out previously.	The timeout is detected when the service is next invoked, a system exception is returned, and the processing is terminated.
FTP adapter ^{#4}	Response monitoring timer	This timer monitors the time from command transmission until response in a control connection.	The connection is closed and the FTP communication terminates with an error. Thereafter, a system exception is returned and the processing is terminated.
	Server connection awaiting timer	This timer monitors the time until the FTP server returns a connection establishment request in the Active mode.	The connection is closed and the FTP communication terminates with an error. Thereafter, a system exception is returned and the processing is terminated.
	Data transmission and reception monitoring timer	This timer monitors the time from the start until the end of data transfer.	The connection is closed and the FTP communication terminates with an error. Thereafter, a system exception is returned and the processing is terminated.
File operations adapter ^{#5}	File conversion monitoring timer	This timer monitors the conversion processing time during the partitioned reading of a large-sized file.	The connection is closed and the FTP communication terminates with an error. Thereafter, a system exception is returned and the processing is terminated.
All	Request response monitoring timer	This timer monitors the time from the invocation of the FTP reception until the return of a response.	An error response is returned to the FTP client. However, the execution continues until the running business processes are terminated.

#1

For details on the timer management functionality of the FTP inbound adapter, see "8.5.2 Monitoring a timeout".

#2

For details on the timer management functionality of the FTP reception, see "8.4.9 Monitoring a timeout".

#3

For details on the overall timer management functionality of the business processes and HCSC server, see "4.1 Setting up a timer for an HCSC server".

#4

For details on the timer management functionality of the FTP adapter, see "*8.6.5 Timer management*".

#5

You can set up the timer management functionality for the file operations adapter in the file operations adapter execution-environment property file.

For details, see "File operations adapter execution-environment property file" in "Service Platform Reference Guide".

8.11.3 Transaction design

To simplify recovery from error, design transactions in FTP integration with the following policy:

- Consider the process from the data transfer request from the FTP client until the response that the data transfer is complete, as a one-phase transaction.
- The file and resource recovery is executed by the FTP reception, FTP adapter, and file operations adapter that handle files.

(1) Scope of a transaction

The transactions are managed by using EJB transactions in order to synchronize the content of response to the FTP client, such as normal and abnormal termination, and the processing status of the business process.

This subsection describes the scope of a transaction for each file transfer pattern.

(a) For synchronous business processes

The following figure shows the transaction scope for synchronous business processes.

For details on the file transfer patterns using synchronous business processes, see "*8.3.1(1) Procedure of file transfer using synchronous business processes*" and "*8.3.2 Transfer of files from an FTP server to an FTP client*".

Figure 8–42: For synchronous business processes (To transfer files from an FTP client to an FTP server)

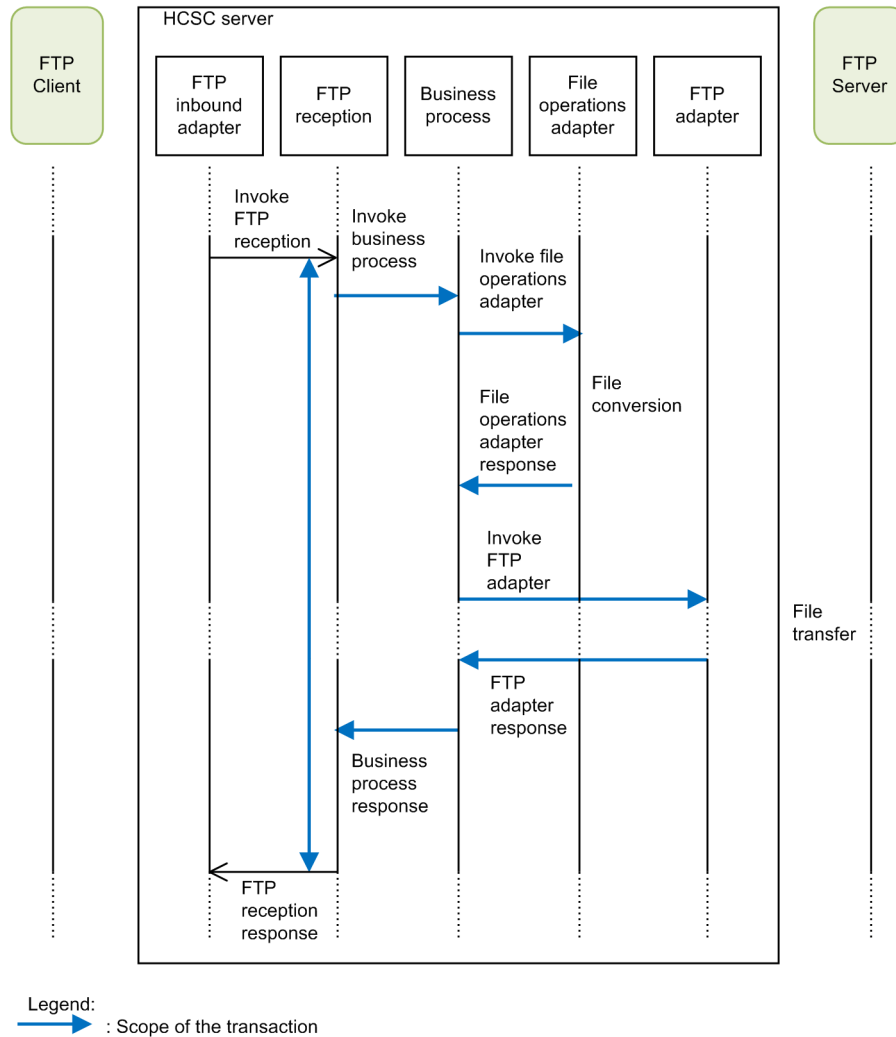
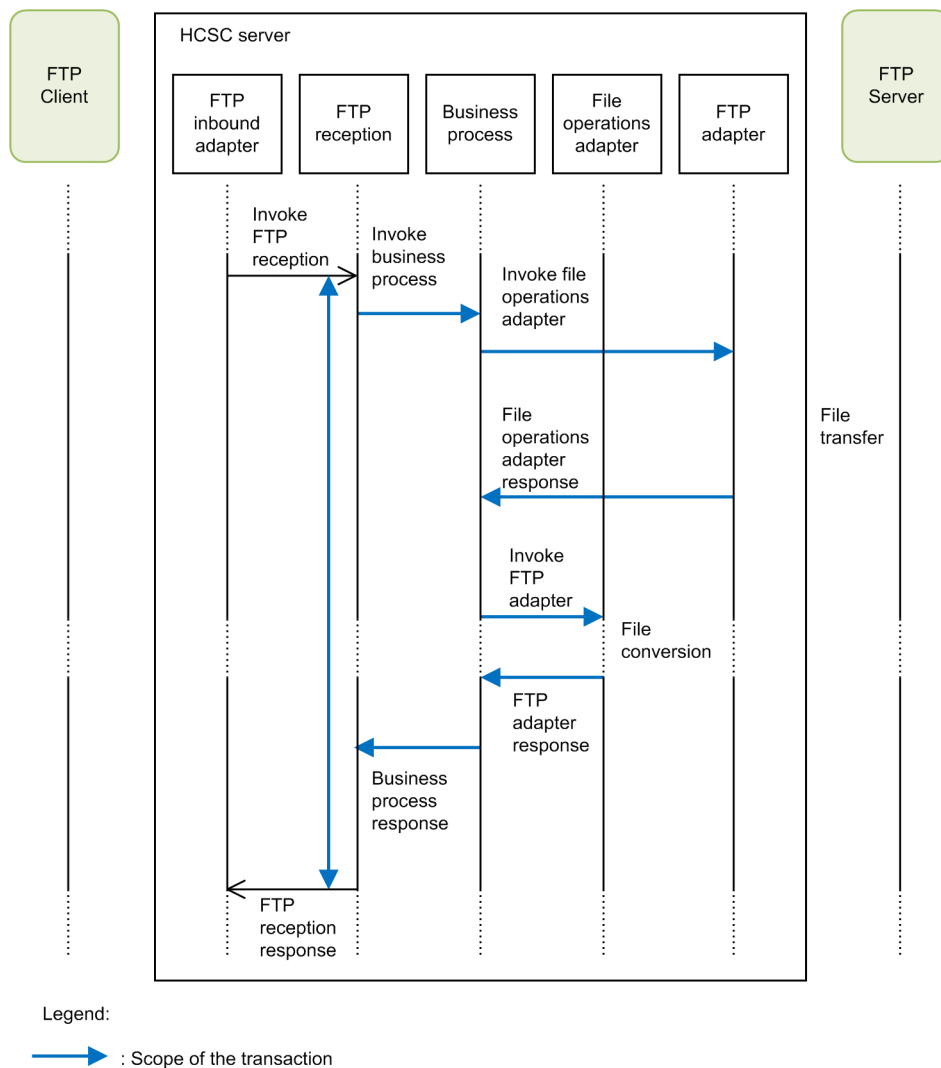


Figure 8–43: For synchronous business processes (To transfer files from an FTP server to an FTP client)

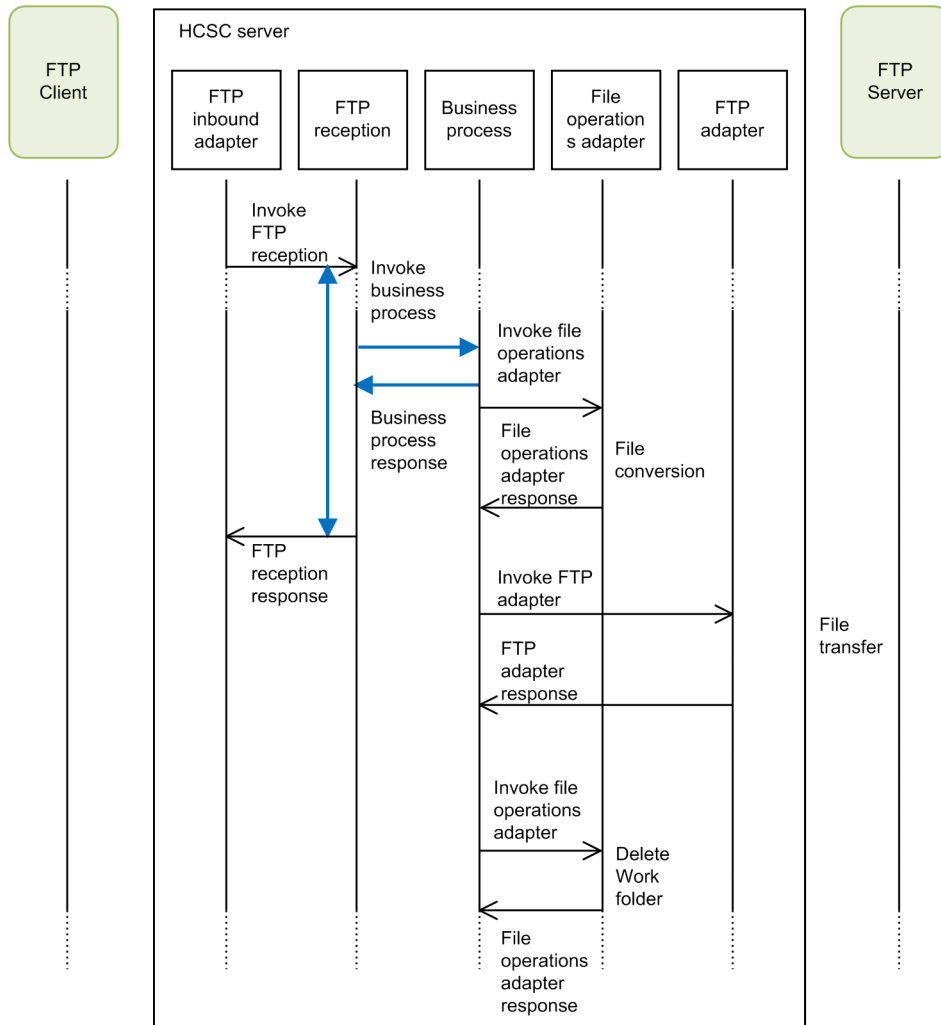


(b) For asynchronous business processes

The following figure shows the transaction scope for the asynchronous business processes.

For details on the file transfer patterns using asynchronous business processes, see "8.3.1(2) Procedure of file transfer using asynchronous business processes".

Figure 8–44: For asynchronous business processes



Legend:

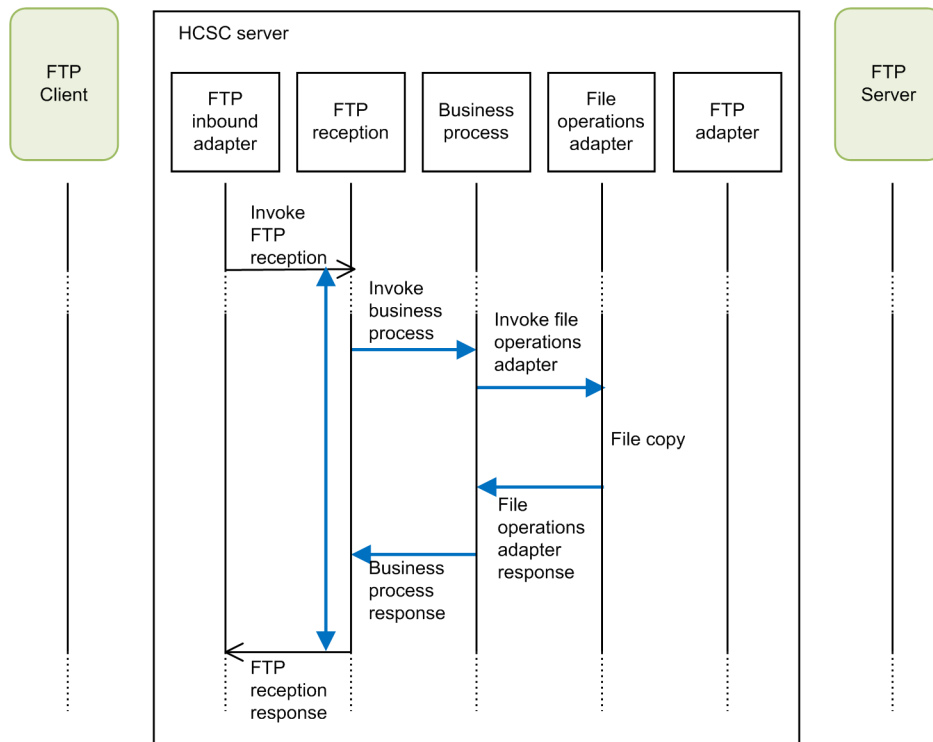
: Scope of the transaction

(c) For the transfer of files from an FTP client to another FTP client

The following figure shows the transaction scope for the transfer of files from an FTP client to another FTP client.

For details on the file transfer pattern from an FTP client to another FTP client, see "8.3.3 *Transfer of files from an FTP client to another FTP client*".

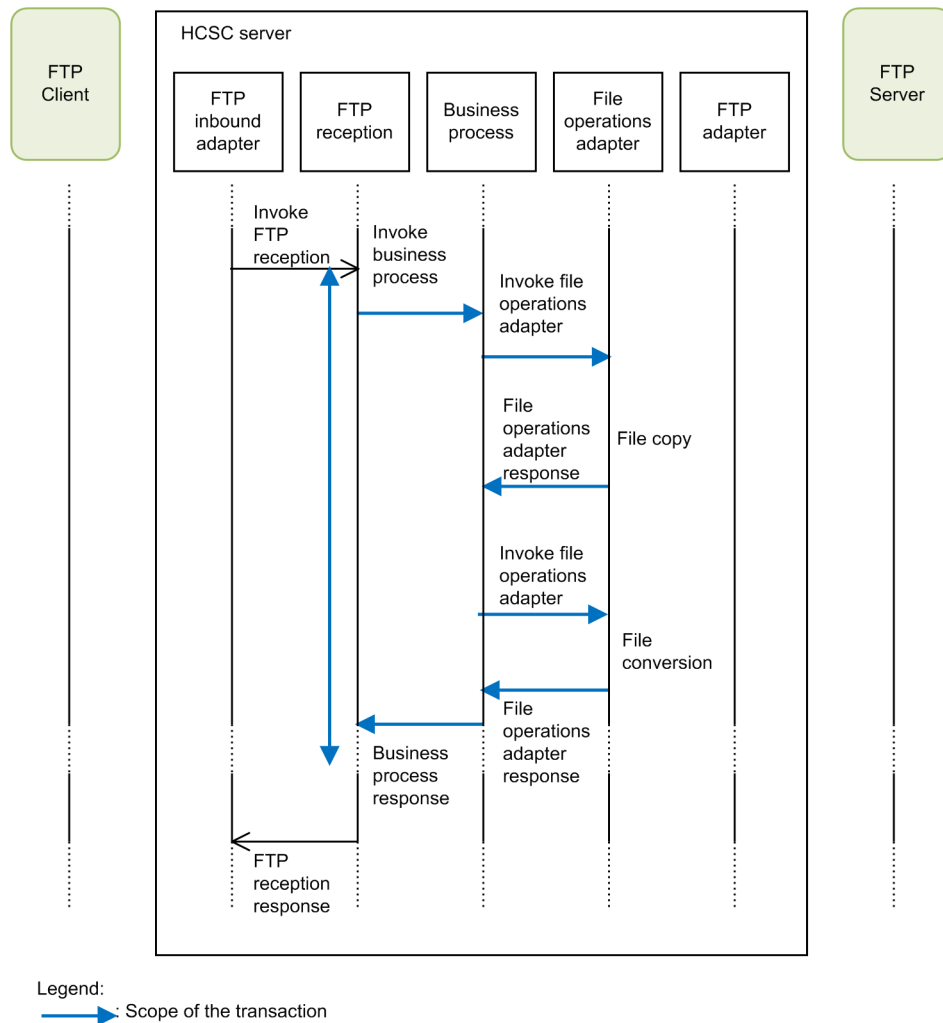
Figure 8–45: For the transfer of files from an FTP client to another FTP client (To transfer files to the common folder in the HCSC server)



Legend:

→ : Scope of the transaction

Figure 8–46: For the transfer of files from an FTP client to another FTP client (To obtain files from the common folder in the HCSC server)



(2) Deleting files when an error occurs

In FTP integration, when an error occurs, each component deletes the output files to match the processing results and file status.

The following table describes the processing in FTP integration and the processing in the corresponding components:

Table 8–30: Handling of files in FTP integration

Processing	Component	Assumed cause of error	Handling of files in the case of error	
			Input file	Output file
File transmission from FTP client	FTP reception	<ul style="list-style-type: none"> File transfer interrupted Network error Insufficient disk space Disk error 	--	Delete
File reception from FTP client	FTP reception	<ul style="list-style-type: none"> File transfer interrupted Network error Insufficient disk space Disk error 	State before processing is guaranteed	--

Processing	Component	Assumed cause of error	Handling of files in the case of error	
			Input file	Output file
File transmission to FTP server	FTP adapter	<ul style="list-style-type: none"> • Network error • Insufficient disk space • Disk error 	State before processing is guaranteed	--
File reception from FTP server	FTP adapter	<ul style="list-style-type: none"> • Network error • Insufficient disk space • Disk error 	--	Delete
File conversion	File operations adapter	<ul style="list-style-type: none"> • Conversion error • Insufficient disk space • Disk error 	State before processing is guaranteed	Delete
File replication	File operations adapter	<ul style="list-style-type: none"> • Insufficient disk space • Disk error 	State before processing is guaranteed	Delete

Legend:

--: Not applicable.

Note that if the file deletion process does not terminate normally due to the following causes, restart the system: #1#2

- The file lock cannot be released
- The files cannot be accessed due to hardware failure such as network error and disk error
- The files cannot be deleted because the system is down

#1

If the file being written to cannot be deleted when an error occurs, the exclusive lock for that file is not released. Therefore, the error does not reach the files to be deleted due to access from the other business processes.

#2

To delete the error files on a shared folder, you must stop the entire system configuring the cluster.

(3) Points to be considered on designing a business process

To consider the entire transfer processing for one file as one-phase transaction, you must consider the updated resources as one. In FTP integration, you cannot perform two-phase commit to match the synchronization of updates for multiple resources. Therefore, when you design a business process, you must consider the following points:

- To update multiple resources in one business process, you must design the update between the resources so that the business process matches the synchronization.
- If an error occurs after returning a response to the FTP client in an asynchronous business process, you must perform roll forward processing by restarting the business process.

8.11.4 Troubleshooting design

Design the system so that you can collect the log and trace required for investigation when an error occurs.

The following table describes the log and trace types that can be collected in FTP integration:

Table 8–31: List of information collected for troubleshooting

Information type	Collected contents	Collected components			
		FTP inbound adapter	FTP reception	FTP adapter	File operations adapter
Message log	Messages such as those for HCSC server operations and those for information, warning, and error during operations are collected.	Y	Y	Y	Y
Request trace	The following trace is collected: <ul style="list-style-type: none"> • Receipt of HCSC server request/ response • Business process invocation/ response • Service invocation from adapter/ response 	--	Y	Y	Y
Performance analysis trace	The trace indicating the state of progress of processing between components is collected.	Y	Y	Y	Y
User message trace	The trace of user messages at the determined collection points is collected.	--	Y	Y	Y
Maintenance log	The information on the internal processing of the product is collected with the aim of isolation and investigation of error.	--	Y	Y	Y
Protocol trace	Information on the FTP commands sent and received in the control connection with the FTP client or FTP server is collected as a trace of the communication protocol of HCSC server and external systems. You can also collect the information on the data connection as and when required.	Y	--	Y	--

Legend:

Y: Collected.

--: Not collected.

For details on the output destination and output contents of the log and trace that you can collect for FTP integration, see "7.7.11 Troubleshooting in the FTP linkage system" in "Service Platform System Setup and Operation Guide".

Reference note

To reduce the loss of time in file transfer due to slow down when using FTP integration, it is recommended that you prepare an investigative tool to be used when slow down occurs. The following table describes the checking methods:

Table 8–32: Methods for investigating slowdown in FTP integration

Assumed cause	Checking method
Network delay	Check with protocol trace and operation statistics for the OS.
Disk I/O delay	Check with performance analysis trace and operation statistics for the OS.
Lock pending	Check with the maintenance log.
Frequent retries	Check with the maintenance log.
CPU problem	Check with the operation statistics for the OS.

Appendixes

A. TP1/Client/J Function

INTENTIONALLY DELETED

A.1 Remote procedure call

INTENTIONALLY DELETED

A.2 Transaction control

INTENTIONALLY DELETED

A.3 TCP/IP communication function

INTENTIONALLY DELETED

A.4 Function for receiving one-way communication from server

INTENTIONALLY DELETED

A.5 TP1/Web connection function

INTENTIONALLY DELETED

A.6 Function for changing dynamic definition

INTENTIONALLY DELETED

A.7 TCP/IP connection establishment monitoring function

INTENTIONALLY DELETED

A.8 Function for connecting to DCCM3

- INTENTIONALLY DELETED

A.9 XA resource services function

INTENTIONALLY DELETED

A.10 Troubleshooting function

INTENTIONALLY DELETED

A.11 Data compression function

INTENTIONALLY DELETED

A.12 Source host specification function

INTENTIONALLY DELETED

A.13 Function for fixing reception port

INTENTIONALLY DELETED

A.14 Host switching function

INTENTIONALLY DELETED

B. TP1/Client/J definition

INTENTIONALLY DELETED

B.1 Overview of TP1/Client/J definition

INTENTIONALLY DELETED

B.2 TP1/Client/J environment definition details

INTENTIONALLY DELETED

C. Troubleshooting of TP1/Client/J

INTENTIONALLY DELETED

C.1 Output contents of trace file

INTENTIONALLY DELETED

C.2 UAP trace

INTENTIONALLY DELETED

C.3 Data trace

INTENTIONALLY DELETED

C.4 Error trace

INTENTIONALLY DELETED

C.5 Method trace

INTENTIONALLY DELETED

C.6 Debug trace

INTENTIONALLY DELETED

C.7 Performance analysis trace

INTENTIONALLY DELETED

D. Glossary

Terminology used in this manual

For the terms used in the manual, see *Application Server and BPM/ESB Platform Terminology Guide*.

Index

Numerics

1-to-1 node switching system 19

A

accessing database using DB adapters 85
access to message queues using Message Queue adapter 112
activity 211
activity (functionality for creating business processes) 5
add (write mode) 99
Arranging business processes in a hierarchy 6
asynchronous execution of activity that is defined after
reply activity 257
authentication when sending email messages from mail
adapter 127

B

basic structure (invoking service components) 28
batch (read mode) 98
business process, managing execution status of 5
business process access control 339
business processes, (functionality for creating business
processes) 4
business processes that are not to be made persistent 215
business processes that are to be made persistent 215
business process flow 207
business process functionality 195

C

Cache functionality for XML messages 300
catching fault by using SOAP fault operation definition file
268
changing connection destination of service components 267
changing timeout value of service components 265
Changing value of user message wherein UOC is used 283
checking file size after sending and receiving files 342, 353
checking operation status from application 271
cluster software 18
combination of variables (message type) when data is not
transformed 276
commands supported in FTP integration 381
communication model and business processes that can be
used with FTP receptions 335
communication model of business process 207
communication through proxy servers 178
communication using chunk transmission 179
communication using connection continuity (Keep-Alive)
178
complementing name space prefixes specified in attribute
values of XML messages 274
compressing and extracting HTTP body 180
configuration pattern of operating environment in cluster
configuration 22
configuring cluster of HCSC server 18

connecting to a system by using a general custom adapter
192
connecting to client using HTTP communication 128
connecting to database 85
connecting to FTP client and FTP server 117
connecting to mail server 119
connecting to message queues 101
connecting to OpenTP1 95
connecting to services by using database queue 77
connecting to services by using Web Services (SOAP
communication) 41
connecting to services using HTTP communication 156
connecting to services using Session Beans 63
connecting to services using WS-R (WS-Reliability) 67
connecting to system by using custom reception 190
connecting to system handling file 96
connection with Object Wrapper system 100
cookie information in (Web service (SOAP
communications)) 48
correlation set (functionality for creating business
processes) 5
corresponding relation between request message (header)
and HTTP adapter execution environment (common)
property file 186
creatingservice adapters (functionality for creating service
adapters) 3
creating service requester 10
creating work folder and intermediate files 338
CTM (redundant configuration of HCSC server) 18
custom adapter 3

D

databases that can be connected 316
data compression function (TP1/Client/J) 400
data trace (TP1/Client/J) 403
data transformation (business process) 198
data transformation definition (functionality for creating
data transformation definition) 6
data validation functionality 307
dead message queue (MDB (DB queue)) 83
dead message queue (MDB (WS-R)) 76
debug trace (TP1/Client/J) 403
defined exception 268
defining message format of any format 275
deployment definition (functionality for defining
deployment) 8
deployment of HCSC components 7
design for troubleshooting 388
detailed information of process instance execution log,
referencing 17
detecting failure by linking with JP1/IM 312
development environment, functional overview of 2
dynamically change in connection destination information
for DB adapter 90
dynamic change in connection destination in Web service
(SOAP communication) 57

E

- element 2
- encoding and / decoding HTTP requests and HTTP responses (HTTP reception) 152
- error trace (TP1/Client/J) 403
- example of file adapter operation 97
- example of HTTP reception configuration 134
- example of Message Queue reception configuration 108
- example of online system using HTTP reception 129
- example of sending GET method request (HTTP reception) 129
- example of sending POST method request (HTTP reception) 130
- example of system using HTTP adapter 183
- examples of file transfer patterns using FTP integration 325
- exclusive control while reading and writing files 99
- exclusive reference 99
- exclusive update 99
- executing FTP commands 351
- executing LIST and NLST commands by specifying list command options 341
- execution environment, functional overview of 12

F

- failure analysis functionality 306
- file access using common folder 379
- file access using work folder 379
- file access using work folder and common folder 379
- file compression operation 374
- file conversion operation 359
- file conversion using character code conversion UOC 370
- file expansion operation 376
- file format handled as binary data in file adapter 98
- file format handled as XML data in file adapter 98
- file formats that can be used in adapters 98
- file operations adapter 324
- file operations adapter functionality 357
- file or folder deletion operation 373
- file replication operation 372
- fixed length format 98
- flow for distributed development using import component function 24
- flow of checking operation status 271
- flow of development in FTP integration system 386
- flow of FTP adapter processing 351
- flow of identifying location of system exception by using general fault 281
- flow of invoking service component with message format different from service component side 36
- flow of process when any format is set 275
- folder specification used for sending and receiving files 353
- FTP adapter 324
- FTP adapter functionality 350
- FTP command reply timeout (FTP adapter) 355
- FTP inbound adapter 324
- FTP inbound adapter functionality 342
- FTP reception 324
- FTP reception functionality 334
- function (functionality for creating data transformation definition) 7

- functionality for acquiring failure information 306
- functionality for connecting to various types of systems 27
- functionality for creating business processes 4
- functionality for creating data transformation definitions 6
- functionality for creating message format 2
- functionality for creating user-defined reception 2
- functionality for debugging business processes 10
- functionality for defining deployment 7
- functionality for deploying HCSC components 15
- functionality for deploying to starting or stopping to deleting HCSC components, in a batch 9
- functionality for distributing services 12
- functionality for executing business processes 13
- functionality for Failure Checking 305
- functionality for invoking various services 12
- functionality for managing execution logs 17
- functionality for managing operations 15
- functionality for setting up execution environment 14
- functionality of Cosminexus Service Platform 1
- functionality to transfer files by integrating with FTP (FTP integration) 321
- functionality used by linking with other products 311
- function for changing dynamic definition (TP1/Client/J) 400
- function for connecting to DCCM3 (TP1/Client/J) 400
- function for fixing reception port (TP1/Client/J) 401
- function for receiving one-way communication from server (TP1/Client/J) 400

G

- general faults for converting system exceptions to faults 280
- generating request ID 337
- glossary 404

H

- HA monitor (redundant configuration of HCSC server) 19
- HCSC component validation 2
- host switching function (TP1/Client/J) 401
- HTTP client and Web server supported by HTTP reception 128
- HTTP reception request process 138
- HTTP reception response process 143
- HTTP request body process (HTTP request) 141
- HTTP request header process (HTTP reception) 140
- HTTP response body process (HTTP reception) 147
- HTTP response header process (HTTP reception) 143

I

- identification information 38
- idle timeout (FTP inbound adapter) 343
- importing repositories at time of distributed development 24
- Importing the BPEL created in BPMN tool 6
- inheriting Cookie information using HTTP adapter 175
- inheriting HTTP header in (Web service (SOAP communication)) 51
- invoking service components (Basic procedure) 28
- invoking service components (flow of messages (MDB (DB queue))) 78
- invoking service components (flow of messages (MDB (WS-R))) 67

invoking service components (handling response message (MDB (DB queue))) 81
 invoking service components (handling response message (MDB (WS-R))) 70
 invoking service components (relationship between user messages and stubs) 63
 invoking service components (relationship between user messages and WSDL) 42
 invoking service components using HTTP adapter 156
 invoking service components using Message Queue reception 101
 invoking service component using HTTP reception 128

J

Java invocation (business process) 197

L

length tag format 98
 linkage with JP1 312
 Listen timeout (FTP adapter) 356
 load balancer 18
 load-balancing functionality (redundant configuration of HCSC server) 18

M

mail information that can be set on mail adapter 125
 management of execution logs by linking with HiRDB 317
 managing files and folders used for each operation 377
 managing FTP commands executed before and after transfer and list 340
 managing HTTP adapter timer 182
 managing HTTP reception timer 153
 managing HTTP request files 150
 managing the mail adapter timer 126
 mapping (functionality for creating data transformation definition) 7
 mapping between general fault messages and message log 282
 mechanism of business process 207
 message tracking functionality 306
 message format (functionality for creating message format) 2
 message log 306
 method trace (TP1/Client/J) 403
 modes for file operations 98
 monitoring processes through JP1 313
 monitoring timeout [FTP inbound adapter] 342
 monitoring timeout [FTP reception] 341
 multiplicity 290
 Multiplicity of the HCSC server 290
 Multiplicity related to database access 293
 Multiplicity related to Web Services (SOAP communication) 295
 multiplicity related to XML analysis process 298

N

names and number of queues (MDB (DB queue)) 83
 names and number of queues (MDB (WS-R)) 76

new (write mode) 99
 node switching when error occurs 20

O

Object Access adapter 100
 obtaining file list from FTP server 331
 operations and FTP commands supported in FTP reception 334
 operations supported by file operations adapter 357
 output contents of trace file (TP1/Client/J) 403
 overview of business processes 196
 overview of component-common UOC 283
 overview of data validation functionality 307
 overview of error design 388
 overview of FTP integration 322
 overview of TP1/Client/J definition 402
 overview of WebSphere MQ 101

P

packaging 8
 partitioning storage of execution logs 317
 performance analysis trace 306
 performance analysis trace (TP1/Client/J) 403
 performance upgrade function 289
 points to be considered for distributed development 26
 points to be considered on using FTP integration 384
 points to be considered when using HTTP reception 154
 points to be considered (when using SessionBean adapter) 66
 pre-cache function of business process 301
 precautions on checking operation status 272
 precautions on using user message trace functionality 310
 preconditions (connecting to mail server) 119
 preconditions (connecting to services using HTTP communication) 156
 preventing duplicate message delivery 110
 procedure for invoking service components 31
 procedure of file transfer from FTP client to FTP server (when asynchronous business processes are used) 327
 procedure of file transfer from FTP client to FTP server (when synchronous business processes are used) 326
 process instance, re-executing 17
 process instance execution log, deleting 17
 process instance execution log, managing 17
 process instance execution log, searching 17
 process instances (operating with business processes) 208

R

reading and writing files using adapter 97
 read mode 98
 reception and response (business process) 196
 redundant configuration of HCSC server (cluster software) 18
 redundant configuration of HCSC server (load-balancing functionality) 18
 redundant configuration of HCSC server (N-to-1 cluster configuration) 21
 re-executing business process 251
 re-executing business processes (commands) 251

- re-executing business processes (service requesters (Session Bean)) 252
- re-executing business processes (service requesters (SOAP communication)) 251
- re-executing business processes (windows in operation environment) 251
- relation between HTTP requests and request messages 157
- relation between HTTP responses and response messages 169
- relation between Message Queue reception and business processes 107
- relation between Message Queue reception and MQ resource adapter 104
- relation between Message Queue reception and queues 105
- relationship with application server (EJB containers) 63
- relationship with Reliable Messaging (MDB (DB queue)) 77
- relationship with Reliable Messaging (MDB (WS-R)) 67
- Relationship with the Communication Infrastructure for Web Services 41
- remote procedure call (TP1/Client/J) 400
- repetition (business process) 201
- request and response messages for each operation 359
- request line process (HTTP reception) 139
- request trace 306
- role of correlation set 214
- roles of variables 213

S

- scope of FTP integration 381
- scoping of data validation function 308
- secure connection using WWW authentication 181
- send option of Message Queue reception 102
- separator format 98
- service adapter 3
- service invocation (business process) 197
- service requester 10
- setting message rollback frequency (Message Queue reception) 112
- setting up timer 260
- setting up cluster 14
- setting up data validation functionality 308
- setting up HCSC server (functionality for setting up execution environment) 14
- setting up timer for HCSC Server 260
- setting up user message trace functionality 310
- simultaneous conversion of multiple data records 369
- SOAP message configuration in (Web service (SOAP communication)) 54
- source host specification function (TP1/Client/J) 401
- specification method for transfer commands 383
- standby processing (business process) 205
- standby processing using business processes 254
- status codes (HTTP reception) 148
- status line process (HTTP reception) 143
- system configuration for using FTP integration 324
- system using mail adapter 119

T

- TCP/IP communication function(TP1/Client/J) 400

- TCP/IP connection establishment monitoring function (TP1/Client/J) 400
- timeout in data reception (FTP inbound adapter) 347
- timeout in data transfer or acquisition (FTP adapter) 356
- timeout in data transmission (FTP inbound adapter) 348
- timeout in establishing control connection (FTP adapter) 354
- timeout in establishing data connection (FTP adapter) 355
- timeout in establishing data connection (FTP inbound adapter) 346
- timeout in obtaining lock (FTP adapter) 355
- timer design 388
- timer management 354
- timers that can be set up on an HCSC server 260
- TP1/Client/J definition 402
- TP1/Client/J environment definition details 402
- TP1/Client/J function 400
- TP1/Web connection function (TP1/Client/J) 400
- transaction (MDB (DB queue)) 83
- transaction (MDB (WS-R)) 72
- transaction (Session Bean) 65
- transaction (Web Services (SOAP communication)) 48
- transaction design 390
- transaction of business process 216
- transaction of scope activity 244
- transactions of invoke service activity 237
- transactions of reply activity 245
- transactions performed while using Message Queue reception 110
- transactions to be performed while using HTTP reception 153
- transactions when FTP reception is used 336
- transactions when OFF is specified for compatibility with business process status and ON is specified for compatibility with invoke service activity status 225
- transactions when OFF is specified in both compatibility with business process status and compatibility with invoke service activity status 233
- transactions when on is specified in compatibility with business process status 217
- transactions when using mail adapter 124
- transaction when you select settings to commit at start time and end time of scope 238
- transaction control (TP1/Client/J) 400
- transfer of files from FTP client to another FTP client 329
- transfer of files from FTP client to FTP server 325
- transfer of files from FTP server to FTP client 328
- troubleshooting design 396
- troubleshooting functionality 306
- troubleshooting function (TP1/Client/J) 400
- troubleshooting of TP1/Client/J 403
- types of variables 213

U

- UAP trace (TP1/Client/J) 403
- usage example of component-common UOC 286
- user-defined reception (functionality for creating user-defined reception) 3
- user message trace functionality 310

V

variable 5

W

while using message queue reception 110

Windows Server Failover Cluster (redundant configuration
of HCSC server) 19

write mode 99

X

XA resource services function (TP1/Client/J) 400

XML format 98

XML parser pool function of data transformation process
302