

uCosminexus Service Platform

First Step Guide

User's Guide and Operator's Guide

3020-3-Y41-40(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.

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

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

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
		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)

Abbreviation			Full name or meaning
Windows	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*.

■ Issued

Oct. 2015: 3020-3-Y41-40(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

Part 1: BASIC

1	Before Using Sample Programs	1
1.1	Overview of this manual	2
1.2	What you can expect from this manual	3
2	Overview of Sample Programs	5
2.1	System configurations of sample programs	6
2.2	Components of sample programs	8
2.3	Processing details of sample programs	9
2.3.1	Calling a service component from the service requester	9
2.3.2	Calling a service component from the business process	9
2.3.3	Integrating processes	10
3	Preparing the Environment for Sample Programs	13
3.1	Overview of installation and setup	14
3.2	Installing Service Architect and preparing for setting it up	15
3.2.1	Installing Service Architect	15
3.2.2	Preparing Eclipse	16
3.3	Using Eclipse Setup to set up the Eclipse environment	17
3.4	Setting up the development environment	20
3.4.1	Setting up Eclipse	20
3.4.2	Installing WSDL4J	23
3.5	Setting up the execution environment	25
3.5.1	Building the test environment	26
3.5.2	Starting the test environment	27
3.5.3	Logging in to Management Server Remote Management	28
3.5.4	Setting up Eclipse	28
3.5.5	Importing Eclipse projects	35
3.5.6	Deploying the web project	38
3.5.7	Deploying definitions to the HCSC server	40
4	Executing Sample Programs	47
4.1	Executing sample programs	48
4.2	Operation when business processes are not applied	49
4.3	Operation when a business process is applied	51

4.4 Operation when processes of multiple services are integrated	53
--	----

Part 2: APPLICATION

5

Experiencing the Development of Sample Programs	55
5.1 Procedure for developing sample programs	56
5.2 Creating the HCSCTE project	58
5.3 Defining service adapters	61
5.3.1 Creating a service adapter	61
5.3.2 Validating and packaging a service adapter	64
5.3.3 Defining deployment of a service adapter	65
5.4 Defining business processes	67
5.4.1 Creating business processes	67
5.4.2 Validating and packaging a Hello business process	76
5.4.3 Defining deployment of a Hello business process	76
5.5 Developing the product arrangement system	77
5.5.1 Defining the stock management service adapter	77
5.5.2 Defining the delivery reception service adapter	79
5.5.3 Defining the product arrangement business process	81
5.5.4 Validating and packaging components	101
5.5.5 Defining deployment of components	101
5.6 Debugging the product arrangement system	102
5.7 Preparing for running the developed sample program	110
5.8 Defining data transformation by using a Java program	111
5.8.1 Overview of defining the CustomFunction sample program	112
5.8.2 Preparing the custom function	112
5.8.3 Modifying the Hello service adapter	119
5.8.4 Modifying the Hello business process	126

6

Deleting the Environment for Sample Programs	131
6.1 Deleting projects	132
6.1.1 Undeploying and deleting web projects	132
6.1.2 Deleting definitions deployed to the HCSC server	133
6.2 Stopping the test environment	135
6.3 Unsetup and uninstallation	136
6.3.1 Undoing setup of the test environment	136
6.3.2 Undoing setup of Eclipse	136
6.3.3 Uninstalling Service Architect	138

Appendixes	139
A. Configuration of sample program files	140
A.1 Configuration of the HelloServiceAdapter sample program	140
A.2 Configuration of the HelloBusinessProcess sample program	141
A.3 Configuration of the HelloProductArrangement sample program	142
A.4 Configuration of the CustomFunction sample program	144
B. Collecting the Information Output When Eclipse Setup Was Executed	145
C. Glossary	146
Index	147

1

Before Using Sample Programs

This chapter provides an overview of this manual and what you can expect from it.

1.1 Overview of this manual

This manual gives you experience in performing tasks from creating the service platform environment to executing sample programs through actual operations on a machine.

Chapter 2 describes the three sample programs that are provided by this product, and the components of each sample program.

Chapter 3 describes the procedure for creating the service platform environment and the preparation for executing the three sample programs.

Chapter 4 describes the procedure for executing the sample programs that were prepared in Chapter 3.

Chapter 5 describes the procedure for creating programs (sample programs used in Chapters 3 and 4) in the development environment.

Chapter 6 describes the procedure for deleting the service platform and sample program environments that were prepared in Chapter 3.

The chapters of this manual are intended to be read sequentially. Read through the chapters in the organized order, while operating the machine.

Tip

Conventions: Fonts and symbols

This manual uses the following conventions for screens and operations:

Text formatting	Convention
<i>Italic</i>	Italic characters indicate a placeholder for text to be provided by the user or system.
Monospace	Monospace characters indicate text that the user enters without change, or text (such as messages) output by the system.

Conventions: Types of text in syntax

The following defines the types of text in syntax:

Type	Definition
Alphabetic	A-Z and a-z
Alphanumeric	A-Z, a-z, and 0-9
String	Any string of characters

Conventions: Installation directory

The default installation directory for the service platform is `C:\Program Files\Hitachi\Cosminexus`. If you do not want to use the default installation directory, replace *service-platform-installation-directory* with the name of the desired directory in this manual.

Conventions: Language pack

This manual uses the Eclipse menu item names for which Babel Japanese Language Pack (version 2012-07-21) provided by Eclipse Babel Project was applied. The menu item names might differ depending on the version of the language pack.

1.2 What you can expect from this manual

This manual gives you experience in using Eclipse to perform tasks from setting up the environment for sample programs to developing and actually executing the sample programs.

The following explains what to expect from each chapter:

- Experiences in Chapter 3
You can install Service Architect, use Eclipse Setup to set up the Eclipse environment with HCSCTE plug-ins installed, and use HCSC Easy Setup to set up the execution environment.
- What to expect from Chapter 4
You can use the environment that you set up in Chapter 3 to execute sample programs from a web browser.
- Experiences in Chapter 5
By performing the procedures as described, you can use the development environment for the service platform to gain experience in performing a sequence of steps, from developing to executing three sample programs.
During development of sample programs, you can perform operations related to service adapters, business processes, data conversion definition, and other basic functions for using the service platform. You can also perform operations related to debugging functions provided by the service platform, by using a sample program as an example.
- Experiences in Chapter 6
You can delete the project that was prepared in Chapter 3, stop the test environment, undo setup of Eclipse, and uninstall Service Architect.

2

Overview of Sample Programs

This chapter provides an overview of sample programs. This chapter also describes learning points and objectives of the sample programs.

2.1 System configurations of sample programs

The service platform provides sample programs that execute the following three types of processing:

- Calling a service component from the service requester
- Calling a service component from the business process
- Integrating processes

The following describes the processing of each sample program:

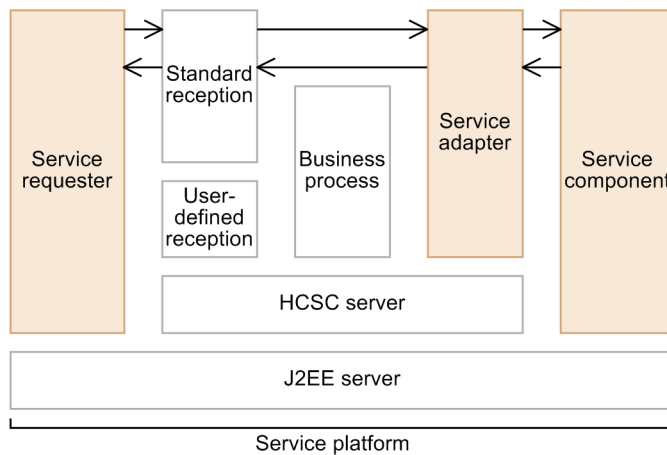
(1) Calling a service component from the service requester

Sample program name: `HelloServiceAdapter`

This sample program calls a service component from the service requester via the service adapter. With this sample program, you will learn about service adapter definitions.

The following figure shows the system configuration of `HelloServiceAdapter`:

Figure 2–1: System configuration of `HelloServiceAdapter`



Legend:

- : Flow of tasks for a request or response that calls a service component
- : Component provided by this sample program

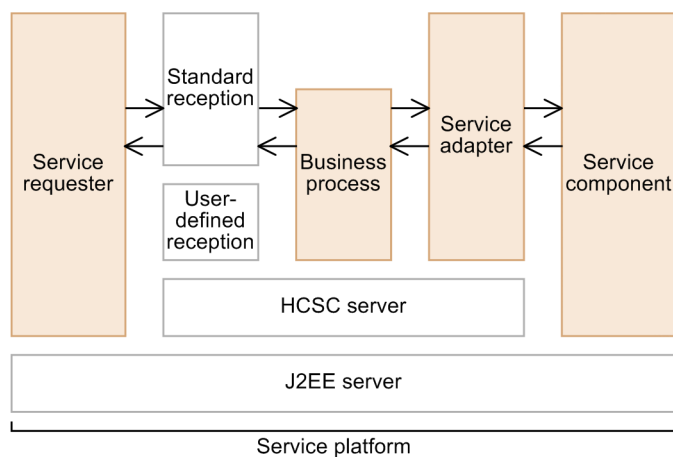
(2) Calling a service component from the business process

Sample program name: `HelloBusinessProcess`

This sample program calls a service component from the business process via the service adapter. With this sample program, you will learn about business process definitions.

The following figure shows the system configuration of `HelloBusinessProcess`:

Figure 2–2: System configuration of HelloBusinessProcess



Legend:

- : Flow of tasks for a request or response that calls a service component
 [Orange Box] : Component provided by this sample program

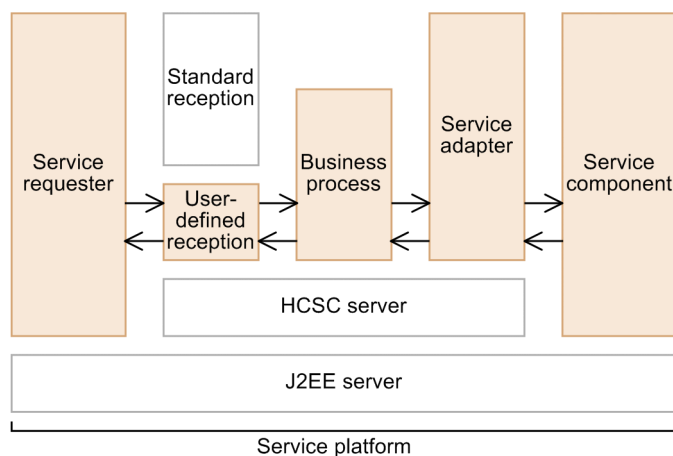
(3) Integrating processes

Sample program name: HelloProductArrangement

This sample program calls a service component that allocates product stock or arranges for delivery from the business process via the service adapter. With this sample program, you will learn about definitions that are similar to the definitions for actual jobs.

The following figure shows the system configuration of HelloProductArrangement:

Figure 2–3: System configuration of HelloProductArrangement



Legend:

- : Flow of tasks for a request or response that calls a service component
 [Orange Box] : Component provided by this sample program

2.2 Components of sample programs

This section describes the roles of the sample program components shown in *2.1 System configurations of sample programs*.

- **Service requester**
An application program that uses a service. This component sends a service calling request (request message) to the HCSC server. The service requester in the sample program uses SOAP Communication Infrastructure.
- **Standard reception**
A function (interface) for receiving a service component request message from the service requester. This component is provided as a function of the HCSC server.
- **User-defined reception**
A function (interface) for receiving a service component request message from the service requester. For this component, unlike the standard reception provided as a function of the HCSC server, the user can define any interface.
- **Business process**
The processing order and processing conditions of services, and other information defined as a sequence of tasks.
- **Service adapter**
A function that receives a request from the service requester or business process and calls a service. This component exists on the HCSC server. This component returns the results of calling a service to the business process or service requester as a response message.
- **Service component**
A business-processing program. This component executes the processing requested by the service requester.
- **HCSC server**
A server function that manages the service adapter and business process to execute a service. This component exists in the service platform.
- **J2EE server**
A server function required to execute a J2EE application (application consisting of JSPs, servlets, enterprise beans, and other items).

2.3 Processing details of sample programs

This section describes the processing details of the components that make up each sample program.

2.3.1 Calling a service component from the service requester

The following figure and table describe the processing details of the `HelloServiceAdapter` sample program:

Figure 2–4: Processing details of the `HelloServiceAdapter` sample program

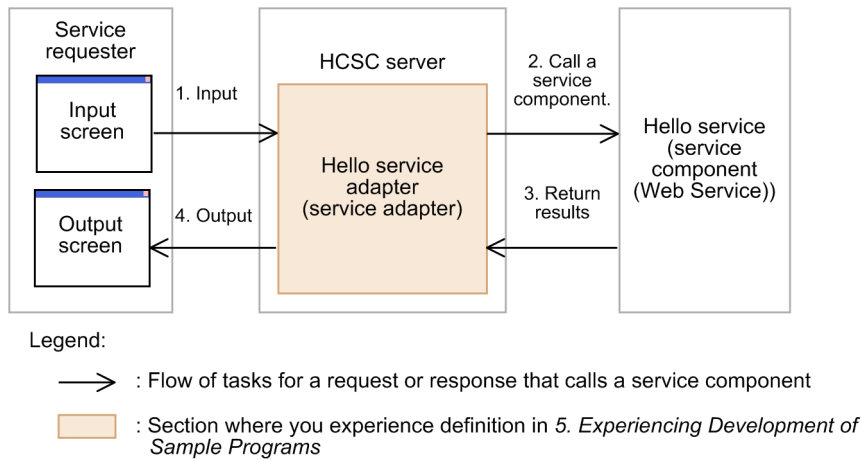


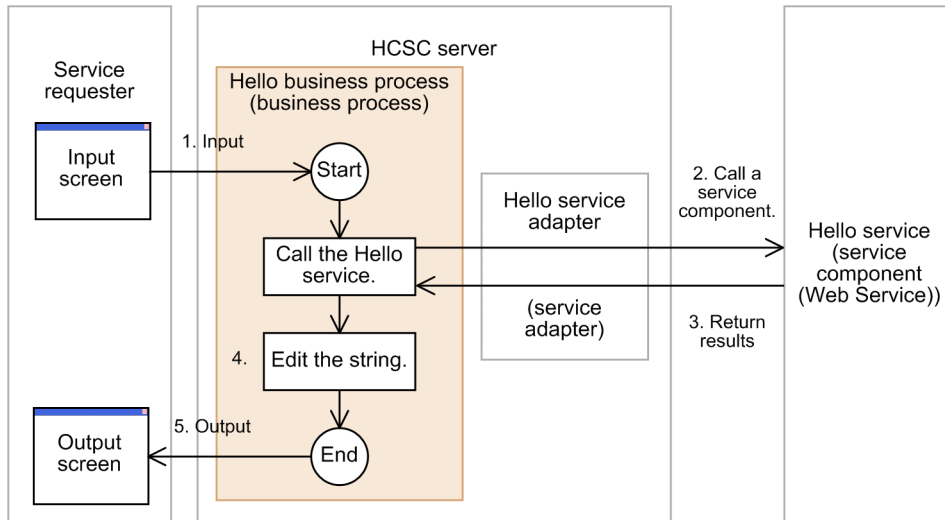
Table 2–1: Processing contents of the `HelloServiceAdapter` sample program

Component	Type	Description
Service requester	HTML Servlet	<ul style="list-style-type: none"> Displays the input screen, and sends a Hello service calling request to the service adapter. Receives the results of calling the Hello service from the service adapter, and displays them on the output screen.
Hello service adapter	Service adapter	<ul style="list-style-type: none"> Receives a request from the service requester, and calls the Hello service. Returns the response of the Hello service to the service requester.
Hello service	Web Service	<ul style="list-style-type: none"> In response to a call from the service adapter, edits the entered string and returns the string to the Hello service adapter.

2.3.2 Calling a service component from the business process

The following figure and table describe the processing details of the `HelloBusinessProcess` sample program:

Figure 2–5: Processing details of the HelloBusinessProcess sample program



Legend:

→ : Flow of tasks for a request or response that calls a service component

: Section where you experience definition in 5. *Experiencing Development of Sample Programs*

Table 2–2: Processing contents of the HelloBusinessProcess sample program

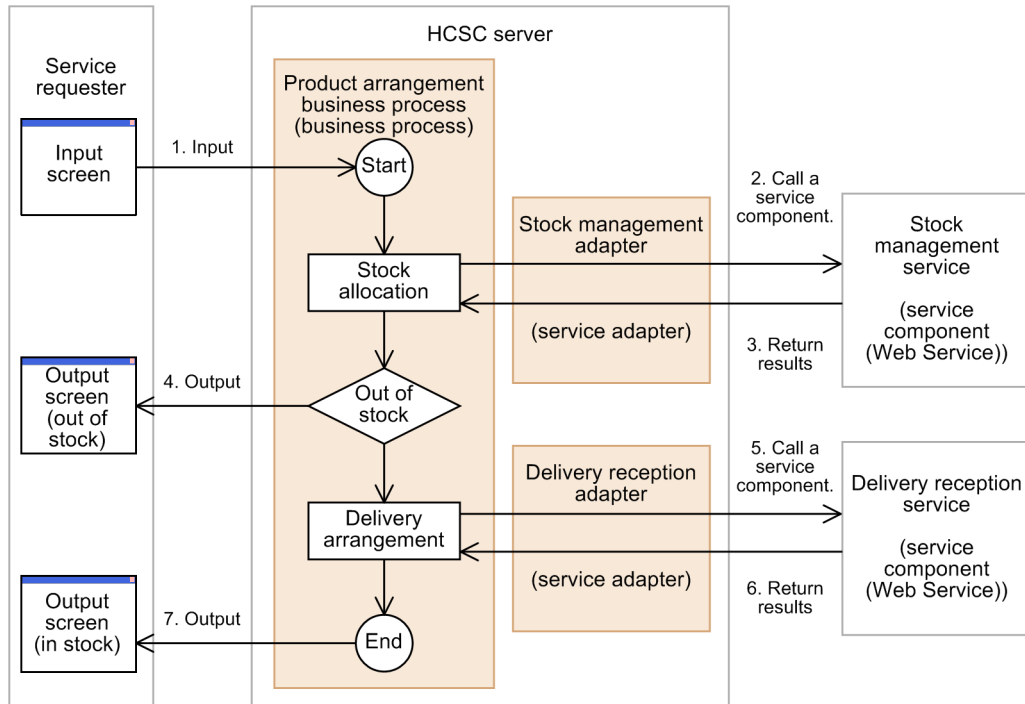
Component	Type	Description
Service requester	HTML Servlet	<ul style="list-style-type: none"> Displays the input screen, and calls the Hello business process. Obtains (as the return value) the string edited by the Hello business process, and displays the string on the output screen.
Hello business process	Business process	<ul style="list-style-type: none"> Receives a request from the service requester, and calls the Hello service via the Hello service adapter. Obtains (as the return value) the results of calling the Hello service via the service adapter. Edits the string (return value), and returns it to the service requester.
Hello service adapter	Service adapter	<ul style="list-style-type: none"> Receives a request from the Hello business process, and calls the Hello service. Returns the response of the Hello service to the Hello business process.
Hello service	Web Service	<ul style="list-style-type: none"> In response to a call from the Hello business process, edits the entered string and returns the string to the Hello business process.

2.3.3 Integrating processes

The `HelloProductArrangement` sample program introduced here assumes that shop staff use a business system to perform stock allocation and delivery arrangement when securing products.

The following figure and table describe the processing details of the `HelloProductArrangement` sample program:

Figure 2–6: Processing details of the HelloProductArrangement sample program



Legend:

→ : Flow of tasks for a request or response that calls a service component

: Section where you experience definition in 5. *Experiencing Development of Sample Programs*

Table 2–3: Processing contents of the HelloProductArrangement sample program

Component	Type	Description
Service requester	HTML Servlet JSP	<ul style="list-style-type: none"> Displays the input screen, and calls the product arrangement business process. Obtains (as the return value) the results of calling the stock management service, and displays them on the output screen. Obtains (as the return value) the results of calling the delivery reception service, and displays them on the output screen.
Product arrangement business process	Business process	<ul style="list-style-type: none"> Receives a request from the service requester, and calls the stock management service via the stock management adapter. Receives a request from the service requester, and calls the delivery reception service via the delivery reception adapter. Obtains the results of calling the stock management service via the stock management adapter, and returns them to the service requester. Obtains the results of calling the delivery reception service via the delivery reception adapter, and returns them to the service requester.
Stock management adapter	Service adapter	<ul style="list-style-type: none"> Receives a request from the product arrangement business process, and calls the stock management service. Returns the response of the stock management service to the product arrangement business process.

2. Overview of Sample Programs

Component	Type	Description
Delivery reception adapter	Service adapter	<ul style="list-style-type: none">• Receives a request from the product arrangement business process, and calls the delivery reception service.• Returns the response of the delivery reception service to the product arrangement business process.
Stock management service	Web Service	<ul style="list-style-type: none">• In response to a call from the product arrangement business process, allocates product stock and returns the allocation number to the product arrangement business process.
Delivery reception service	Web Service	<ul style="list-style-type: none">• In response to a call from the product arrangement business process, returns the delivery number obtained as a result of delivery arrangement.

3

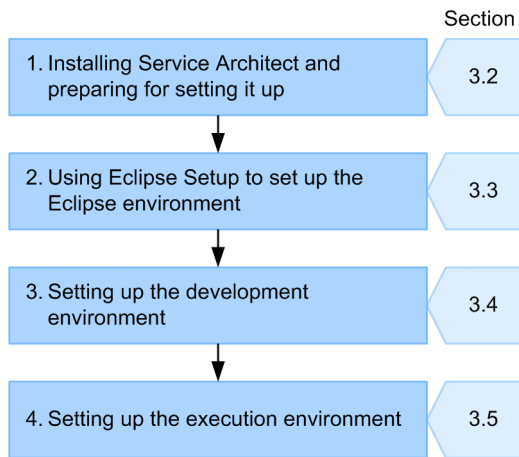
Preparing the Environment for Sample Programs

This chapter describes how to build the environment required to execute sample programs.

3.1 Overview of installation and setup

The following figure shows an overview of building the environment for executing sample programs:

Figure 3–1: Overview of building the environment for executing sample programs



1. Installing Service Architect and preparing for setting it up

Use the installer to install Service Architect. Also install Eclipse, which is required for setup. For details, see 3.3 *Using Eclipse Setup to set up the Eclipse environment*.

2. Using Eclipse Setup to set up the Eclipse environment

Use Eclipse Setup to set up the Eclipse-based development environment with HCSCTE plug-ins installed. For details, see 3.2 *Installing Service Architect and preparing for setting it up*.

3. Setting up the development environment

Specify the settings that are necessary to develop J2EE applications by using Eclipse. For details, see 3.4 *Setting up the development environment*.

4. Setting up the execution environment

Use HCSC Easy Setup of Service Architect to build the execution environment. For details, see 3.5 *Setting up the execution environment*.

The following sections describe the above stages.

3.2 Installing Service Architect and preparing for setting it up

This section describes how to install Service Architect and how to prepare Eclipse, which is required to set up Service Architect.

3.2.1 Installing Service Architect

This subsection describes the procedure for installing Service Architect.

! Important note

Before installing Service Architect, exit all Windows applications. In addition, make sure that you are logged on as a member of the Administrators group.

1. Set the installation CD-ROM in the CD-ROM drive.

The Hitachi Integrated Installer dialog box appears, displaying the following message: The selected software will be installed.

If the Hitachi Integrated Installer dialog box does not appear, use Explorer to double-click HCD_INST.EXE in the CD-ROM directory.

2. Select **uCosminexus Service Architect**, and then click the **Install** button.

The Confirm Starting of Installer - Hitachi Integrated Installer dialog box appears, displaying the following message: "Installation will start. Do you want to continue?"

3. Click the **OK** button.

The **Welcome to the uCosminexus Service Architect Setup Program** page appears.

4. Click the **Next** button.

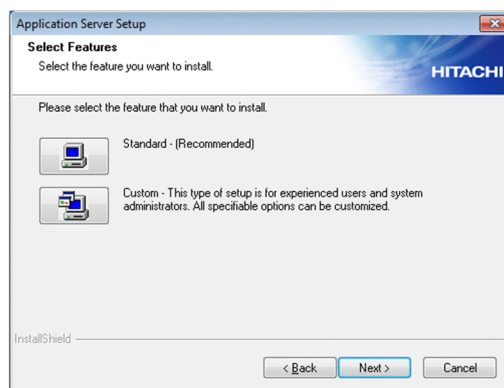
The **Choose Destination Location** page appears.

5. If you do not want to use the default installation directory, select the desired installation directory, and then click the **Next** button.

Tip

When you specify the installation directory that is not the default one, make sure that the directory path name is 50 or fewer alphanumeric characters.

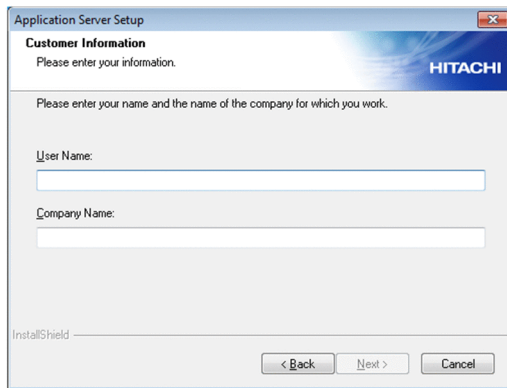
The **Select Features** page appears.



6. Click the button on the left of **Standard - (Recommended)**.

The **Customer Information** page appears.

3. Preparing the Environment for Sample Programs



7. Enter **User Name** and **Company Name**, and then click the **Next** button.

The **Select Program Folder** page appears.

8. If you do not want to use the default program folder, select the desired program folder, and then click the **Next** button.

The **Start the installation** page appears.

9. Check the settings that you specified, and then, if no problems exist, click the **Next** button.

Installation starts. When installation finishes, the Completing the Setup dialog box appears. Installation might take a few minutes.

10. Click the **Finish** button.

The dialog box asking you whether to immediately restart the OS appears.

11. Click the **Yes** button.

The OS restarts, and installation of Service Architect finishes.

Now you have installed Service Architect.

3.2.2 Preparing Eclipse

Prepare the environment for using Eclipse. The following shows the procedure for preparing the Eclipse archive file:

1. Obtain the Eclipse archive file.

Obtain the Eclipse archive file shown below. Although it comes with Service Architect, you can also download it from the Eclipse.org download site.

For Windows x86 (including the WOW64 environment):

Eclipse IDE for Java EE Developers 4.2.1 (eclipse-jee-juno-SR1-win32.zip)

For Windows x64:

Eclipse IDE for Java EE Developers 4.2.1 (eclipse-jee-juno-SR1-win32-x86_64.zip)

2. Store the Eclipse archive file.

Store the obtained Eclipse archive file in the following directory:

service-platform-installation-directory\ADP\archives

Now you are ready to install Eclipse.

Reference note

The Eclipse language pack comes with Service Architect. If you store the language pack in the directory that contains the Eclipse archive file, you can localize Eclipse into Japanese.

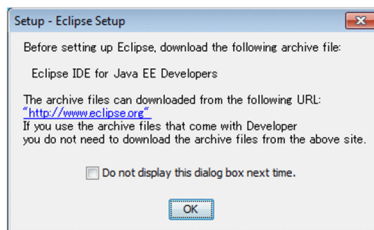
3.3 Using Eclipse Setup to set up the Eclipse environment

Use Eclipse Setup to build the Eclipse environment.

The following shows the procedure for building the Eclipse environment:

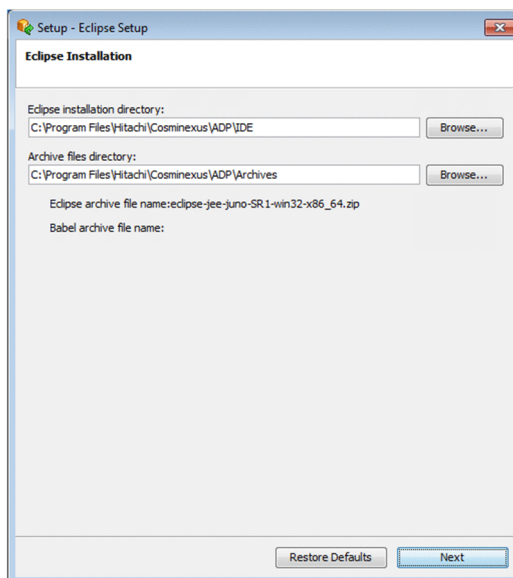
1. Start the machine as a user with administrator privileges. (In Windows Vista or later, start the machine in Admin mode.)
2. From the Windows **Start** menu, select **Programs, Cosminexus, First Setup**, and then **Setup Eclipse**.

The dialog box asking you whether the Eclipse archive file was prepared appears.



3. In the dialog box that appears, click the **OK** button.

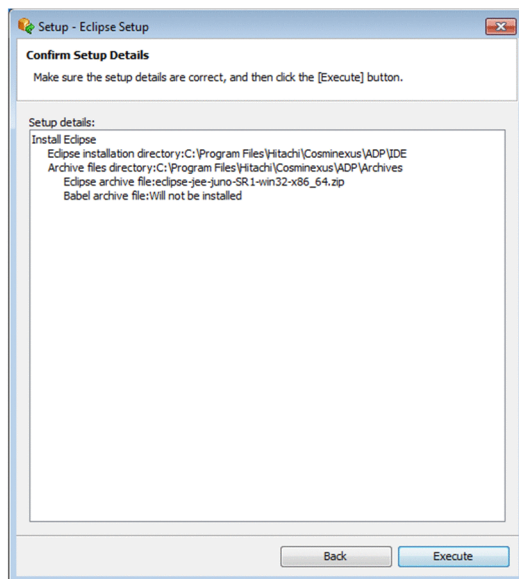
The **Eclipse Installation** page appears.



4. On the **Eclipse Installation** page, specify the Eclipse installation directory and the directory that contains the Eclipse archive file, and then click the **Next** button.

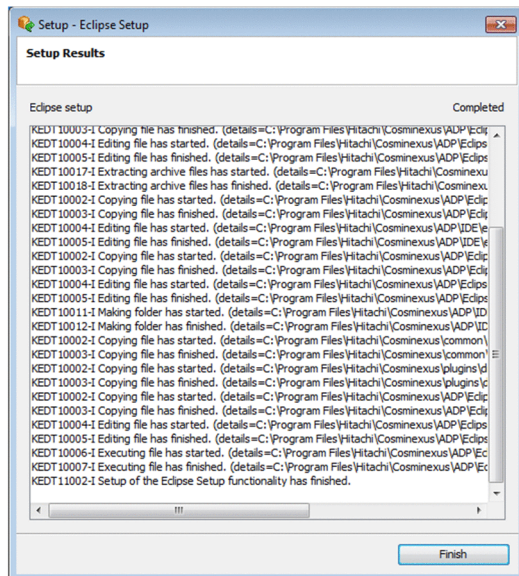
The **Confirm Setup Details** page appears.

3. Preparing the Environment for Sample Programs



5. On the **Confirm Setup Details** page, confirm that the displayed information is correct, and then click the **Execute** button.

The **Progress Status** page appears and setup starts. When setup finishes, the **Setup Results** page appears. Eclipse setup might take a few minutes.



6. On the **Setup Results** page, click the **Finish** button.

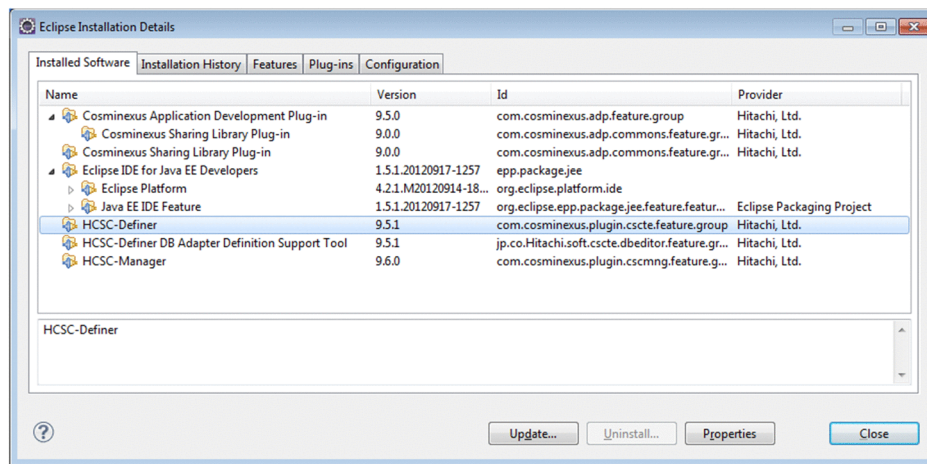
The Setup - Eclipse Setup dialog box closes and Eclipse setup finishes.

A shortcut to Eclipse is created on the desktop.

Now you have built the Eclipse environment.

Confirm that Eclipse was set up correctly.

1. Start Eclipse.
2. From the Eclipse menu, select **Help**, **About Eclipse**, and then **Installation Details** to display the Eclipse Installation Details dialog box.
Confirm that the version number displayed for **HCSC-Definer** is the version number of the Eclipse instance that you installed.



Reference note

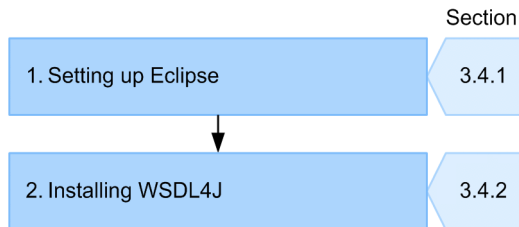
How to check the setup log

The events that occurred when Eclipse Setup is executed are recorded in the Eclipse setup log. For details about how to check the setup log, see *Appendix B. Collecting the Information Output When Eclipse Setup Was Executed*.

3.4 Setting up the development environment

This section describes the setup that is required to develop J2EE applications. The following figure shows an overview of setup:

Figure 3–2: Overview of setup for the development environment



The following is a brief description of the stages in the above figure:

1. Setting up Eclipse

In this stage, you set the JDK version to be used in Eclipse and specify the settings for outputting the information about local variables. For details, see *3.4.1 Setting up Eclipse*.

2. Installing WSDL4J

In this stage, you install WSDL4J. WSDL is required to parse and generate documents. For details, see *3.4.2 Installing WSDL4J*.

The following subsections describe the stages of development environment setup in the same order as in the above figure.

3.4.1 Setting up Eclipse

This subsection shows the procedure for checking the JDK version set in Eclipse. This subsection also shows the procedure for specifying the settings for outputting information about the local variables that are in the J2EE application.

(1) Checking the version of JDK

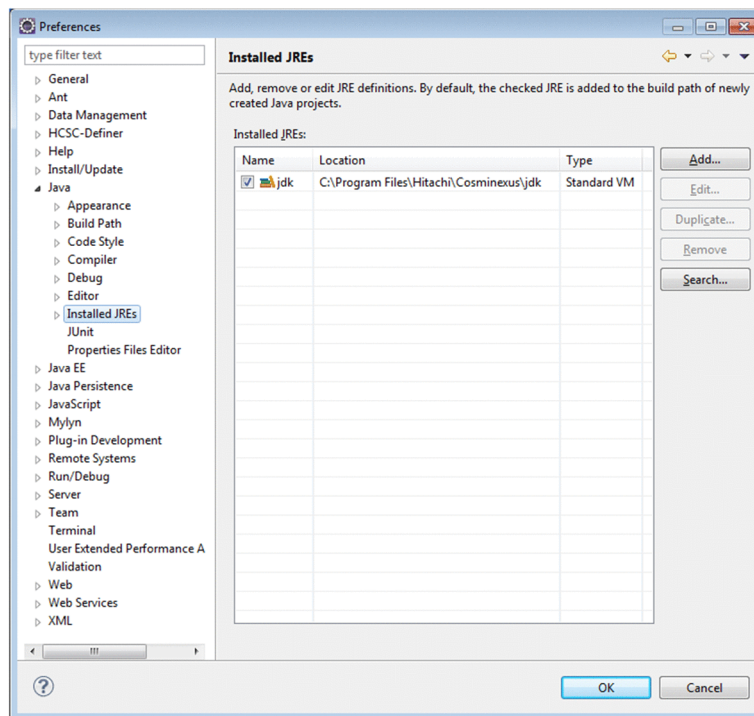
Check whether the JDK version to be used for development in the Eclipse environment is a JDK version provided by Service Architect. The following shows the check procedure:

1. From the Eclipse menu, select **Window**, and then **Preferences**.

The Preferences dialog box appears.

2. In the left pane, select **Java**, and then **Installed JREs**.

The **Installed JREs** page is displayed in the right pane.

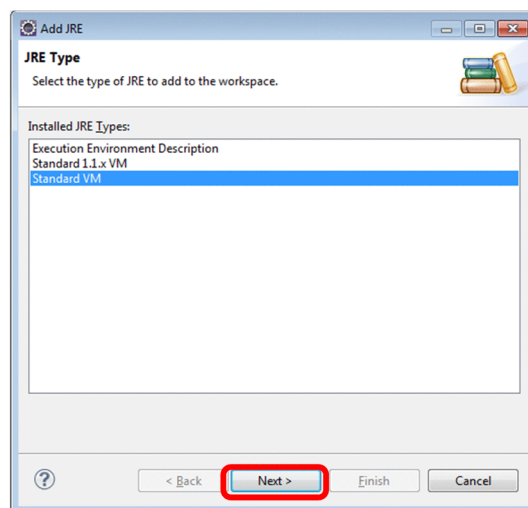


3. Check whether a JDK version provided by Service Architect is displayed in the **Installed JREs** list.
Check whether the following path is displayed in the **Location** column:

`service-platform-installation-directory\jdk`

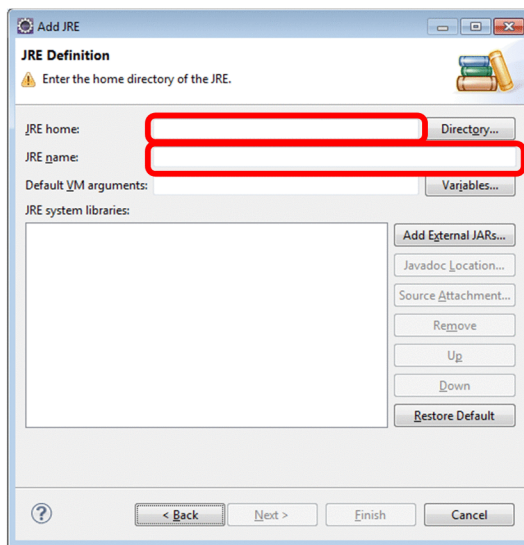
If the path is not displayed:

Click the **Add** button. The **JRE Type** page appears.



Select **Standard VM**, and then click the **Next** button. The **JRE Definition** page appears.

3. Preparing the Environment for Sample Programs



Enter *service-platform-installation-directory\jdk* in **JRE home**, enter *jdk* in **JRE name**, and then click the **Finish** button. After you add the entry, select the check box in the **Name** column.

If the path is displayed:

Check whether the check box in the **Name** column for the appropriate version is selected. If that check box is not selected, select it.

Note that if two or more JDK versions have been installed, the check box for the appropriate version (*service-platform-installation-directory\jdk*) might not be selected. If that check box is not selected, select it.

4. Click the **OK** button.

The settings are saved.

(2) Specifying the settings for outputting the information about local variables

You can specify the Eclipse compiler settings so that the information about local variables in the J2EE application is output as a stack trace when an exception occurs.

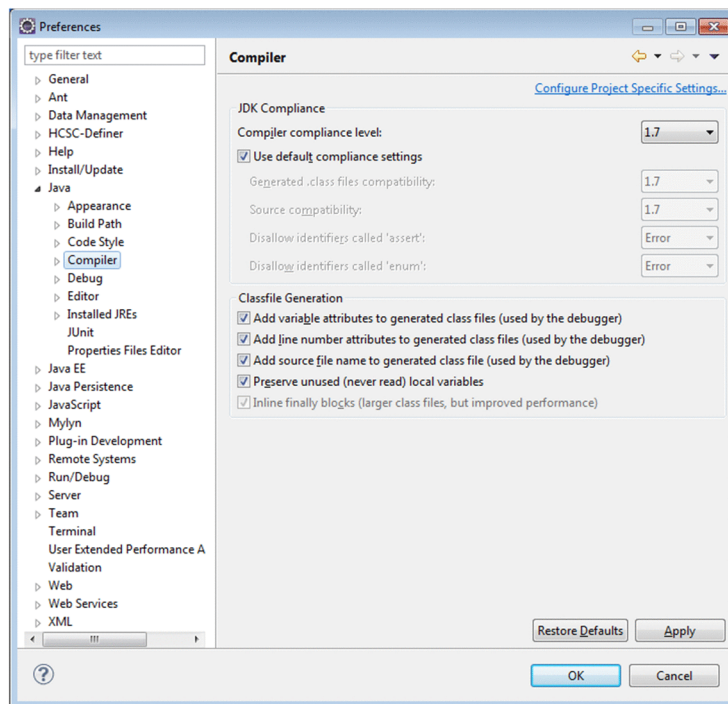
The following shows the procedure for specifying the compiler settings to output the information about local variables as a stack trace:

1. From the Eclipse menu, select **Window**, and then **Preferences**.

The Preferences dialog box appears.

2. In the left pane, select **Java**, and then **Compiler**.

The **Compiler** page is displayed.



3. Specify the following settings:

Item name		Value to be specified
JDK Compliance	Compiler compliance level	Select 1.7.
Classfile Generation		Select Add variable attributes to generated class files (used by the debugger) . For other items, select the necessary options according to the information that you want to output.

Optionally, specify the following settings:

Item name		Value to be specified
JDK Compliance	Use default compliance settings	Specify the settings that you want to use for the compiler. <ul style="list-style-type: none"> If this item is selected: The settings that are appropriate for the level specified by Compiler compliance level are applied. If this item is not selected: Manually specify the following options: Generated .class file compatibility Source compatibility Disallow identifiers called 'assert' Disallow identifiers called 'enum'

4. Click the **Apply** or **OK** button.
The settings are saved.

3.4.2 Installing WSDL4J

Install WSDL4J, which is required to parse and generate WSDL documents. WSDL4J is supplied with a CD-ROM different from the Service Architect CD-ROM.

1. Create a work directory that will be temporarily used to install WSDL4J.

3. Preparing the Environment for Sample Programs

In this example, create `C:\WSDL4J_work`.

2. Copy the `wsdl4j-bin-1.5.1.zip` file from the WSDL4J CD-ROM to the WSDL4J work directory that you created in step 1.

The `wsdl4j-bin-1.5.1.zip` file is stored in the following directory:

CD-ROM-drive:\WSDL4J

3. Decompress the `wsdl4j-bin-1.5.1.zip` file that was copied to the WSDL4J work directory.
4. Copy `wsdl4j.jar` (JAR library file), `license.html` (license file), and `Readme`, which were extracted from the compressed file.

Files to be copied:

WSDL4J-work-directory\wsdl4j-bin-1.5.1\wsdl4j-1_5_1\lib\wsdl4j.jar

WSDL4J-work-directory\wsdl4j-bin-1.5.1\wsdl4j-1_5_1\license.html

WSDL4J-work-directory\wsdl4j-bin-1.5.1\wsdl4j-1_5_1\Readme

Copy-destination directory:

service-platform-installation-directory\c4web\lib

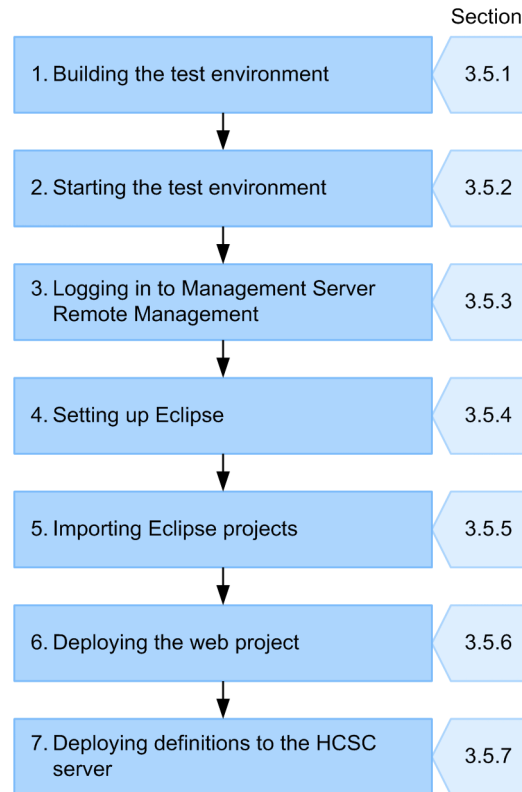
5. Delete the WSDL4J work directory that you created in step 1.

3.5 Setting up the execution environment

This section describes the procedure for using HCSC Easy Setup of Service Architect to set up the execution environment (test environment).

The following figure shows an overview of setup:

Figure 3–3: Overview of setup for the execution environment



The following is a brief description of the stages in the above figure:

1. Building the test environment
In this stage, you use HCSC Easy Setup to build the test environment. For details, see *3.5.1 Building the test environment*.
2. Starting the test environment
In this stage, you start the test environment that you built. For details, see *3.5.2 Starting the test environment*.
3. Logging in to Management Server Remote Management
In this stage, you log in from Eclipse to Management Server Remote Management to perform the task in step 4. For details, see *3.5.3 Logging in to Management Server Remote Management*.
4. Setting up Eclipse
In this stage, you set library paths, create a server runtime, and select the J2EE server. For details, see *3.5.4 Setting up Eclipse*.
5. Importing an Eclipse project
In this stage, you import the Eclipse projects for the sample programs. For details, see *3.5.5 Importing Eclipse projects*.
6. Deploying the web project
In this stage, you deploy the web project that is appropriate for the sample program you will use. For details, see *3.5.6 Deploying the web project*.
7. Deploying definitions to the HCSC server

3. Preparing the Environment for Sample Programs

In this stage, you deploy definitions (that are appropriate for the sample program you will use) to the HCSC server. The definitions to be deployed differ depending on the sample program. For details, see 3.5.7 *Deploying definitions to the HCSC server*.

The following subsections describe the stages of execution environment setup in the same order as in the above figure.

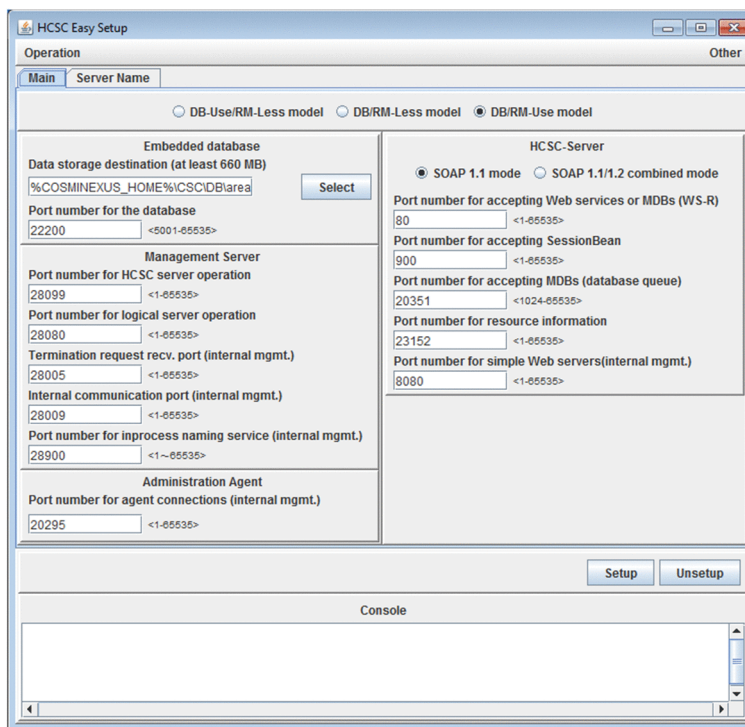
3.5.1 Building the test environment

To build the test environment, use HCSC Easy Setup. The following shows how to build the test environment:

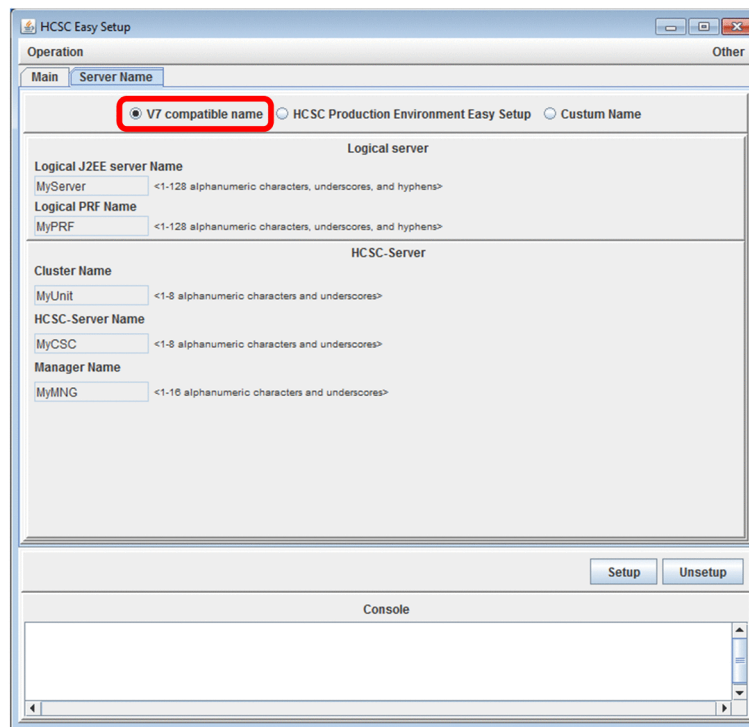
1. Make sure that Eclipse is not running.
2. From the **Start** menu, select **Programs, Cosminexus[#], First Setup**, and then **Setup Testing Environment**.
The **Main** page of the HCSC Easy Setup window opens.
Select the **DB/RM-Use model** radio button.

#

This program folder name might have been changed. If it has been changed, select the changed program folder name.



3. Click the **Server Name** tab.
The **Server Name** page appears.
4. Select the **V7 compatible name** radio button.
The names of the logical server and HCSC server are changed as follows:
 - **V7 compatible name** radio button
 - **Logical J2EE server Name** = MyServer
 - **Logical PRF Name** = MyPRF
 - **Cluster Name** = MyUnit
 - **HCSC-Server Name** = MyCSC
 - **Manager Name** = MyMNG



- Click the **Setup** button.

Setup of the test environment starts. The progress of setup is displayed in the Console of the HCSC Easy Setup window. Setup of the test environment normally terminates when the Console displays a message that indicates the end of setup by HCSC Easy Setup.

Note:

If an error is displayed in the Console and setup of the test environment terminates abnormally, you must perform re-setup. The re-setup procedure differs depending on whether the **Setup** button in the HCSC Easy Setup window was enabled or disabled when setup terminated abnormally.

If the **Setup** button was enabled:

Click the **Setup** button to perform setup again.

If the **Setup** button was disabled:

Click the **Unsetup** button to undo setup. After that, perform setup again.

- From the menu of the HCSC Easy Setup window, select **Operation** and then **End** to close the window.

3.5.2 Starting the test environment

The following shows how to start the test environment that you built:

- From the **Start** menu, select **Programs, Cosminexus[#]**, and then **Start Database** to start the embedded database of the test environment.

When you start the database, the following message might be output: KFPS01853-W Hostname=*host-name*, unable to execute pdstart command, unit state not OFFLINE. This message is output because the embedded database is already started when the sample program is executed again on the same machine. You do not need to take any actions when this message is output.

- From the **Start** menu, select **Programs, Cosminexus[#]**, and then **Start Test Server** to start Performance Tracer, the J2EE server, and the HCSC server (including the standard reception and user-defined reception) in the test environment.

#

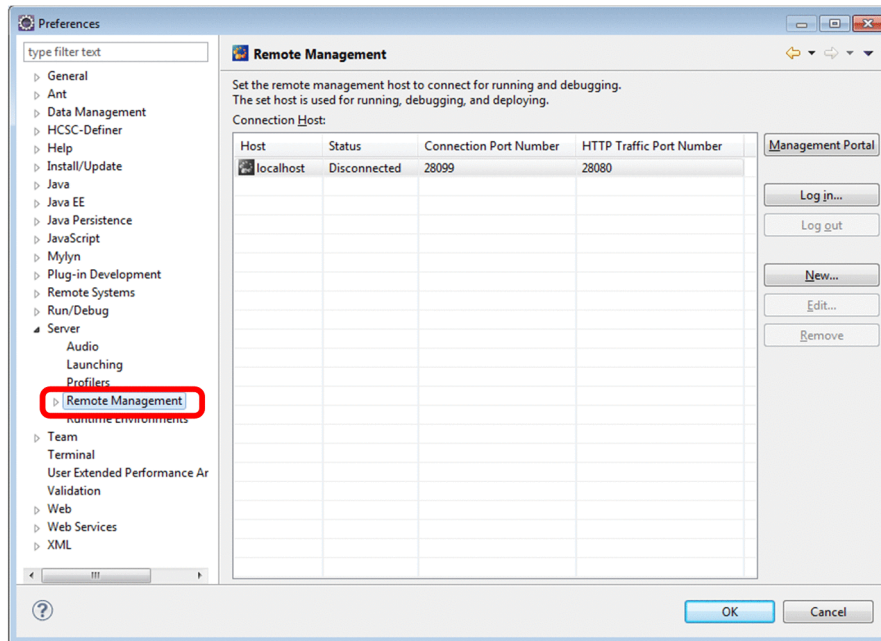
This program folder name might have been changed. If it has been changed, select the changed program folder name.

To terminate using Service Architect, stop the active test environment, and then exit Eclipse. For details about how to stop the test environment, *6.2 Stopping the test environment*.

3.5.3 Logging in to Management Server Remote Management

The following shows the procedure for logging in to Management Server Remote Management:

1. Start Eclipse.
2. From the Eclipse menu, select **Window**, and then **Preferences**.
The Preferences dialog box appears.
3. In the left-pane tree view, select **Server**, and then **Remote Management**.
The **Remote Management** page appears.



4. Select the host of the Management Server to be connected (localhost), and then click the **Log in** button.
The login window appears.
5. Enter admin in **Administrator ID** and **Password**, and then click the **OK** button.
You are connected to Management Server Remote Management.
6. Click the **OK** button.

3.5.4 Setting up Eclipse

In this stage, you set library paths, create a server runtime, and select the J2EE server.

(1) Setting library paths

Set the following service platform library paths:

```
C:\Program Files\Hitachi\Cosminexus\CC\client\lib\j2ee-javax.jar
C:\Program Files\Hitachi\Cosminexus\jasp\lib\csmjasp.jar
C:\Program Files\Hitachi\Cosminexus\c4web\lib\hitjaxrpc.jar
C:\Program Files\Hitachi\Cosminexus\c4web\lib\hitc4web.jar
```

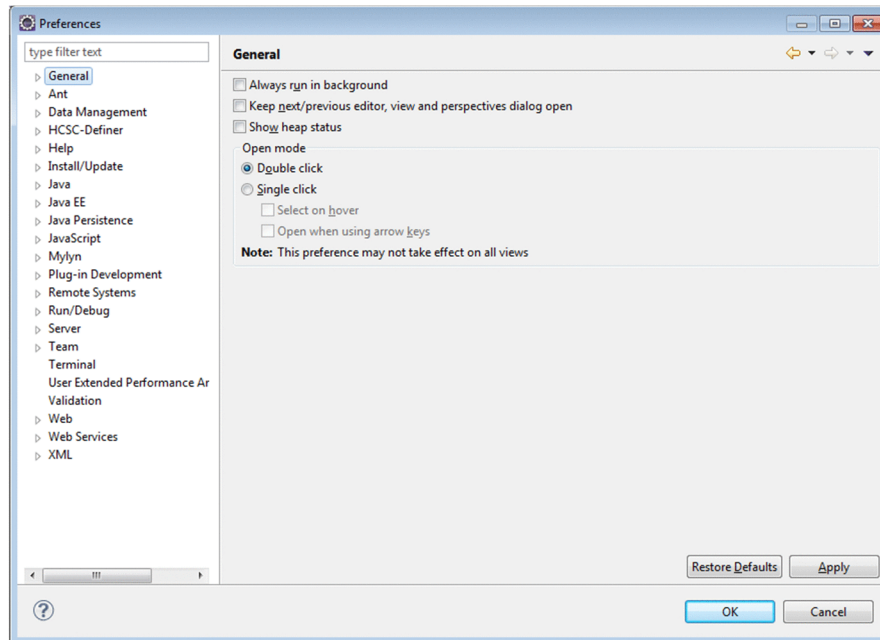
The service platform library path C:\Program Files\Hitachi\Cosminexus is the default installation directory of the service platform. If a directory that is not the default installation directory is specified during

installation of the service platform, you must replace the default installation directory with the directory specified during installation.

The following shows the procedure for setting the library paths:

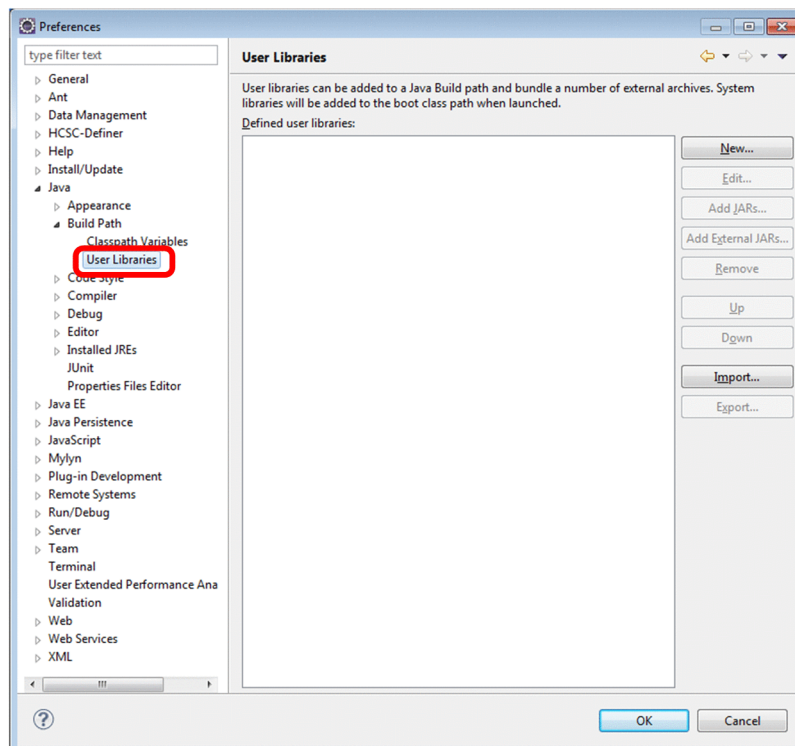
1. From the Eclipse menu, select **Window**, and then **Preferences**.

The Preferences dialog box appears.



2. In the left-pane tree view, select **Java**, **Build Path**, and then **User Libraries**.

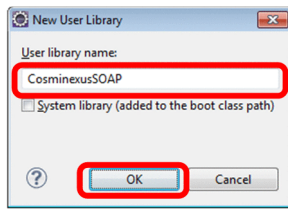
The **User Libraries** page appears in the right pane.



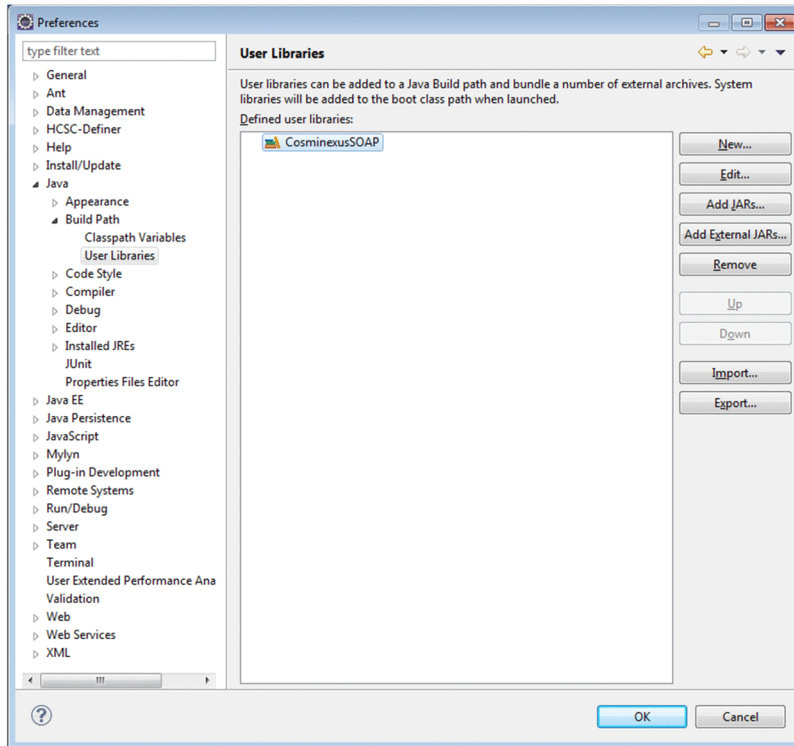
3. Click the **New** button.

The New User Library dialog box opens.

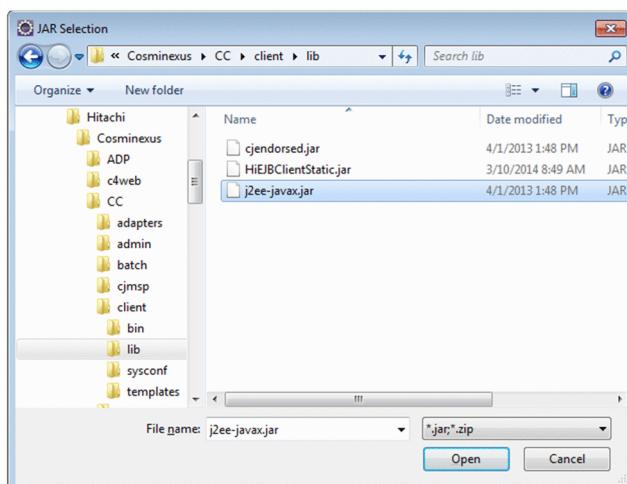
3. Preparing the Environment for Sample Programs



Enter **CosminexusSOAP** in **User library name**, and then click the **OK** button.



4. On the **User Libraries** page, select **CosminexusSOAP**, and then click the **Add External JARs** button. The JAR Selection dialog box opens.

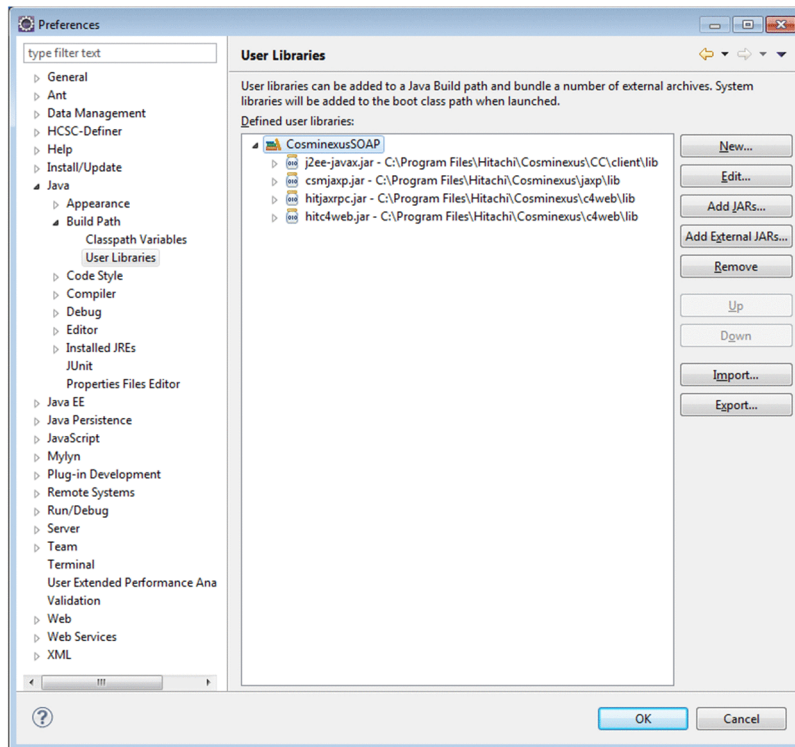


5. Select *service-platform-installation-directory*\CC\client\lib\j2ee-javax.jar, and then click the **Open** button.

The **j2ee-javax.jar** library is added.

6. Using the same procedure as above, add the following JAR files by clicking the **Add External JARs** button:

- `service-platform-installation-directory\jaxp\lib\csmjaxp.jar`
- `service-platform-installation-directory\c4web\lib\hitjaxrpc.jar`
- `service-platform-installation-directory\c4web\lib\hitc4web.jar`



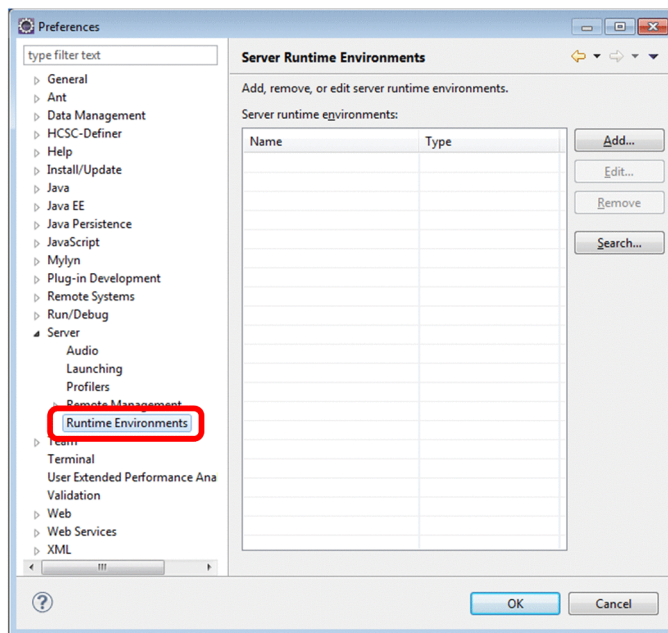
7. Click the **OK** button.
8. Restart Eclipse.

(2) Creating a server runtime

Create a server runtime before you can operate the J2EE server. A server runtime is a collection of environment settings, such as the J2EE server installation destination. The following shows the creation procedure:

1. From the Eclipse menu, select **Window**, and then **Preferences**.
The Preferences dialog box appears.
2. From the left-pane tree view, select **Server**, and then **Runtime Environments**.
The **Server Runtime Environments** page appears in the right pane.

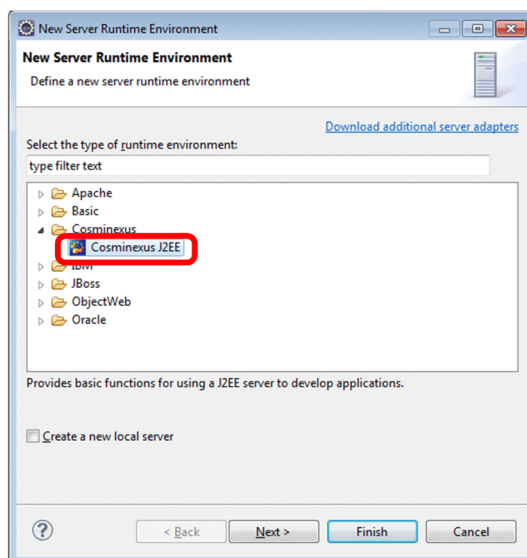
3. Preparing the Environment for Sample Programs



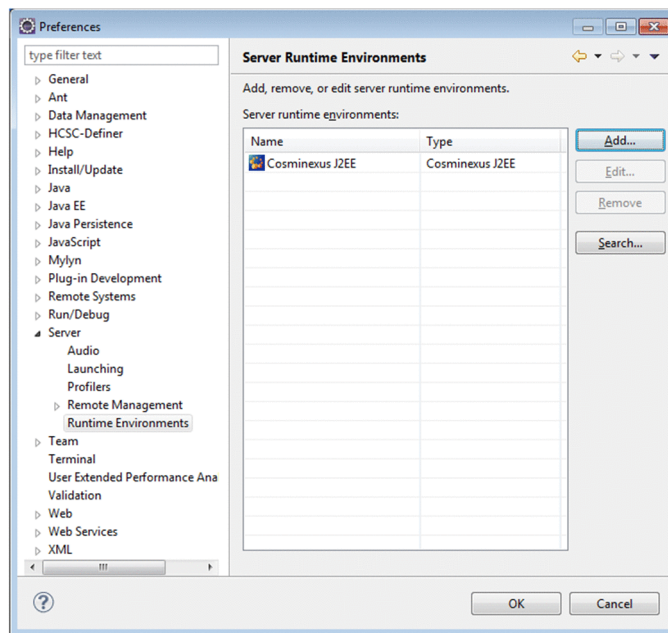
3. Click the **Add** button.

The New Server Runtime Environment dialog box opens.

4. In the tree view, select **Cosminexus**, and then **Cosminexus J2EE**.



5. Click the **Finish** button.



6. Confirm that **Cosminexus J2EE** was added to the **Server runtime environments** list, and then click the **OK** button.

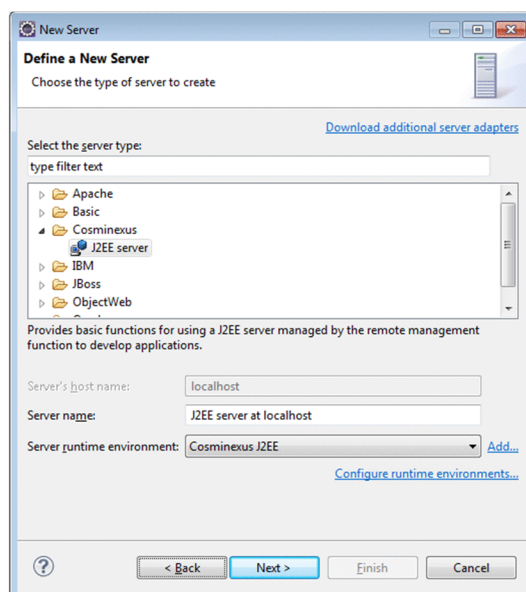
The settings are saved.

(3) Selecting the J2EE server

Select the J2EE server that you will operate from Eclipse. For the J2EE server selected here, you perform start and stop operations, or you deploy J2EE applications from Eclipse.

The following shows the setup procedure:

1. From the Eclipse menu, select **File, New,** and then **Other.**
The New dialog box appears.
2. Select **Server,** and then **Server,** and then click the **Next** button.
The New Server dialog box opens, displaying the **Define a New Server** page.



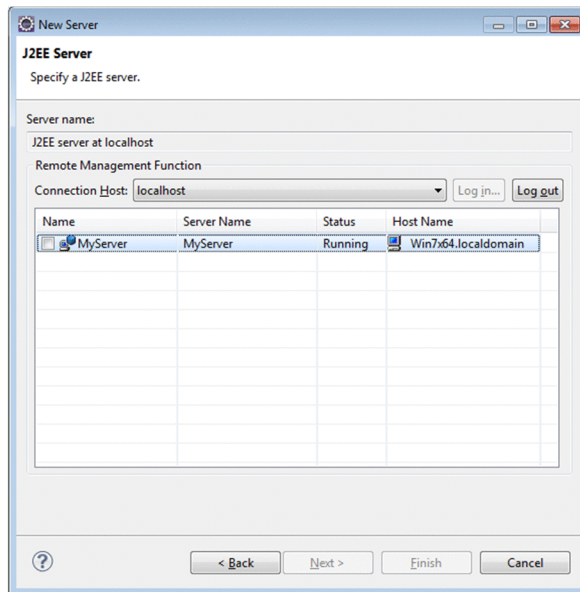
3. Specify the following settings:

3. Preparing the Environment for Sample Programs

Item name	Value to be specified
Select the server type	Under Cosminexus , select J2EE server .
Server name	You do not need to specify this item because the server name on the J2EE Server page is used.
Server runtime environment	Specify the server runtime Cosminexus J2EE .

4. Click the **Next** button.

The **J2EE Server** page appears.

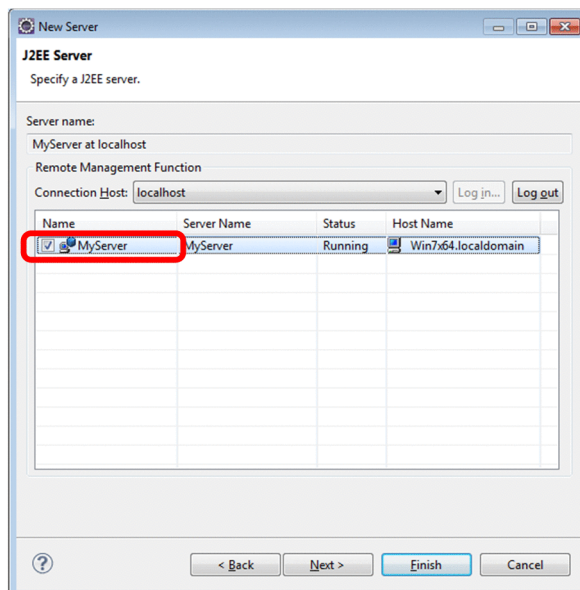


5. In the **Remote Management Function** area, from the **Connection Host** drop-down list, select the connection-destination host. If you are not logged in to the connection-destination host, click the **Log in** button.

Enter **admin** in **Administrator ID** and **Password**, and then click the **OK** button.

You are connected to Management Server Remote Management. The connection-destination J2EE server appears.

6. Select the check box of the J2EE server to be used (MyServer).



7. Click the **Finish** button.

The settings are saved.

3.5.5 Importing Eclipse projects

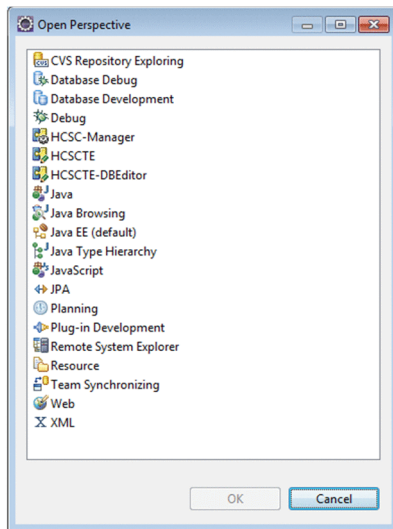
A separate Eclipse project is provided for each sample program. Import the Eclipse projects for the sample programs. This subsection describes the import procedure in the case where you will use the `HelloServiceAdapter` sample program.

Reference note

If the imported project includes an XML file for which no applicable XML schema is registered in Eclipse, a warning is generated. However, you can use the XML file without any problems.

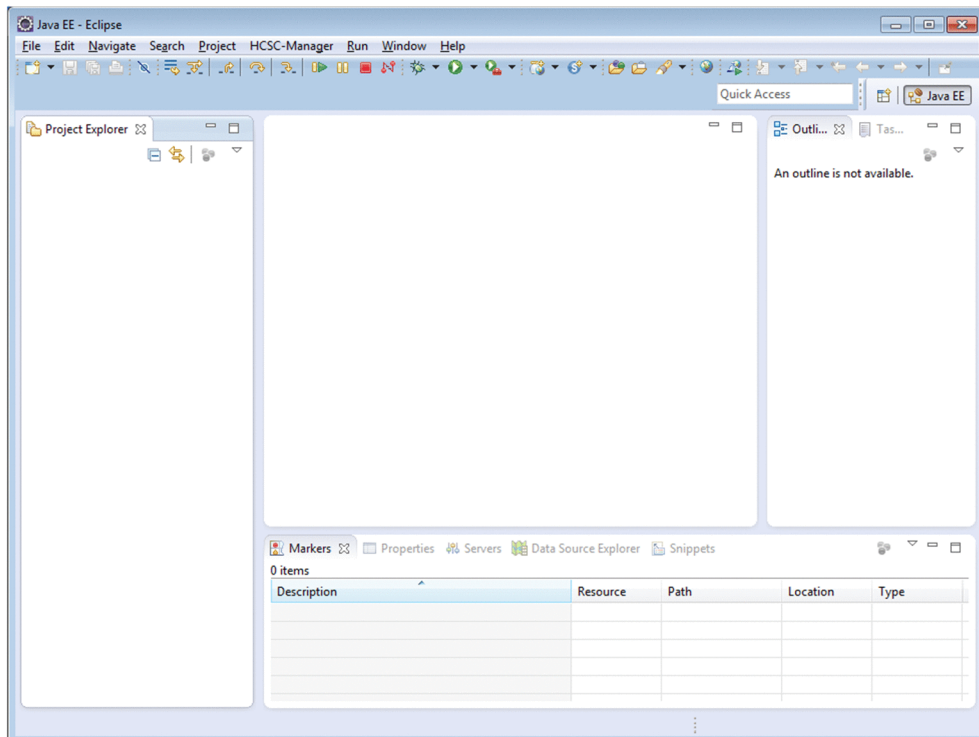
1. From the Eclipse menu, select **Window, Open Perspective, and then Other**.

The Open Perspective dialog box opens.



2. Select **Java EE (default)**, and then click the **OK** button.

The Java EE perspective opens.

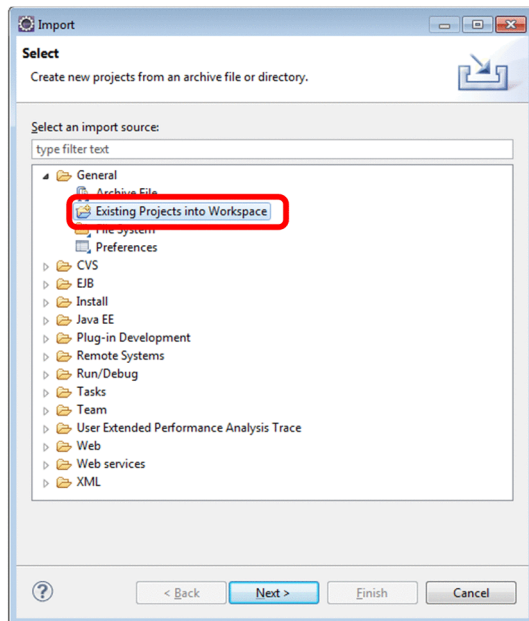


3. Preparing the Environment for Sample Programs

3. From the menu, select **File**, and then **Import**.

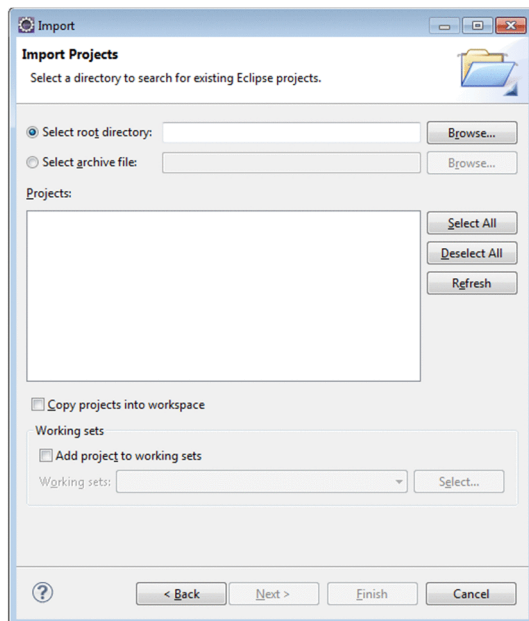
The **Select** page appears.

4. In the tree view, select **General**, and then **Existing Projects into Workspace**.



5. Click the **Next** button.

The **Import Projects** page appears.



6. Select the **Select root directory** radio button, and then click the **Browse** button.

The dialog box for selecting a directory opens.

7. Select the directory that contains the target sample program, and then click the **OK** button. In this example, the directory you select is as follows: *service-platform-installation-directory*\CSCTE\Samples\HelloServiceAdapter

The **Import Projects** page appears again. Make sure that the check boxes of all projects are selected.

The locations of other sample programs are as follows:

HelloBusinessProcess sample program:

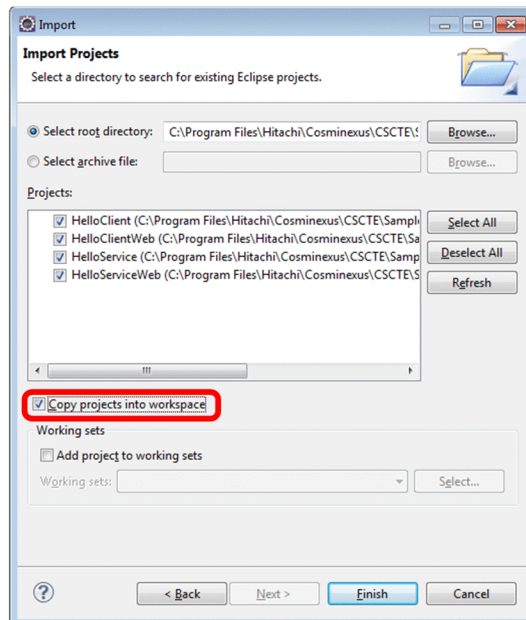
service-platform-installation-directory\CSCTE\Samples\HelloBusinessProcess

HelloProductArrangement sample program:

service-platform-installation-directory\CSCTE\Samples\ProductStock

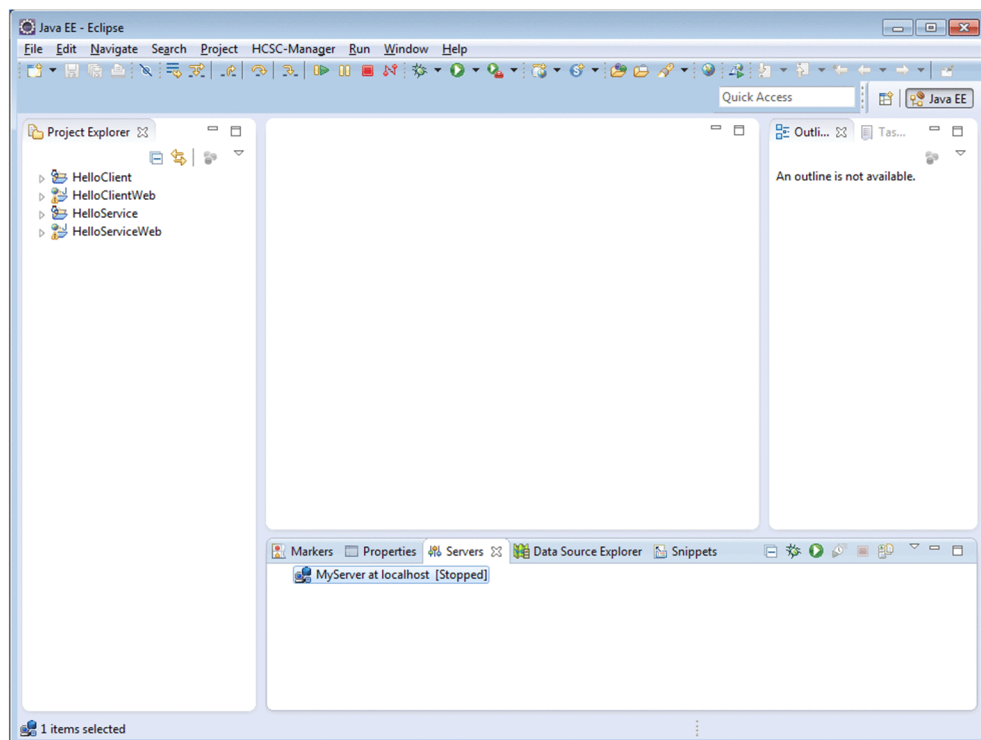
8. If the **Copy projects into workspace** check box is not selected, select it.

If you do not select this check box, the sample programs might be deleted. Make sure that this check box is selected.



9. Click the **Finish** button.

The selected projects are imported into the perspective.



3.5.6 Deploying the web project

The web project needs to be deployed for each sample program. This subsection describes the web project deployment procedure in the case where you will use the `HelloServiceAdapter` sample program.

1. From the menu, select **Window, Open Perspective, and then Other.**

The Open Perspective dialog box opens.

2. Select **Java EE (default)**, and then click the **OK** button.

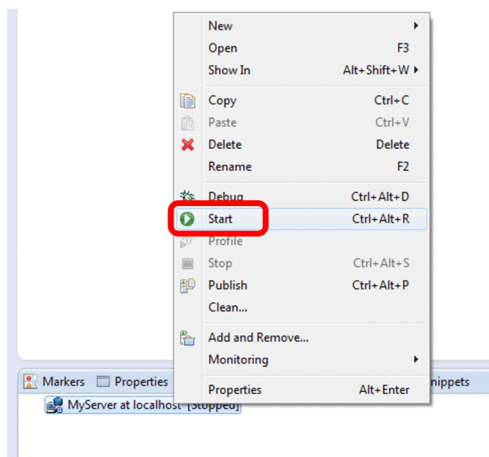
The Java EE perspective opens.

3. In the **Servers** view, right-click **MyServer at localhost**, and then select **Start**.

At this time, you might see an error dialog box that reports the server `MyServer` is externally running and asks you to stop the server. In this case, from the **Start** menu, select **Programs, Cosminexus[#]**, and then **Stop Test Server** to stop Performance Tracer, the J2EE server, and the HCSC server (including the standard reception and user-defined reception) in the test environment. Then, in the **Servers** view, right-click **MyServer at localhost**, and then select **Start** again.

#

This program folder name might have been changed. If it has been changed, select the changed program folder name.



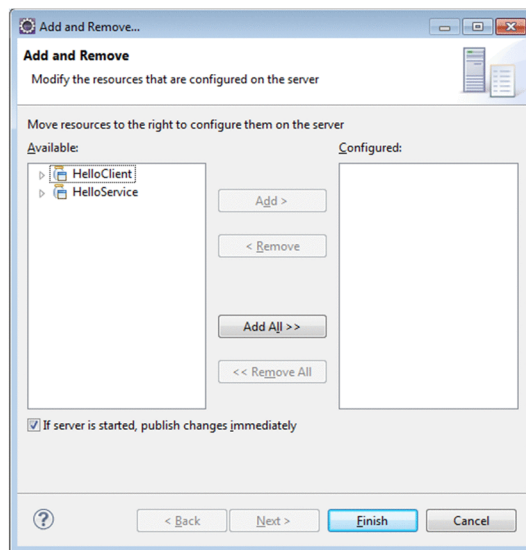
The login window appears.

4. Enter `admin` in both **Administrator ID** and **Password**, and then click the **OK** button.

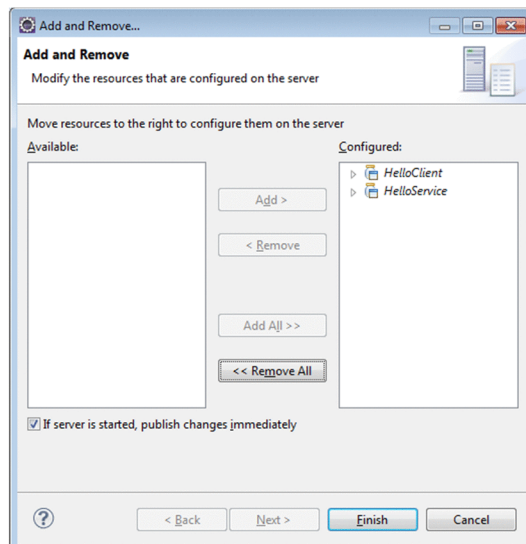
A message indicating that processing is in progress appears, and then the server starts.

5. In the **Servers** view, right-click **MyServer at localhost**, and then select **Add and Remove**.

The Add and Remove dialog box appears.



6. In the **Available** list box, select **HelloClient** and **HelloService**, and then click the **Add** button.

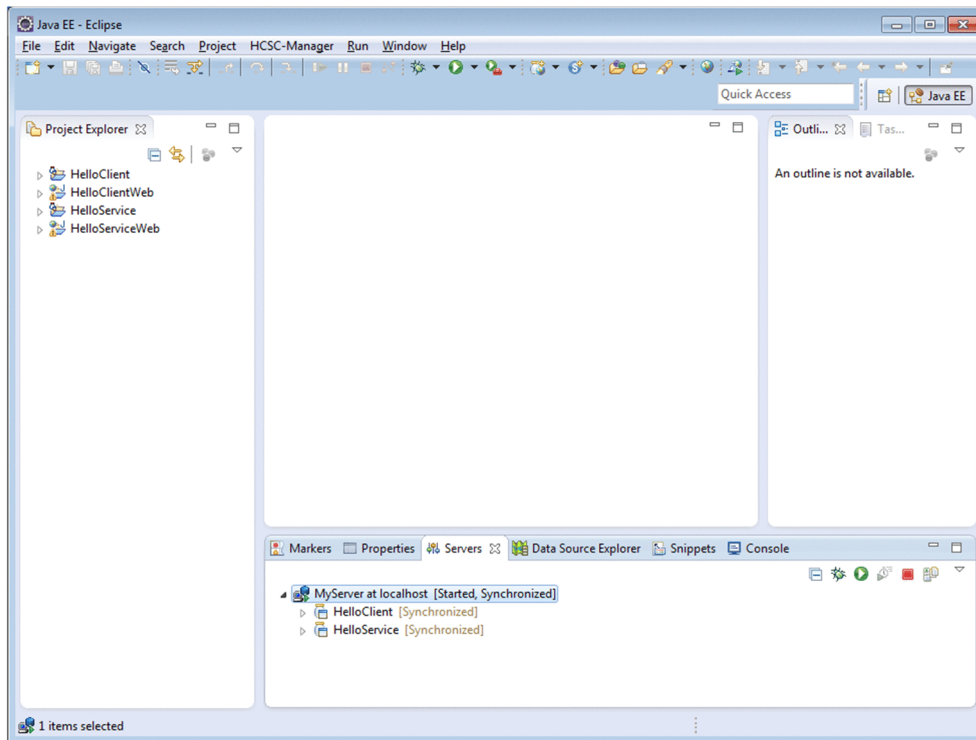


HelloClient and **HelloService** moves from the **Available** list box to the **Configured** list box.

7. Click the **Finish** button.

After processing terminates, the Java EE perspective appears again. Confirm that the `HelloClient` and `HelloService` projects are displayed under **MyServer at localhost** in the **Servers** view.

3. Preparing the Environment for Sample Programs



8. In the **Servers** view, right-click **MyServer at localhost**, and then select **Stop**.
9. From the **Start** menu, select **Programs, Cosminexus[#]**, and then **Start Test Server** to start Performance Tracer, the J2EE server, and the HCSC server (including standard reception and user-defined reception) in the test environment.

#

This program folder name might have been changed. If it has been changed, select the changed program folder name.

3.5.7 Deploying definitions to the HCSC server

Deploy definitions to the HCSC server for each sample program. The definitions to be deployed differ depending on the sample program. This subsection describes the deployment procedure in the case where you will use the `HelloServiceAdapter` sample program.

(1) Creating HCSCTE projects

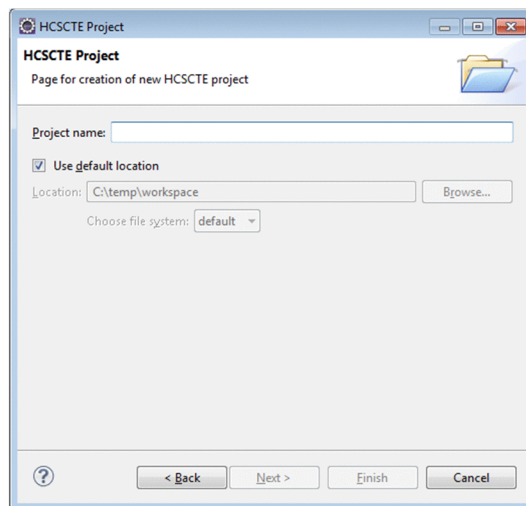
Before deploying definitions, you must create HCSCTE projects.

! Important note

An HCSCTE project is required for each program. When you develop multiple programs, you must use a separate workspace for the HCSCTE project of each program. If you create multiple HCSCTE projects in the same workspace, the programs will not operate correctly.

The following shows the procedure for creating an HCSCTE project:

1. From the menu, select **File, New**, and then **Project**.
The New Project dialog box appears.
2. Select **HCSCTE Project**, and then click the **Next** button.
The HCSCTE Project dialog box appears, displaying the page for creating a new HCSCTE project.



3. Specify the following items, and then click the **Next** button:

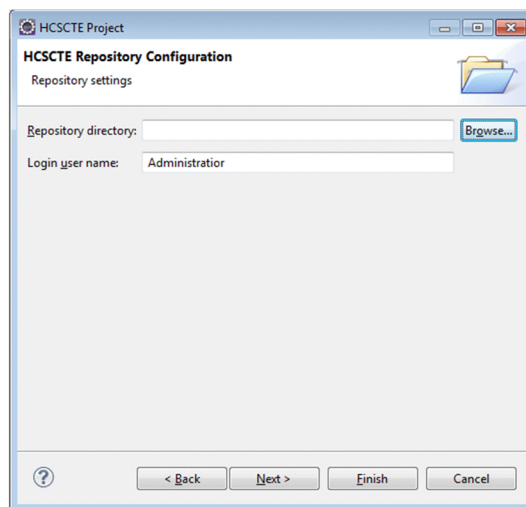
Project name

Specify any name. In this example, specify HCSCTE.

Use default location

Select the **Use default location** check box.

The HCSCTE Project dialog box appears, displaying the page for setting the HCSCTE repository.



4. Specify the following items, and then click the **Finish** button:

Repository directory

Specify the directory in which to store repository information. In this example, specify `C:\work\HelloServiceAdapter\repository`. Note the following points when you specify the repository directory:

- Do not specify the same path for the repository directory path and project path.
- Use an absolute path to specify the path.
- The specified absolute path is normalized, and then the length is verified with the normalized path.

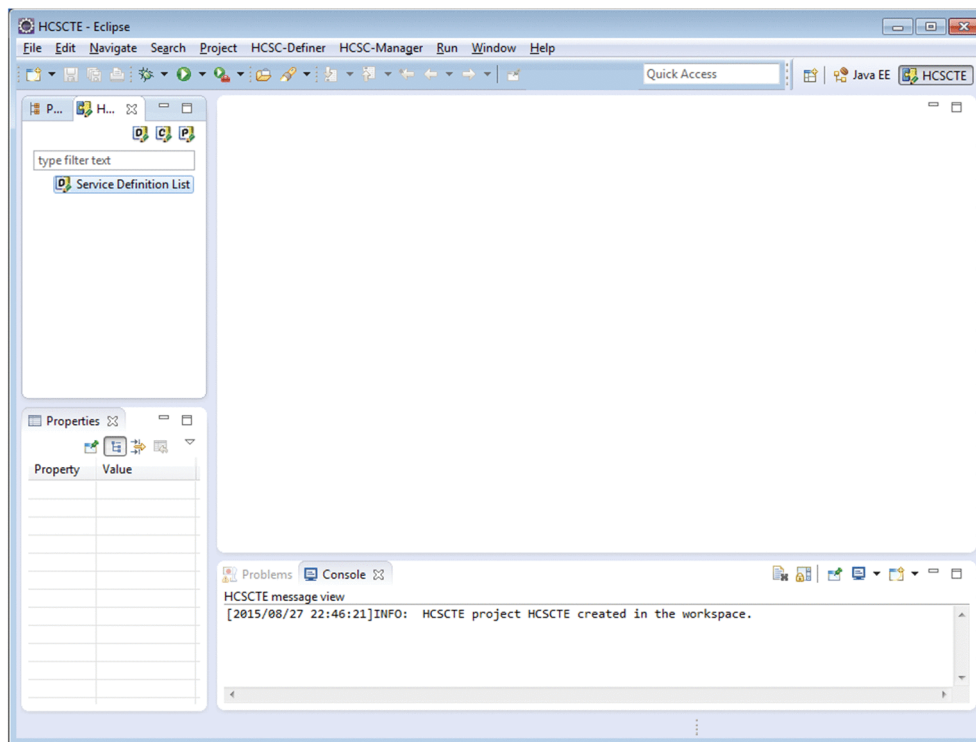
Login user name

Specify the user name that will be used for logging in to the repository. The user name can contain 1 to 16 alphanumeric characters.

If a dialog box asking you whether to open the associated perspective appears, click the **Yes** button.

An HCSCTE project is created, and the perspective for the project opens.

3. Preparing the Environment for Sample Programs

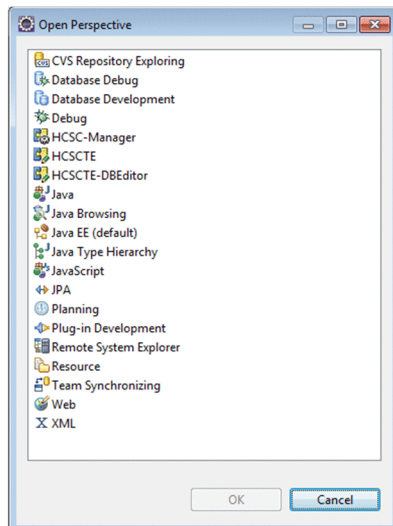


(2) Exporting the system configuration definition from the operation environment

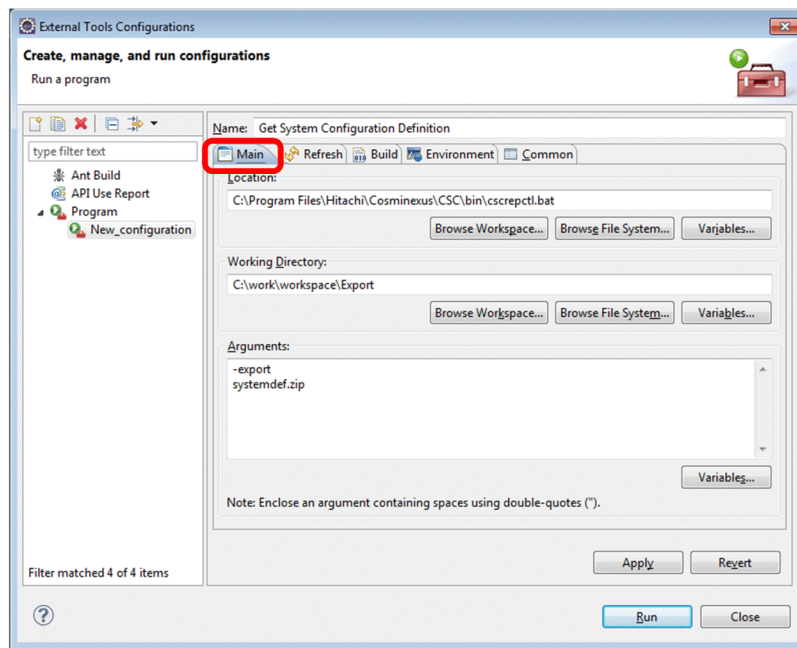
To import the system configuration definition from the execution environment to the development environment, export the repository information of the operation environment to a file compressed in ZIP format.

1. From the Eclipse menu, select **Window**, **Open Perspective**, and then **Other**.

The Open Perspective dialog box opens.



2. Select **HCSCTE**, and then click the **OK** button.
The HCSCTE perspective opens.
3. From the Eclipse menu, select **Run**, **External Tools**, and then **External Tools Configurations**.
The External Tools Configurations dialog box appears.
4. In the left pane, right-click **Program**, and then select **New**.
The **Create, manage, and run configurations** page appears.



Select the **Main** tab, and then enter the following information:

Item name	Value to be set
Name	Specify any name. In this example, enter <code>Get System Configuration Definition</code> .
Location	Enter the following file: <code>\${env_var: COSMINEXUS_HOME}\csc\bin\cscreptcl.bat</code> Alternatively, click the Browse File System button, and select the following file: <code>service-platform-installation-directory\csc\bin\cscreptcl.bat</code>
Working Directory	Specify a directory to which the file specified with the argument entered next will be output.
Arguments	<code>-export</code> <code>any-output-file-name.zip</code> In this example, enter <code>systemdef.zip</code> .

5. Click the **Run** button.

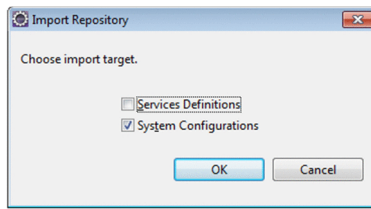
The command is registered and executed. As a result, the repository information of the operation environment is exported as a ZIP file to the specified output-destination directory.

(3) Importing the system configuration definition into the development environment

From the export file that contains the operation environment's repository information, import only the system configuration definition to the development environment.

- From the Eclipse menu, select **HCSC-Definer**, **Repository management**, and then **Import repository**.
The dialog box confirming that the repository will be overwritten appears.
- Click the **Yes** button.
The Import Repository dialog box for selecting the ZIP file that contains repository information appears.
Specify the ZIP file named `systemdef.zip` that was exported from the operation environment.
- Click the **Open** button.
The Import Repository dialog box for selecting the definition information to be imported appears.

3. Preparing the Environment for Sample Programs



4. Select the **System Configurations** check box, and then click the **OK** button.
When the system configuration definition of the execution environment is imported, a dialog box reporting that processing was normally completed appears.
5. Click the **OK** button.

(4) Importing a sample program

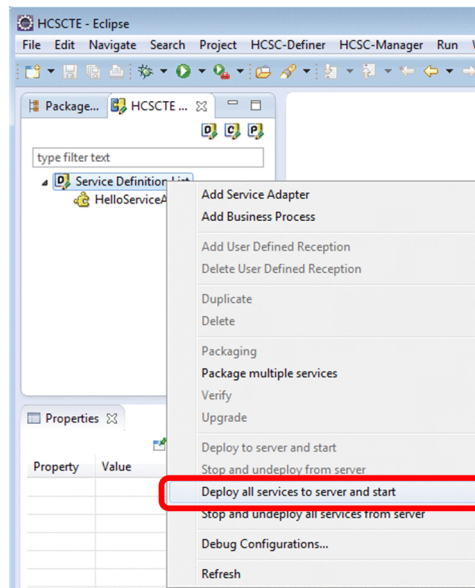
Import the repository information for the sample program.

1. Select **HCSC-Definer**, **Repository management**, and then **Import repository**.
The dialog box confirming that the repository will be overwritten appears.
2. Click the **Yes** button.
The Import Repository dialog box for selecting the ZIP file that contains repository information appears. Select the following ZIP file:
`service-platform-installation-directory\CSCTE\Samples\HelloServiceAdapter\Repository\HelloServiceAdapter.zip`
The repository information of other sample programs is stored in the following files:
HelloBusinessProcess sample program:
`service-platform-installation-directory\CSCTE\Samples\HelloBusinessProcess\Repository\HelloBusinessProcess.zip`
HelloProductArrangement sample program:
`service-platform-installation-directory\CSCTE\Samples\ProductStock\Repository\ProductStock.zip`
3. Click the **Open** button.
The Import Repository dialog box for selecting the definition information to be imported appears. Select only the **Services Definitions** check box, and then click the **OK** button.

(5) Deploying HCSC components

Deploy and start HCSC components.

1. In the tree view, right-click **Service Definition List**, and then select **Deploy all services to server and start**.



If you are not logged in, the account authentication window appears. Perform step 2.

2. Enter **admin** in both **User ID** and **Password**, and then click the **OK** button.

A message indicating that processing is in progress appears, and then a message reporting the results appears.

3. Click the **OK** button.

You have now completed deployment.

(6) Confirming HelloServiceAdapter and starting the standard reception

Confirm that HelloServiceAdapter is running, and then start the standard reception.

1. From the menu, select **Window, Show View**, and then **Other**.

The Show View dialog box appears.

2. In the Show View dialog box, under **HCSC-Manager**, select **HCSC-Manager View**, and then click the **OK** button.

The **HCSC-Manager** view appears.

3. In the **HCSC-Manager** view, right-click **HCSC-Manager(Login)**, and then select **Login**.

The login window appears.

4. Enter **admin** in both **Administrator ID** and **Password**, and then click the **OK** button.

Login to HCSC - Manager finishes.

5. Confirm that HelloServiceAdapter is running.

In the **HCSC-Manager** view, expand **HCSC-Manager(Login)**, **HCSC-Domain**, **MyUnit(LBCluster)**, and then **MyCSC[localhost:28099]**. Then, double-click **HelloServiceAdapter** to display the **Information** page. Click the **Operations** tab to open the **Operations** page. Confirm that **Status** is **active**.

For the other sample programs, confirm that the following services are running:

HelloBusinessProcess sample program:

Confirm that HelloServiceAdapter and HelloBusinessProcess are running.

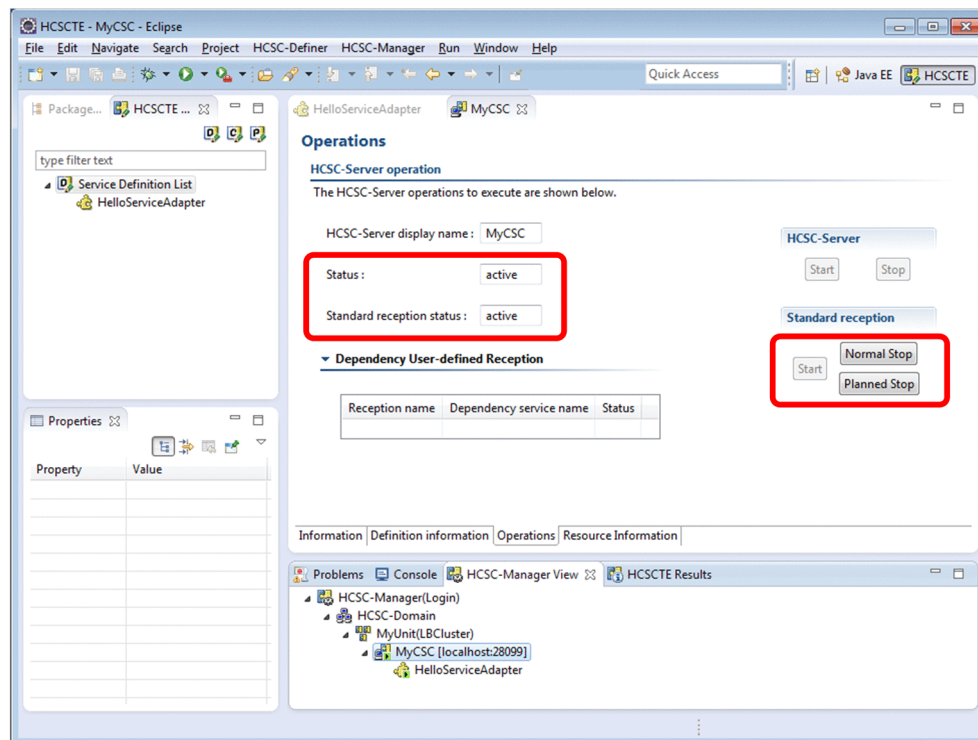
HelloProductArrangement sample program:

Confirm that ProductStock, DeliveryReceipt, and InventoryManagement are running.

6. Confirm that the standard reception has started.

In the **HCSC-Manager** view, expand **HCSC-Manager(Login)**, **HCSC-Domain**, and then **MyUnit(LBCluster)**. Then, double-click **MyCSC[localhost:28099]** to display the **Operations** page for **MyCSC**. Click the **Start** button to change the status of **Standard reception** to **active**. When the status of **Standard reception** becomes **active**, the **Start** button is disabled.

3. Preparing the Environment for Sample Programs



7. In the **HCSC-Manager** view, right-click **HCSC-Manager(Login)**, and then select **Logout**.

A message confirming that you will log out from HCSC-Manager appears.

8. Click the **OK** button.

Logout from HCSC - Manager finishes, and HCSC-Manager (Logout) is displayed.

You are now ready to run sample programs.

4

Executing Sample Programs

This chapter describes how to execute sample programs.

4.1 Executing sample programs

After preparing the environment for sample programs, execute sample programs.

Important note

For sample programs, separate projects are provided for each type of processing. Therefore, to execute a sample program after executing another program, use the following procedure:

1. Delete the projects for the previously executed sample program.
For details about how to delete projects, see *6.1 Deleting projects*.
2. Import Eclipse projects for the sample program to be executed next.
For details about how to import Eclipse projects, see *3.5.5 Importing Eclipse projects*.
3. Deploy web projects for the sample program to be executed next.
For details about how to deploy web projects, see *3.5.6 Deploying the web project*.
4. Deploy definitions of the sample program to be executed next to the HCSC server.
For details about how to deploy definitions to the HCSC server, see the procedure in *3.5.7(4) Importing a sample program* and later.
5. Execute a sample program.

See one of the following sections based on the sample program you want to execute:

To execute the `HelloServiceAdapter` sample program:

See *4.2 Operation when business processes are not applied*.

To execute the `HelloBusinessProcess` sample program:

See *4.3 Operation when a business process is applied*.

To execute the `HelloProductArrangement` sample program:

See *4.4 Operation when processes of multiple services are integrated*.

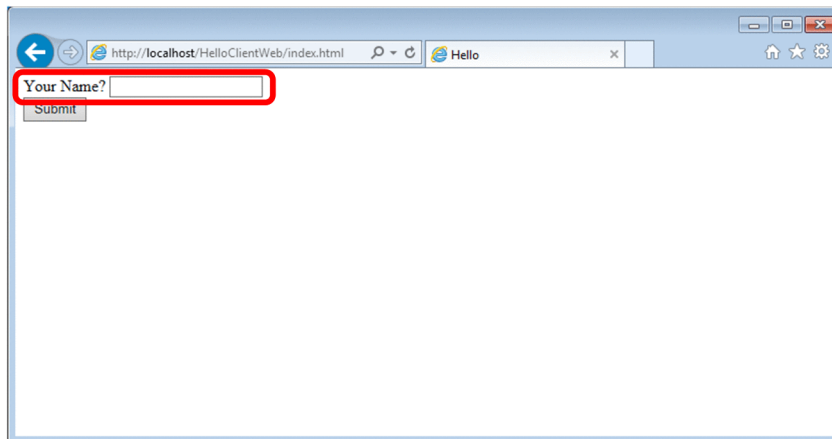
Important note

To execute a sample program, the test environment must be started. For details about how to start the test environment, see *3.5.2 Starting the test environment*. If you do not execute a sample program, stop the test environment. For details about how to stop the test environment, see *6.2 Stopping the test environment*.

4.2 Operation when business processes are not applied

Execute the HelloServiceAdapter sample program by performing the following procedure.

1. Enter the following URL into the browser:
<http://localhost/HelloClientWeb/index.html>
 The HelloServiceAdapter sample program is started.
2. Enter a character string for **Your Name?**



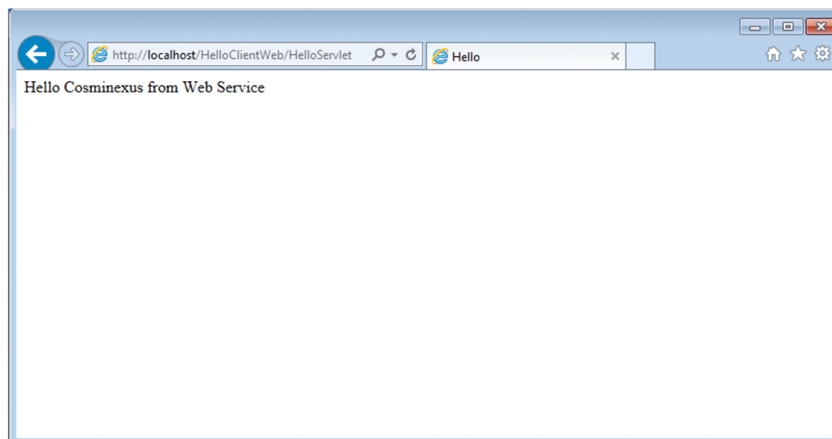
3. Click the **Submit** button.
 The following is displayed:

Normal response

The following is displayed on the screen:

Hello **entered-character-string** from Web Service

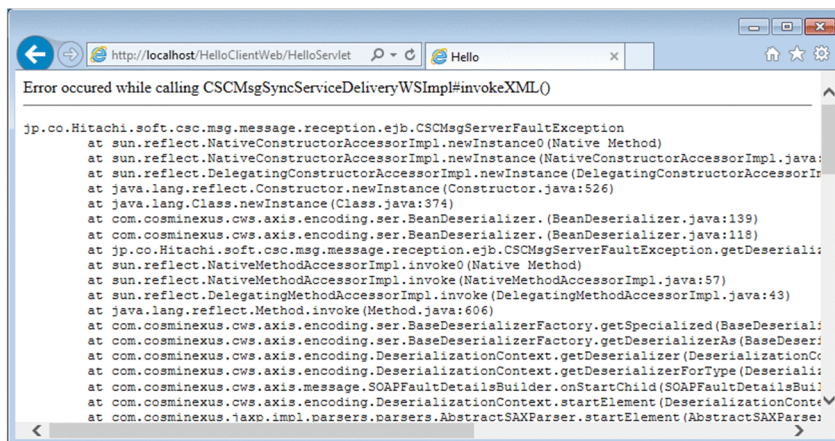
: Space



If an error occurred

Details about the error are displayed.

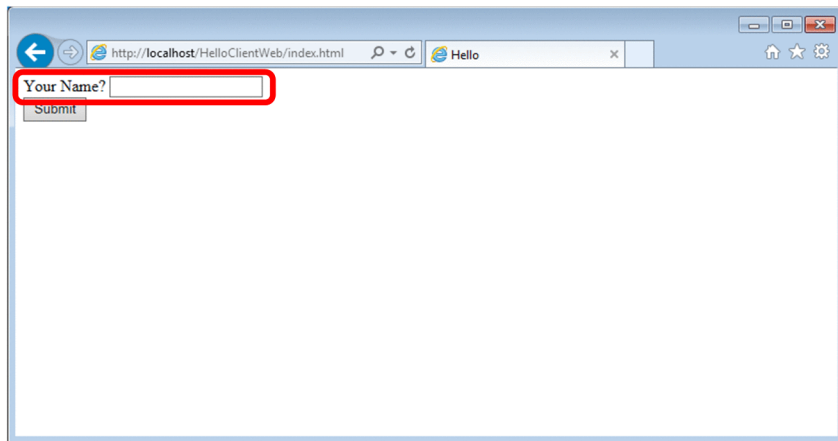
4. Executing Sample Programs



4.3 Operation when a business process is applied

Execute the `HelloBusinessProcess` sample program by performing the following procedure.

1. Enter the following URL into the browser:
`http://localhost/HelloClientWeb/index.html`
 The `HelloBusinessProcess` sample program is started.
2. Enter a character string for **Your Name?**.

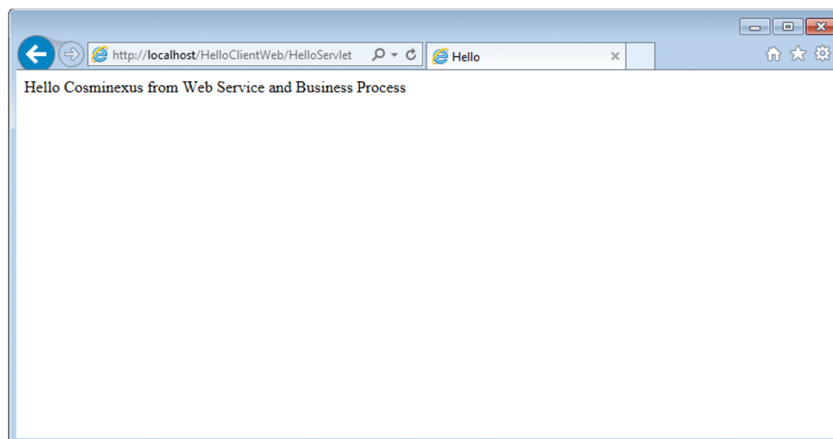


3. Click the **Submit** button.
 The following is displayed:

Normal response

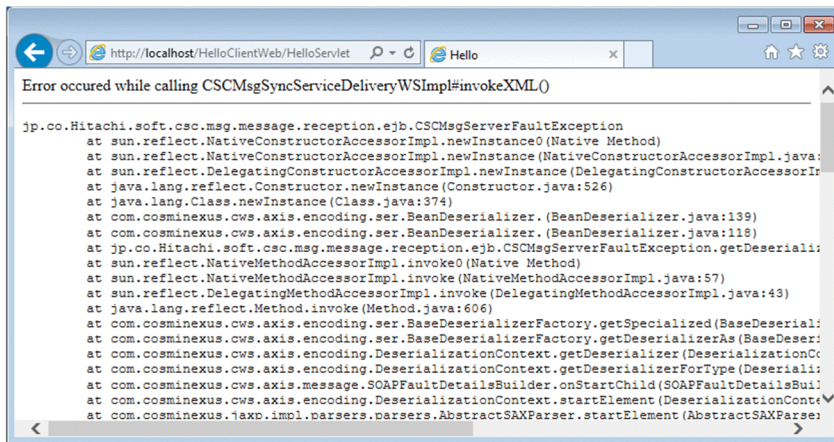
The following is displayed on the screen:

Hello entered-character-string from Web Service and Business Process
 : Space



If an error occurred
 Details about the error are displayed.

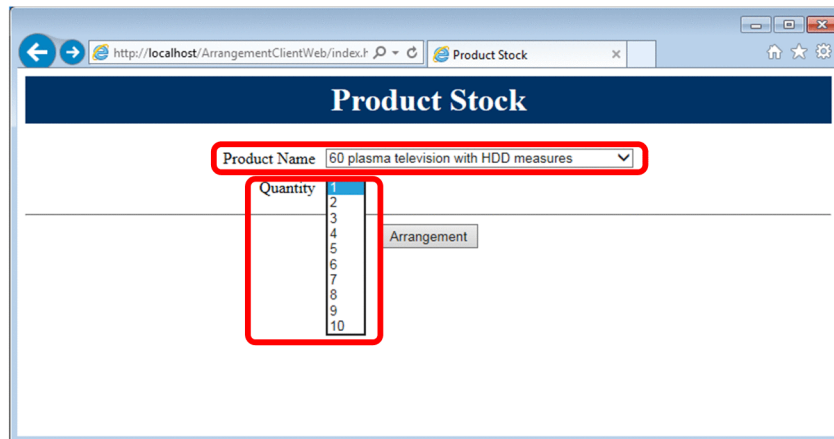
4. Executing Sample Programs



4.4 Operation when processes of multiple services are integrated

Execute the product arrangement sample program by performing the following procedure.

1. Enter the following URL into the browser:
`http://localhost/ArrangementClientWeb/index.html`
 The HelloProductArrangement sample program is started.
2. Select the product name and quantity.

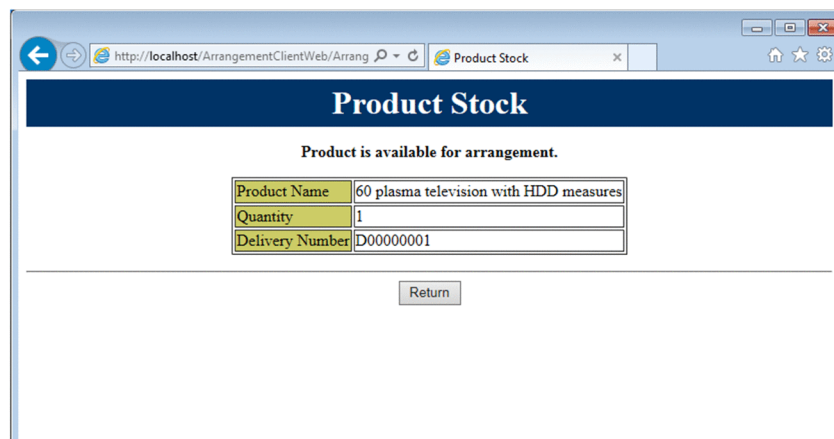


3. Click the **Arrangement** button.

The following is displayed:

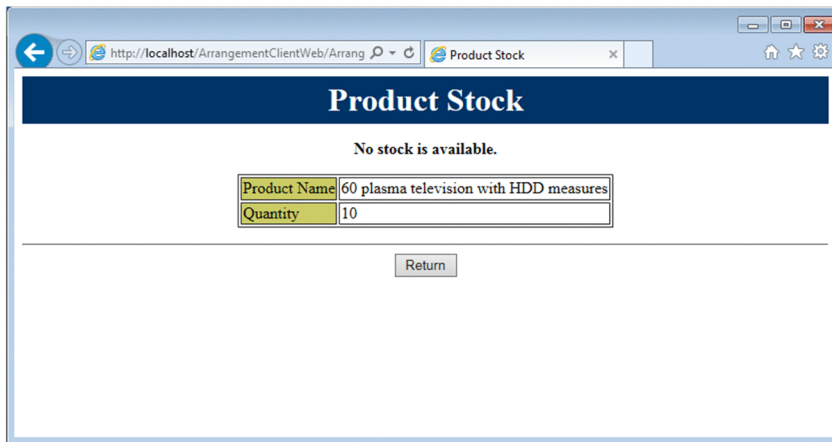
Normal response (arrangement completed)

A message indicating that arrangement is completed appears. The product name, quantity, and delivery number are also displayed.



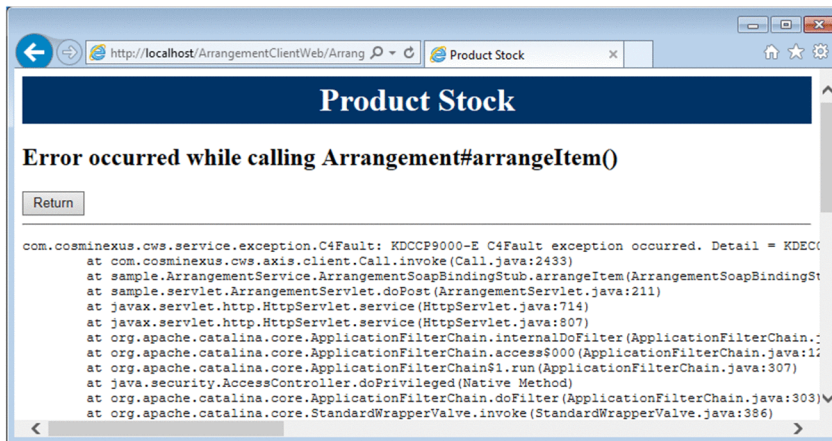
Normal response (out of stock)

A message indicating that there is no stock appears. The product name and quantity are also displayed.



If an error occurred

Details about the error are displayed.



How to reset the quantity of product stock to the initial value

For the HelloProductArrangement sample program, the total quantity of stock for each product is 10. If arrangements are completed, the quantity is reduced for the number of products arranged. If there is no stock, restart the server or redeploy the J2EE project. The quantity of stock returns to 10. The following describes how to restart the server and how to redeploy the J2EE project.

How to restart the server

1. From the Eclipse menu, select **Window, Open Perspective, and then Other**.
The Open Perspective dialog box appears.
2. Select **Java EE (default)**, and then click **OK**.
The **Java EE** perspective appears.
3. In the **Servers** view, right-click **MyServer at localhost**, and then select **Restart**.

How to redeploy the J2EE project

1. From the Eclipse menu, select **Window, Open Perspective, and then Other**.
The Open Perspective dialog box appears.
2. Select **Java EE (default)**, and then click **OK**.
The Java EE perspective appears.
3. In the **Servers** view, right-click **MyServer at localhost**, and then select **Add and Remove**.
The Add and Remove dialog box appears.
Undeploy the J2EE project, and then deploy it again. For details about undeployment, see 6.1.1(1) *Undeploying web projects*. For details about deployment, see 3.5.6 *Deploying the web project*.

5

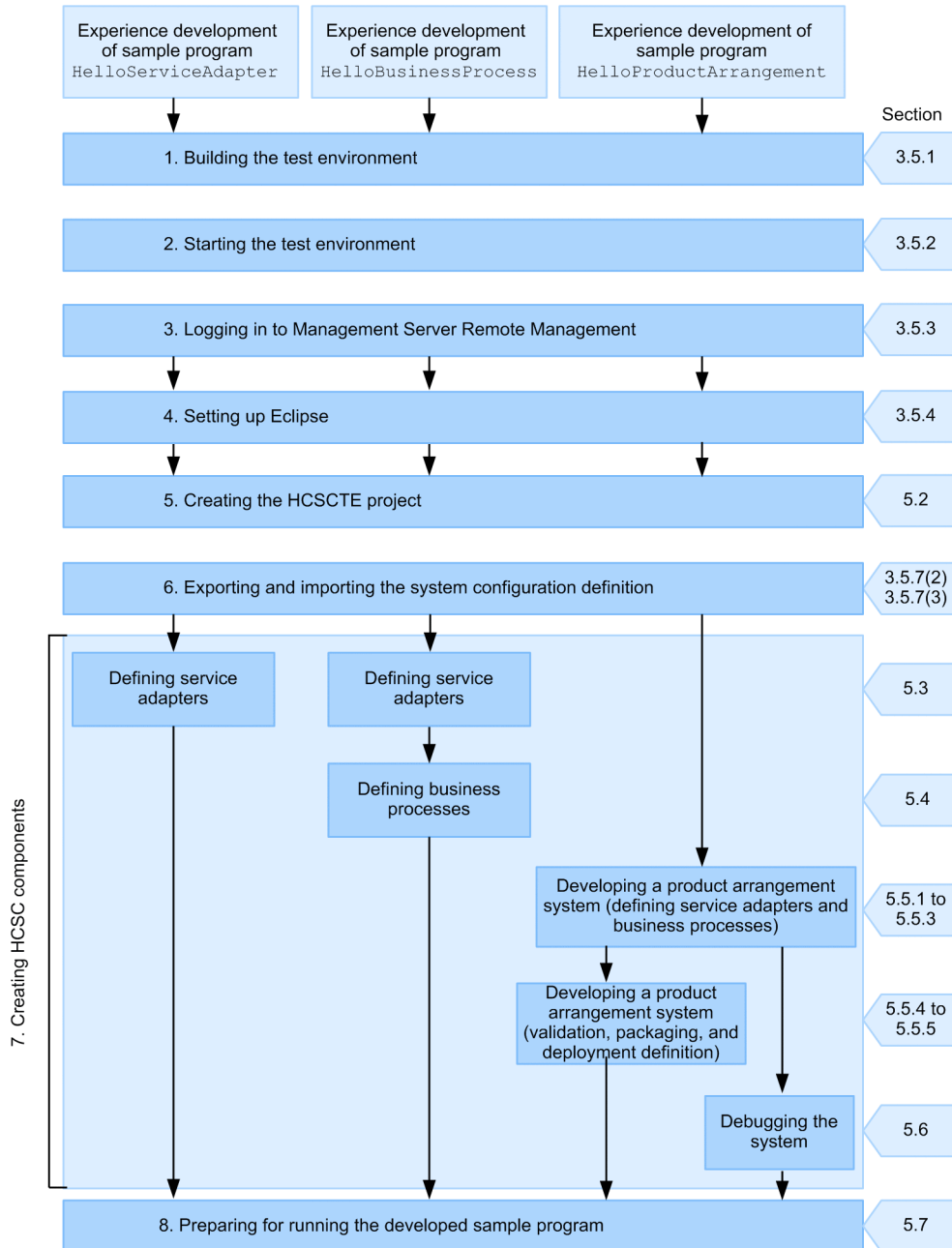
Experiencing the Development of Sample Programs

This chapter describes how to define service adapters and business processes by using provided sample programs.

5.1 Procedure for developing sample programs

The following figure shows the procedure for developing sample programs.

Figure 5–1: Procedure for developing sample programs



1. Building the test environment

In this section, use HCSC Easy Setup to build the test environment. If the test environment has already been built, undo setup from the HCSC Easy Setup window, and then rebuild the test environment. For details, see 3.5.1 *Building the test environment*.

2. Starting the test environment

In this section, start the test environment that you built. For details, see 3.5.2 *Starting the test environment*.

3. Logging in to Management Server Remote Management

In this section, log in from Eclipse to Management Server Remote Management. For details, see *3.5.3 Logging in to Management Server Remote Management*.

4. Setting up Eclipse

In this section, set up Eclipse for each sample program. For details, see *3.5.4 Setting up Eclipse*.

5. Creating the HCSCTE project

In this section, create a project and set properties before defining service adapters and business processes. For details, see *5.2 Creating the HCSCTE project*.

6. Exporting and importing the system configuration definition

In this section, export the system configuration definition of the execution environment, and then import the system configuration definition into the development environment. For details, see *3.5.7(2) Exporting the system configuration definition from the operation environment* and *3.5.7(3) Importing the system configuration definition into the development environment*.

7. Creating HCSC components

In this section, create HCSC components, such as a service adapter that calls a service component and a business process that calls multiple service components. In addition, debug the created HCSC components in the test environment. For details, see the following sections:

- *5.3 Defining service adapters*
- *5.4 Defining business processes*
- *5.5 Developing the product arrangement system*
- *5.6 Debugging the product arrangement system*

8. Preparing for running the developed sample program

In this section, validate operation of the developed sample program by using the sample service requester and service component provided by Service Architect. For details, see *5.7 Preparing for running the developed sample program*.

The following sections describe the above stages.

This chapter also describes how to develop a sample program that defines data transformation using a Java program. For details, see *5.8 Defining data transformation by using a Java program*.

5.2 Creating the HCSCTE project

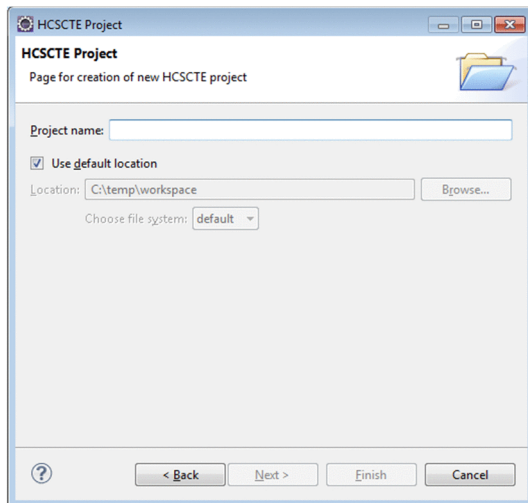
In this section, create an HCSC project before defining service adapters and business processes.

! Important note

An HCSCTE project is required for each program. When you develop multiple programs, you must use a separate workspace for the HCSCTE project of each program. If you create multiple HCSCTE projects in the same workspace, the programs will not operate correctly.

The following is the procedure for creating an HCSCTE project.

1. Start Eclipse.
2. From the menu, select **File**, **New**, and then **Project**.
The New Project dialog box appears.
3. Select **HCSCTE Project**, and then click the **Next** button.
The HCSCTE Project dialog box appears, and then the page for creating a new HCSCTE project appears.



4. Specify the following items, and then click the **Next** button.

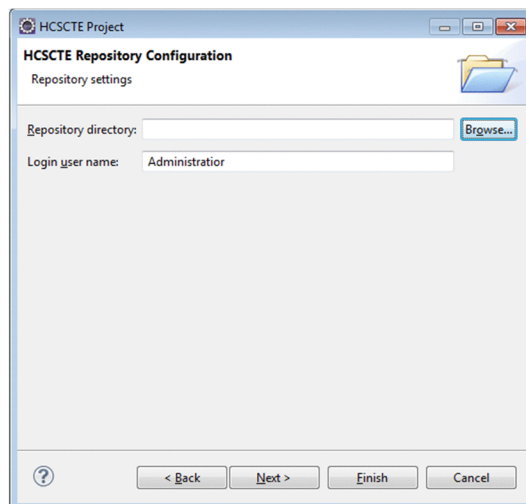
Project name

Specify any name. In this example, specify HCSCTE.

Use default location

Select the **Use default location** check box.

The HCSCTE Project dialog box appears, and then the page for setting the HCSCTE repository appears.



5. Specify the following items, and then click the **Finish** button.

Repository directory

Specify the directory in which to store repository information. Note the following points when you specify the repository directory:

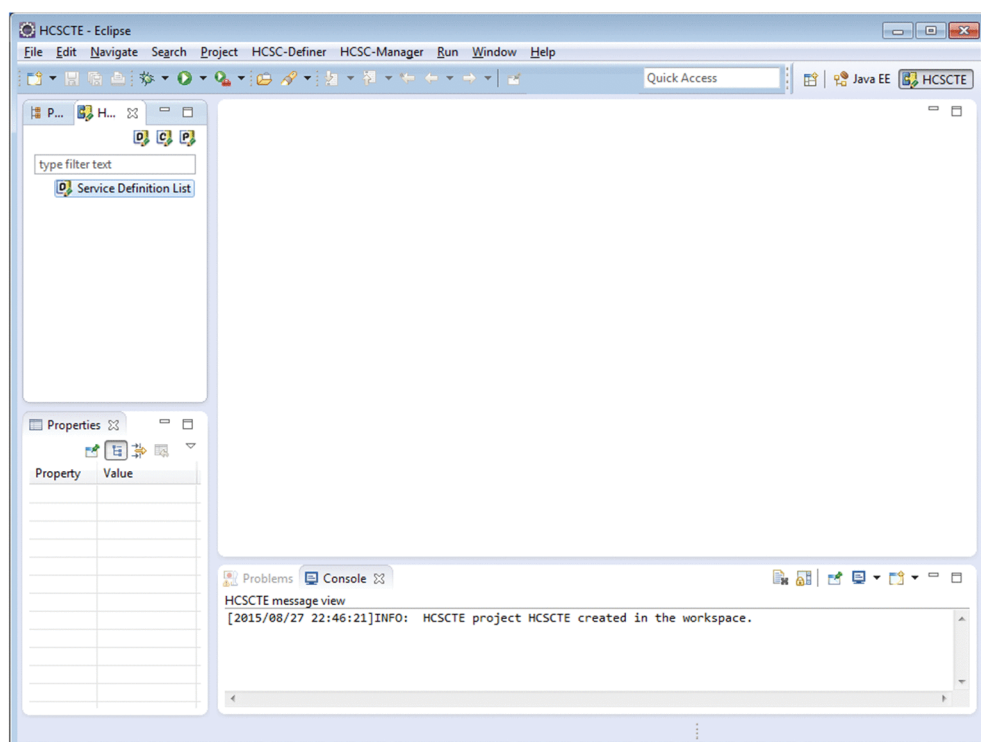
- Do not specify the same path for the repository directory path and project path.
- Use an absolute path to specify the path.
- The specified absolute path is normalized, and then the length is verified with the normalized path.

Login user name

Specify the user name that will be used for logging in to the repository. The user name can consist of 1 to 16 alphanumeric characters.

If a dialog box asking you whether to switch the perspective appears, click the **Yes** button.

An HCSCTE project is created, and then the perspective for the project opens.



After you have completed the above procedure, export and import the system configuration definition. For details about the procedure, see 3.5.7(2) *Exporting the system configuration definition from the operation environment* and 3.5.7(3) *Importing the system configuration definition into the development environment*.

5.3 Defining service adapters

In this section, define a service adapter by using the `HelloServiceAdapter` sample program, which calls a service adapter from the service requester.

You need to define a service adapter according to the service component to be called. The following table shows the values that you need to set when using the `HelloServiceAdapter` sample program to define a service adapter.

Table 5–1: Values that need to be set when using the `HelloServiceAdapter` sample program to define a service adapter

Item	Value to be set	Description
Service component type	<code>Web Service</code>	Specify the type of service adapter that is set according to the type of service component to be called. Because the service component type of this sample program is a Web Service, specify <code>Web Service</code> .
Service name	<code>HelloServiceAdapter</code>	Specify the name of the service adapter to be defined.
WSDL file	<code>HelloService.wsdl</code>	Specify the WSDL file to be used. For the Hello service adapter, use <code>HelloService.wsdl</code> (WSDL for the Hello service).
Port specification	<code>Hello</code>	Specify the port of the Hello service adapter.
Service ID	<code>HelAdp</code>	Specify the ID of the Hello service adapter.
Client definition file	<code>c4webcl.properties</code>	Specify the client definition file. The client definition file controls the client-side behavior. The name of this file is fixed to <code>c4webcl.properties</code> . The client definition file of this sample program contains the following entry: <code>c4web.logger.log_file_prefix=HelloService</code> This entry sets <code>HelloService</code> as the prefix of the trace file and application log.

Note:

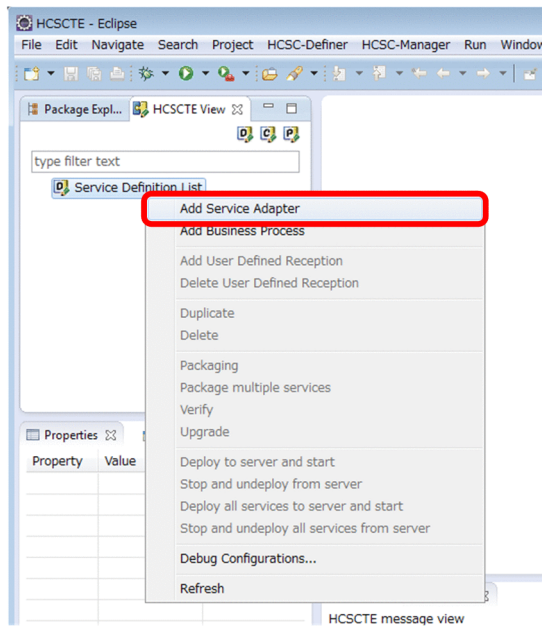
For details about the location of each file, see *A.1 Configuration of the HelloServiceAdapter sample program*.

5.3.1 Creating a service adapter

The following is the procedure for creating the Hello service adapter.

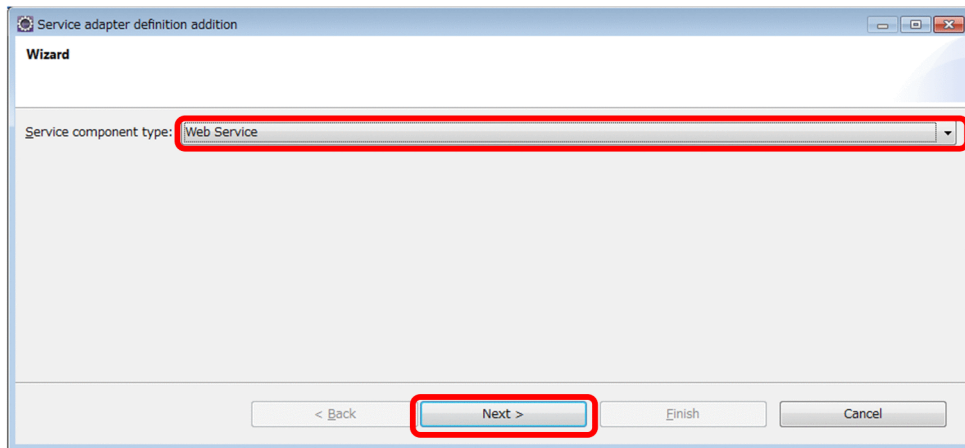
1. Start Eclipse.
2. In the tree view, select and right-click **Service Definition List**, and then select **Add Service Adapter**.

5. Experiencing the Development of Sample Programs



The dialog box for setting the type of service to be used from the service adapter that you are creating appears.

3. Select **Web Service** as the type of service component, and then click the **Next** button.



The dialog box for entering the information that is necessary for adding a SOAP adapter appears.

4. Enter HelloServiceAdapter as the service name.
5. Click the ... button.

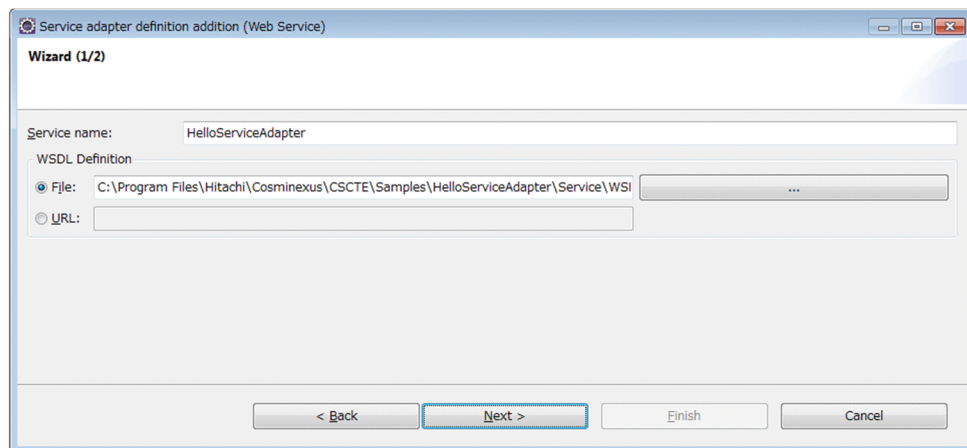
The **Open** dialog box appears.

6. Select HelloService.wsdl, and then click the **Open** button.

HelloService.wsdl is located in the following directory:

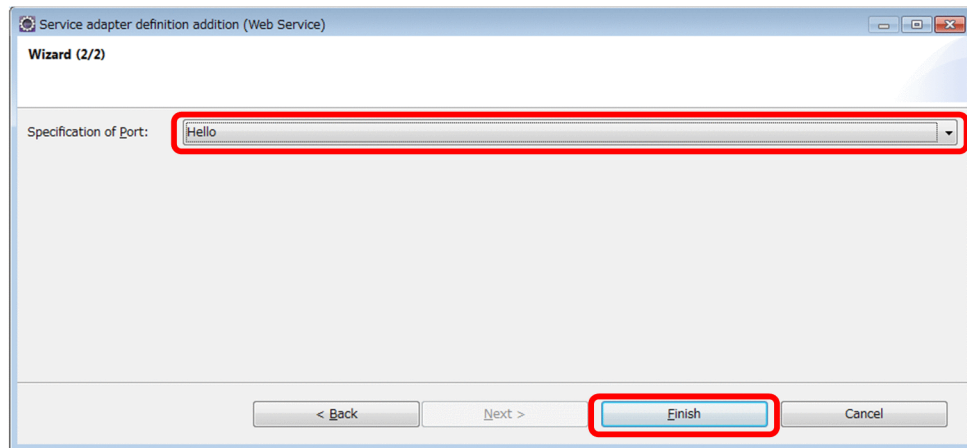
service-platform-installation-directory\CSCTE\Samples\HelloServiceAdapter\Service\WSDL

7. Click the **Next** button.



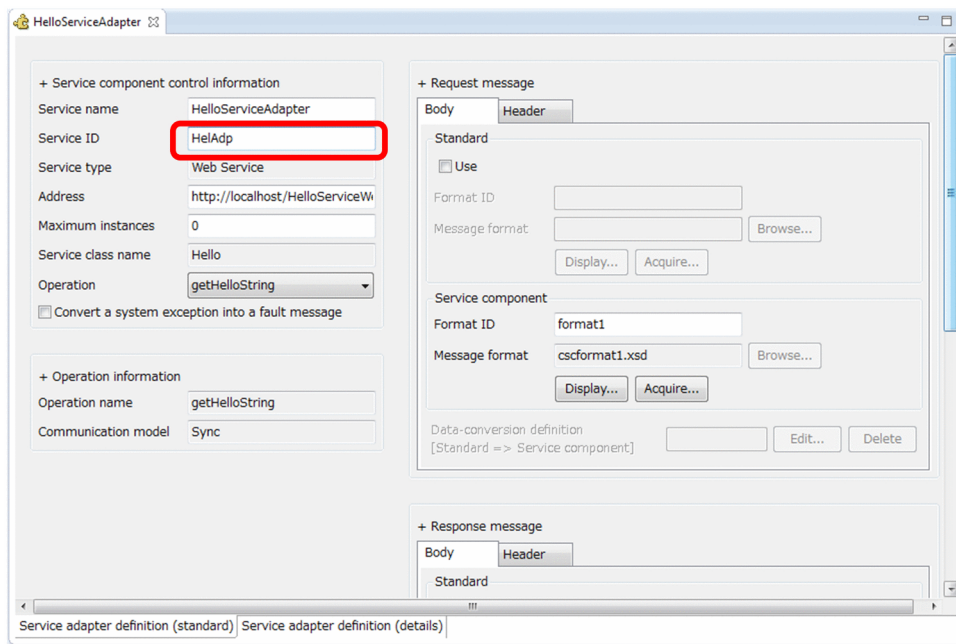
The dialog box for specifying the port appears.

8. Select **Hello** from the **Specification of Port** drop-down list, and then click the **Finish** button.



The service adapter for calling the Hello service (HelloServiceAdapter) is created, and then the service adapter definition (standard) window appears.

9. Change the service ID to HelAdp.

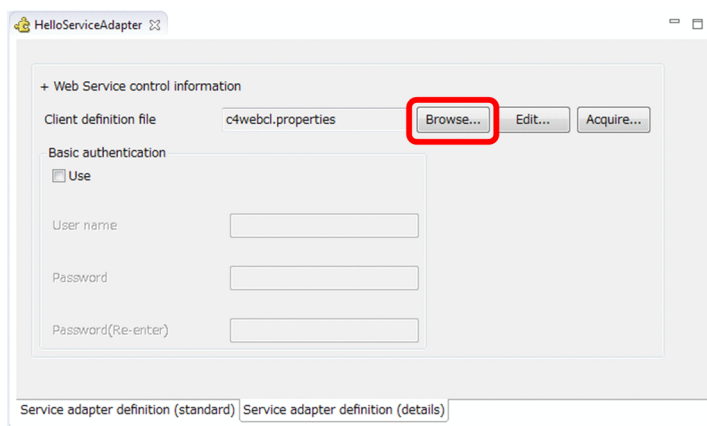


In the service adapter definition (standard) window, the information that has been read from the specified WSDL, such as the service type and access-target address, is displayed. In addition, the message format that has been automatically generated from WSDL is displayed (in **Message format**) in the **Request message** and **Response message** areas. The contents of the message format can be checked by clicking the **Display** button.

10. Click the **Service adapter definition (details)** tab at the bottom of the window.

The service adapter definition (details) window appears. In the service adapter definition (details) window, specify the client definition file.

11. In the service adapter definition (details) window, click the **Browse** button.



12. Specify `c4webcl.properties` as the client definition file.

For the HelloServiceAdapter sample program, the location of the above file is as follows:

`service-platform-installation-directory\CSCTE\Samples\HelloServiceAdapter\Service\HelloService\c4webcl.properties`

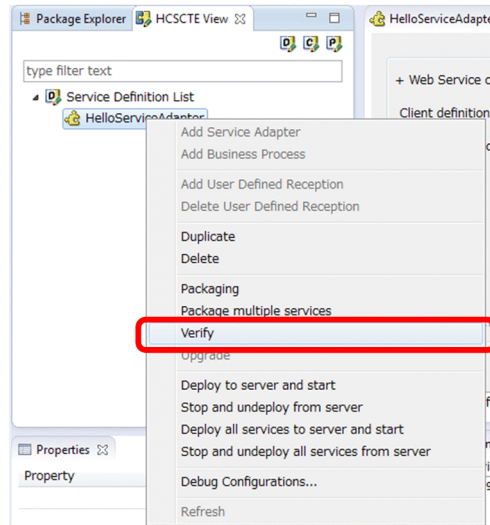
13. From the menu, select **File**, and then **Save**.

The service adapter definition is now completed.

5.3.2 Validating and packaging a service adapter

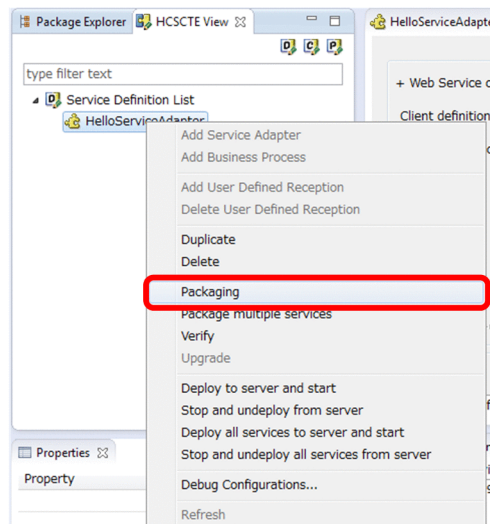
When you have created the Hello service adapter, validate that it is defined correctly, and then package it. The following is the procedure for validating and packaging a service adapter.

1. In the tree view, select and right-click **HelloServiceAdapter**, and then select **Verify**.



The validation results are displayed in the Console view. If an error occurs, correct it according to the message.

2. In the tree view, select and right-click **HelloServiceAdapter**, and then select **Packaging**.



Packaging starts. When processing finishes, a message reporting the processing results appears.

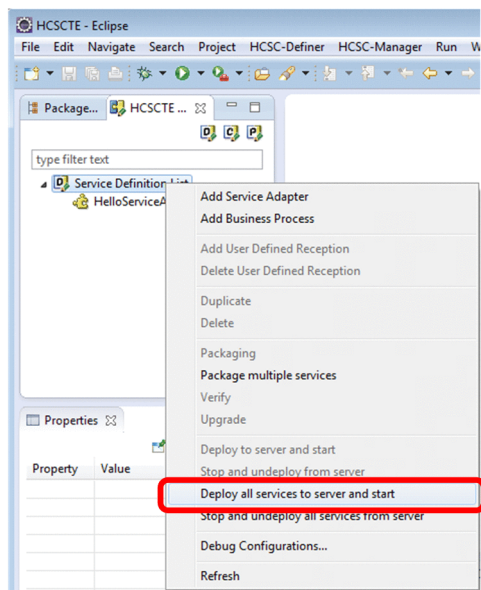
3. Perform either of the following operations:
 - If the packaging is successful, click the **OK** button.
 - If the packaging fails, take action according to the message, and then re-execute packaging.

5.3.3 Defining deployment of a service adapter

When you have packaged a service adapter, define its deployment. The following shows the deployment definition procedure.

1. In the tree view, right-click **Service Definition List**, and then select **Deploy all services to server and start**.

5. Experiencing the Development of Sample Programs



If you are not logged in, the account authentication window appears. Perform step 2.

2. Enter admin in both **User ID** and **Password**, and then click the **OK** button.

A message indicating that processing is in progress appears, and then a message reporting the results appears.

5.4 Defining business processes



In this section, define a business process by using the `HelloBusinessProcess` sample program, which calls a service adapter from a business process.



! Important note




Before you define a business process, define the service adapter. The service adapter that you use for the `HelloBusinessProcess` sample program is the Hello service adapter. For details about how to define the Hello service adapter, see 5.3 *Defining service adapters*.

5.4.1 Creating business processes

For a business process, define how the request received from the service requester will be processed. The Hello business process of the `HelloBusinessProcess` sample program executes processing as follows:

1. The Hello business process receives the character string entered from the service requester.
2. The Hello service is called via the Hello service adapter.
3. When the Hello service is called, the following character string is concatenated to the received one:  and 

Business  Process ( : single-byte space)

Generated string: `string-output-by-Hello-service`  and  Business  Process

4. The concatenation result is returned to the service requester, and is displayed in the output window.

Define a business process of the `HelloBusinessProcess` sample program as follows:

1. Add a new business process.
2. Set variables.
3. Deploy activities[#].
4. Define activities[#].
5. Finish defining the business process.

#

An activity is a component that defines an overview of the processing of a business process.

(1) Adding a business process

The following table shows the values that you need to set when adding a business process.

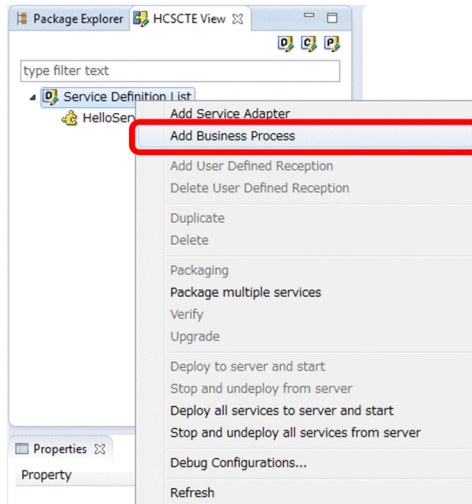
Table 5–2: Values that need to be set when adding a business process

Item	Value to be set	Description
Business process name	<code>HelloBusinessProcess</code>	Specify the name of the business process.
Status persistence	<code>yes</code>	Specify whether to leave records in the database. Records left in the database can be used to check the progress of a process. For this sample program, you leave records in the database. Therefore, select <code>yes</code> .
Import check box in the BPEL file area	Clear the check box.	Specify whether to import the BPEL file that was created by using a tool in the upper process. If you import the BPEL file, the activities necessary for the business process are automatically displayed. For this sample program, you do not import the file. Therefore, clear the check box.

Item	Value to be set	Description
Service ID	HelBP	Specify the ID of the business process.

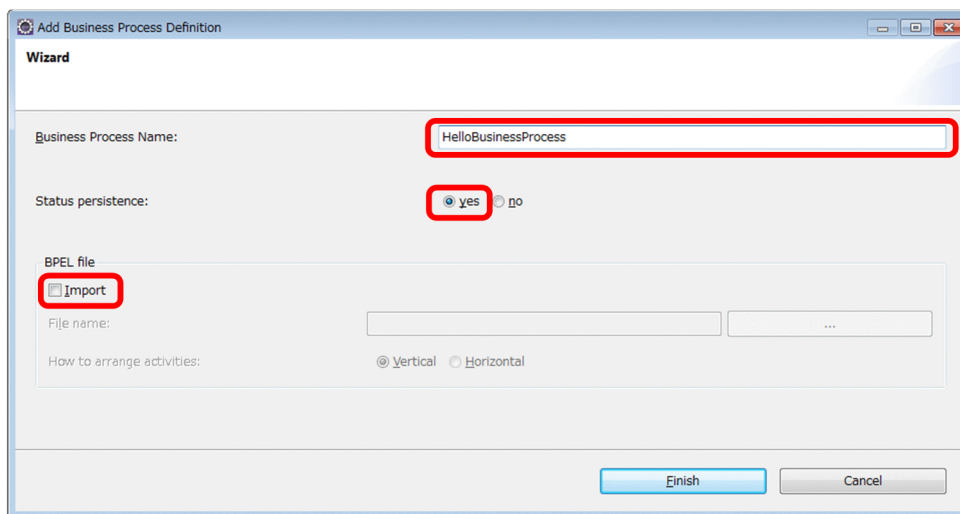
The following is the procedure for adding the Hello business process.

1. In the tree view, select and right-click **Service Definition List**, and then select **Add Business Process**.

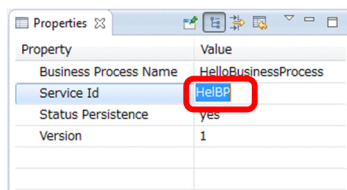


The dialog box for adding a business process definition appears.

2. Enter `HelloBusinessProcess` in **Business Process Name**, and then select **yes** for **Status Persistence**. Clear the **Import** check box in the **BPEL file** area.



3. Click the **Finish** button.
A business process named `HelloBusinessProcess` is created, and then the Define Business Process window appears.
4. In the tree view, select `HelloBusinessProcess`.
A list of properties for the Hello business process is displayed in the properties view.
5. In the properties view, click the cell for the value of the service ID.
A value can be entered in the cell.
6. Change the value to `HelBP`, and then press the **Enter** key.



7. When a message asking you whether you really want to change the value appears, click the **OK** button.

(2) Setting variables

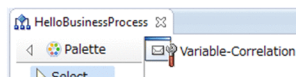
For a business process, variables are used to define activities. Therefore, the variables to be used must be set before activities are defined. The following table shows the variables to be used for the Hello business process.

Table 5–3: Variables to be used for the Hello business process

Variable name	Type	XSD file
InputData	XML	InputData.xsd
OutputData	XML	OutputData.xsd

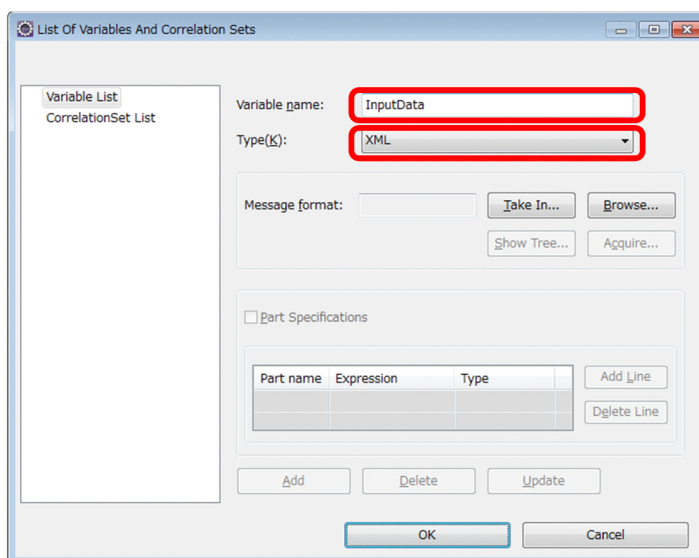
The following is the procedure for setting the variables to be used for the Hello business process.

1. On the canvas of the Define Business Process window, double-click the **Variable-Correlation** icon.



The List Of Variables And Correlation Sets dialog box appears.

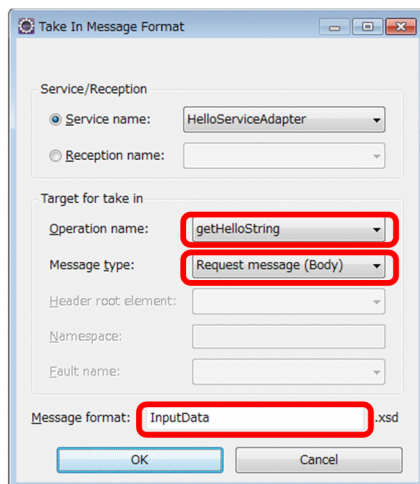
2. In the tree view, select **Variable List**.
3. Enter InputData in **Variable name**, and then select **XML** from the **Type** drop-down list.



4. Click the **Take In** button.

The Take In Message Format dialog box appears.

5. Select **Service name**, and then, from the drop-down list, select **Hello service adapter**.
6. Select **getHelloString** from the **Operation name** drop-down list, and **Request message (Body)** from the **Message type** drop-down list. For **Message format**, enter InputData.



7. Click the **OK** button.

The Take In Message Format dialog box closes.

8. In the List Of Variables And Correlation Sets dialog box, click the **Add** button.

InputData is added to the **Variable List** node in the tree view.

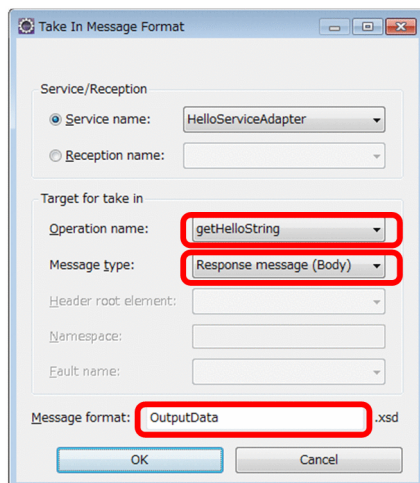
9. In the List Of Variables And Correlation Sets dialog box, select **Variable List**. Enter OutputData in **Variable name**, and then select **XML** from the **Type** drop-down list.

10. Click the **Take In** button.

The Take In Message Format dialog box appears.

11. Select **Service name**, and then, from the drop-down list, select **Hello service adapter**.

12. Select **getHelloString** from the **Operation name** drop-down list, and **Response message (Body)** from the **Message type** drop-down list. For **Message format**, enter OutputData.



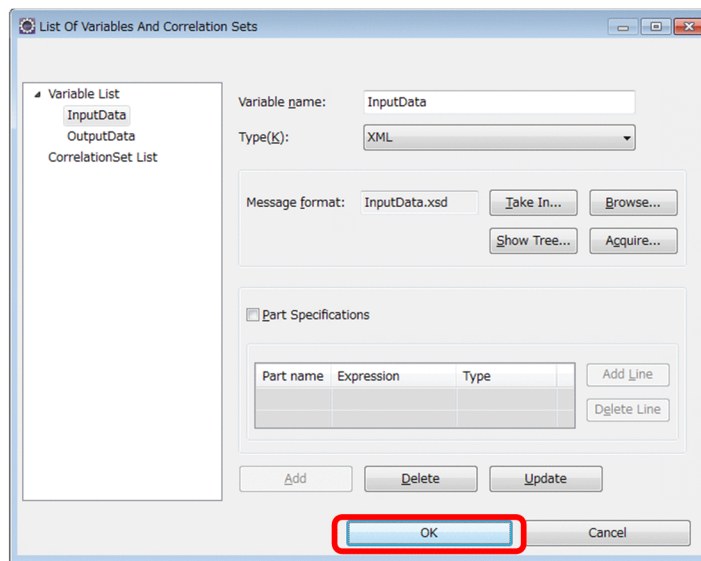
13. Click the **OK** button to close the Take In Message Format dialog box.

14. In the List Of Variables And Correlation Sets dialog box, click the **Add** button.

OutputData is added to the **Variable List** node in the tree view.

15. In the List Of Variables And Correlation Sets dialog box, click the **OK** button.

The variables are now set.




(3) Deploying activities

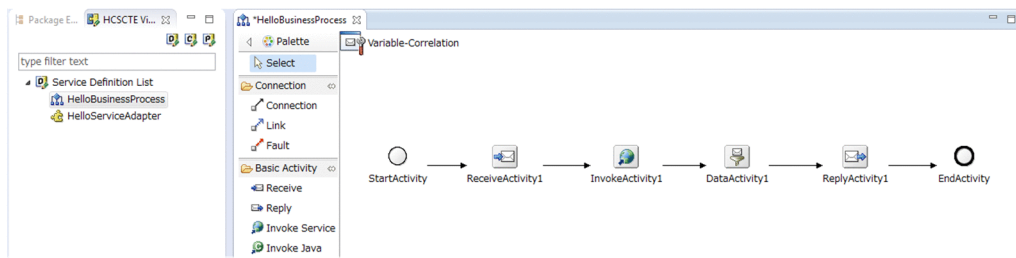
The following table shows the activities that are necessary for the business process of the HelloBusinessProcess sample program.

Table 5–4: Activities necessary for the business process of the HelloBusinessProcess sample program

Value to be set	Description
<i>Receive</i> activity	Receives a response from the service requester
<i>Invoke service</i> activity	Calls the Hello service
<i>Data transformation</i> activity	Edits a string
<i>Reply</i> activity	Returns the processing result to the service requester

The following shows the activity deployment procedure.

- On the palette, click the following activities, and then place them at appropriate positions by clicking them on the canvas.
 - Receive*
 - Invoke service*
 - Data transformation*
 - Reply*
- To connect activities, on the palette, click  **Connection**.
- Click the *start* activity to start the connection.
- Click the *receive* activity as the connection destination.
The *start* activity is now connected to the *receive* activity.
- Chain the activities by connecting adjacent ones (as in steps 2 to 4) from the *receive* activity to the *reply* activity.
Make sure that the activities are chained as follows.



(4) Defining activities

Define each of the activities that were placed on the canvas.

(a) Receive activity

1. Double-click the *receive* activity on the canvas.
The Receive Activity dialog box appears.
2. Enter information as shown in the following figure.

Item	Value to be set	Description
Activity name	Receive	Specify the name of the activity.
Operation name	getHelloString	Specify the name of the operation that is used to call a service component from the service requester.
Body allocated variable	InputData	From the drop-down list, select the variable to be allocated to the body of the request message for the business process.
Header allocated variable	None	Set this item when you allocate a variable to the header of the request message for the business process. This item is not used for this sample program. Therefore, do not set this item.
Correlation set group	None	Set this item when you allocate a correlation set group to an activity. This item is not used for this sample program. Therefore, do not set this item.
Communication model	Sync	Specify the communication model of the operation. The Hello service used for this sample program is a Web Service. Therefore, set Sync.
Instance generation	yes	Select whether to initialize the process when a request message is received. For this sample program, set yes.

3. Click the **OK** button.

(b) Invoke service activity

1. Double-click the *invoke service* activity on the canvas.

The Invoke Service Activity dialog box appears.

2. Enter information as shown in the following figure.

Item	Value to be set	Description
Activity name	HelloService	Enter the name of the activity.
Service name	HelloServiceAdapter	From the drop-down list, select the name of the service component to be called by sending a request message.
Operation name	getHelloString	Among the operations for the service component (Hello service adapter) specified in Service name , select the name of the operation that is to be called from the drop-down list.
Communication model	Sync	The communication model set for the operation specified in Operation name is displayed.
Body allocated variable (in the Request message area)	InputData	From the drop-down list, select the variable to be allocated to the body of the request message that calls the stock management service.
Header allocated variable (in the Request message area)	None	Set this item when you allocate a variable to the header of the request message that calls the stock management service. This item is not used for this sample program. Therefore, do not set this item.
Body allocated variable (in the Response message area)	OutputData	From the drop-down list, select the variable to be allocated to the body of the response message to be received from the synchronization operation.
Header allocated variable (in the Response message area)	None	Set this item when you allocate a variable to the header of the response message to be received from the synchronization operation. This item is not used for this sample program. Therefore, do not set this item.
Correlation set group	None	Set this item when you allocate a correlation set group to an activity. This item is not used for this sample program. Therefore, do not set this item.

3. Click the **OK** button.

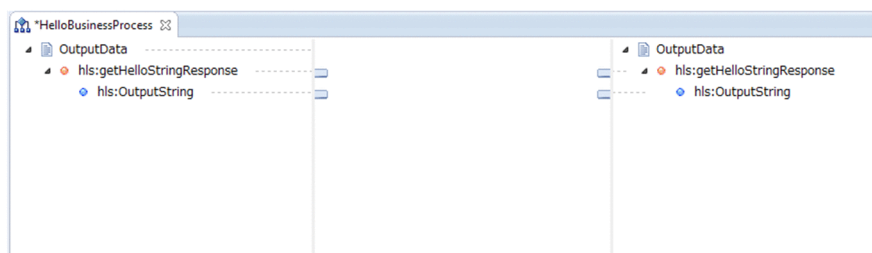
(c) Data transformation activity

1. Double-click the *data transformation* activity on the canvas.
The Data Transformation Activity dialog box appears.
2. Enter information as shown in the following figure.

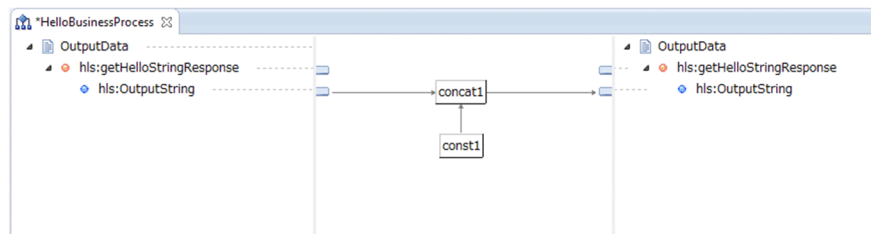
Item	Value to be set	Description
Activity name	EditOutputString	Specify the name of the activity.
Variable (in the Source Variables area)	OutputData	Select the transformation-source variable from the drop-down list, and then click the Add button.
Variable (in the Destination Variable area)	OutputData	Select the transformation-destination variable from the drop-down list.
DataTransDefnFile	EditOutputString	Specify a name for the data transformation definition file.

3. Click the **OK** button.
4. Right-click the data transformation activity on the canvas, and then select **Launch mapping definition**.
The Select Root Element dialog box appears.
5. As the root element of OutputData (schema logical name) for **Source**, select `hls:getHelloStringResponse` from the drop-down list. As the root element of OutputData (schema logical name) for **Target**, select `hls:getHelloStringResponse`.

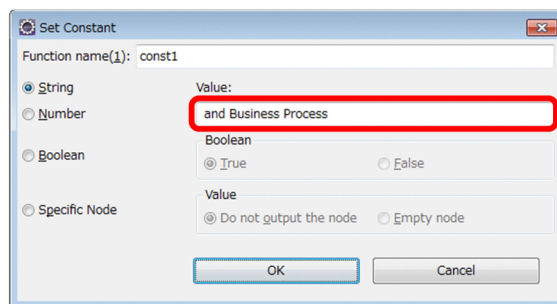
6. Click the **OK** button.
The data transformation definition window appears.



7. On the palette of the data transformation definition window, click **concat**. Then, on the canvas, click between the transformation source and destination to place the **concat** there.
8. On the palette of the data transformation definition window, click **const**. Then, on the canvas, click between the transformation source and destination to place the **const** there.
9. On the palette of the data transformation definition window, select **Mapping**.
10. Click the node adapter of the transformation-source node as the mapping source.
11. Click **concat** as the mapping destination.
A mapping line is set.
12. In the same way as steps 9 to 11, set a mapping line from **concat** to the node adapter of the transformation-destination node.
Do not set a mapping line from **const** to **concat** first. Doing so will change the order of output strings.
13. In the same way as steps 9 to 11, set a mapping line from **const** to **concat**.



14. On the palette of the data transformation definition window, click **Select**.
15. Double-click **const**.
The Set Constant dialog box appears.
16. Select **String**, and then enter the following character string: `△ and △ Business △ Process (△ :`
single-byte space)



17. Click the **OK** button.

(d) Reply activity

1. Double-click the *reply* activity on the canvas.
The Reply Activity dialog box appears.
2. Enter information as shown in the following figure.

The screenshot shows a 'Reply Activity' dialog box with the following fields and values:

- Activity name: Reply
- Operation name: getHelloString
- Body allocated variable: OutputData (with an 'Edit...' button)
- Header allocated variable: Setting(H)... (with a 'Setting(H)...' button)
- Correlation set group: Setting... (with a 'Setting...' button)
- Fault name: (empty)

At the bottom are 'OK' and 'Cancel' buttons.

Item	Value to be set	Description
Activity name	Reply	Specify the name of the activity.
Operation name	getHelloString	Specify the name of the operation specified for the corresponding receive activity.
Body allocated variable	OutputData	From the drop-down list, select the variable to be allocated to the body of the response message.
Header allocated variable	None	Set this item when you allocate a variable to the header of the response message. This item is not used for this sample program. Therefore, do not set this item.
Correlation set group	None	Set this item when you allocate a correlation set group to an activity. This item is not used for this sample program. Therefore, do not set this item.
Fault name	None	Define the reply activity as fault processing, and then specify the fault name to be used when a response message that indicates that a fault occurred in the service requester is received. No fault processing is used for this sample program. Therefore, do not set this item.

3. Click the **OK** button.

4. When you have defined all activities, from the menu, select **File** and then **Save**. The business process is now defined.

5.4.2 Validating and packaging a Hello business process

When you have created a business process, validate that it is defined correctly, and then package it. The validation and packaging procedures are the same as those for a Hello service adapter. For details about validation and packaging, see *5.3.2 Validating and packaging a service adapter*.

5.4.3 Defining deployment of a Hello business process

When you have packaged a business process, define its deployment. The deployment procedure is the same as that for a Hello service adapter. For details about deployment definitions, see *5.3.3 Defining deployment of a service adapter*.

5.5 Developing the product arrangement system

In this section, define a business process by using the `HelloProductArrangement` sample program, which calls a service adapter from a business process whose processing is close to an actual job.

For the `HelloProductArrangement` sample program, define the following three components:

- Stock management service adapter
- Delivery reception service adapter
- Product arrangement business process

5.5.1 Defining the stock management service adapter

Use the WSDL file for the stock management service (`InventoryManagementService.wsdl`) to define the stock management service adapter. The following table shows the values that you need to set when defining the stock management service adapter.

Table 5–5: Values that need to be set when defining the stock management service adapter

Item	Value to be set	Description
Service component type	Web Service	Specify the type of service adapter that is set according to the type of service component to be called. Because the service component type of this sample program is a Web Service, also specify <code>Web Service</code> as the service adapter type.
Service name	StockManagement	Specify the name of the service adapter.
WSDL file	<code>InventoryManagementService.wsdl</code>	The WSDL file defines the method of writing what functions the Web Service has and what requests you need to send to use those functions, etc. Use the WSDL file for the stock management service (<code>InventoryManagementService.wsdl</code>) to create the stock management service adapter.
Port specification	<code>InventoryManager</code>	Specify the port of the stock management service adapter.
Service ID	<code>InvAdp</code>	Specify the ID of the stock management service adapter.
Client definition file	<code>c4webcl.properties</code> (This file contains the following entry: <code>c4web.logger.log_file_prefix=InventoryManagementService</code>)	The client definition file controls the client-side behavior. The user creates this file with the name <code>c4webcl.properties</code> . This sample program provides a client definition file that sets <code>InventoryManagementService</code> as the prefix of the trace file and application log.

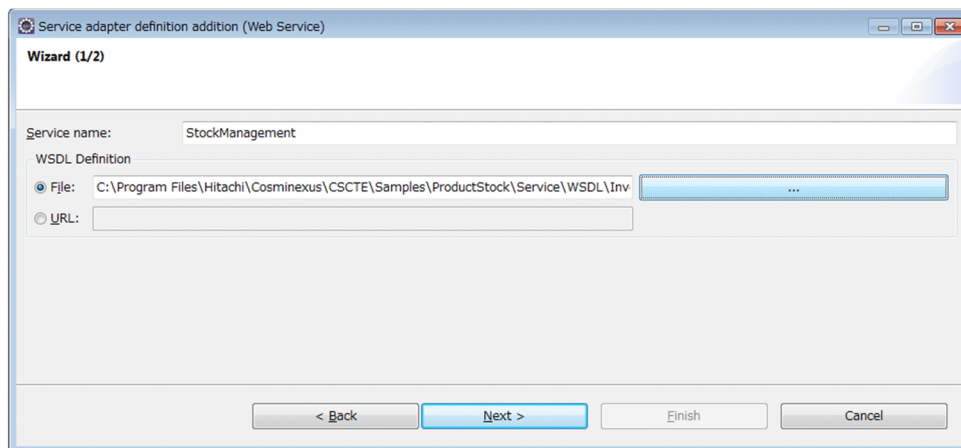
Note:

For details about the location of each file, see *A.3 Configuration of the HelloProductArrangement sample program*.

The following is the procedure for adding and defining the stock management service adapter.

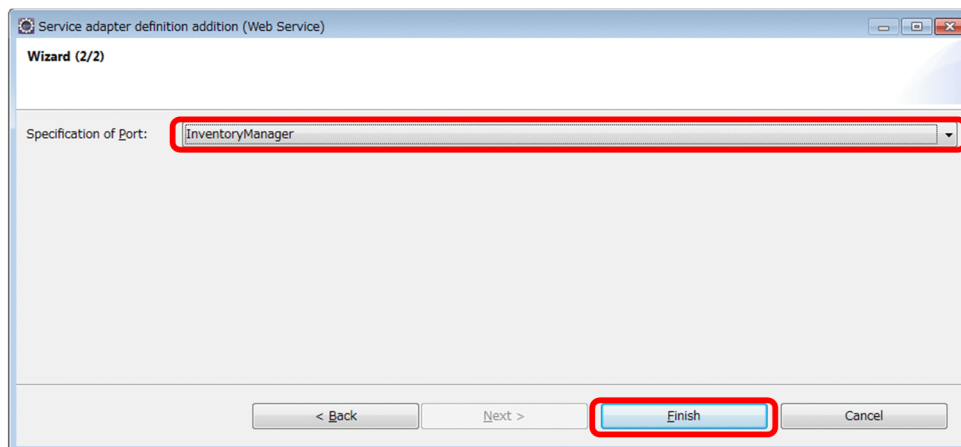
1. In the tree view, select and right-click **Service Definition List**, and then select **Add Service Adapter**.
The dialog box for setting the type of service to be used from the service adapter to be added appears.
2. From the drop-down list, select **Web Services**, and then click the **Next** button.
The dialog box for entering the information that is necessary for adding a SOAP adapter appears.
3. Enter `StockManagement` as the service name, specify `InventoryManagementService.wsdl` as the WSDL file, and then click the **Next** button.

5. Experiencing the Development of Sample Programs



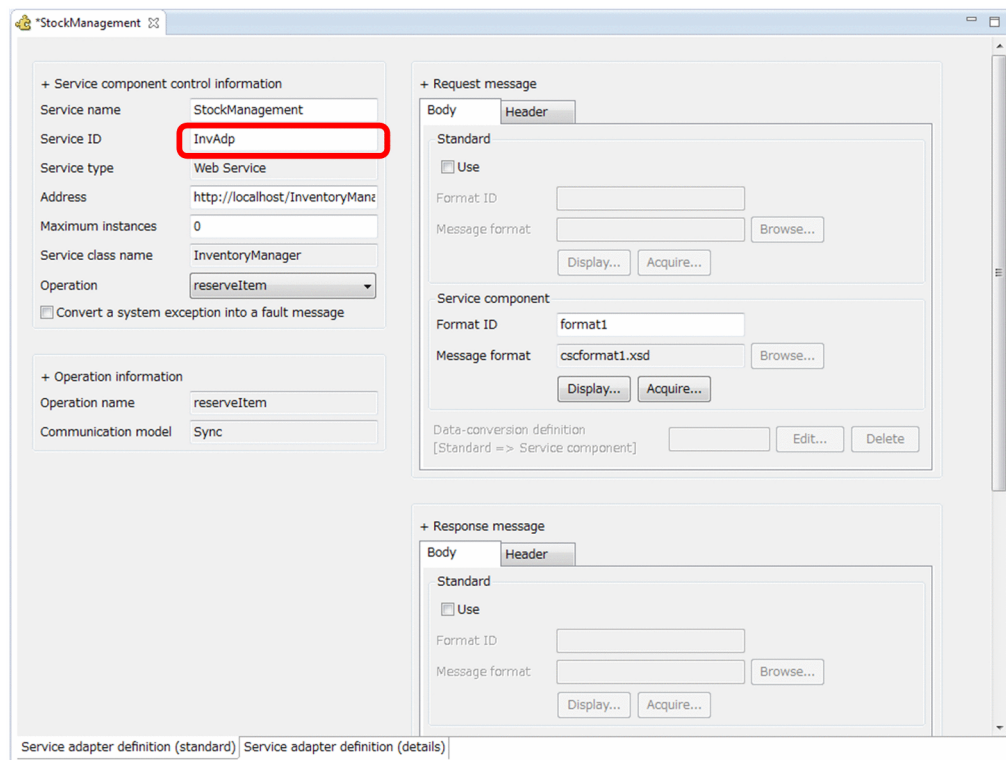
The dialog box for specifying the port appears.

- From the **Specification of Port** drop-down list, select **InventoryManager**, and then click the **Finish** button.



A service adapter named `StockManagement` service adapter is created, and then the service adapter definition window appears.

- In the service adapter definition (standard) window, change the service ID to `InvAdp`.



6. In the service adapter definition (details) window, click the **Browse** button, and then specify `c4webcl.properties` as the client definition file.

For the `HelloProductArrangement` sample program, the location of the above file is as follows:

`service-platform-installation-directory\CSCTE\Samples\ProductStock\Service
InventoryManagementService\c4webcl.properties`

7. From the menu, select **File**, and then **Save**.

5.5.2 Defining the delivery reception service adapter

Use the WSDL file for the delivery reception service (`DeliveryService.wsdl`) to define the delivery reception service adapter. The following table shows the values that you need to set when defining the delivery reception service adapter.

Table 5–6: Values that need to be set when defining the delivery reception service adapter

Item	Value to be set	Description
Service component type	Web Service	Specify the type of service adapter that is set according to the type of service component to be called. Because the service component type of this sample program is a Web Service, also specify <code>Web Service</code> as the service adapter type.
Service name	DeliveryReception	Specify the name of the service adapter.
WSDL file	DeliveryService.wsdl	The WSDL file defines the method of writing what functions the Web Service has and what requests you need to send to use those functions, etc. Use the WSDL file for the delivery reception service (<code>DeliveryService.wsdl</code>) to define the delivery reception service adapter.
Port specification	Delivery	Specify the port of the delivery reception service adapter.
Service ID	DelAdp	Specify the ID of the delivery reception service adapter.

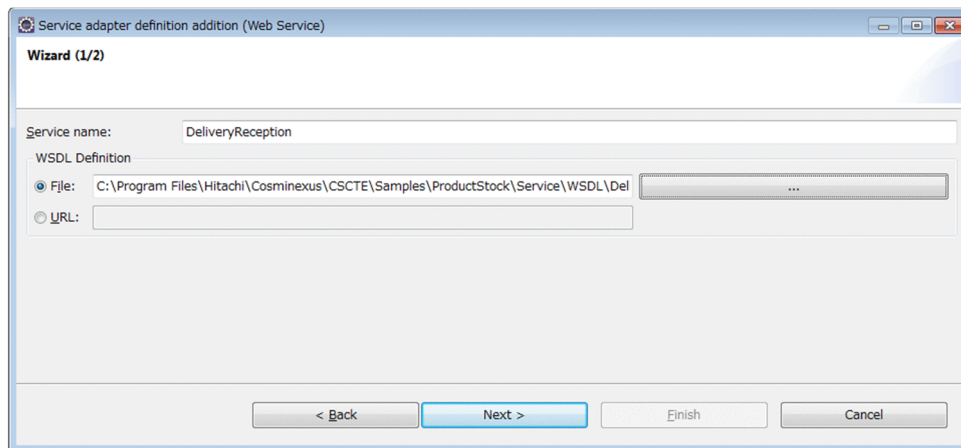
Item	Value to be set	Description
Client definition file	c4webcl.properties (This file contains the following entry: c4web.logger.log_file_prefix=DeliveryService)	The client definition file controls the client-side behavior. The user creates this file with the name c4webcl.properties. This sample program provides a client definition file that sets DeliveryService as the prefix of the trace file and application log.

Note:

For details about the location of each file, see *A.3 Configuration of the HelloProductArrangement sample program*.

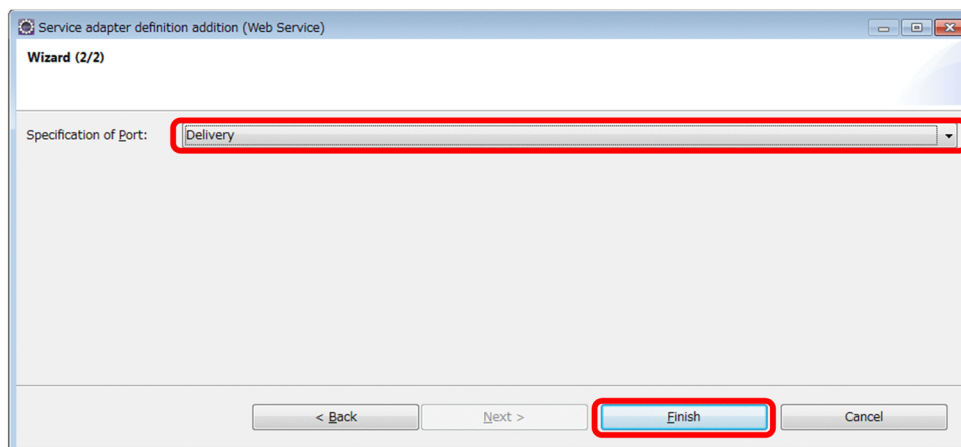
The following is the procedure for adding and defining the delivery reception service adapter.

1. In the tree view, select and right-click **Service Definition List**, and then select **Add Service Adapter**.
The dialog box for setting the type of service to be used from the service adapter to be added appears.
2. From the drop-down list, select **Web Services**, and then click the **Next** button.
The dialog box for entering the information that is necessary for adding a SOAP adapter appears.
3. Enter `DeliveryReception` as the service name, specify `DeliveryService.wsdl` as the WSDL file, and then click the **Next** button.



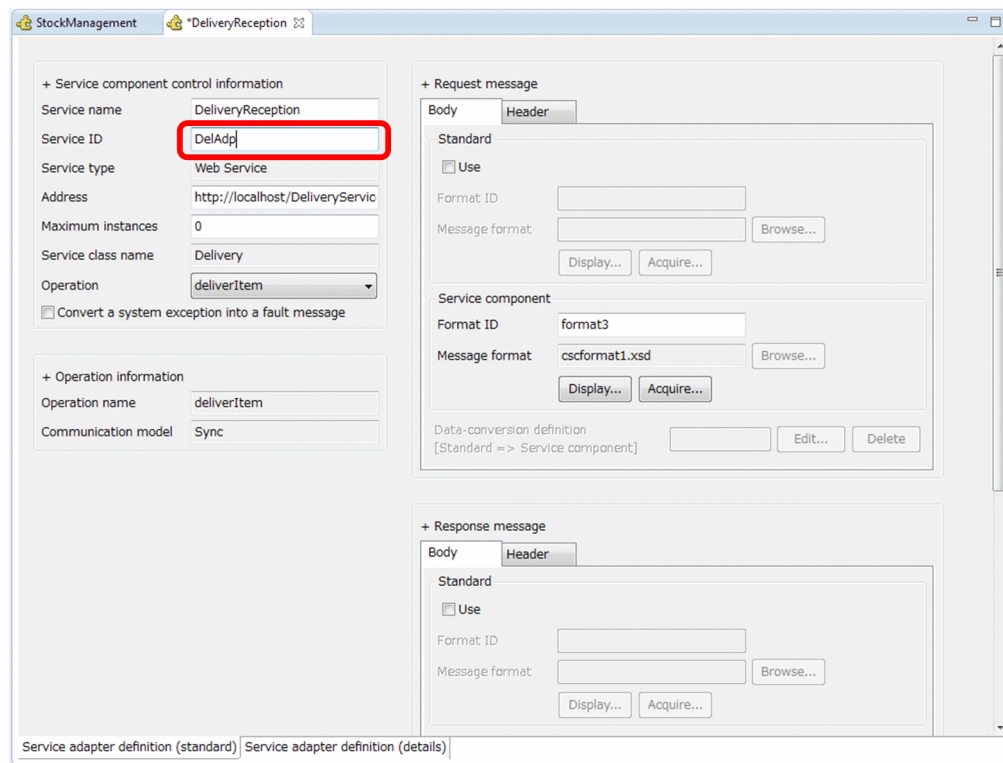
The dialog box for specifying the port appears.

4. From the drop-down list, select **Delivery**, and then click the **Finish** button.



A service adapter named `DeliveryReception` service adapter is created, and then the service adapter definition window appears.

5. In the service adapter definition (standard) window, change the service ID to `DelAdp`.



6. In the service adapter definition (details) window, click the **Browse** button, and then specify `c4webcl.properties` as the client definition file.

For the `HelloProductArrangement` sample program, the location of the above file is as follows:

```
service-platform-installation-directory\CSCTE\Samples\ProductStock\Service\DeliveryService\c4webcl.properties
```

7. From the menu, select **File**, and then **Save**.

5.5.3 Defining the product arrangement business process

The product arrangement business process of the `HelloProductArrangement` sample program executes processing as follows:

1. The product arrangement business process receives the product name and quantity entered from the service requester.
2. The stock management service is called via the stock management service adapter.
3. When the stock management service is called, if the product is out of stock, an asterisk (*), which means *out of stock*, is returned to the service requester.
4. If the product is in stock, the delivery number is obtained by calling the delivery reception service via the delivery reception service adapter.
5. The obtained delivery number is returned to the service requester.

Note that because the product arrangement business process is called by using a user-defined reception, you need to create a user-defined reception by using WSDL (`ArrangementService.wsdl`).

Define a business process of the `HelloProductArrangement` sample program as follows:

1. Add a new business process.
2. Add a user-defined reception.
3. Set variables.
4. Deploy activities.

5. Define activities.
6. Finish defining the business process.

(1) Adding a business process

The following table shows the values that you need to set when adding the product arrangement business process.

Table 5–7: Values that need to be set when adding a business process

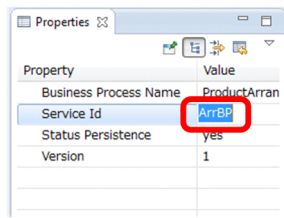
Item	Value to be set	Description
Business process name	ProductArrangement	Specify the name of the business process.
Status persistence	yes	Specify whether to leave records in the database. The records left in the database can be used to check the progress of a process. For this sample program, you leave records in the database. Therefore, select <i>yes</i> .
Import check box in the BPEL file area	Clear the check box.	To import the BPEL file that was created by using a tool in the upper process, select the check box. If you import the file, the activities necessary for the business process are automatically displayed. For this sample program, you do not import the file. Therefore, clear the check box.
Service ID	ArrBP	Specify the ID of the business process.

The following is the procedure for adding the product arrangement business process.

1. In the tree view, select and right-click **Service Definition List**, and then select **Add Business Process**.
The dialog box for adding a business process definition appears.
2. Enter *ProductArrangement* in **Business Process Name**, and then select *yes* for **Status Persistence**. Clear the **Import** check box in the **BPEL file** area.

The screenshot shows the 'Add Business Process Definition' wizard. The 'Business Process Name' field is set to 'ProductArrangement'. Under 'Status persistence', the 'yes' radio button is selected. In the 'BPEL file' section, the 'Import' checkbox is unchecked. The 'File name' field is empty, and the 'How to arrange activities' section has 'Vertical' selected. The 'Finish' button is highlighted.

3. Click the **Finish** button.
A business process named *ProductArrangement* is created, and then the Define Business Process window appears.
4. In the tree view, select **Product Arrangement**.
A list of properties for the *ProductArrangement* business process is displayed in the properties view.
5. In the properties view, click the cell for the value of the service ID.
A value can be entered in the cell.
6. Change the value to *ArrBP*, and then press the **Enter** key.



7. When a message asking you whether you really want to change the value appears, click the **OK** button.

(2) Adding a user-defined reception

The `HelloProductArrangement` sample program receives a request from the service requester by using a reception that has been defined by the user according to the interface of the business process. The interface of the business process includes the operation name and message format to be set for the receive activity and reply activity. The following table shows the values that you need to set when adding a user-defined reception.

Table 5–8: Values that need to be set when adding a user-defined reception

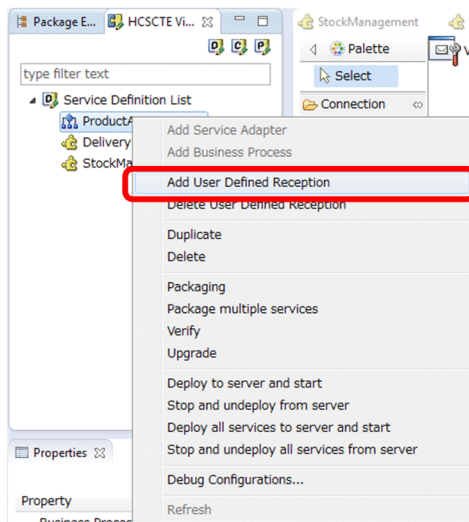
Item	Value to be set	Description
Reception type	SOAP Reception	Select the reception type.
Reception name	ServiceReception	Specify the name of the user-defined reception.
WSDL file	ArrangementService.wsdl	Specify the name of the WSDL file to be used.
Port name	Arrangement	Specify the port name.

Note:

For details about the location of the file, see *A.3 Configuration of the HelloProductArrangement sample program*.

The following is the procedure for adding a user-defined reception for product arrangement.

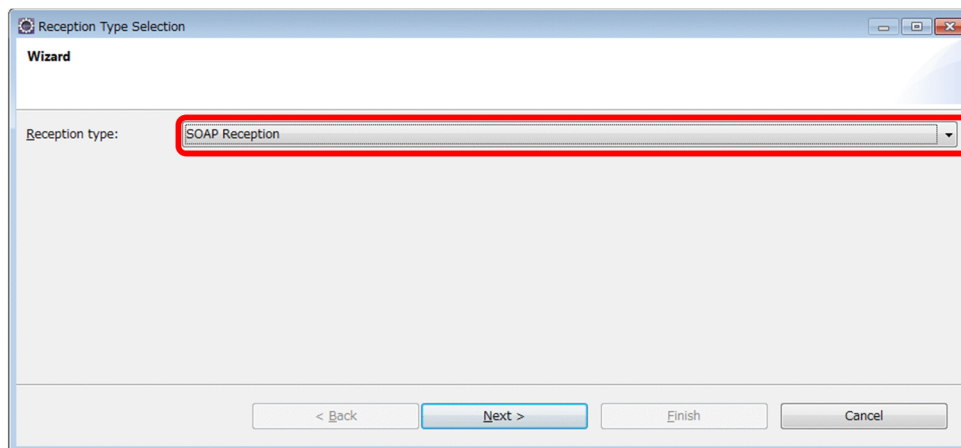
1. In the tree view, select and right-click the **Product Arrangement** business process, and then select **Add User Defined Reception**.



The dialog box for selecting the reception type appears.

2. From the **Reception type** drop-down list, select **SOAP Reception**.

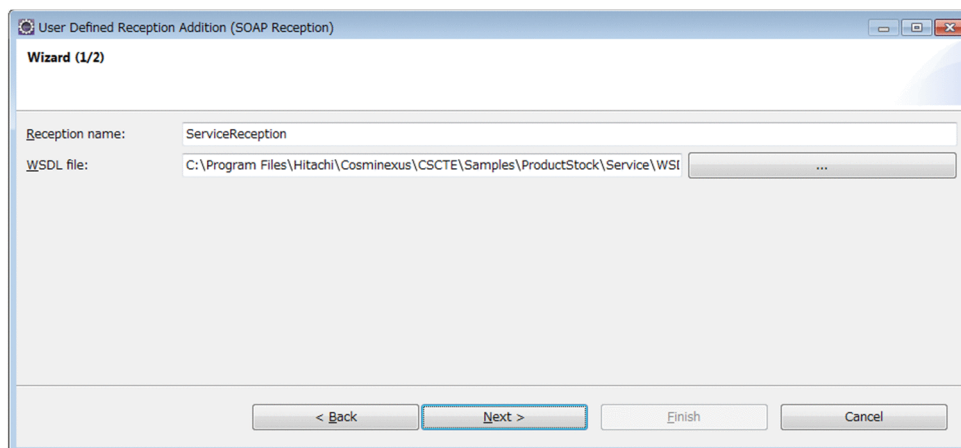
5. Experiencing the Development of Sample Programs



3. Click the **Next** button.

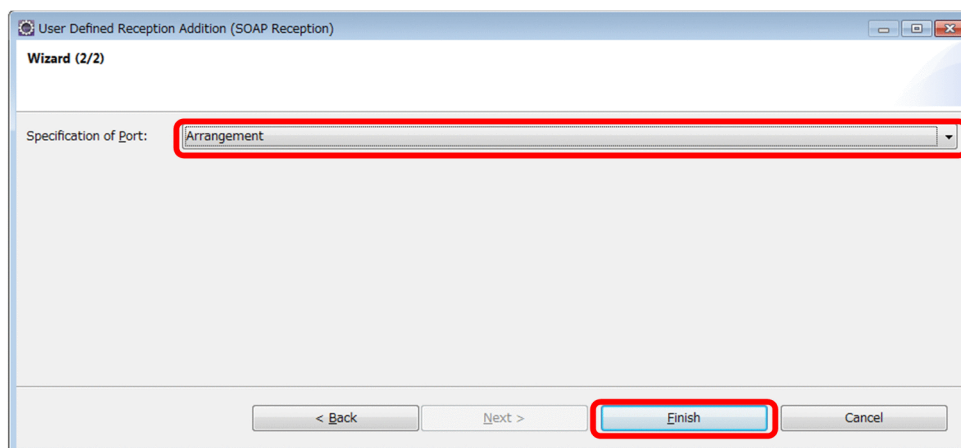
The dialog box for adding the SOAP reception appears.

4. Enter `ServiceReception` as the reception name, and then specify `ArrangementService.wsdl` as the WSDL file.



5. Click the **Next** button.

6. From the drop-down list, select **Arrangement**, and then click the **Finish** button.



The SOAP reception is added to the business process, and then the window for defining a user-defined reception appears.

(3) Setting variables

For a business process, variables are used to define activities. Therefore, the variables to be used must be set before activities are defined. The following table shows the variables to be used for the product arrangement business process.

Table 5–9: Variables to be used for the product arrangement business process

Variable name	Type	XSD file
InputData	XML	InputData.xsd
OutputData	XML	OutputData.xsd
StockAllocationInputData	XML	StockAllocationInputData.xsd
StockAllocationOutputData	XML	StockAllocationOutputData.xsd
DeliveryArrangementInputData	XML	DeliveryArrangementInputData.xsd
DeliveryArrangementOutputData	XML	DeliveryArrangementOutputData.xsd

The following is the procedure for setting the variables to be used for the product arrangement business process.

1. On the canvas of the Define Business Process window, double-click the **Variable-Correlation** icon.
The List Of Variables And Correlation Sets dialog box appears.
2. Select **Variable List**. Enter InputData in **Variable name**, and then select **XML** from the **Type** drop-down list.
3. Click the **Take In** button.
The Take In Message Format dialog box appears.
4. Select **Reception name**, and then, from the drop-down list, select **Service Reception**.
5. Select **arrangeItem** from the **Operation name** drop-down list, and **Request message (Body)** from the **Message type** drop-down list. For **Message format**, enter InputData.

The screenshot shows the 'Take In Message Format' dialog box. It has two main sections: 'Service/Reception' and 'Target for take in'. In the 'Service/Reception' section, 'Reception name' is set to 'ServiceReception'. In the 'Target for take in' section, 'Operation name' is 'arrangeItem' and 'Message type' is 'Request message (Body)'. The 'Message format' is 'InputData.xsd'. The 'OK' button is highlighted with a red box.

6. Click the **OK** button.

The Take In Message Format dialog box closes.

7. In the List Of Variables And Correlation Sets dialog box, click the **Add** button.

The InputData variable is added to the **Variable List** node.

8. In the same way as steps 2 to 7, set the variables OutputData, StockAllocationInputData, StockAllocationOutputData, DeliveryArrangementInputData, and DeliveryArrangementOutputData.

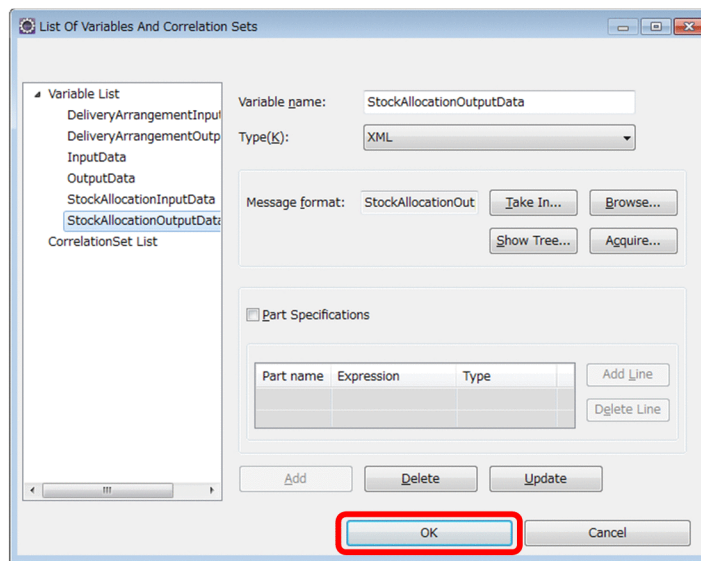
The values to be set are as follows.

All of the types set in the List Of Variables And Correlation Sets dialog box are XML. The values to be set in the Take In Message Format dialog box are as follows.

Item	Variable name				
	OutputData	StockAllocationInputData	StockAllocationOutputData	DeliveryArrangementInputData	DeliveryArrangementOutputData
Service/Reception	Reception name	Service name	Service name	Service name	Service name
Name of Service/Reception	ServiceReception	StockManagement	StockManagement	DeliveryReception	DeliveryReception
Operation name	arrangeItem	reserveItem	reserveItem	deliverItem	deliverItem
Message type	Response message (Body)	Request message (Body)	Response message (Body)	Request message (Body)	Response message (Body)
Message format	Output data	Stock allocation input data	Stock allocation output data	Delivery arrangement input data	Delivery arrangement output data

9. In the List Of Variables And Correlation Sets dialog box, click the **OK** button.

The variables are now set.



(4) Deploying activities

The following table shows the activities that are necessary for the business process of the HelloProductArrangement sample program.

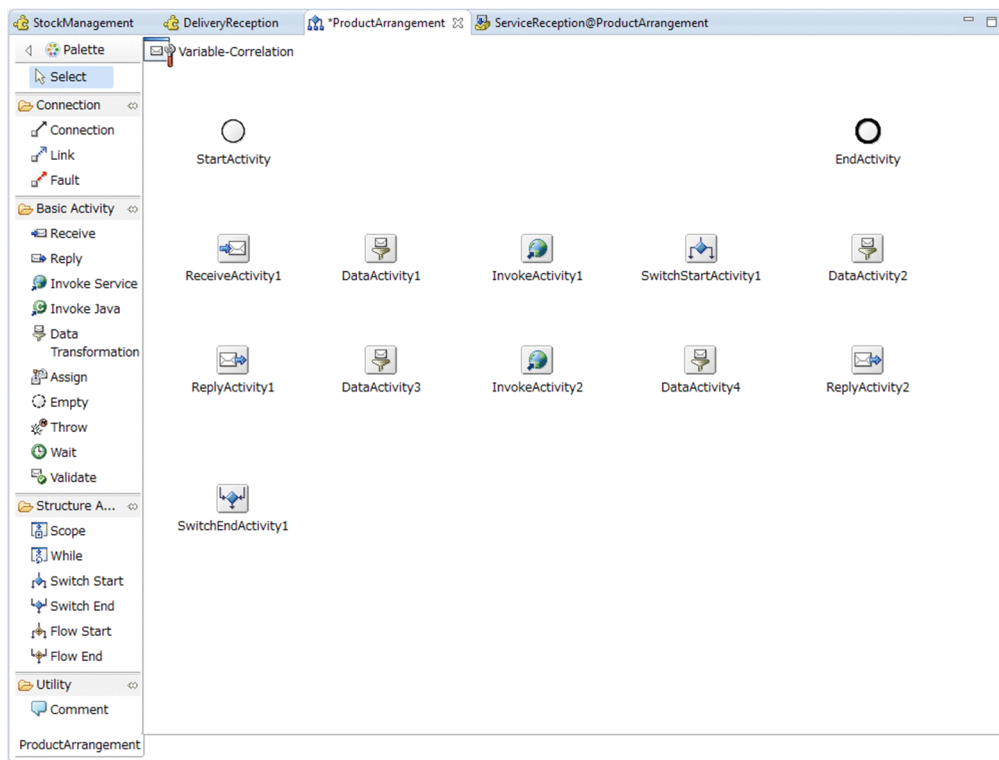
Table 5–10: Activities necessary for the business process of the HelloProductArrangement sample program

Value to be set	Description
<i>Receive activity</i>	Receives a response from the service requester
<i>Data transformation activity</i>	Edits input data, output data, stock allocation data, and delivery arrangement data
<i>Invoke service activity</i>	Calls the stock management service or delivery reception service
<i>Switch start activity</i>	Starts the processing selected according to the condition (whether the product is in stock)
<i>Reply activity</i>	Returns the processing result to the service requester
<i>Switch end activity</i>	Ends the processing selected according to the condition (whether the product is in stock)

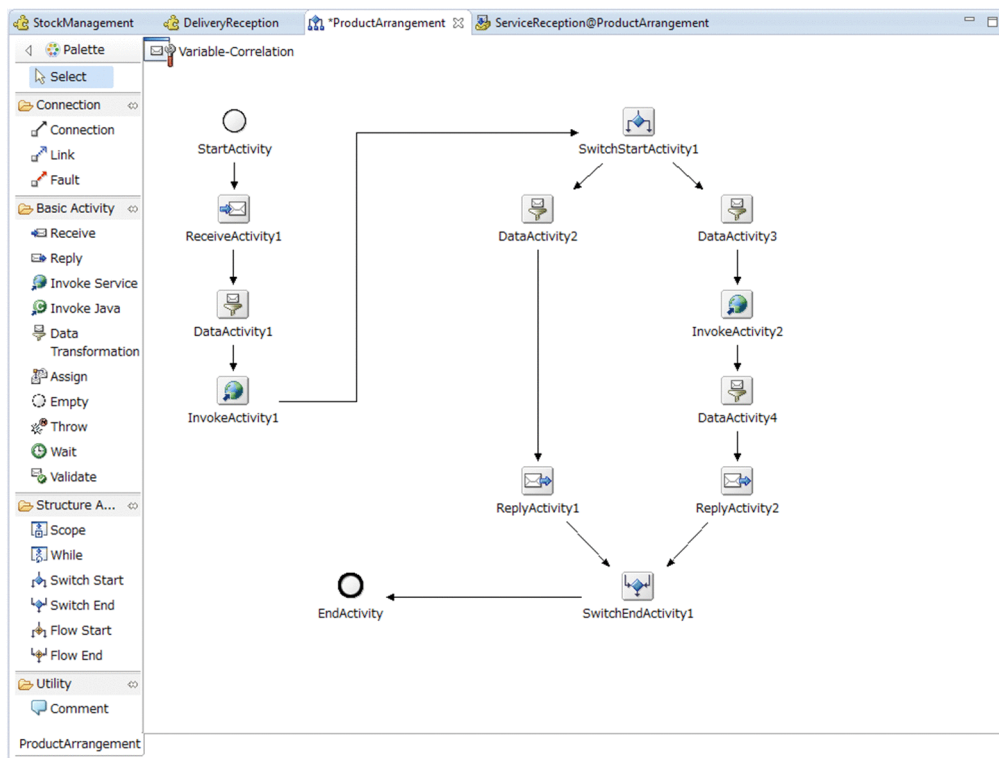
The following shows the activity deployment procedure.

1. Place activities on the canvas as shown in the following figure. To place an activity, click on it on the palette, and then click the position at which to place it on the canvas.

5. Experiencing the Development of Sample Programs



2. To connect activities, click **Connection** on the palette.
3. Click the *start* activity to start the connection.
4. Click the *receive* activity as the connection destination.
The *start* activity is now connected to the *receive* activity.
5. Chain the activities by connecting adjacent ones (as in steps 2 to 4) from the *receive* activity to the *end* activity.
Make sure that the activities are chained as follows.



(5) Defining activities

Define each of the activities that were placed on the canvas.

(a) Receive activity

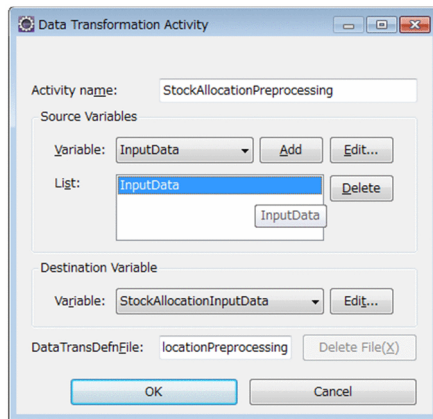
1. Double-click the *receive* activity (**ReceiveActivity1**) on the canvas.
The Receive Activity dialog box appears.
2. Enter information as shown in the following figure.

Item	Value to be set	Description
Activity name	Receive	Enter the name of the activity.
Operation name	arrangeItem	Specify the name of the operation that is used to call the stock management service from the service requester.
Body allocated variable	InputData	From the drop-down list, select the variable to be allocated to the body of the request message for the business process.
Header allocated variable	None	Set this item when you allocate a variable to the header of the request message for the business process. This item is not used for this sample program. Therefore, do not set this item.
Correlation set group	None	Set this item when you allocate a correlation set group to an activity. This item is not used for this sample program. Therefore, do not set this item.
Communication model	Sync	Specify the communication model of the operation. The product arrangement service used for this sample program is a Web Service. Therefore, set Sync.
Instance generation	yes	Select whether to initialize the process when a request message is received. For this sample program, set yes to enable initialization.

3. Click the **OK** button.

(b) Data transformation activity (for preprocessing of stock allocation)

1. Double-click the *data transformation* activity (DataActivity1) on the canvas.
The Data Transformation Activity dialog box appears.
2. Enter information as shown in the following figure.



Item	Value to be set	Description
Activity name	StockAllocationPreprocessing	Enter the name of the activity.
Variable (in the Source Variables area)	InputData	Select the transformation-source variable from the drop-down list, and then click the Add button.
Variable (in the Destination Variable area)	StockAllocationInputData	Select the transformation-destination variable from the drop-down list.
DataTransDefnFile	StockAllocationPreprocessing	Enter the name of the data transformation definition file to be used to transform variables.

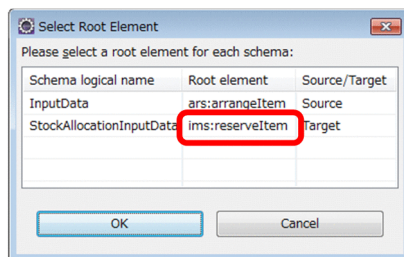
3. Click the **OK** button.

4. Right-click the data transformation activity on the canvas, and then select **Launch mapping definition**.

The Select Root Element dialog box appears.

5. Click the root element of `InputData` (schema logical name) for **Source**, and then, from the drop-down list, select `ars:arrangeItem`.

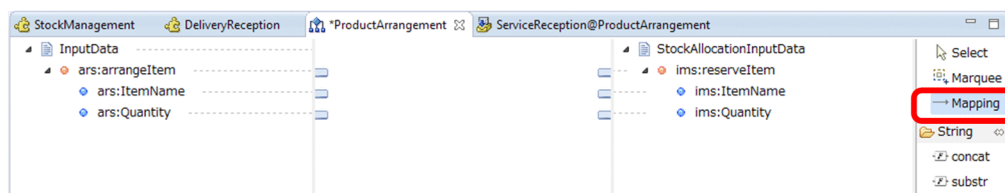
6. Click the root element of `StockAllocationInputData` (schema logical name) for **Destination**, and then, from the drop-down list, select `ims:reserveItem`.



7. Click the **OK** button.

The data transformation definition window appears.

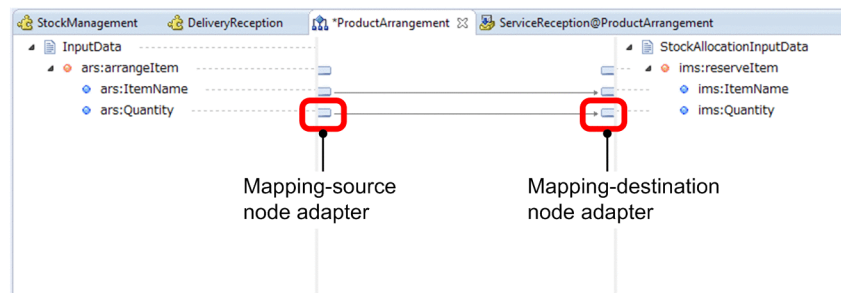
8. On the palette of the data transformation definition window, select **Mapping**.



9. Click the node adapter of the transformation-source node as the mapping source.

10. Click the node adapter of the transformation-destination node as the mapping-destination.

A mapping line is set. The correspondence between the mapping-source and mapping-destination node adapters is as follows.



- (c) Data transformation activity (in the case where the product is out of stock)

1. Double-click the *data transformation* activity (DataActivity2) on the canvas.
The Data Transformation Activity dialog box appears.
2. Enter information as shown in the following figure.

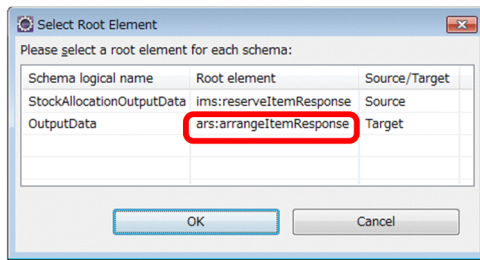
The Data Transformation Activity dialog box is shown with the following configuration:

- Activity name:** Out-Of-Stock-Setting
- Source Variables:**
 - Variable: StockAllocationOutput (Add button)
 - List: StockAllocationOutputData (Delete button)
- Destination Variable:** OutputData (Edit button)
- DataTransDefnFile:** Out-Of-Stock-Setting (Delete File(X) button)
- Buttons:** OK, Cancel

Item	Value to be set	Description
Activity name	Out-Of-Stock-Setting	Enter the name of the activity.
Variable (in the Source Variables area)	StockAllocationOutputData	Select the transformation-source variable from the drop-down list, and then click the Add button.
Variable (in the Destination Variable area)	OutputData	Select the transformation-destination variable from the drop-down list.
DataTransDefnFile	Out-Of-Stock-Setting	Enter the name of the data transformation definition file to be used to transform variables.

3. Click the **OK** button.
4. Right-click the data transformation activity on the canvas, and then select **Launch mapping definition**.
The Select Root Element dialog box appears.
5. Click the root element of StockAllocationOutputData (schema logical name) for **Source**, and then, from the drop-down list, select ims:reserveItemResponse.
6. Click the root element of OutputData (schema logical name) for **Destination**, and then, from the drop-down list, select ars:arrangeItemResponse.

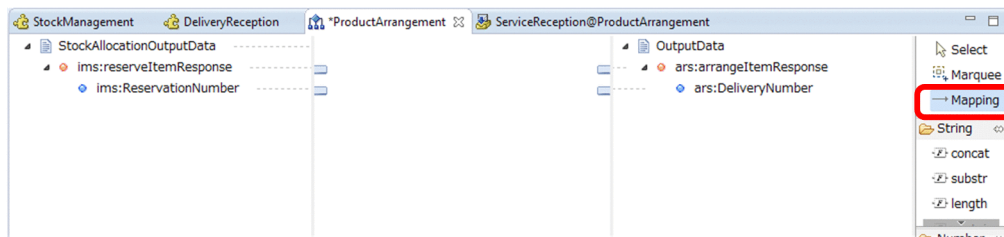
5. Experiencing the Development of Sample Programs



7. Click the **OK** button.

The data transformation definition window appears.

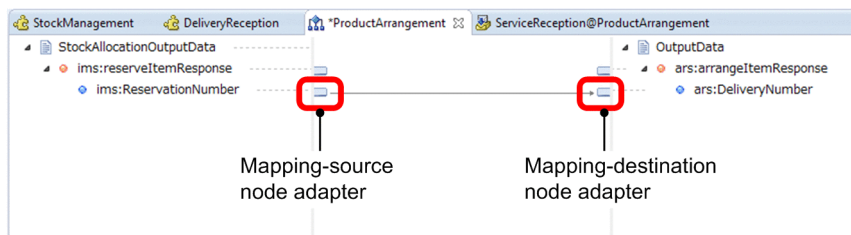
8. On the palette of the data transformation definition window, select **Mapping**.



9. Click the node adapter of the transformation-source node as the mapping source.

10. Click the node adapter of the transformation-destination node as the mapping-destination.

A mapping line is set. The correspondence between the mapping-source and mapping-destination node adapters is as follows.

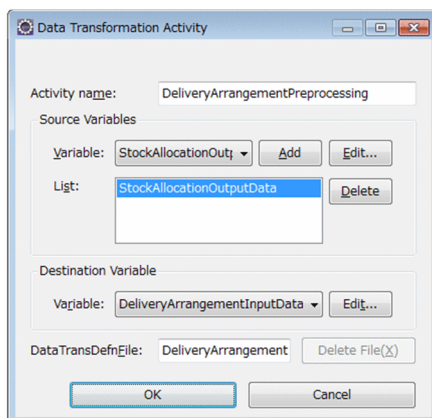


(d) Data transformation activity (for preprocessing of delivery arrangement)

1. Double-click the *data transformation* activity (DataActivity3) on the canvas.

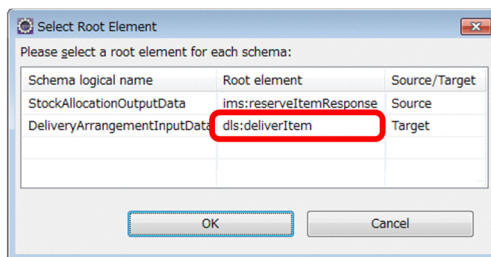
The Data Transformation Activity dialog box appears.

2. Enter information as shown in the following figure.

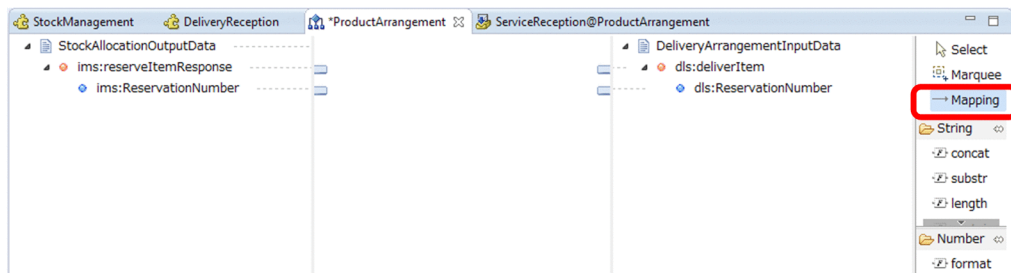


Item	Value to be set	Description
Activity name	DeliveryArrangementPreprocessing	Enter the name of the activity.
Variable (in the Source Variables area)	StockAllocationOutputData	Select the transformation-source variable from the drop-down list, and then click the Add button.
Variable (in the Destination Variable area)	DeliveryArrangementInputData	Select the transformation-destination variable from the drop-down list.
DataTransDefnFile	DeliveryArrangementPreprocessing	Enter the name of the data transformation definition file to be used to transform variables.

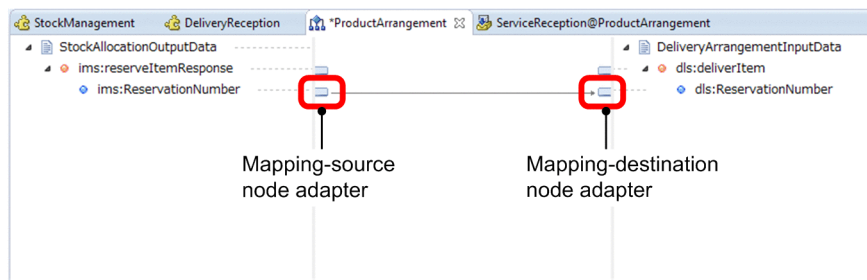
- Click the **OK** button.
- Right-click the data transformation activity on the canvas, and then select **Launch mapping definition**. The Select Root Element dialog box appears.
- Click the root element of StockAllocationOutputData (schema logical name) for **Source**, and then, from the drop-down list, select ims:reserveItemResponse.
- Click the root element of DeliveryArrangementInputData (schema logical name) for **Destination**, and then, from the drop-down list, select dls:deliverItem.



- Click the **OK** button. The data transformation definition window appears.
- On the palette of the data transformation definition window, select **Mapping**.



- Click the node adapter of the transformation-source node as the mapping source.
- Click the node adapter of the transformation-destination node as the mapping destination. A mapping line is set. The correspondence between the mapping-source and mapping destination node adapters is as follows.

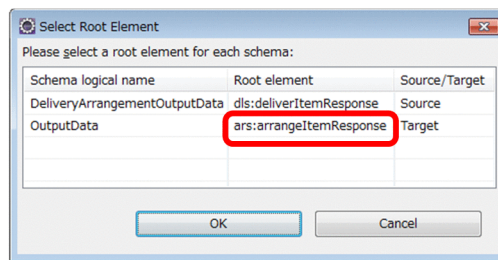


(e) Data transformation activity (for setting a delivery number)

1. Double-click the *data transformation* activity (DataActivity4) on the canvas.
The Data Transformation Activity dialog box appears.
2. Enter information as shown in the following figure.

Item	Value to be set	Description
Activity name	DeliveryNumberSetting	Enter the name of the activity.
Variable (in the Source Variables area)	DeliveryArrangementOutputData	Select the transformation-source variable from the drop-down list, and then click the Add button.
Variable (in the Destination Variable area)	OutputData	Select the transformation-destination variable from the drop-down list.
DataTransDefnFile	DeliveryNumberSetting	Enter the name of the data transformation definition file to be used to transform variables.

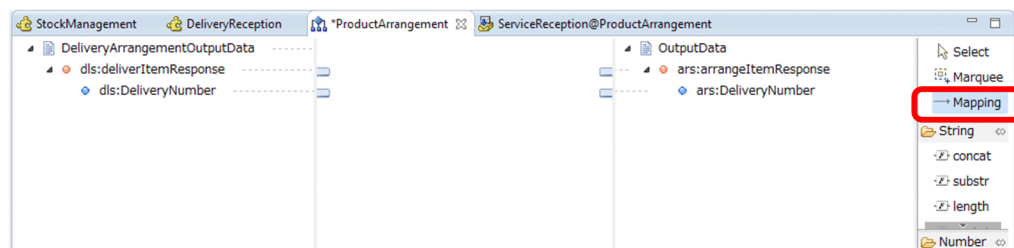
3. Click the **OK** button.
4. Right-click the data transformation activity on the canvas, and then select **Launch mapping definition**.
The Select Root Element dialog box appears.
5. Click the root element of `DeliveryArrangementOutputData` (schema logical name) for **Source**, and then, from the drop-down list, select `dls:deliverItemResponse`.
6. Click the root element of `OutputData` (schema logical name) for **Destination**, and then, from the drop-down list, select `ars:arrangeItemResponse`.



7. Click the **OK** button.

The data transformation definition window appears.

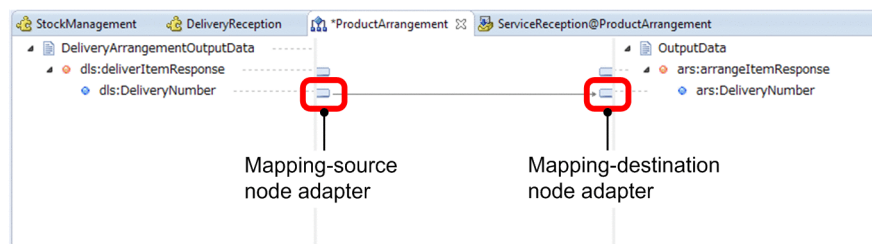
8. On the palette of the data transformation definition window, select **Mapping**.



9. Click the node adapter of the transformation-source node as the mapping source.

10. Click the node adapter of the transformation-destination node as the mapping destination.

A mapping line is set. The correspondence between the mapping-source and mapping destination node adapters is as follows.



(f) Invoke service activity (for stock allocation)

1. Double-click the *invoke service* activity (InvokeActivity1) on the canvas.

The Invoke Service Activity dialog box appears.

2. Enter information as shown in the following figure.

Item	Value to be set	Description
Activity name	StockAllocation	Enter the name of the activity.
Service name	StockManagement	From the drop-down list, select the name of the service component to be called by sending a request message.
Operation name	reserveItem	Among the operations for the service component (stock management) specified in Service name , select the name of the operation that is to be called.
Communication model	Sync	The communication model set for the operation specified in Operation name is displayed.
Body allocated variable (in the Request message area)	StockAllocationInputData	From the drop-down list, select the variable to be allocated to the body of the request message that calls the stock management service.
Header allocated variable (in the Request message area)	None	Set this item when you allocate a variable to the header of the request message that calls the stock management service. This item is not used for this sample program. Therefore, do not set this item.
Body allocated variable (in the Response message area)	StockAllocationOutputData	From the drop-down list, select the variable to be allocated to the body of the response message to be received from the synchronization operation.
Header allocated variable (in the Response message area)	None	Set this item when you allocate a variable to the header of the response message to be received from the synchronization operation. This item is not used for this sample program. Therefore, do not set this item.
Correlation set group	None	Set this item when you allocate a correlation set group to an activity. This item is not used for this sample program. Therefore, do not set this item.

3. Click the **OK** button.

(g) Invoke service activity (for delivery arrangement)

1. Double-click the *invoke service* activity (InvokeActivity2) on the canvas.
The Invoke Service Activity dialog box appears.
2. Enter information as shown in the following figure.

Item	Value to be set	Description
Activity name	DeliveryArrangement	Enter the name of the activity.
Service name	DeliveryReception	From the drop-down list, select the name of the service component to be called by sending a request message.
Operation name	deliverItem	Among the operations for the service component (delivery arrangement) specified in Service name , select the name of the operation that is to be called.
Communication model	Sync	The communication model set for the operation specified in Operation name is displayed.
Body allocated variable (in the Request message area)	DeliveryArrangementInputData	From the drop-down list, select the variable to be allocated to the body of the request message that calls the delivery arrangement service.
Header allocated variable (in the Request message area)	None	Set this item when you allocate a variable to the header of the request message that calls the delivery arrangement service. This item is not used for this sample program. Therefore, do not set this item.
Body allocated variable (in the Response message area)	DeliveryArrangementOutputData	From the drop-down list, select the variable to be allocated to the body of the response message to be received from the synchronization operation.

Item	Value to be set	Description
Header allocated variable (in the Response message area)	None	Set this item when you allocate a variable to the header of the response message to be received from the synchronization operation. This item is not used for this sample program. Therefore, do not set this item.
Correlation set group	None	Set this item when you allocate a correlation set group to an activity. This item is not used for this sample program. Therefore, do not set this item.

3. Click the **OK** button.

(h) Reply activity (if the product is out of stock)

1. Double-click the *reply* activity (**ReplyActivity1**) on the canvas.

The Reply Activity dialog box appears.

2. Enter information as shown in the following figure.

Item	Value to be set	Description
Activity name	Reply_Out-Of-Stock-Error	Enter the name of the activity.
Operation name	arrangeItem	Specify the name of the operation specified for the corresponding receive activity.
Body allocated variable	OutputData	From the drop-down list, select the variable to be allocated to the body of the response message for the business process.
Header allocated variable	None	Set this item when you allocate a variable to the header of the response message for the business process. This item is not used for this sample program. Therefore, do not set this item.
Correlation set group	None	Enter this item when you allocate a correlation set group to an activity. This item is not used for this sample program. Therefore, do not set this item.
Fault name	None	Define the reply activity as fault processing, and then specify the fault name to be used when a response message

Item	Value to be set	Description
Fault name	None	that indicates that a fault occurred in the service requester is received. No fault processing is used for this sample program. Therefore, do not set this item.

3. Click the **OK** button.

(i) Reply activity (in the case where delivery arrangement is successful)

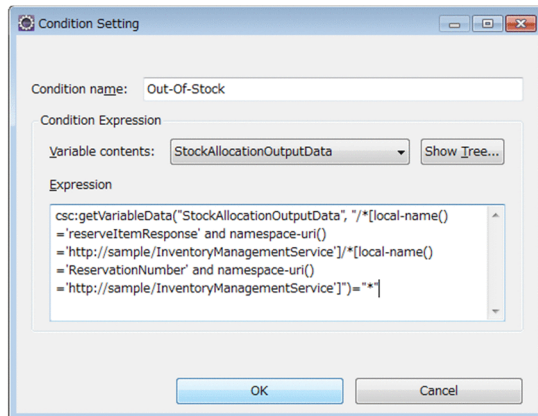
1. Double-click the *reply* activity (ReplyActivity2) on the canvas.
The Reply Activity dialog box appears.
2. Enter information as shown in the following figure.

Item	Value to be set	Description
Activity name	Reply_Arrangement-Success	Enter the name of the activity.
Operation name	arrangeItem	Specify the name of the operation specified for the corresponding receive activity.
Body allocated variable	OutputData	From the drop-down list, select the variable to be allocated to the body of the response message for the business process.
Header allocated variable	None	Set this item when you allocate a variable to the header of the response message for the business process. This item is not used for this sample program. Therefore, do not set this item.
Correlation set group	None	Enter this item when you allocate a correlation set group to an activity. This item is not used for this sample program. Therefore, do not set this item.
Fault name	None	Define the reply activity as fault processing, and then specify the fault name to be used when a response message that indicates that a fault occurred in the service requester is received. No fault processing is used for this sample program. Therefore, do not set this item.

3. Click the **OK** button.

(j) Switch start activity

1. Double-click the *switch start* activity (**SwitchStartActivity1**) on the canvas.
The Switch Activity dialog box appears.
2. Enter **CheckStockAllocationResult** as the activity name.
3. Click the line on which the value of the **Transition destination** column is **Out-Of-Stock-Setting**. Then, click the **To Upper** button to move the line to the top line.
4. Select the top line, and then click the **Condition Setting...** button.
The Set Condition dialog box appears.
5. Enter information as shown in the following figure.

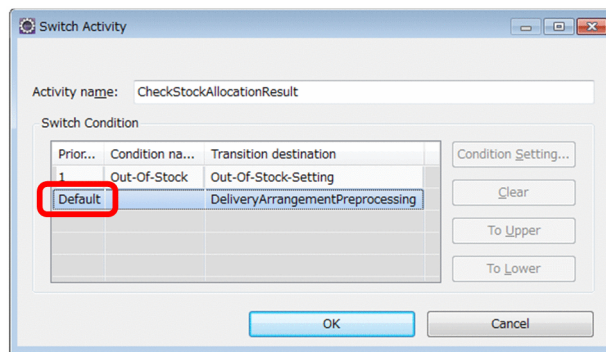


Item	Value to be set	Description
Condition name	Out-Of-Stock	Specify the condition for determining whether the product is out of stock as a result of a stock allocation check.
Variable contents	StockAllocationOutputData	From the drop-down list, select the variable to be used in the condition expression.
Condition expression	csc:getVariableData("StockAllocationOutputData", "/*[local-name() = 'reserveItemResponse' and namespace-uri() = 'http://sample/InventoryManagementService']/*[local-name() = 'ReservationNumber' and namespace-uri() = 'http://sample/InventoryManagementService']")= "*"	Specify the condition expression for determining whether the product is out of stock in XPath expression format.

Note:

Line breaks are not applied to the displayed condition expression.

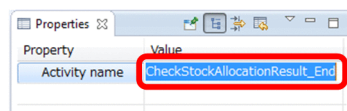
6. Click the **OK** button.
The Switch Activity dialog box appears again.
7. On the line on which the value of the **Transition destination** column is **DeliveryArrangementPreprocessing**, click **Priority**, and then select **Default** from the drop-down list.
If you select **Default**, you do not need to specify the condition settings.



8. Click the **OK** button.

(k) Switch end activity

1. Double-click the *switch end* activity (SwitchEndActivity1) on the canvas. The details of the *switch end* activity are displayed in the properties view.
2. Enter CheckStockAllocationResult_End as the activity name.



3. When you have defined all activities, from the menu, select **File** and then **Save**. The business process is now defined.

5.5.4 Validating and packaging components

When you have created a component, validate that it is defined correctly, and then package it. The validation and packaging targets are the **Stock Management** service adapter, **Delivery Reception** service adapter, and **Product Arrangement** business process in the tree view.

The validation and packaging procedures are the same as those for a Hello service adapter. For details about validation and packaging, see 5.3.2 *Validating and packaging a service adapter*.

5.5.5 Defining deployment of components

When you have packaged a component, define its deployment. The deployment procedure is the same as that for a Hello service adapter. For details about deployment definitions, see 5.3.3 *Defining deployment of a service adapter*.

5.6 Debugging the product arrangement system

In this section, debug the business process that was defined in 5.5.3 *Defining the product arrangement business process*, and then check how the business process will be processed.

The following is an overview of debugging the business process described in this section:

1. Set a break point in an activity.
2. Deploy the service requester.
3. Start the HCSC server.
4. Start the debugger.
5. Send a request from the service requester to start the processing of the process instance.
6. Debug each activity.
7. To debug without calling services, use service emulation.
8. Finish debugging the business process.

(1) Setting a break point

Set a break point in the activity at which to stop processing of the process instance.

In this example, set a break point in the *invoke service* activity (**StockAllocation**).

The following is the procedure for setting a break point.

1. On the canvas of the business process definition window, right-click the *invoke service* activity (**StockAllocation**), and then select **Add Breakpoint**.

A break point is added to the activity. As shown in the following figure, a check mark indicating a break point appears beside the activity icon.



(2) Deploying the service requester

To send a request, import the Eclipse project of the `HelloProductArrangement` sample program, and then deploy the service requester on the J2EE server.

The following is the procedure for deploying the service requester.

1. Stop the HCSC server.

From the **Start** menu, select **Programs, Cosminexus[#]**, and then **Stop Test Server** to stop Performance Tracer, J2EE server, and HCSC server (including the standard reception and user-defined reception) in the test environment.

#

This program folder name might have been changed. If it was changed, select the changed program folder name.

2. From the Eclipse menu, select **Window, Open Perspective**, and then **Other**.

The Open Perspective dialog box opens.

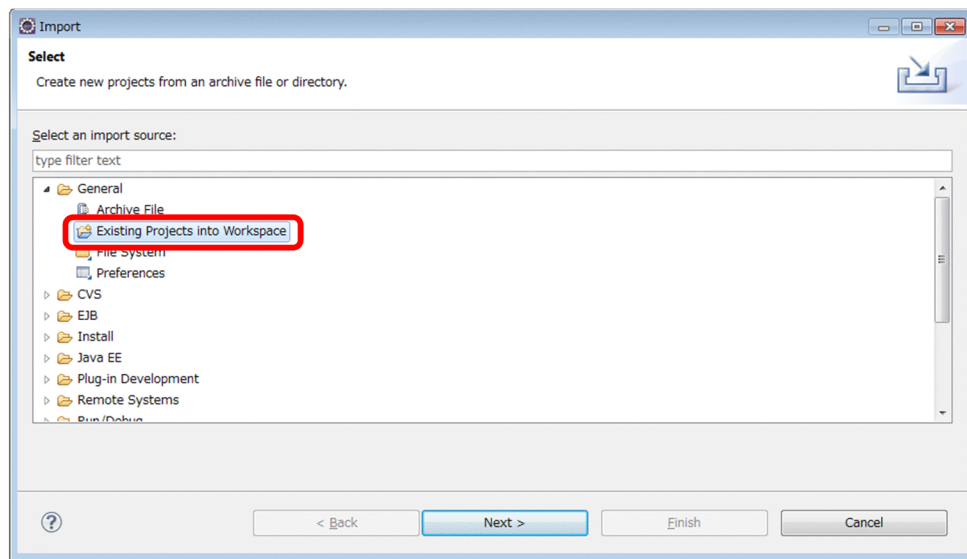
3. Select **Java EE (default)**, and then click the **OK** button.

The Java EE perspective opens.

4. From the menu, select **File**, and then **Import**.

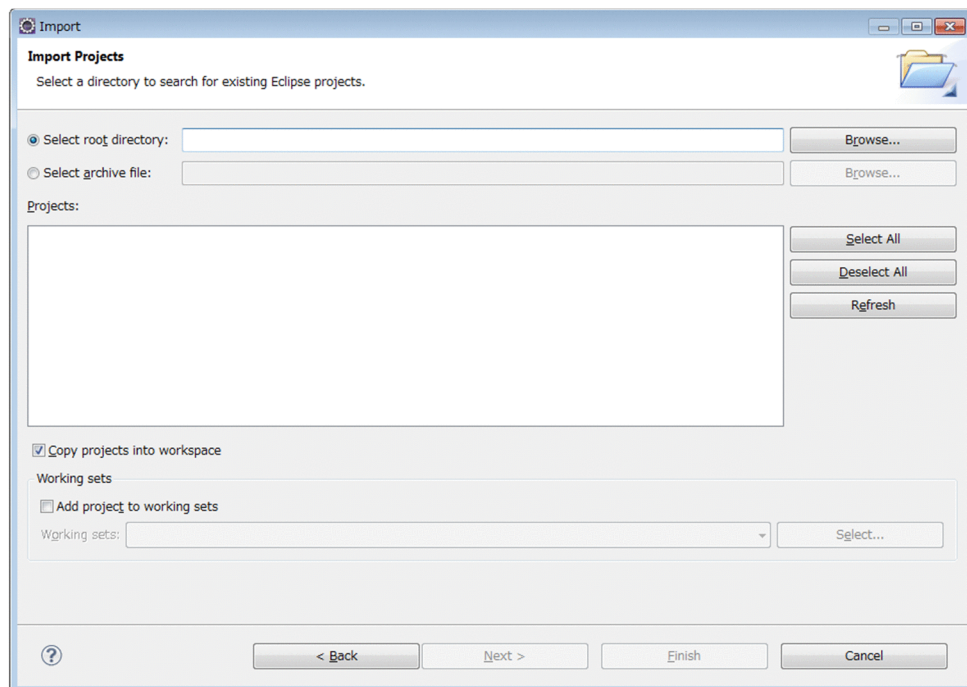
The **Select** page appears.

5. In the tree view, select **General**, and then **Existing Projects into Workspace**.



6. Click the **Next** button.

The **Import Projects** page appears.



7. Select the **Select root directory** radio button, and then click the **Browse** button.

The dialog box for selecting a directory opens.

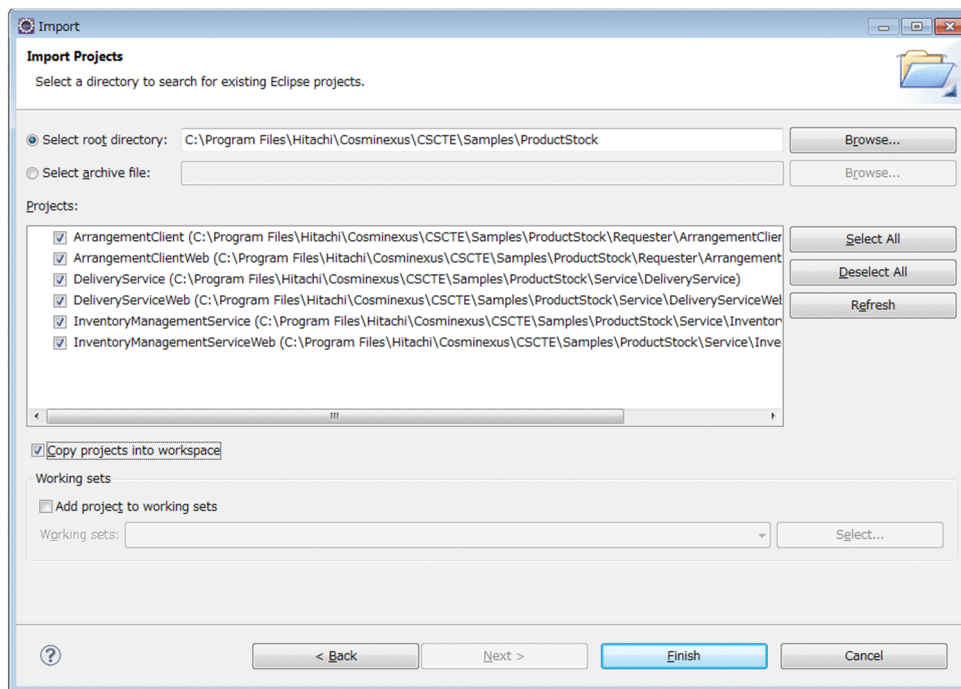
8. Select the directory that contains the target sample program, and then click the **OK** button. In this example, the directory to select is as follows: *service-platform-installation-directory*\CSCTE\Samples\ProductStock

The **Import Projects** page appears again. Make sure that the check boxes of all projects are selected.

9. If the **Copy projects into workspace** check box is not selected, select it.

If you do not select this check box, the sample programs might be deleted. Make sure that this check box is selected.

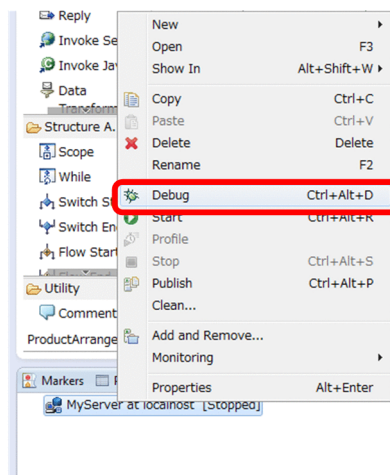
5. Experiencing the Development of Sample Programs



10. Click the **Finish** button.

The selected projects are imported into the workspace.

11. In the **Servers** view, right-click **MyServer at localhost**, and then select **Debug**.

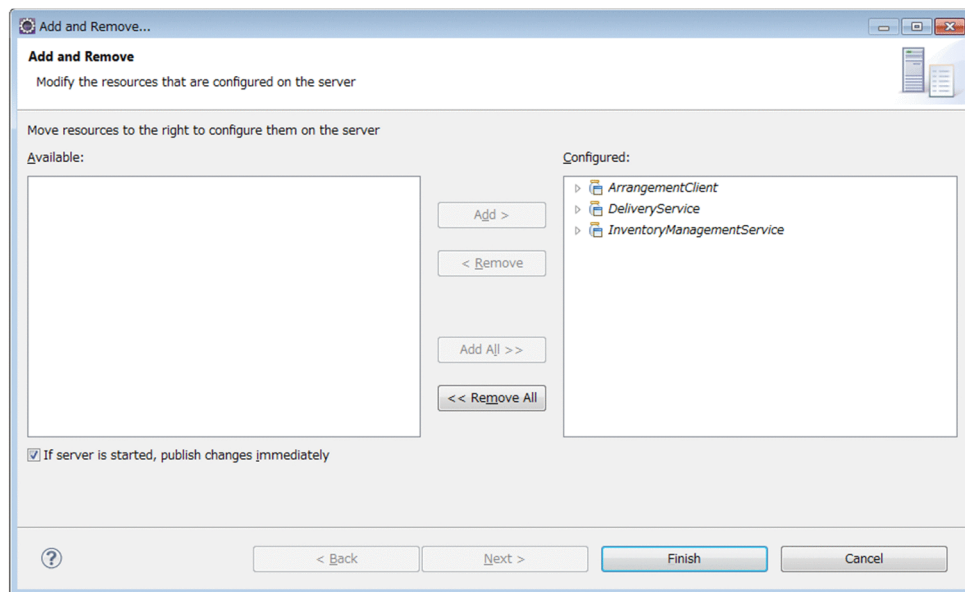


A message indicating that processing is in progress appears, and then the server starts. If no server is displayed, see 3.5.4(3) *Selecting the J2EE server*.

12. In the **Servers** view, right-click **MyServer at localhost**, and then select **Add and Remove**.

The Add and Remove dialog box appears.

13. In the **Available** list box, select **ArrangementClient**, **DeliveryService**, and **InventoryManagementService**, and then click the **Add** button.



14. Click the **Finish** button.

A message indicating that processing is in progress appears. After processing finishes, the Java EE perspective appears again.

When you have completed the setting, confirm that the projects `ArrangementClient`, `DeliveryService`, and `InventoryManagementService` are displayed under **MyServer at localhost** in the **Servers** view.

(3) Starting the HCSC server

Before starting the debugger, start the HCSC server.

1. Start the HCSC server.

From the **Start** menu, select **Programs, Cosminexus[#]**, and then **Start Test Server** to start Performance Tracer, J2EE server, and HCSC server (including the standard reception and user-defined reception) in the test environment.

#

This program folder name might have been changed. If it was changed, select the changed program folder name.

(4) Starting the debugger

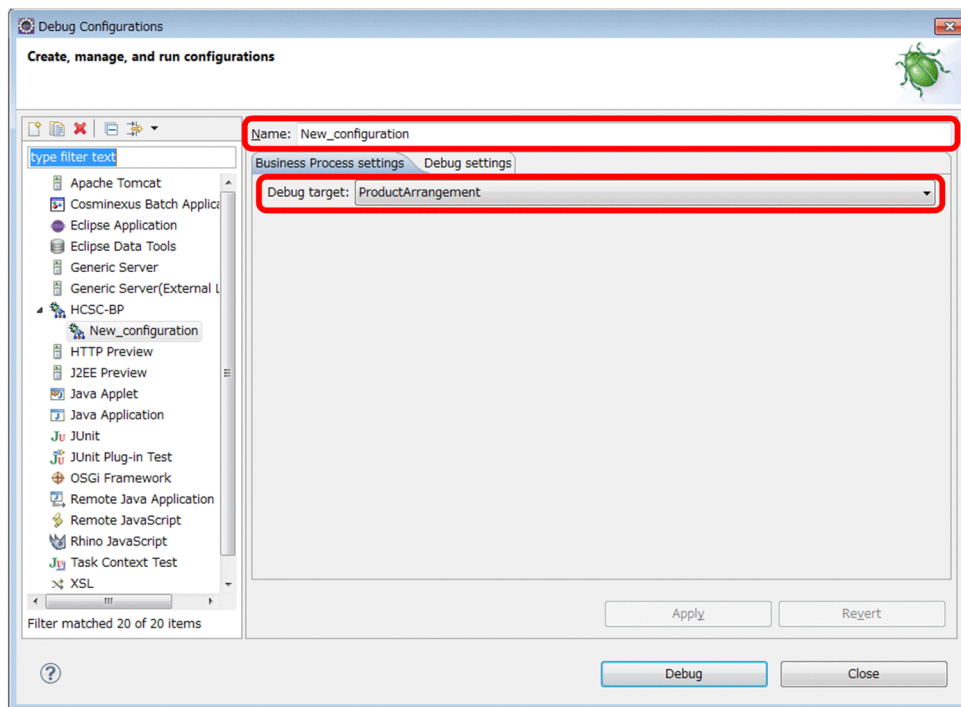
The following is the procedure for starting the debugger.

1. From the Eclipse menu, select **Run**, and then **Debug Configurations**.

The Debug Configurations dialog box appears.

2. In the menu of the Debug Configurations dialog box, right-click **HCSC-BP**, and then select **New**.

3. Enter any name in **Name**, and then select **ProductArrangement** from the **Debug target** drop-down list.



4. Click the **Debug** button.

The account authentication window appears.

5. Enter **admin** in both **User ID** and **Password**, and then click the **OK** button.

A message indicating that processing is in progress appears. When you start the debugger again, the account authentication window does not appear, but processing immediately starts.

6. Click the **OK** button.

The debugger starts.

(5) Sending a request

Send a request from the service requester to start the processing of the process instance.

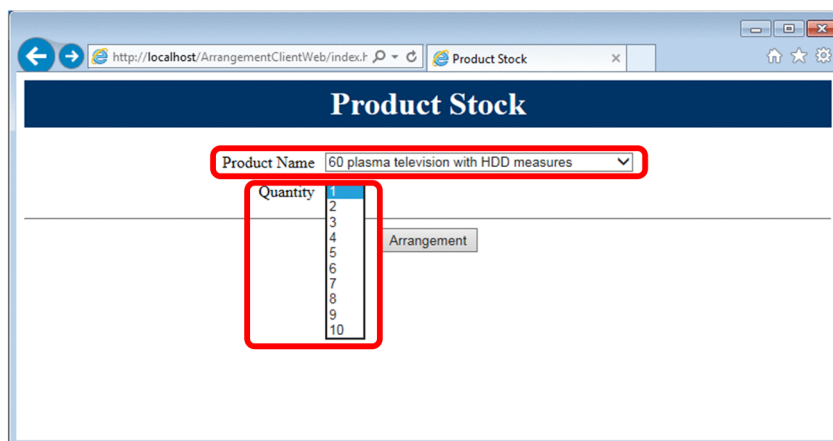
The following is the procedure for sending a request.

1. Access the following URL with a browser:

<http://localhost/ArrangementClientWeb/index.html>

The window for running the HelloProductArrangement sample program opens.

2. Select the product name and quantity.



3. Click the **Arrangement** button.

The dialog box asking you whether to switch the perspective appears.

4. Click the **Yes** button.

The view that displays debugging information and variables appears, and the processing of the process instance starts.

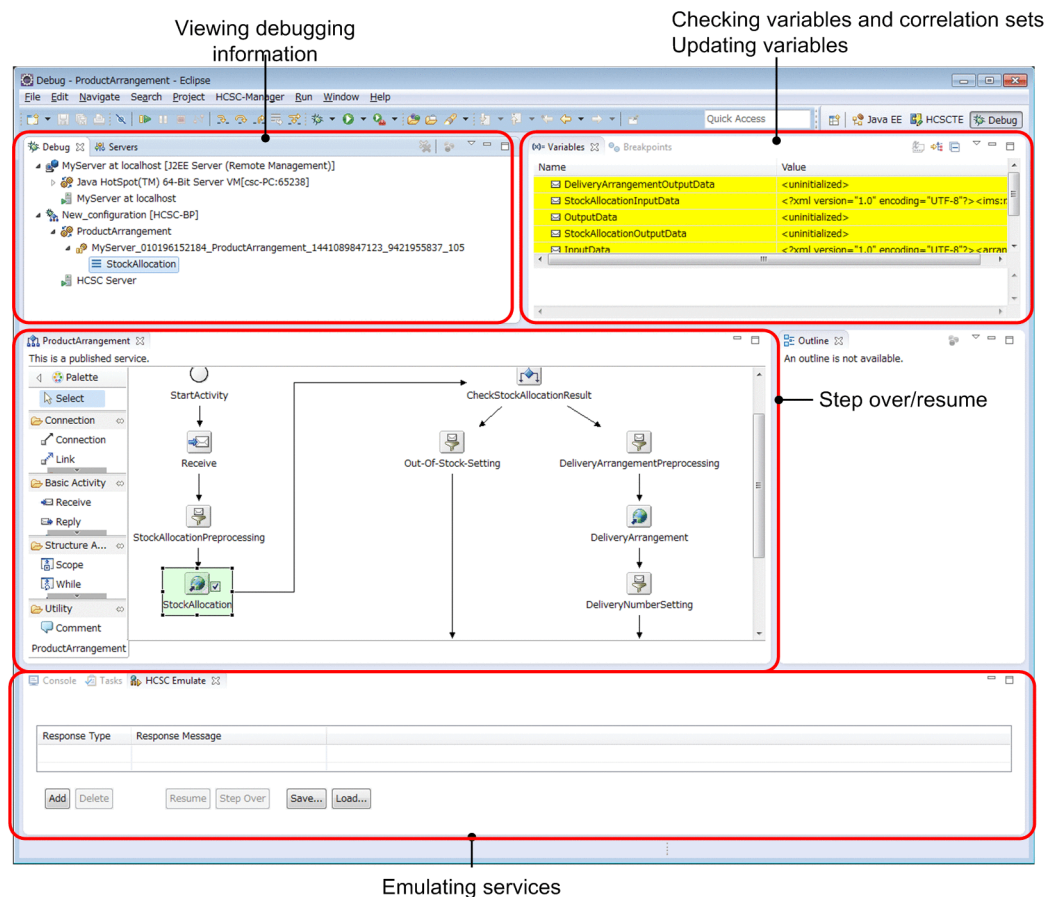
The processing of the process instance stops at the *invoke service* activity (**StockAllocation**). On the canvas of the business process definition window, the icon color of the activity changes as shown as follows to indicate that processing is temporarily stopped.



(6) Debugging the business process

If processing of the process instance is temporarily stopped, you can debug in the following window.





Figure 5-2: Business process debugging window



The following table shows the operations that can be performed during debugging of a business process.

Table 5-11: Operations that can be performed during debugging of a business process

Item	Description
Check variables and correlation sets	You can check the variables and correlation sets that are currently used in the business process.
Update variables	You can update the values of the variables that are currently used in the business process.

Item	Description
Step over and resume	<p>In the Debug view, you can perform the operations below. Use these operations to debug each activity.</p> <ul style="list-style-type: none"> • Step-in () Moves processing to the next activity. If the step-in operation is performed for a <i>switch start</i> activity, processing moves to the first-branch activity. • Step-over () If the step-over operation is performed for a <i>switch start</i> activity, the activities up to the corresponding <i>switch end</i> activity are processed at one time. If the step-over operation is performed for an activity other than a <i>switch start</i> activity, the same operation as the step-in operation is performed. • Step-return () If the step-return operation is performed for an activity that is subordinate to a <i>scope</i> activity or <i>while</i> activity, the activities up to the next activity to that <i>scope</i> or <i>while</i> activity are processed. • Resume () Resumes the processing of the process instance up to the activity at which the next break point is set.
Emulate services	<p>You can emulate services by using already-created response messages.</p> <p>For details about the procedure for emulating services, see (7) <i>Service emulation</i>.</p>

(7) Service emulation

To debug without calling services, enable service emulation. If you enable service emulation, you can use already-created response messages instead of calling actual services.

Service emulation allows you to execute a business process even when services called by the business process do not exist.

You can enable service emulation while the processing of the process instance is temporarily stopped. This subsection describes the procedure for emulating the StockAllocation service on the assumption that the processing of the process instance is temporarily stopped at an *invoke service* activity (**StockAllocation**).

The following is the procedure for emulating the StockAllocation service.


1. Create an XML file.
2. Display the HCSC emulation view.
3. Use the created XML file to execute service emulation.

Before you execute service emulation, you must start the debugger and send a request from the service requester.

(a) Creating an XML file

Create an XML file that will be used as a service response.

The following shows how to create an XML file.

1. From the Eclipse menu, select **File**, **New**, and then **Other**. In the dialog box that appears, select **General**, and then **File**.
The New File dialog box appears.
2. In the New File dialog box, select the directory in which to save the XML file.
3. In **File name**, enter any file name (with a file name extension of `.xml`).
4. Click the **Finish** button.
An XML editor appears.
5. In the XML editor, click the **Source** tab, and then enter the following code ( : single-byte space):

```
<?xml version="1.0" encoding="UTF-8"?><reserveItemResponse xmlns="http://
sample/InventoryManagementService"><ReservationNumber>R00000001</
ReservationNumber></reserveItemResponse>
```

- From the menu, select **File**, and then **Save**.

The XML file is saved.

(b) Displaying the HCSC emulation view

You can emulate services in the HCSC emulation view.

The following is the procedure for displaying the HCSC emulation view.

- From the Eclipse menu, select **Window**, **Show View**, and then **Other**.
The Show View dialog box appears.
- Select **Debug**, and then **HCSC Emulate**. Then, click the **OK** button.
The HCSC emulation view appears.

(c) Emulating a service

- In the **Debug** view, select **Stock Allocation** (*invoke service* activity).
- In the HCSC emulation view, click the **Add** button.
A line is added to the table in the HCSC emulation view.
- Select the **Response Type** cell, and then select **Normal Response**.
- Select the **Response Message** cell, and then click the ... button.
The **Select files** dialog box appears.
- Select the created XML file, and then click the **OK** button.
- Click the **Step Over** button.
The response from the *invoke service* activity (**StockAllocation**) is emulated.
The processing of the process instance proceeds to the *switch start* activity (**CheckStockAllocationResult**).

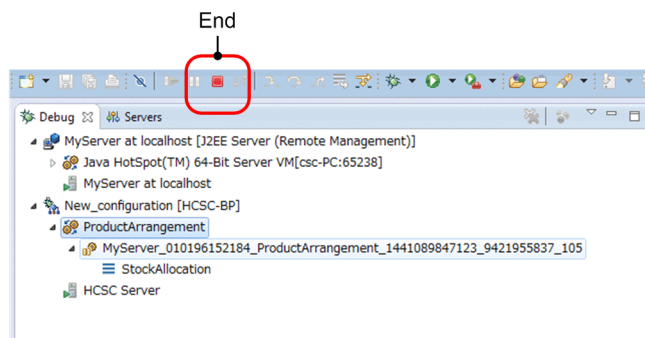
To proceed to the subsequent activities, use the **Debug** view. For details, see (6) *Debugging the business process*.

(8) Finishing the debugging the business process

Finish debugging the business process.

The following is the procedure for finishing the debugging of the business process.

- In the **Debug** view, select **ProductArrangement**, and then click the **Finish** icon.



Debugging of the business process finishes.

5.7 Preparing for running the developed sample program

In this section, validate operation of the developed sample program by using the sample service requester and service component provided by Service Architect.

To verify operation of the sample program, first, perform the procedures in *3.5.5 Importing Eclipse projects* and *3.5.6 Deploying the web project*. Then, execute the sample program by following the procedure in *Chapter 4. Executing Sample Programs*.

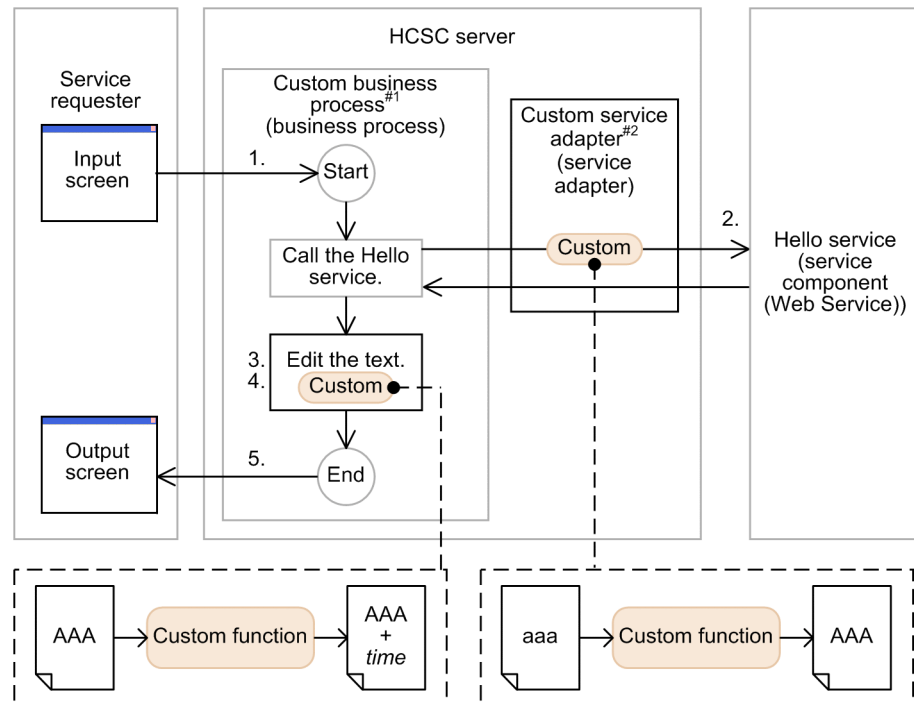
5.8 Defining data transformation by using a Java program

In this section, use the CustomFunction sample program to develop a program that uses a Java program.

Development with the CustomFunction sample program uses a custom function. The custom function defines data transformations by calling a Java program in which certain data processing is defined.

The following figure shows an overview of the processing of the CustomFunction sample programs.

Figure 5-3: Processing of the CustomFunction sample program



Legend:

→ : Flow of tasks for a request or response that calls a service component

Custom : Custom function





[] : Processing by the custom function

#1: Created by partially changing the Hello business process

#2: Created by partially changing the Hello service adapter

The CustomFunction sample program executes processing as follows:

1. The Custom business process receives the character string entered from the service requester.
2. The business process calls the Hello service via the Hello service adapter. At this time, the custom function is used to convert all lowercase characters to uppercase characters.
3. When the Hello service is called, the following character string is concatenated to the received one: **△** and **△**
Business **△** Process (**△** : single-byte space)
4. In addition, the custom function is used to add the execution date and time to the end of the character string (in yyyy/MM/dd HH:mm:ss.SSS format).

```
Generated string: uppercase-string-output-by-Hello-service  and  Business   
Process  yyyy/MM/dd HH:mm:ss.SSS
```

5. The concatenation result is returned to the service requester, and then it is displayed in the output window.

5.8.1 Overview of defining the CustomFunction sample program

The following describes the general procedure for defining the CustomFunction sample program.

Note that you create the CustomFunction sample program by partially modifying the Hello service adapter and Hello business process. Therefore, you must create the Hello service adapter and Hello business process before creating the CustomFunction sample program. For details about how to create the Hello service adapter and Hello business process, see [5.3 Defining service adapters](#) and [5.4 Defining business processes](#).

Define the CustomFunction sample program as follows:

1. Prepare the custom function.
2. Change the Hello service adapter.
3. Change the Hello business process.

The following subsections describe the stages of developing the CustomFunction sample program above.

5.8.2 Preparing the custom function

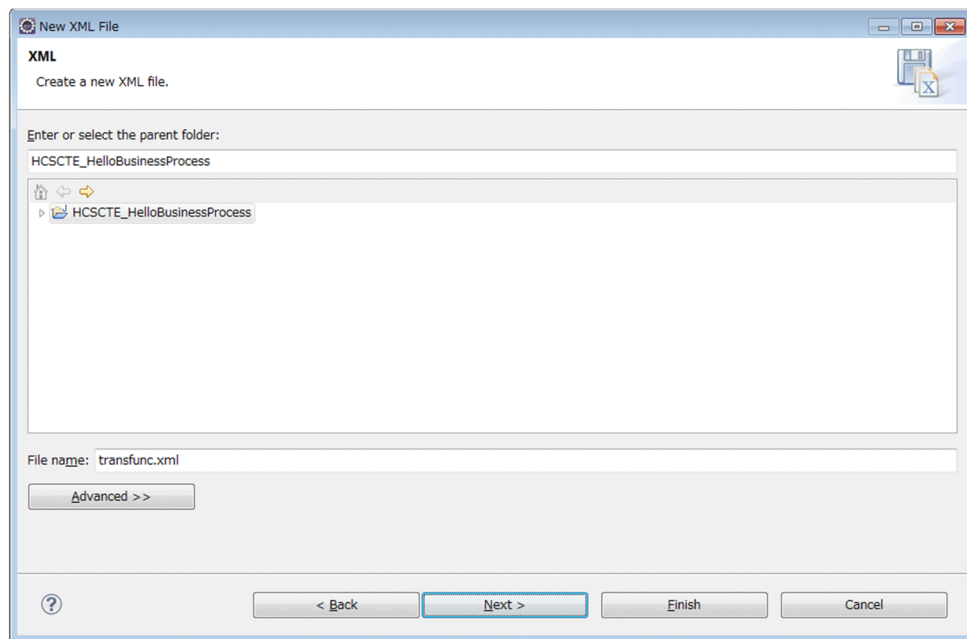
If you choose to use the custom function, create the following two items, which are used for the custom function:

- Transformation function definition file that defines the configuration of the Java program to be called
- Java program

The following describes the procedure for creating the items above.

(1) Creating the transformation function definition file

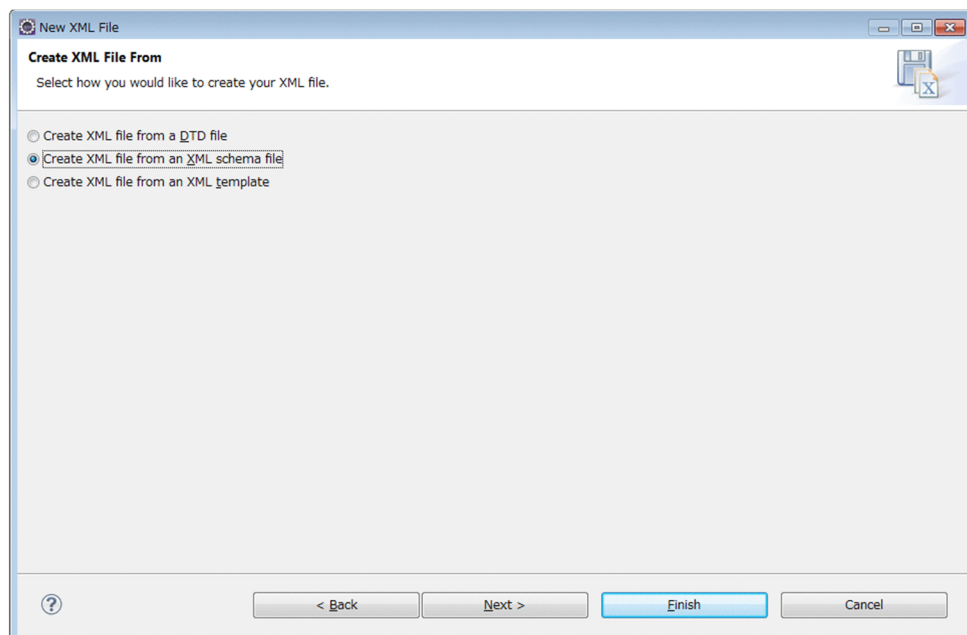
1. From the Eclipse menu, select **File**, **New**, and then **Other**.
The New dialog box appears.
2. Select **XML**, and then **XML File**. Then, click the **Next** button.
The **XML** page appears.



3. Specify the directory that will contain the transformation function definition file and the desired transformation function definition file. Then, click the **Next** button.

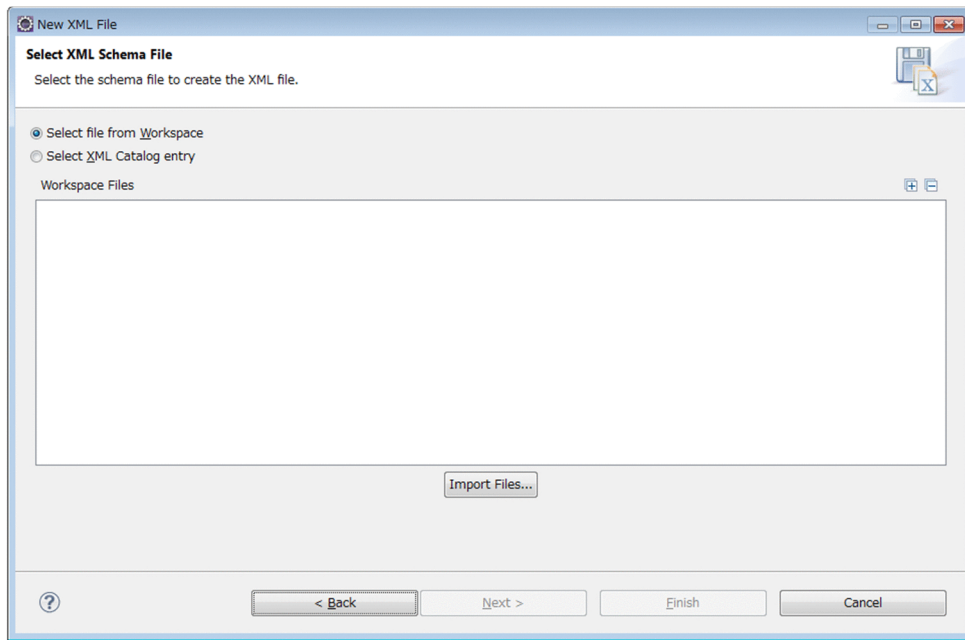
In this example, the directory that will store the transformation function definition file is HCSCTE_HelloBusinessProcess (HCSCTE project folder), and the transformation function definition file to be used is transfunc.xml.

The **Create XML File From** page appears.



4. Select the **Create XML file from an XML schema file** radio button, and then click the **Next** button. The **Select XML Schema File** page appears.

5. Experiencing the Development of Sample Programs

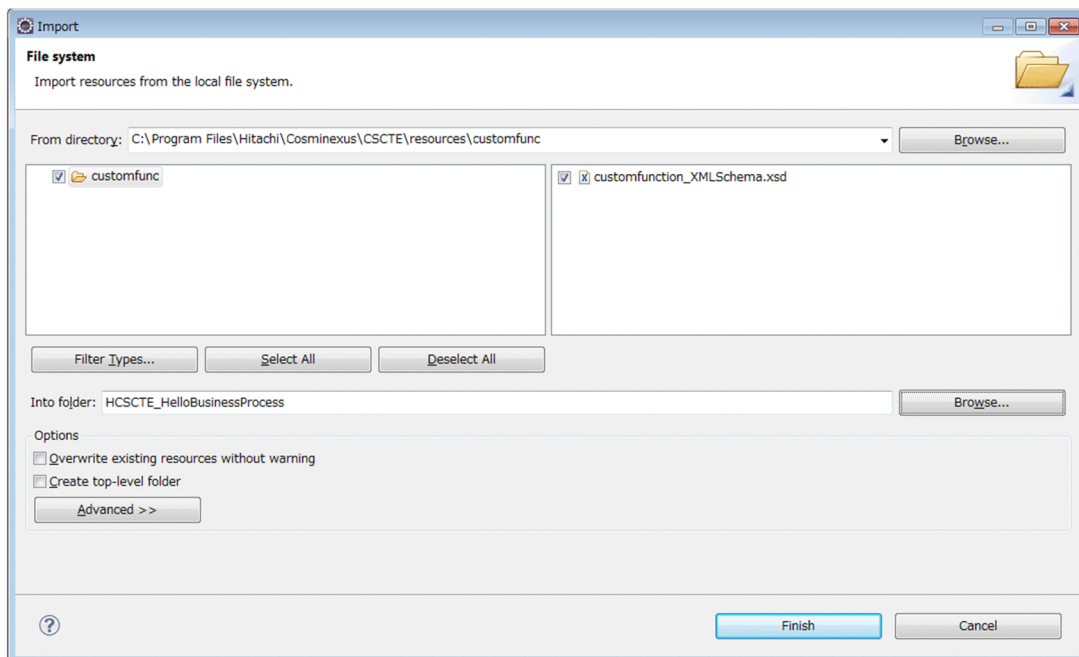


5. Click the **Import Files** button.

The Import dialog box appears.

6. For **From directory**, specify the following directory:

service-platform-installation-directory\CSCTE\resources\customfunc



7. Select the check boxes of **customfunc** and **customfunction_XMLSchema.xsd**.

8. For **Into folder**, specify the folder to which the schema file will be imported.

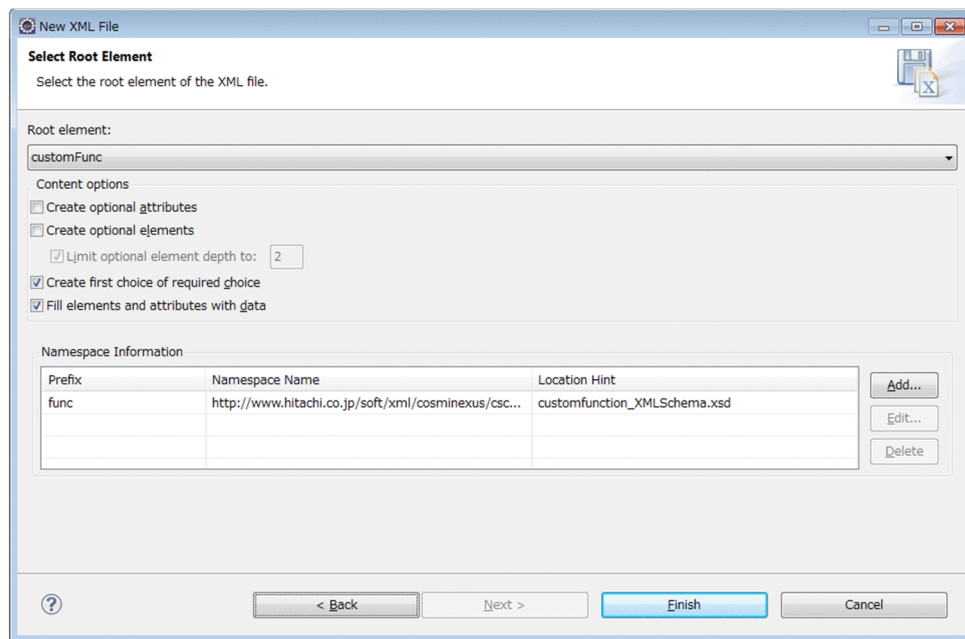
In this example, specify **HCSCTE_HelloBusinessProcess**.

9. Click the **Finish** button.

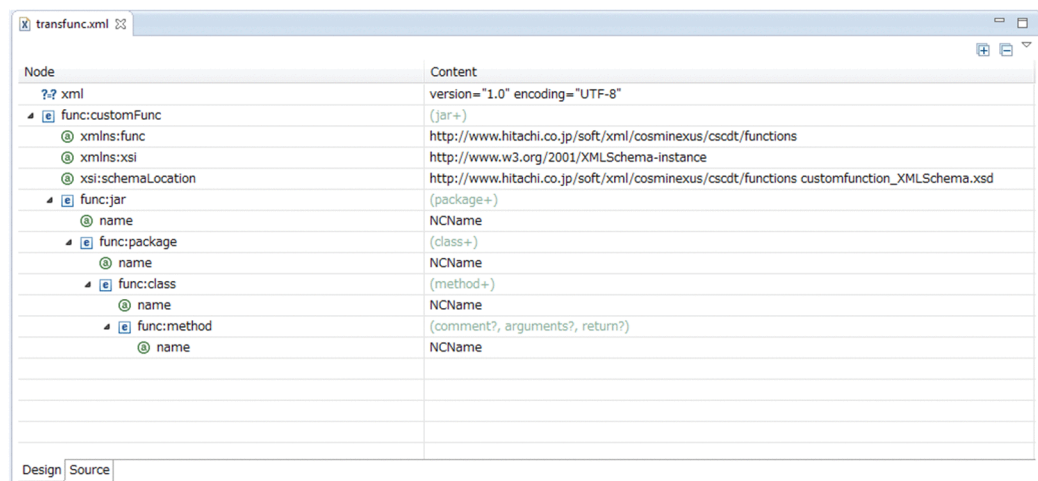
The **Select XML Schema File** page appears again.

10. Click the **Next** button.

The **Select Root Element** page appears.



11. From **Root element** drop-down list, select **customFunc**, and then click the **Finish** button.
A model file of the transformation function definition file is generated.



12. Define necessary information for the model transformation function definition file.
To add an element, select and right-click the parent item in the node column. Select the position at which to add the element.
You can edit the information of the element in the content column.
The following shows the information to be defined.

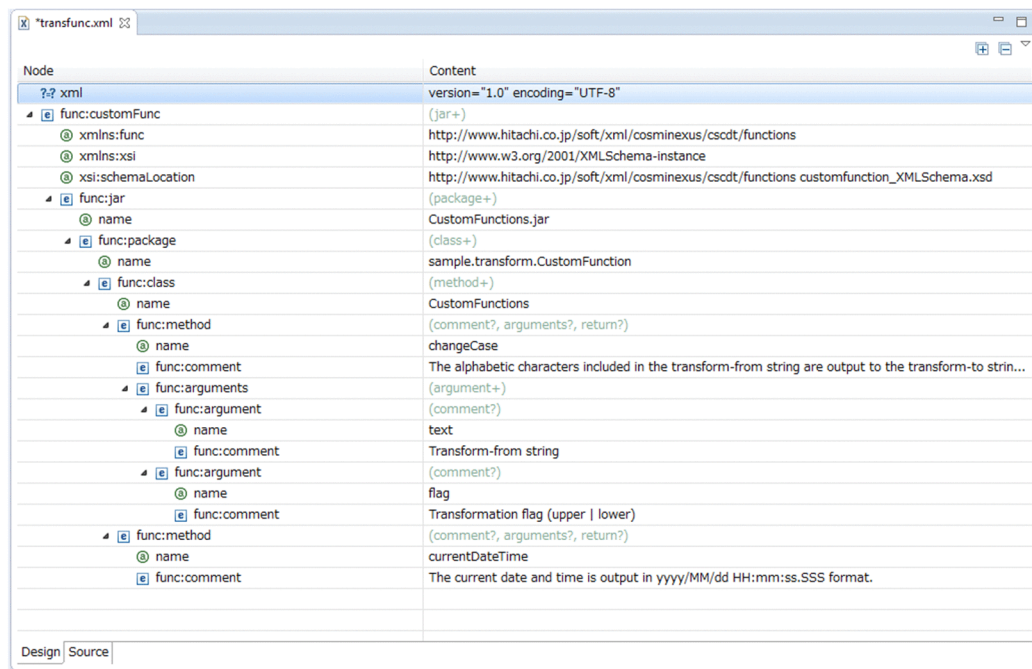


Table 5–12: Information to be defined in the transformation function definition file

Element	Information to be defined
xml	version="1.0" encoding="UTF-8"
func:customFunc	--
xmlns:func	http://www.hitachi.co.jp/soft/xml/cosminexus/cscdt/functions
xmlns:xsi	http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation	http://www.hitachi.co.jp/soft/xml/cosminexus/cscdt/functions customfunction_XMLSchema.xsd
func:jar	--
name	CustomFunctions.jar
func:package	--
name	sample.transform.CustomFunction
func:class	--
name	CustomFunctions
func:method	--
name	changeCase
func:comment	The alphabetic characters included in the transform-from string are output to the transform-to string by changing all lowercase letters to uppercase letters, or vise versa, according to the specified transformation flag.
func:arguments	--
func:argument	--
name	text
func:comment	Transform-from string

Element	Information to be defined
func:argument	--
name	flag
func:comment	Transformation flag (upper lower)
func:method	--
name	currentDateTime
func:comment	The current date and time is output in <i>yyyy/MM/dd HH:mm:ss.SSS</i> format.

Legend:

--: Nothing needs to be defined.

13. From the Eclipse menu, select **File**, and then **Save**.

The transformation function definition file has now been created.

(2) Creating a Java program

In the CustomFunction sample program, you create a custom function that calls the following Java program.

Figure 5–4: Java program called by the custom function

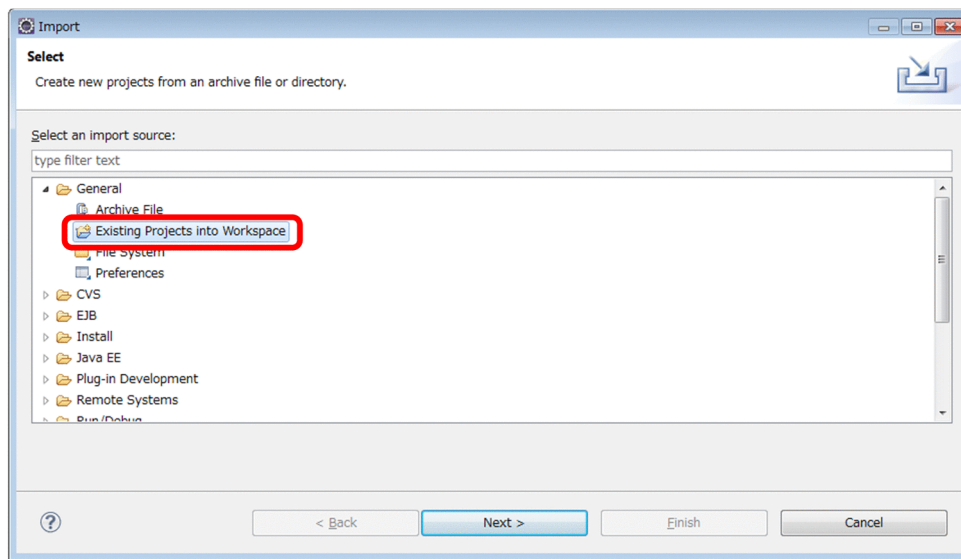
CustomFunctions.jar	← JAR file name
sample.transform.CustomFunction	← Package name
CustomFunctions	← Class name
String changeCase	← Method that converts uppercase letters in the string to lowercase letters, or vise versa, according to which of the following flags is specified: upper: Converts lowercase letters to uppercase letters. lower: Converts uppercase letters to lowercase letters.
String currentDateTime	← Method that outputs the current date and time in the following format: <i>yyyy/MM/dd HH:mm:ss.SSS</i>

The CustomFunction sample program provides a source file (CustomFunctions.java) that contains the Java code above. Import and package this source file to create a Java program to be called by the custom function.

The following is the procedure for creating a Java program.

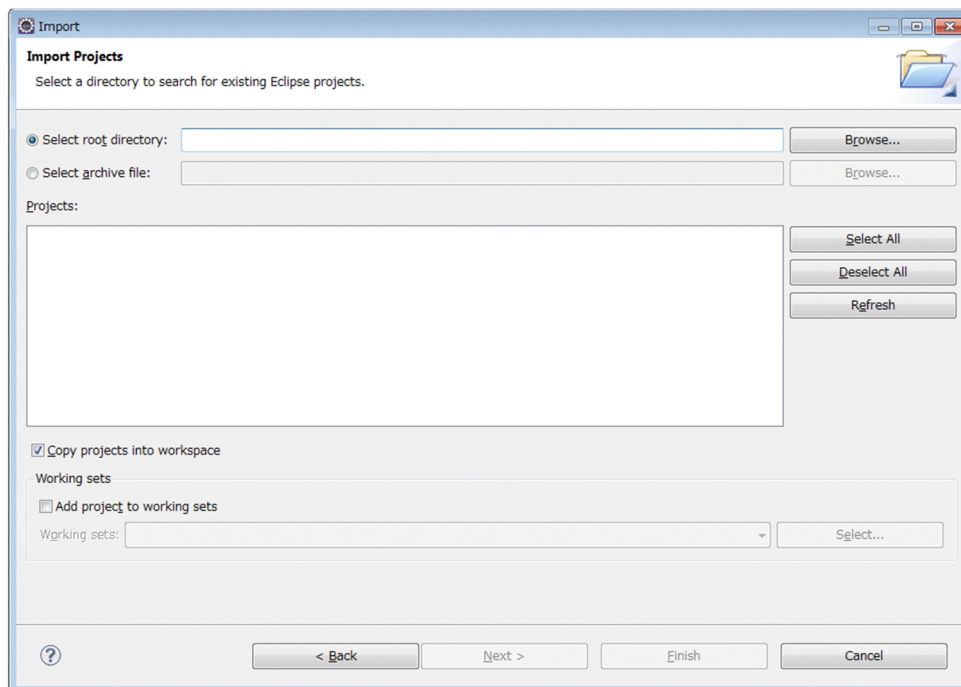
1. From the Eclipse menu, select **Window**, **Open Perspective**, and then **Other**.
The Open Perspective dialog box opens.
2. Select **Java**, and then click the **OK** button.
The Java perspective opens.
3. From the Eclipse menu, select **File**, and then **Import**.
The **Select** page appears.
4. In the tree view, select **General**, and then **Existing Projects into Workspace**.

5. Experiencing the Development of Sample Programs



5. Click the **Next** button.

The **Import Projects** page appears.



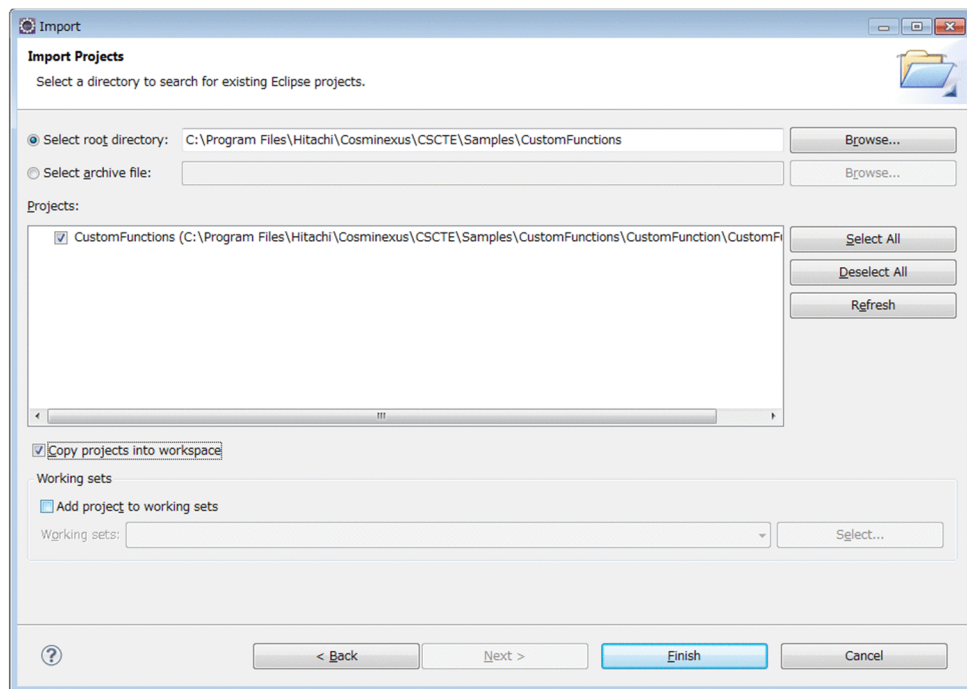
6. Select the **Select root directory** radio button, and then click the **Browse** button.

The dialog box for selecting a directory opens.

7. Select the directory that contains the CustomFunction sample program, and then click the OK button. In this example, the directory to select is as follows: *service-platform-installation-directory*\CSCTE\Samples\CustomFunctions\CustomFunction\CustomFunctions

The **Import Projects** page appears again.

8. Select the **Copy projects into workspace** check box.



9. Click the **Finish** button.

Import of the Java program is completed.

10. In the **Package Explorer** view, under the **CustomFunctions** project, right-click **build.xml**. Then, select **Run**, and then **Ant Build**.

The Java program is packaged, and then a JAR file (`CustomFunctions.jar`) is created in the following directory:

`eclipse-workspace-directory\CustomFunctions\build\lib`

If **Ant Build** does not appear, from the Eclipse menu, select **Run**, **External Tools**, and then **Open External Tools Dialog**. In the dialog box that appears, specify the **Ant Build** settings.

11. Copy the created JAR file to the following location:

`service-platform-installation-directory\CSC\userlib\customfunc`

The Java program has now been created.

5.8.3 Modifying the Hello service adapter

The `CustomFunction` sample program changes the case of alphabetic data to uppercase when calling a service via the service adapter. To implement this processing, you need to define data transformation in the Hello service adapter by modifying the definition of the Hello service adapter.

The following is the procedure for modifying the Hello service adapter.

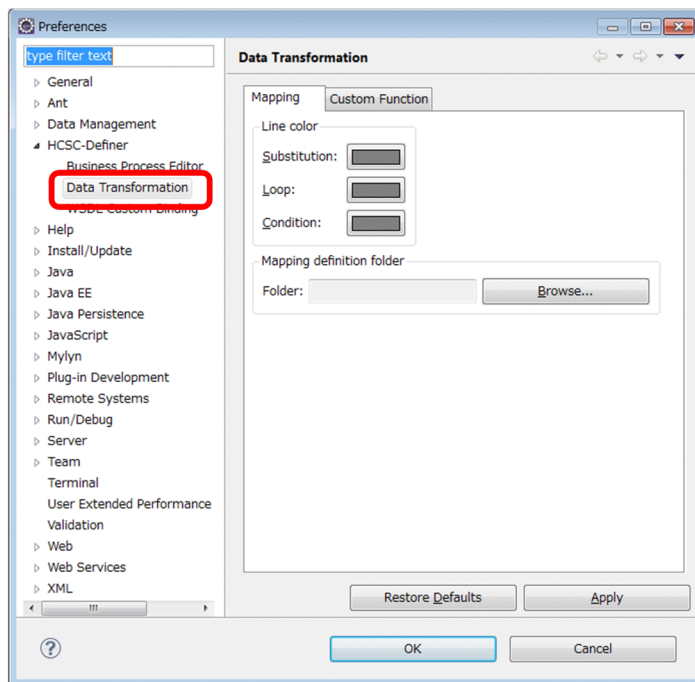
1. From the Eclipse menu, select **Window**, and then **Preferences**.

The Preferences dialog box appears.

2. In the left pane, select **HCSC-Definer**, and then **Data Transformation**.

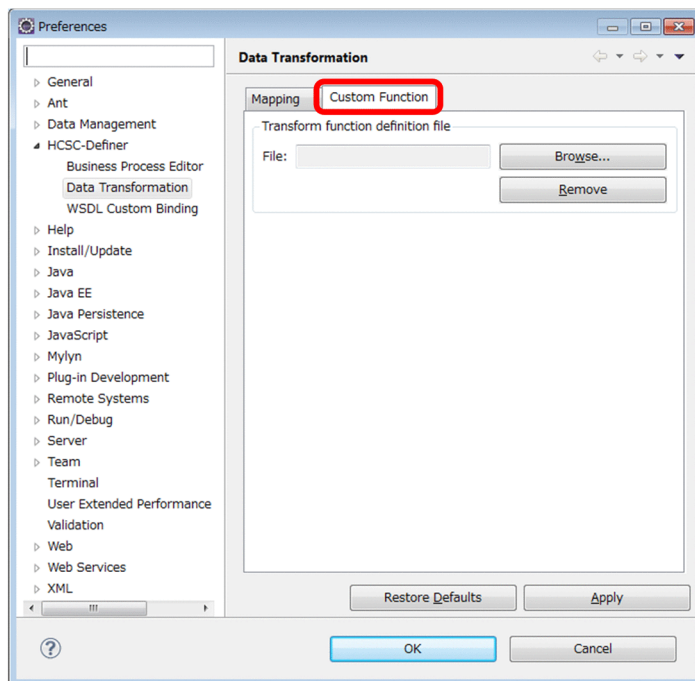
The setting items that are specified in the data transformation definition window are displayed.

5. Experiencing the Development of Sample Programs



3. Click the **Custom Function** tab.

The **Custom Function** tab in the Preference dialog box opens.

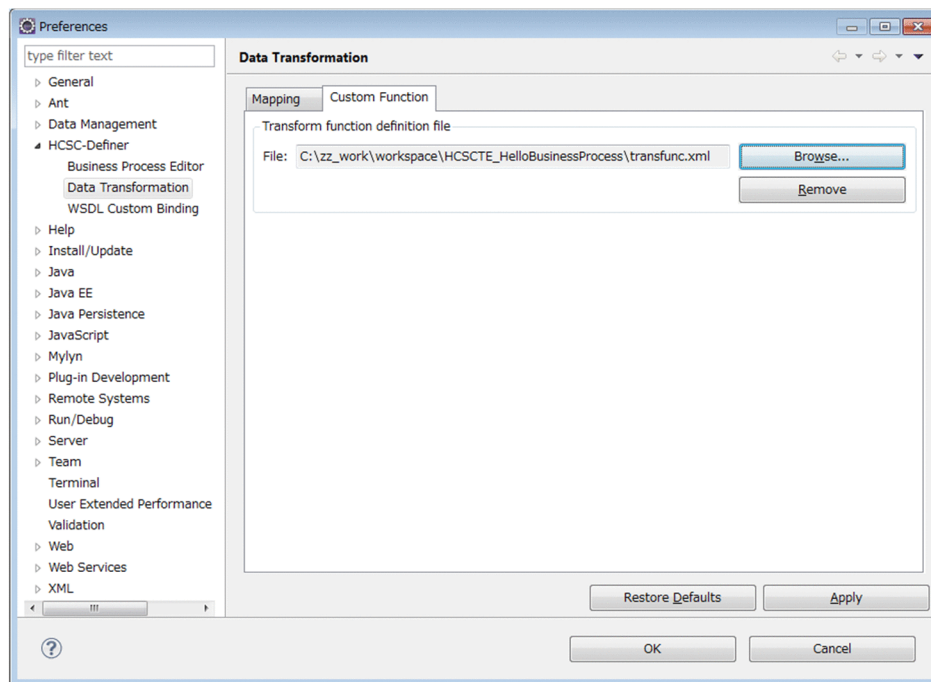


4. Click the **Browse** button.

The dialog box for selecting a file opens.

5. Select the transformation function definition file that you created in 5.8.2(1) *Creating the transformation function definition file*, and then click the **OK** button.

The transformation function definition file is selected.



6. Click the **OK** button.

The transformation function definition file is registered.

7. Click the **HelloServiceAdapter** tab.

The service adapter definition (standard) window defined during development of the Hello service adapter appears.

5. Experiencing the Development of Sample Programs

Service adapter definition (standard) | Service adapter definition (details)

Service component control information

Service name: HelloServiceAdapter
Service ID: HelAdp
Service type: Web Service
Address: http://localhost/HelloServiceW
Maximum instances: 0
Service class name: Hello
Operation: getHelloString
☐ Convert a system exception into a fault message

Operation information

Operation name: getHelloString
Communication model: Sync

+ Request message

Body | Header

Standard

☒ Use

Format ID:
Message format: Browse...
Display... Acquire...

Service component

Format ID: format4
Message format: cscformat1.xsd Browse...
Display... Acquire...

Data-conversion definition
[Standard => Service component] Edit... Delete

+ Response message

Body | Header

Standard

☐ Use

Format ID:
Message format: Browse...
Display... Acquire...

Service component

Format ID: format3
Message format: cscformat1.xsd Browse...
Display... Acquire...

8. In the **Request message** area, in the **Standard** group, select the **Use** check box. Then, for **Format ID**, **Message format**, and **Data-conversion definition**, set the values listed below.

The screenshot shows the 'HelloServiceAdapter' configuration window. The 'Request message' section is expanded, showing 'Standard' with 'Use' checked, 'Format ID' set to 'hello01', and 'Message format' set to 'HelloService.xsd'. The 'Data-conversion definition' section shows 'TRANSFORM' selected. The 'Response message' section is also expanded, showing 'Standard' with 'Use' unchecked, 'Format ID' set to 'format3', and 'Message format' set to 'cscformat1.xsd'.

No.	Item	Value to be set
1	Format ID	hello01
2	Message format	service-platform-installation-directory\CSCTE\Samples\CustomFunctions\CustomFunction\Schema\HelloService.xsd
3	Data-conversion definition	TRANSFORM

9. Click the **Edit** button.

The Select Root Element dialog box appears.

10. As the root elements for **Source** and **Destination**, select the following values from the drop-down list:

- Root element of `HelloService.xsd` (schema logical name) for **Source**: `hls:getHelloString`
- Root element of `cscformat1.xsd` (schema logical name) for **Destination**: `hls:getHelloString`

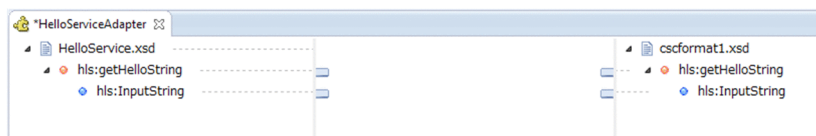
The screenshot shows the 'Select Root Element' dialog box. It contains a table with the following data:

Schema logical name	Root element	Source/Target
HelloService.xsd	hls:getHelloString	Source
cscformat1.xsd	hls:getHelloString	Target

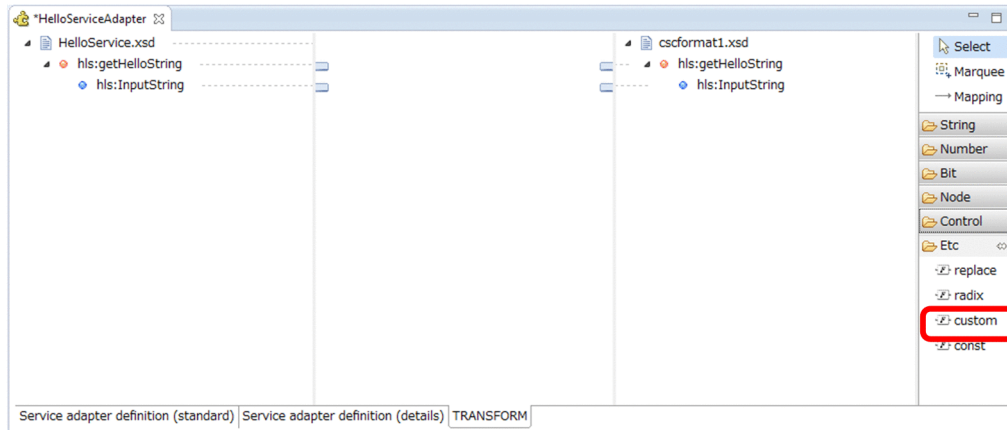
The 'OK' button is highlighted.

11. Click the **OK** button.

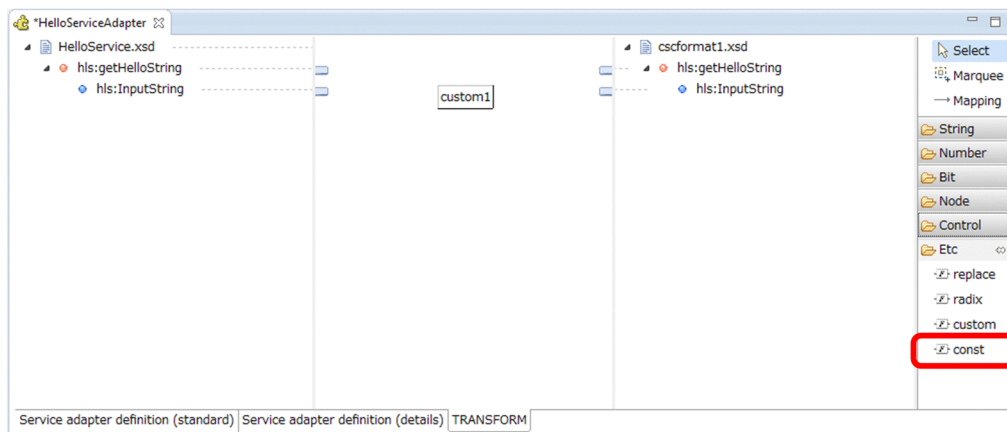
The data transformation definition window appears.



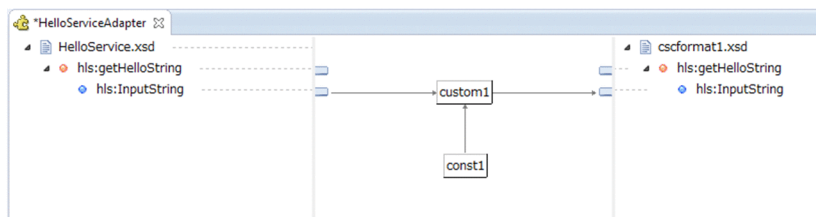
12. On the palette of the data transformation definition window, click **custom**. Then, on the canvas, click between the transformation source and destination to place the **custom** there.



13. On the palette of the data transformation definition window, click **const**. Then, on the canvas, click between the transformation source and destination to place the **const** there.

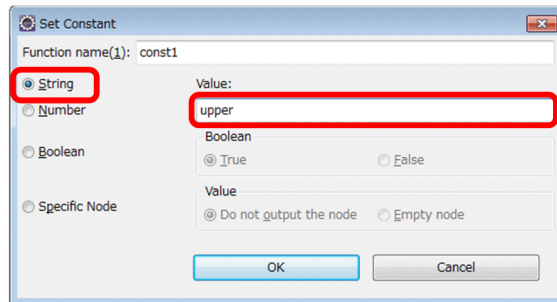


14. On the palette of the data transformation definition window, click **Mapping**.
15. Click the node adapter of the transformation-source node as the mapping source.
16. Click **custom** as the mapping destination.
- A mapping line is set.
17. In the same way as steps 14 to 16, set a mapping line from **custom** to the node adapter of the transformation-destination node.
18. In the same way as steps 14 to 16, set a mapping line from **const** to **custom**.

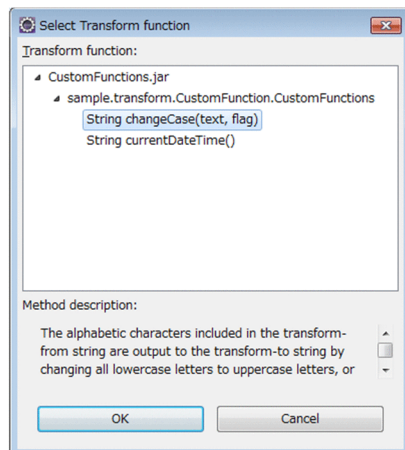


19. On the palette of the data transformation definition window, click **Select**.

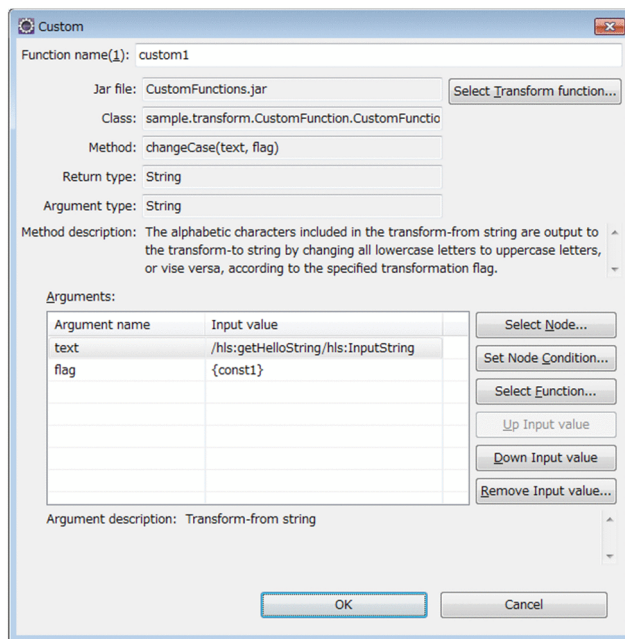
20. On the palette of the data transformation definition window, double-click **const**.
The Set Constant dialog box appears.
21. Select **String**, and then enter **upper**.



22. Click the **OK** button.
The Set Constant dialog box closes.
23. In the data transformation definition window, double-click **custom**.
The Custom dialog box appears.
24. Click the **Select Transform function** button.
The Select Transform function dialog box appears.



25. In the transformation function tree, select **String changeCase(text,flag)**, and then click the **OK** button.
The Select Transform function dialog box closes, and then the Custom dialog box re-appears.



26. Confirm that **Argument name** and **Input value** are defined as shown below.

If the combination of **Argument name** and **Input value** is different from the combination shown below, use the **Up Input value** or **Down Input value** button to combine them correctly.

No.	Argument name	Input value
1	text	/hls:getHelloString/hls:InputString
2	flag	{const1}

27. Click the **OK** button.

The Custom dialog box closes.

28. From the Eclipse menu, select **File**, and then **Save**.

The Hello service adapter has now been modified completely.

29. Validate and package the modified service adapter.

For details about validation and packaging, see 5.3.2 *Validating and packaging a service adapter*.

30. Define deployment of the modified service adapter.

For details about deployment definitions, see 5.3.3 *Defining deployment of a service adapter*.

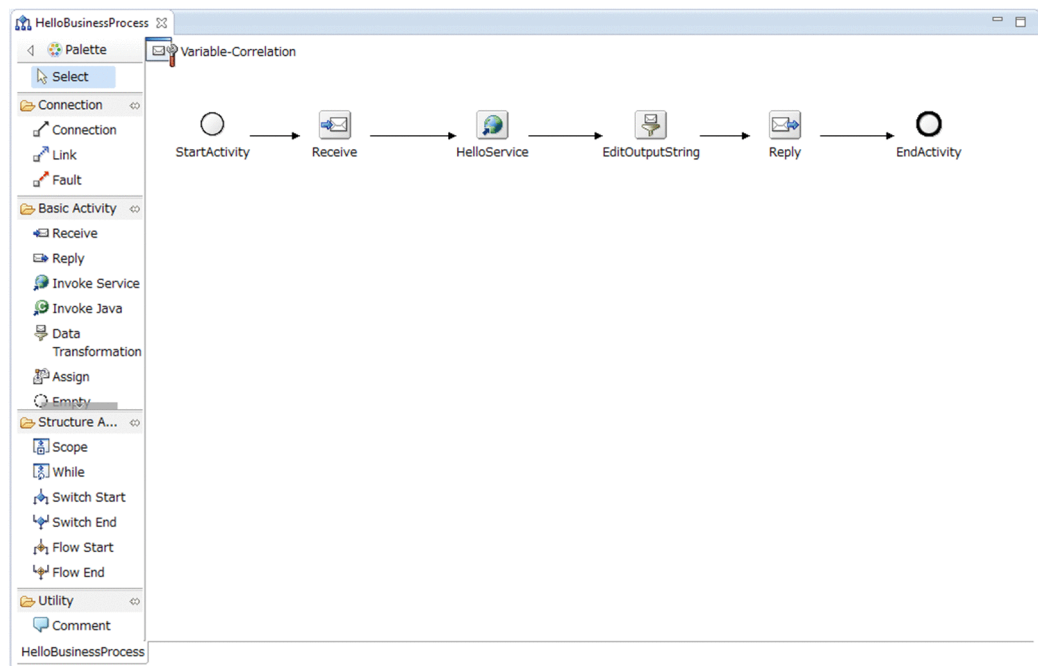
5.8.4 Modifying the Hello business process

The CustomFunction sample program adds the time that the business process was executed to the end of the output string. To implement this processing, you need to modify the data transformation definition of the Hello business process.

The following is the procedure for modifying the Hello business process.

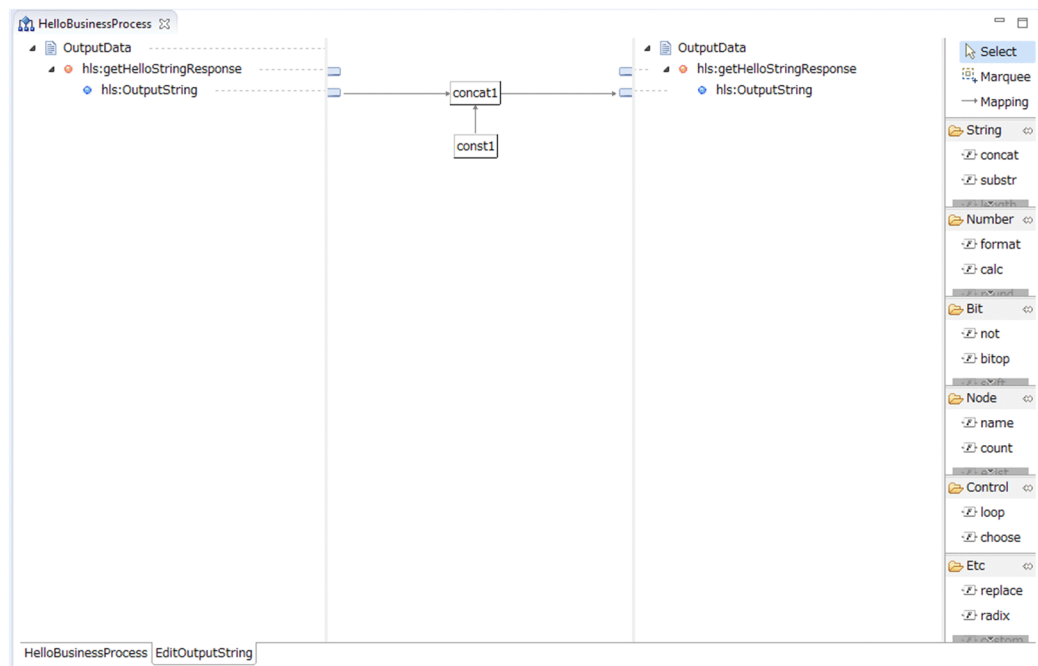
1. Click the **HelloBusinessProcess** tab.

The business process definition window defined during Hello business process development appears.

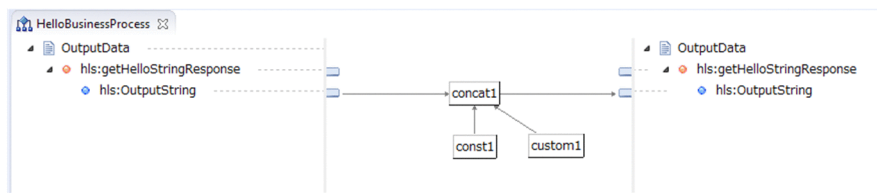


2. Click the **EditOutputString** tab.

The data transformation definition window defined during Hello business process development appears.



3. On the palette of the data transformation definition window, click **custom**. Then, on the canvas, click between the transformation source and destination to place the **custom** there.
 4. On the palette of the data transformation definition window, click **Mapping**.
 5. Click **custom** as the mapping source.
 6. Click **concat** as the mapping destination.
- A mapping line is set.



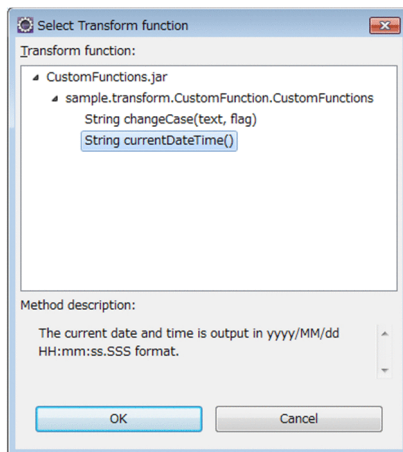
7. On the palette of the data transformation definition window, click **Select**.

8. In the data transformation definition window, double-click **custom**.

The Custom dialog box appears.

9. Click the **Select Transform function** button.

The **Select Transform function** dialog box appears.



10. In the transformation function tree, select **String currentDateTime()**, and then click the **OK** button.

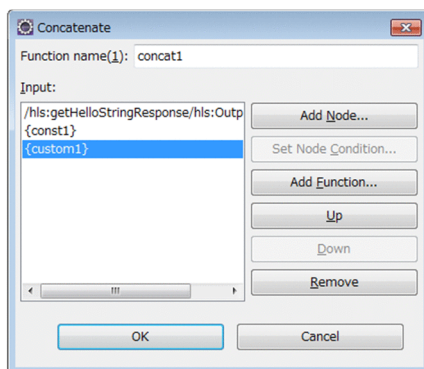
The **Select Transform function** dialog box closes, and the then Custom dialog box re-appears.

11. Click the **OK** button.

The Custom dialog box closes.

12. In the data transformation definition window, double-click **concat**.

The Concatenate dialog box appears.



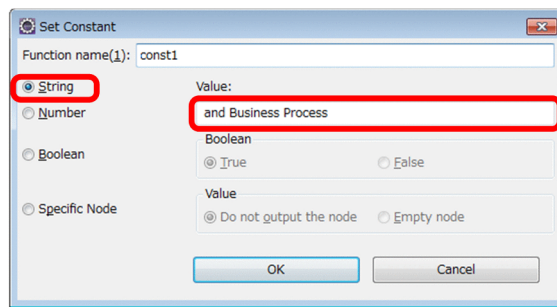
13. Confirm that **{custom1}** has been added at the bottom of the **Input** list box, and then click the **OK** button.

The Concatenate dialog box closes.

14. On the palette of the data transformation definition window, double-click **const**.

The Set Constant dialog box appears.

15. Select **String**, and then enter the following character string: **△ and △ Business △ Process (△ :**
single-byte space)



16. Click the **OK** button.

The Set Constant dialog box closes.

17. From the Eclipse menu, select **File**, and then **Save**.

The Hello business process has now been modified completely.

18. Validate and package the modified business process.

For details about validation and packaging, see 5.3.2 *Validating and packaging a service adapter*.

19. When you have modified a business process, define its deployment.

For details about deployment definitions, see 5.3.3 *Defining deployment of a service adapter*.

To execute the CustomFunction sample program that you developed

Use the following procedure to execute the sample program in the same way as in the example of executing the HelloBusinessProcess sample program:

1. Perform the procedure in 3.5.5 *Importing Eclipse projects*.
2. Perform the procedure in 3.5.6 *Deploying the web project*.
3. Perform the procedure in 4.3 *Operation when a business process is applied*.

6

Deleting the Environment for Sample Programs

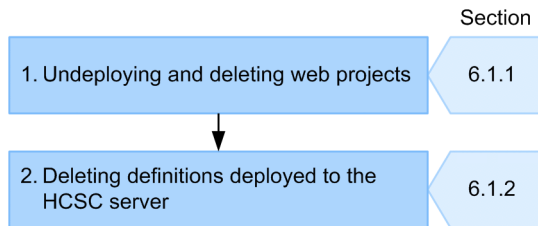
This chapter describes how to delete the environment for sample programs.

6.1 Deleting projects

The projects to be deleted differ depending on the sample program. This section describes how to delete projects, taking an example of the `HelloServiceAdapter` sample program. To run the same sample program (you previously used) or another sample program after deleting all sample programs, you must begin with importing Eclipse projects. For details about how to import Eclipse projects, see [3.5.5 Importing Eclipse projects](#).

The following figure shows an overview of deleting projects:

Figure 6–1: Overview of deleting projects



An overview of each process is provided below. Before you start deleting projects, start Eclipse.

1. Undeploying and deleting web projects

For each sample program, undeploy the web projects deployed in [3.5.6 Deploying the web project](#). Then, delete the Eclipse projects imported in [3.5.5 Importing Eclipse projects](#). For details, see [6.1.1 Undeploying and deleting web projects](#).

2. Deleting definitions deployed to the HCSC server

For each sample program, delete definitions deployed to the HCSC server in [3.5.7 Deploying definitions to the HCSC server](#). For details, see [6.1.2 Deleting definitions deployed to the HCSC server](#).

The following subsections describe how to delete projects in the order shown above.

6.1.1 Undeploying and deleting web projects

This subsection describes how to delete web projects, taking an example of the `HelloServiceAdapter` sample program.

(1) Undeploying web projects

The following describes how to undeploy web projects:

! Important note

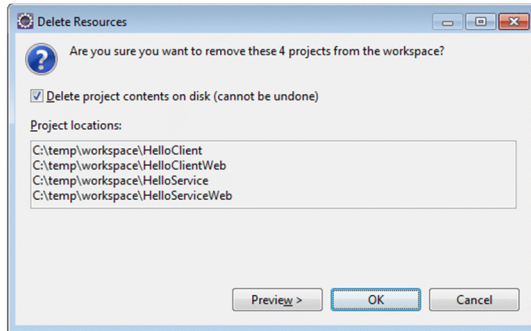
Do not delete the projects created in [3.5.7\(1\) Creating HCSCTE projects](#).

1. From the Eclipse menu, click **Window**, **Open Perspective**, and then **Other**.
The Open Perspective dialog box appears.
2. Select **Java EE (default)**, and then click the **OK** button.
The Java EE perspective appears.
3. In the **Servers** view, right-click **MyServer at localhost**, and then select **Add and Remove**.
The Add and Remove dialog box appears.
4. In the **Configured** list box, select **HelloClient** and **HelloService**, and then click the **Remove** button.
HelloClient and **HelloService** are moved from the **Configured** list box to the **Available** list box.
5. Click the **Finish** button.
A message indicating that operation is in progress appears.
After processing terminates, the Java EE perspective appears again. Make sure that the **HelloClient** and **HelloService** projects have been deleted from under **MyServer at localhost** in the **Servers** view.

(2) Deleting web projects

The following describes how to delete web projects:

1. In the **Project Explorer** view, select the imported Eclipse project.
For details about the Eclipse projects imported for each sample program, see *3.5.5 Importing Eclipse projects*.
2. Right-click, and then select **Delete**.
The Delete Resources dialog box that asks you to confirm deletion appears.

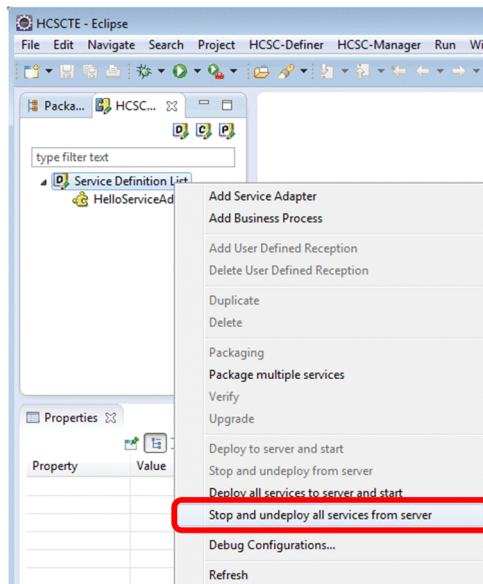


3. Select the **Delete project contents on disk** check box, and then click the **OK** button.
The selected Eclipse project is deleted.

6.1.2 Deleting definitions deployed to the HCSC server

This subsection describes how to delete the service adapter deployed to the HCSC server, taking an example of the HelloServiceAdapter sample program.

1. In the HCSCTE tree view, right-click **Service Definition List**, and then select **Stop and undeploy all services from server**.



The account authentication window appears.

2. Enter **admin** in both **User ID** and **Password**, and then click the **OK** button.
A message indicating that operation is in progress appears. Then, a message indicating the results appears. This completes deletion of the HCSC component deployed to the HCSC server.
3. To completely delete the definitions from the HCSCTE project, select **HCSC-Definer**, **Repository management**, and then **Initialize repository**.

6. Deleting the Environment for Sample Programs

When the dialog box that asks you to confirm initialization appears, click the **Yes** button. Definitions are now completely deleted.

6.2 Stopping the test environment

To terminate use of Service Architect, stop the active test environment, and then exit Eclipse.

To stop the test environment:

1. From the **Start** menu, select **Programs, Cosminexus[#]**, and then **Stop Test Server** to stop Performance Tracer, the J2EE server, and the HCSC server (including the standard reception and user-defined reception) in the test environment.
2. From the **Start** menu, select **Programs, Cosminexus[#]**, and then **Stop Database** to stop the embedded database in the test environment.

#

If this program folder name has been changed, select the changed program folder name.

Important note

If you exit Eclipse before stopping the test environment, process-termination processing is performed to terminate the test environment even after the Eclipse window is closed. Therefore, when you start Eclipse the next time, an error message indicating that another process is using the HCSCTE temporary directory might appear. The following describes what to do if an error message appears:

1. Open Windows Task Manager.
 2. Wait until the `eclipse.exe` and `javaw.exe` processes, which are running Eclipse, disappear from the list of processes.
 3. After confirming that the processes have disappeared, start Eclipse.
-

6.3 Unsetup and uninstallation

This section describes how to undo setup of the test environment and Eclipse, and how to uninstall Service Architect.

6.3.1 Undoing setup of the test environment

Use HCSC Easy Setup to undo setup of the test environment. To undo setup of the test environment:

1. Stop the test environment if it is active. Exit Eclipse if it is running.
2. From the **Start** menu, select **Programs, Cosminexus[#], First Setup**, and then **Setup Testing Environment**.
The **Main** page of the HCSC Easy Setup window appears.
3. Click the **Unsetup** button.
Unsetup of the test environment starts, and then finishes after a while.

#

If this program folder name has been changed, select the changed program folder name.

6.3.2 Undoing setup of Eclipse

Use Eclipse Setup to undo setup of an environment that was built by using Eclipse Setup.

The following table lists the items that are deleted when setup is undone by using Eclipse Setup:

Table 6–1: Items deleted when setup is undone by using Eclipse Setup

Item	Deleted?
Eclipse Platform	Deleted ^{#1}
Eclipse configuration folder (folder for each user)	Not deleted ^{#2}
Eclipse workspace	Not deleted
Shortcut to Eclipse	Deleted

#1

All data in the target folder, including user-created files, will be deleted.

#2

The configuration folder and workspace are created by Eclipse. Therefore, they are not deleted by undoing setup by using Eclipse Setup provided by Developer. If you want to delete them, you have to do so manually. For details, see (2) *Undoing setup manually*.

If you have uninstalled and then re-installed Service Architect before undoing setup of an environment built by using Eclipse Setup, see (2) *Undoing setup manually*.

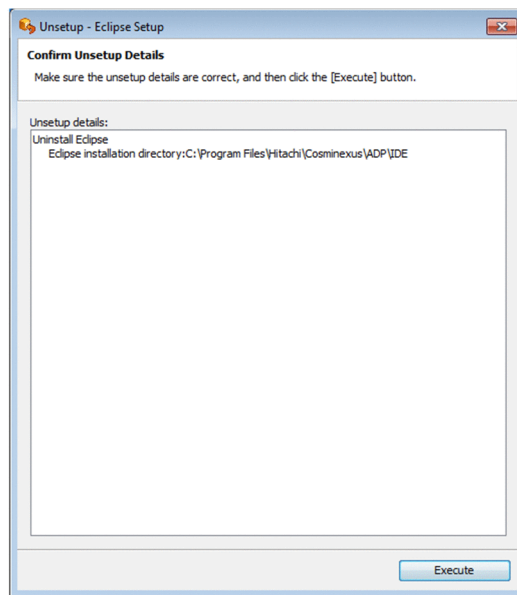
! Important note

Before you start undoing setup, exit Eclipse. If you undo setup while Eclipse is running, the Eclipse installation directory will not be deleted. If the Eclipse installation directory remains after setup is undone, manually delete the directories and files in *Eclipse-installation-directory\ eclipse*.

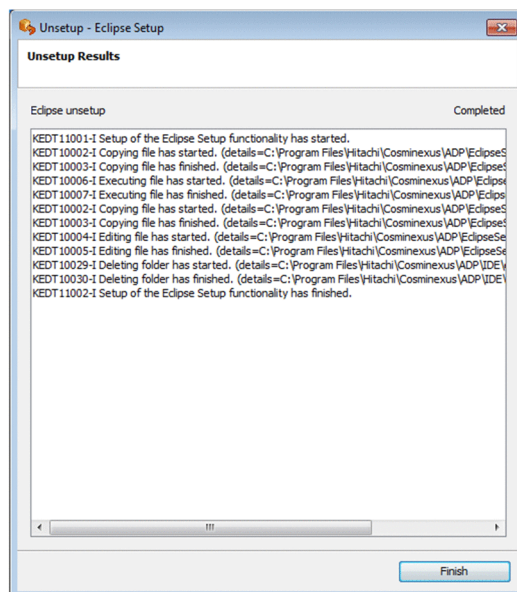
(1) Using Eclipse Setup

To use Eclipse Setup to undo setup:

1. From the **Start** menu, select **Programs, Cosminexus, First Setup**, and then **Unsetup Eclipse**.
Eclipse Setup starts, and then the **Verify Unsetup** page appears in the Unsetup - Eclipse Setup dialog box.



2. Check the information displayed in the **Unsetup details** area, and then click the **Execute** button.
The **Progress Status** page appears.
When unsetup processing terminates, the **Unsetup Completed** page appears.



3. Click the **Finish** button.
The Unsetup - Eclipse Setup dialog box closes.

! Important note

If you undo setup in Windows Vista or later, the shortcut to `eclipse.exe` that was added to the desktop during setup is deleted. However, the shortcut icon might remain on the screen (as if not deleted). In this case, refresh the desktop to delete the shortcut from the screen.

(2) Undoing setup manually

If you have uninstalled and then re-installed Service Architect before undoing setup of the environment built by using Eclipse Setup, manually undo that environment. To manually undo setup of the test environment:

1. Delete the shortcut to Eclipse.

The administrator deletes the following files:

In Windows XP:

`C:\Documents and Settings\All Users\Desktop\Eclipse.lnk`

In Windows Vista or later

`C:\Users\Public\Desktop\Eclipse.lnk`

2. Delete the Eclipse configuration folder.

The following shows the location of the folder:

In Windows XP:

`C:\Documents and Settings\All Users\ADP`

In Windows Vista or later

`C:\Users\All Users\ADP`

3. Delete the Eclipse folder.

The following shows the default location of the folder:

service-platform-installation-directory\ADP\IDE\eclipse

6.3.3 Uninstalling Service Architect

Uninstall Service Architect. For uninstallation, you must have Administrator permissions or administrator privileges.

Before uninstalling Service Architect, you need to undo setup of Eclipse.

1. From the Windows **Start** menu, select **Cosminexus**, and then **uCosminexus Service Architect Uninstall**.

A dialog box that asks you to confirm uninstallation appears.

2. Click the **Yes** or **No** button.

If you click the **Yes** button

Uninstallation starts, and then all the Service Architect configuration software products are deleted.

If you click the **No** button

A dialog box for selecting the configuration software to be uninstalled appears. Select the configuration software to uninstall, and then click the **Next** button. Uninstallation starts, and then the selected configuration software is deleted.

Appendixes

A. Configuration of sample program files

The following table lists the locations of the sample programs:

Table A–1: Storage location of sample programs

Sample program name	Storage location
HelloServiceAdapter	<i>service-platform-installation-directory</i> \CSCTE\Samples\HelloServiceAdapter
HelloBusinessProcess	<i>service-platform-installation-directory</i> \CSCTE\Samples\HelloBusinessProcess
HelloProductArrangement	<i>service-platform-installation-directory</i> \CSCTE\Samples\ProductStock
CustomFunction	<i>service-platform-installation-directory</i> \CSCTE\Samples\CustomFunctions

The sections below describe the configuration of files for each sample program.

Bold text indicates files that are installed in a sample program. Files not in bold text are automatically created.

A.1 Configuration of the HelloServiceAdapter sample program

```

HelloServiceAdapter
|-Repository ..... Repository directory
| |-HelloServiceAdapter.zip ..... Repository
|-Requester ..... Service requester directory
| |-HelloClient ..... Eclipse Java enterprise application project
| | .project
| |+.settings
| | | org.eclipse.wst.common.component
| | | org.eclipse.wst.common.project.facet.core.xml
| | +EarContent
| | | +META-INF
| | | application.xml
| |-HelloClientWeb ..... Eclipse Java Web project
| | .classpath
| | .project
| |+.settings
| | | .jsdtscope
| | | org.eclipse.jdt.core.prefs
| | | org.eclipse.wst.common.component
| | | org.eclipse.wst.common.project.facet.core.xml
| | | org.eclipse.wst.jsdt.ui.superType.container
| | | org.eclipse.wst.jsdt.ui.superType.name
| +build
| +classes
| +src
| +jp
| | +co
| | | +Hitachi
| | | | +soft
| | | | | +csc
| | | | | +msg
| | | | | +message
| | | | | +reception
| | | | | |-ejb ..... Stores client stubs that are generated based on WSDL
| | | | | | CSCMsgServerFaultException.java
| | | | | | CSCMsgSyncServiceDeliveryWSImpl.java
| | | | | | CSCMsgSyncServiceDeliveryWSImplService.java
| | | | | | CSCMsgSyncServiceDeliveryWSImplServiceLocator.java
| | | | | | CSCMsgSyncServiceDeliveryWSImplSoapBindingStub.java
| +sample
| +servlet
| | HelloServlet.java ..... Service component invocation servlet
| +xml
| | DomUtil.java
| | XmlErrorHandler.java
| +WebContent
| | index.html ..... Input window
| +META-INF
| | MANIFEST.MF
| +WEB-INF
| | web.xml ..... DD

```

```

|      +lib
|-Service ..... Service component directory
| |-HelloService ..... Eclipse Java enterprise application project
| |   .project
| |   c4webcl.properties ..... Client definition file used by the Hello service adapter
| +.settings
| |   org.eclipse.wst.common.component
| |   org.eclipse.wst.common.project.facet.core.xml
| +EarContent
| |   +META-INF
| |   |   application.xml
| |-HelloServiceWeb ..... Eclipse Java Web project
| |   .classpath
| |   .project
| +.settings
| |   .jsdtscope
| |   org.eclipse.jdt.core.prefs
| |   org.eclipse.wst.common.component
| |   org.eclipse.wst.common.project.facet.core.xml
| |   org.eclipse.wst.jsdt.ui.superType.container
| |   org.eclipse.wst.jsdt.ui.superType.name
| +build
| | +classes
| | +src
| | +sample
| | |   +HelloService
| | |   |   Hello.java ..... Server skeleton
| | |   |   HelloSoapBindingImpl.java ..... Hello service (server skeleton)
| +WebContent
| |   +META-INF
| |   |   MANIFEST.MF
| |   +WEB-INF
| |   |   server-config.xml ..... Service deploy definition file (server skeleton)
| |   |   web.xml ..... DD
| |   +lib
+WSDL
|   cscmsg_ws.wsdl
|   HelloService.wsdl ..... WSDL for the Hello service

```

A.2 Configuration of the HelloBusinessProcess sample program

```

HelloBusinessProcess
|-Repository ..... Repository directory
|   HelloBusinessProcess.zip ..... Repository
|-Requester ..... Service requester directory
| |-HelloClient ..... Eclipse Java enterprise application project
| |   .project
| +.settings
| |   org.eclipse.wst.common.component
| |   org.eclipse.wst.common.project.facet.core.xml
| +EarContent
| |   +META-INF
| |   |   application.xml
| |-HelloClientWeb ..... Eclipse Java Web project
| |   .classpath
| |   .project
| +.settings
| |   .jsdtscope
| |   org.eclipse.jdt.core.prefs
| |   org.eclipse.wst.common.component
| |   org.eclipse.wst.common.project.facet.core.xml
| |   org.eclipse.wst.jsdt.ui.superType.container
| |   org.eclipse.wst.jsdt.ui.superType.name
| +build
| | +classes
| | +src
| | |   +jp
| | |   |   +co
| | |   |   |   +Hitachi
| | |   |   |   |   +soft
| | |   |   |   |   |   +csc
| | |   |   |   |   |   |   +msg
| | |   |   |   |   |   |   |   +message
| | |   |   |   |   |   |   |   |   +reception
| | |   |   |   |   |   |   |   |   |   |-ejb ..... Stores client stubs that are generated based on WSDL
| | |   |   |   |   |   |   |   |   |   |   CSCMsgServerFaultException.java
| | |   |   |   |   |   |   |   |   |   |   CSCMsgSyncServiceDeliveryWSImpl.java
| | |   |   |   |   |   |   |   |   |   |   CSCMsgSyncServiceDeliveryWSImplService.java
| | |   |   |   |   |   |   |   |   |   |   CSCMsgSyncServiceDeliveryWSImplServiceLocator.java

```

```

| | | CSCMsgSyncServiceDeliveryWSImplSoapBindingStub.java
| |+sample
| | | +servlet
| | | | HelloServlet.java ..... Service component invocation servlet
| | | +xml
| | | | DomUtil.java
| | | | XmlErrorHandler.java
| |+WebContent
| | | | index.html ..... Input window
| | | +META-INF
| | | | MANIFEST.MF
| | | +WEB-INF
| | | | web.xml ..... DD
| | | +lib
|-Service ..... Service component directory
| |-HelloService ..... Eclipse Java enterprise application project
| | | .project
| | | | c4webcl.properties ..... Client definition file used by the Hello service adapter
| | | +.settings
| | | | org.eclipse.wst.common.component
| | | | org.eclipse.wst.common.project.facet.core.xml
| | | +EarContent
| | | | +META-INF
| | | | | application.xml
| |-HelloServiceWeb ..... Eclipse Java Web project
| | | .classpath
| | | .project
| | | +.settings
| | | | .jsdtscope
| | | | org.eclipse.jdt.core.prefs
| | | | org.eclipse.wst.common.component
| | | | org.eclipse.wst.common.project.facet.core.xml
| | | | org.eclipse.wst.jsdt.ui.superType.container
| | | | org.eclipse.wst.jsdt.ui.superType.name
| | | +build
| | | | +classes
| | | | +src
| | | | +sample
| | | | | +HelloService
| | | | | | Hello.java ..... Server skeleton
| | | | | | HelloSoapBindingImpl.java ..... Hello service (server skeleton)
| | | +WebContent
| | | | +META-INF
| | | | | MANIFEST.MF
| | | | +WEB-INF
| | | | | server-config.xml ..... Service deploy definition file (server skeleton)
| | | | | web.xml ..... DD
| | | | +lib
| | | +WSDL
| | | | cscmsg_ws.wsdl
| | | | HelloService.wsdl ..... WSDL for the Hello service

```

A.3 Configuration of the HelloProductArrangement sample program

```

ProductStock
|-Repository ..... Repository directory
| | ProductStock.zip ..... Repository
|-Requester ..... Service requester directory
| |-ArrangementClient ..... Eclipse Java enterprise application project
| | | .project
| | | +.settings
| | | | org.eclipse.wst.common.component
| | | | org.eclipse.wst.common.project.facet.core.xml
| | | +EarContent
| | | | +META-INF
| | | | | application.xml
| |-ArrangementClientWeb ..... Eclipse Java Web project
| | | .classpath
| | | .project
| | | +.settings
| | | | .jsdtscope
| | | | org.eclipse.jdt.core.prefs
| | | | org.eclipse.wst.common.component
| | | | org.eclipse.wst.common.project.facet.core.xml
| | | | org.eclipse.wst.jsdt.ui.superType.container
| | | | org.eclipse.wst.jsdt.ui.superType.name
| | | +build
| | | | +classes
| | | | +src

```

```

| | +sample
| | | +ArrangementService
| | | | Arrangement.java ..... Client stub
| | | | ArrangementService.java ..... Client stub
| | | | ArrangementServiceLocator.java ..... Client stub
| | | | ArrangementSoapBindingStub.java ..... Client stub
| | | +servlet
| | | | ArrangementServlet.java ..... Service component invocation servlet
| | +WebContent
| | | error.jsp ..... Error response window
| | | index.html ..... Input window
| | | result.jsp ..... Normal response window
| | +META-INF
| | | MANIFEST.MF
| | +WEB-INF
| | | web.xml ..... DD
| | +lib
+Service
| -DeliveryService ..... Eclipse Java enterprise application project
| | .project
| | | c4webcl.properties ..... Client definition file used by the delivery reception
service adapter
| | +.settings
| | | org.eclipse.wst.common.component
| | | org.eclipse.wst.common.project.facet.core.xml
| | +EarContent
| | | +META-INF
| | | | application.xml
| | -DeliveryServiceWeb ..... Eclipse Java Web project
| | | .classpath
| | | .project
| | +.settings
| | | .jsdtscope
| | | org.eclipse.jdt.core.prefs
| | | org.eclipse.wst.common.component
| | | org.eclipse.wst.common.project.facet.core.xml
| | | org.eclipse.wst.jsdt.ui.superType.container
| | | org.eclipse.wst.jsdt.ui.superType.name
| | +build
| | | +classes
| | | +src
| | | +sample
| | | | +DeliveryService
| | | | | Delivery.java ..... Server skeleton
| | | | | DeliverySoapBindingImpl.java ..... Delivery reception service (server skeleton)
| | | +WebContent
| | | | +META-INF
| | | | | MANIFEST.MF
| | | | +WEB-INF
| | | | | server-config.xml ..... Service deploy definition file (server skeleton)
| | | | | web.xml ..... DD
| | | | +lib
| | -InventoryManagementService ... Eclipse Java enterprise application project
| | | .project
| | | | c4webcl.properties ..... Client definition file used by the inventory management
service adapter
| | | +.settings
| | | | org.eclipse.wst.common.component
| | | | org.eclipse.wst.common.project.facet.core.xml
| | | +EarContent
| | | | +META-INF
| | | | | application.xml
| | -InventoryManagementServiceWeb ..... Eclipse Java Web project
| | | .classpath
| | | .project
| | +.settings
| | | .jsdtscope
| | | org.eclipse.jdt.core.prefs
| | | org.eclipse.wst.common.component
| | | org.eclipse.wst.common.project.facet.core.xml
| | | org.eclipse.wst.jsdt.ui.superType.container
| | | org.eclipse.wst.jsdt.ui.superType.name
| | +build
| | | +classes
| | | +src
| | | +sample
| | | | +InventoryManagementService
| | | | | InventoryManager.java ..... Server skeleton
| | | | | InventoryManagerSoapBindingImpl.java ... Inventory management service (server
skeleton)
| | | +WebContent
| | | | +META-INF
| | | | | MANIFEST.MF

```

```
| +WEB-INF
| | server-config.xml ..... Service deploy definition file (server skeleton)
| | web.xml ..... DD
| +lib
+WSDL
  ArrangementService.wsdl ..... WSDL for HelloProductArrangement
  DeliveryService.wsdl ..... WSDL for the delivery reception service
  InventoryManagementService.wsdl ..... WSDL for the inventory management service
```

A.4 Configuration of the CustomFunction sample program

```
CustomFunctions
|-Repository ..... Repository directory
|  HelloCustomFunction.zip ..... Repository
+CustomFunction
  +CustomFunctions
  | .classpath
  | .project
  | build.xml ..... Ant build definition file
  +bin
  +build
  | +classes
  | +lib
  | CustomFunctions.jar ..... Custom function jar file
  +src
  | +sample
  |   +transform
  |     +CustomFunction
  |       CustomFunctions.java ..... Sample source
+Configuration
| CustomFunctions.xml ..... Transformation function definition file
+Schema
  HelloService.xsd ..... Schema used to correct the Hello service adapter
```

B. Collecting the Information Output When Eclipse Setup Was Executed

Information specified when Eclipse Setup was executed, including setup details, configuration changes, and unsetup details, is saved in a file as *setup log*.

The following describes how to check the setup log, and the information recorded in the setup log:

- How to check the setup log

From the Windows **Start** menu, select **Programs**, **Cosminexus**, **First Setup**, and then **Eclipse Setup Log**. The setup log is displayed.

- Information recorded in the setup log

You can check the following information in the setup log in the Eclipse environment:

- Settings specified when the Eclipse environment was set up
- Settings of the Eclipse environment for which setup was undone

These entries are stored in one file in the order in which operations were executed. Therefore, the most recent information is stored at the end of the file. Even if an error occurred or processing was interrupted, the log information is stored.

While Eclipse Setup is being executed, error messages are displayed in the Eclipse Setup console window. If error messages are output, take action according to the manual *Application Server Messages*.

C. Glossary

Terms used in the manual

See the *Application Server and BPM/ESB Platform Terminology Guide*.

Index

A

activities, defining 72, 89
activities, deploying 71, 87
ArrangementService.wsdl 81

B

business process 8
business process, adding 67, 82
business processes, creating 67
business processes, defining 67

C

calling service component from business process 9
calling service component from service requester 9
Compiler (page) 22
components, defining deployment of 101
components, validating and packaging 101
conventions: fonts and symbols 2
conventions: installation directory 2
conventions: language pack 2
conventions: types of text in syntax 2
custom function, preparing 112
CustomFunction sample program, configuration of 144
CustomFunction sample program, general procedure for defining 112
CustomFunction sample program, processing of 111

D

data transformation, defining (by using Java program) 111
definitions 40
definitions deployed to HCSC server, deleting 133
delivery reception service adapter, defining 79
DeliveryService.wsdl 79
development environment, setting up 20

E

Eclipse, preparing 16
Eclipse, setting up (development environment) 20
Eclipse, setting up (execution environment) 28
Eclipse, undoing setup of 136
Eclipse projects, importing 35
Eclipse Setup, using (to set up Eclipse environment) 17
execution environment, setting up 25

G

glossary 146

H

HCSC Easy Setup 26
HCSC server 8

HCSC server, deploying definitions to 40
HCSCTE project, creating 58
Hello business process, defining deployment of 76
Hello business process, modifying 126
Hello business process, validating and packaging 76
HelloBusinessProcess (sample program), processing contents 10
HelloBusinessProcess (sample program), processing details 10
HelloBusinessProcess (sample program), system configuration of 7
HelloBusinessProcess sample program, configuration of 141
HelloBusinessProcess sample program (operation procedure) 51
HelloProductArrangement (sample program), processing contents 11
HelloProductArrangement (sample program), processing details 11
HelloProductArrangement (sample program), system configuration of 7
HelloProductArrangement sample program, configuration of 142
Hello service adapter, modifying 119
HelloServiceAdapter (sample program), processing contents 9
HelloServiceAdapter (sample program), processing details 9
HelloServiceAdapter (sample program), system configuration of 6
HelloServiceAdapter sample program, configuration of 140
HelloServiceAdapter sample program (operation procedure) 49

I

installation and setup, overview of (Service Architect) 14
Installed JREs (page) 20
installing Service Architect 15
integrating processes 10
InventoryManagementService.wsdl 77

J

J2EE server 8
J2EE server, selecting 33
Java program, creating 117
JDK, checking version of 20

L

library paths, setting 28
local variables, specifying settings for outputting information about 22

M

Management Server Remote Management, logging in to 28

O

- operation when business processes are not applied 49
- operation when business process is applied 51
- operation when processes of multiple services are integrated 53
- overview of this manual 2

P

- product arrangement business process, defining 81
- product arrangement sample program (operation procedure) 53
- product arrangement system, debugging 102
- product arrangement system, developing 77
- projects, deleting 132

S

- sample program, preparing for running 110
- sample program files, configuration of 140
- sample programs, before using 1
- sample programs, components of 8
- sample programs, deleting environment for 131
- sample programs, executing 47, 48
- sample programs, experiencing the development of 55
- sample programs, location of 140
- sample programs, overview of 5
- sample programs, preparing environment for 13
- sample programs, procedure for developing 56
- sample programs, processing details of 9
- sample programs, system configurations of 6
- server runtime, creating 31
- service adapter 8
- service adapter, creating 61
- service adapter, defining deployment of 65
- service adapter, validating and packaging 64
- service adapters, defining 61
- Service Architect, installing and setting up 15
- Service Architect, uninstalling 138
- service component 8
- service requester 8
- setup log 145
- standard reception 8
- stock management service adapter, defining 77

T

- test environment, building 26
- test environment, starting 27
- test environment, stopping 135
- test environment, undoing setup of 136
- transformation function definition file, creating 112

U

- unsetup and uninstallation 136
- user-defined reception 8
- user-defined reception, adding 83

V

- variables, setting 69, 85

W

- web project, deploying 38
- web projects, undeploying and deleting 132
- what you can expect from this manual 3
- WSDL4J, installing 23