FUJITSU

FUJITSU Software
NetCOBOL V11.1

NetCOBOL Studio User's Guide

Windows(64)

B1WD-3309-03ENZ0(00)
February 2017

Preface

NetCOBOL Studio is an integrated development environment for COBOL programs.

Intended Readers

This documentation provides information for COBOL program developers who use NetCOBOL Studio. For details on general functions of Eclipse, see "Workbench User Guide" in the Help information. Readers of this documentation are assumed to have a basic knowledge of COBOL programming and the Windows operating system.

Organization of This Documentation

This documentation is organized as follows:

Chapter 1 Overview

Provides an overview of NetCOBOL Studio, describes its starting method, and explains its development workflows.

Chapter 2 Tutorial

Provides a tutorial explaining the basic operations of NetCOBOL Studio.

Chapter 3 COBOL Perspective

Describes the types of views that are displayed when a COBOL project is created.

Chapter 4 Project

Explains how to create a project.

Chapter 5 Editor

Explains the COBOL editor that is used to edit COBOL source files.

Chapter 6 Build Function

Explains how to build a COBOL program using the build tools provided in NetCOBOL Studio.

Chapter 7 Debugging Function

Explains the startup configuration used to debug programs, and the functions provided by the interactive debugger.

Chapter 8 Execution Function

Explains the startup configuration used for project execution.

Chapter 9 Remote Development Function

Explains the procedure for remote development of COBOL programs that run on a server.

Chapter 10 Differences between Eclipse 3.4 and Eclipse 4.3

Explains the distinction of Eclipse 4.3 base and Eclipse 3.4 base.

Chapter 11 CORBA Development Support Function

Explains CORBA application development using the GUI function that the "Interstage Studio COBOL plug-in" offers.

Appendix A Compile Options

Explains the compiler options.

Appendix B Troubleshooting Guide

Explains how to resolve NetCOBOL Studio issues.

Appendix C Handling of Workspace and Project

Explains the handling of workspace and project.

Appendix D Various Specification Formats used in Debugging

Explains various specification formats used in the Watch view.

Appendix E Transition from Project Manager

Explains the transfer of COBOL applications from NetCOBOL project manager for Windows(x86) to NetCOBOL Studio, and the project organization converting command that is a transference support tool.

Abbreviations

The following abbreviations are used in this manual:

Product Name	Abbreviation
Microsoft(R) Windows Server(R) 2016 Datacenter Microsoft(R) Windows Server(R) 2016 Standard Microsoft(R) Windows Server(R) 2016 Essentials	Windows Server 2016
Microsoft(R) Windows Server(R) 2012 R2 Datacenter Microsoft(R) Windows Server(R) 2012 R2 Standard Microsoft(R) Windows Server(R) 2012 R2 Essentials Microsoft(R) Windows Server(R) 2012 R2 Foundation	Windows Server 2012 R2
Microsoft(R) Windows Server(R) 2012 Datacenter Microsoft(R) Windows Server(R) 2012 Standard Microsoft(R) Windows Server(R) 2012 Essentials Microsoft(R) Windows Server(R) 2012 Foundation	Windows Server 2012
Microsoft(R) Windows Server(R) 2008 R2 Foundation 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 2008 R2
Windows(R) 10 Education Windows(R) 10 Home Windows(R) 10 Pro Windows(R) 10 Enterprise	Windows 10 or Windows 10(x64)
Windows(R) 8.1 Windows(R) 8.1 Pro Windows(R) 8.1 Enterprise	Windows 8.1 or Windows 8.1(x64)
Windows(R) 7 Home Premium Windows(R) 7 Professional Windows(R) 7 Enterprise Windows(R) 7 Ultimate	Windows 7 or Windows 7(x64)
Oracle Solaris	Solaris
Red Hat(R) Enterprise Linux(R) 5 (for Itanium)	Linux or Linux(Itanium)
Red Hat(R) Enterprise Linux(R) 5(for Intel64) Red Hat(R) Enterprise Linux(R) 6(for Intel64)	Linux or Linux(x64)

- In this manual, when all the following products are indicates, it is written as "Windows(x64)" or "Windows".
    - Windows Server 2016
    - Windows Server 2012 R2
    - Windows Server 2012
    - Windows Server 2008 R2
    - Windows 10(x64)
    - Windows 8.1(x64)
    - Windows 7(x64)

Trademarks

NetCOBOL is a trademark or registered trademark of Fujitsu Limited or its subsidiaries in the United States or other countries or in both.

Eclipse is a trademark of Eclipse Foundation, Inc.

Microsoft, Windows, and Internet Explorer are registered trademarks of Microsoft Corporation in the United States and other countries.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners. Oracle Solaris might be described as Solaris, Solaris Operating System, or Solaris OS.

Linux is a registered trademark of Linus Torvalds in the United States and other countries.

Red Hat, RPM and all Red Hat-based trademarks are trademarks or registered trademarks of Red Hat, Inc. in the United States and other countries.

Intel and Itanium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

All other product names mentioned herein are the trademarks or registered trademarks of their respective owners.

Export Regulation

Exportation/release of this document may require necessary procedures in accordance with the regulations of your resident country and/or US export control laws.

February 2017

Contents

Chapter 1 Overview

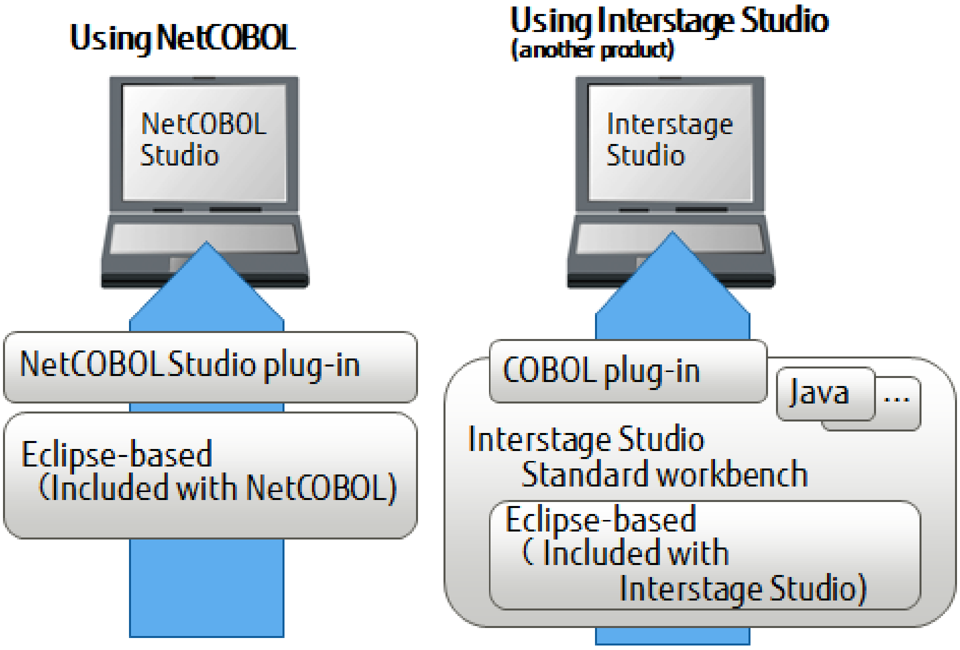
This chapter provides an overview of NetCOBOL Studio and explains its development workflows.

1.1 NetCOBOL Studio

NetCOBOL Studio is an integrated development environment for COBOL programs.

NetCOBOL Studio includes the Assist functions for COBOL programming based on Eclipse, an open-source GUI development environment. With NetCOBOL, users can efficiently perform a series of development work steps, including editing COBOL source programs, and building, debugging, and executing COBOL programs.

There are two ways of using NetCOBOL Studio. One way is to do NetCOBOL application development. The other way is to integrate the COBOL plug-in of NetCOBOL Studio into Interstage Studio, which provides an Integrated Development Environment that includes Java language.



Using NetCOBOL
NetCOBOL Studio
NetCOBOL Studio plug-in
Eclipse-based (Included with NetCOBOL)
Using Interstage Studio (another product)
Interstage Studio
COBOL plug-in
Java
...
Interstage Studio Standard workbench
Eclipse-based (Included with Interstage Studio)

Note

- In Windows 64bit NetCOBOL Studio, a 64-bit COBOL application can be developed.
- NetCOBOL Studio is an integrated development environment designed for the COBOL language. The Java development environment and the plug-in development environment that are provided as Eclipse functions are not supported.

1.1.1 Using NetCOBOL

A typical installation of the NetCOBOL development package installs the necessary NetCOBOL Studio components.

Compatibility

In Eclipse, the upward compatibility of a workspace is guaranteed. When a workspace created in NetCOBOL Studio based on Eclipse 3.x is opened in NetCOBOL Studio based on Eclipse 4.3, it is converted automatically.

However, it is not backwards compatible. Workspaces that have been converted to Net COBOL Studio based on Eclipse 4.3 cannot be opened by NetCOBOL Studio based on Eclipse 3.x.

(*): Eclipse 3.x is 3.2 or 3.4.

When two or more developers and development machines share one workspace for development work, be sure both machines are using the latest version of NetCOBOL Studio.

1.1.2 Using Interstage Studio

The COBOL development function of NetCOBOL Studio can be used from Interstage Studio by integrating the COBOL plug-in into the Eclipse 3.4 workbench provided in Interstage Studio Standard-J Edition V10 or later.

Interstage Studio is not bundled with NetCOBOL. It must be purchased separately.

Installing the COBOL plug-in

Install the "COBOL plug-in for Interstage Studio" in addition to Interstage Studio and the NetCOBOL development package. Refer to "Interstage Studio COBOL Plug-in SOFTWARE RELEASE GUIDE".

Eclipse-based

Use Eclipse version is 3.4- from Interstage Studio.

Do not use the Eclipse 4.3 workbench specific functions. For details of the difference between the Eclipse 3.4 workbench and the Eclipse 4.3 workbench, see "Chapter 10 Differences between Eclipse 3.4 and Eclipse 4.3".

1.2 Functional Overview

NetCOBOL Studio supports the following COBOL program development functions:

- Development environment management functions

  NetCOBOL Studio manages the development environment and development resources of each COBOL program together in a unit called a project. Projects are managed in folders called workspaces.

  Note

  When developing a 32-bit COBOL application, use NetCOBOL Studio for Windows(x86). When developing a 64-bit COBOL application, use NetCOBOL Studio for Windows(x64).

- Edit functions

  NetCOBOL Studio provides many editing functions such as highlighting key words, displaying and resequencing line numbers, displaying horizontal, vertical, and overview rulers, displaying differences in source code, support of fixed and variable reference formats, automatic indentation, undo/redo, cut/copy/paste, find/replace, etc.

- Build and execution functions

  NetCOBOL Studio provides facilities to set COBOL compiler options and link options, and add version information and set icons for programs using the resource compiler. Tools for dependency analysis, automatic build, and manual build are provided. Compilation errors are displayed, and the line of source code containing a selected error can be displayed in the NetCOBOL Studio Editor.

- Debug functions

  The interactive debugger can detect logic errors in program processing. It can be used to set breakpoints to halt execution of a program, and to verify execution of a program by confirming data item values. The debugging of multithreaded programs is supported; however, the debugger does not support multiple debug sessions.

- Remote development functions

  COBOL programs for the Solaris OS, Linux (Itanium), Linux(64) or Windows(64) servers can be developed remotely. After performing standalone tests on a local computer, add the settings for remote development to the project for a smooth transition to test builds and links on a server.

NetCOBOL Studio users must understand the following main basic concepts:

Projects

NetCOBOL Studio manages the resources and information necessary for program development in individual units called "projects". NetCOBOL Studio creates and manages the following projects.

- COBOL projects

  A "COBOL project" is used for COBOL program development. One COBOL project manages the resources of one target (executable program or library). To develop a COBOL program, a COBOL project must first be created.

- COBOL resource projects

  A "COBOL resource project" is used for management of the library file and the descriptor file.

- COBOL solution projects

  A "COBOL solution project" is used for management of multiple projects. Specify the common compiler option for multiples projects.

You can easily create these projects interactively with the help of a wizard.

See

Project Import

A project in another workspace can be used by importing it. For details, refer to "C.3 Importing Project".

Workspaces

A workspace is a folder for storing one or more projects, and for storing setting information for a development environment shared by the projects.

For workspace setting and switching, refer to "Setting and Switching of Workspace". For the default workspace, refer to "C.1 Default workspace".

Note

- Projects for 32-bit and 64-bit COBOL applications cannot be created in the same workspace. Separate workspace folders must be specified for 32-bit and 64-bit COBOL projects.

- Workspaces cannot be shared between NetCOBOL Studio and Interstage Studio.

- The workspace created with NetCOBOL Studio cannot be opened with Interstage Studio. Also, the workspace created with Interstage Studio cannot be opened with NetCOBOL Studio.

Perspectives

The operation windows of NetCOBOL Studio consist of editor windows and multiple information display windows (each of these is called a "view"). The type and layout of a displayed view are managed under a concept called "perspectives" each of which is appropriate for the target work.

The COBOL perspective and Debug perspective, which has been prepared for debugging, are used in COBOL program development.

1.3 Starting NetCOBOL Studio

Start NetCOBOL Studio as follows:

Select the up arrow > **Apps** > **Fujitsu NetCOBOL V11(x64)** > **NetCOBOL Studio(x64)** from the **Start** menu.

If the start screen appears, select the **Run** button. The NetCOBOL Studio window is displayed. The contents of the start screen are shown below.

Table 1.1 NetCOBOL Studio start screen

Item	Description
Run	Starts NetCOBOL Studio. The folder name of the workspace that NetCOBOL Studio uses can be confirmed in the following way. - Place the mouse cursor on the **Run** button. - Click the **Setup** button and the workspace folder name is displayed.
Setup...	Used to make operating environment settings for NetCOBOL Studio. When the **Setup** button is clicked, the **Setup Configuration** dialog box appears. For information about setting the workspace folder, refer to "Appendix C Handling of Workspace and Project". For details on the environment settings, see the Help for the **Setup Configuration** dialog box.
Cancel	Closes the start screen and does not start NetCOBOL Studio.

When the **Always show Launch dialog** checkbox is selected in the **Setup Configuration** dialog box, this screen is displayed.

Table 1.2 "Setup Configuration" dialog box

Item	Description
Workspace folder name	Used to select the folder for the workspace. NetCOBOL Studio uses the selected workspace folder. To change the workspace folder, - Select the folder name and push the F2 key, or - Edit by clicking the left mouse button. The following characters cannot be used for the workspace folder name. `"#", "%", ";"`
Add	Adds the existing workspace folder to the workspace list. The dialog box displayed is varies according to the operating system and the Internet Explorer version. When specify a non-existent folder name, a new workspace folder is created.
Delete	Deletes the workspace folder name from the workspace list. Even if the workspace folder name is deleted from the list, the actual folder is not deleted. Use Explorer to delete unnecessary folders.
Execution option	Used to specify a startup option for NetCOBOL Studio. For a normal startup, do not specify an option. -clean Deletes the cache data used by Eclipse. When a plug-in is added, updated, or deleted, you should start NetCOBOL Studio by specifying this option. -nosplash Does not display the splash screen at start-up of NetCOBOL Studio. -showlocation Displays the workspace location in the NetCOBOL Studio title bar. -vmargs JavaVMoption Specifies the option for JavaVM. Only "-Xmx option" and "-XX:MaxPermSize option" can be specified for *JavaVMoption*. NetCOBOL Studio assumes that "-Xmx512M -XX:MaxPermSize=256M" is specified.

Item	Description
	For option details, refer to the NetCOBOL Studio help. "Workbench User's Guide" > "Task" > "Running Eclipse" > "Eclipse Startup Parameters" Note: In NetCOBOL Studio, only the above option can be used.
Always show Launch dialog	When checked, the start screen is displayed. When not checked, the start screen is not displayed. In a non-display state, when NetCOBOL Studio is started while holding the shift key, the start screen is displayed.

1.4 Development Flows

This section explains the development work patterns and flows of COBOL programming using NetCOBOL Studio.

1.4.1 Development patterns

NetCOBOL Studio provides two patterns of development:

- Local development flow

  Development pattern for a COBOL program that runs on a local personal computer.

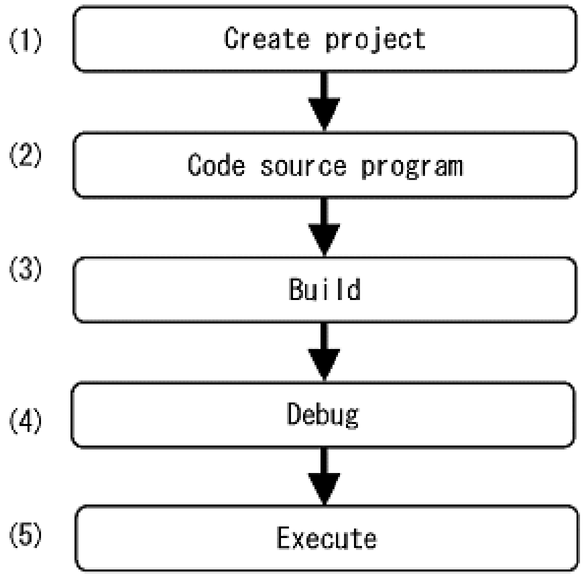
- Remote development flow

  Development pattern for a COBOL program that runs on a server.

  Remote development requires that information specific to the target server be set in each project. A project with a local development pattern can be changed to a remote development pattern by adding server information to the project.

1.4.2 Local development flow

This section explains the flow of COBOL program development on a local personal computer.



1. Create a project

   NetCOBOL Studio manages each COBOL program under development as a "project." The resources (e.g., source files, libraries) required for COBOL program development are managed in blocks in units of projects. A single target (executable program or dynamic link library) is managed with each project. Using a wizard, projects can be defined easily and efficiently. Precompiler settings (e.g. command, parameter) can be specified for COBOL programs including input sources for the precompiler.

2. Code a source program

   NetCOBOL Studio provides a wizard for creating source program templates. Various Assist functions of the COBOL editor can be used to edit COBOL source programs.

3. Build

   Compiling and linking are performed according to the information defined in the project (e.g., compile options, link options).

4. Debug

   The interactive debugger provides various functions, such as setting program breakpoints and referencing and setting data item values. These functions enable the programmer to efficiently debug the COBOL program.

5. Execute

   The program can be executed after the runtime environment information required for COBOL program execution is set.

1.4.3 Remote development flow

For details on remote development of a COBOL program, refer to 9.1.3 Flow of Remote Development".

Chapter 2 Tutorial

This chapter explains the basic operation flow of NetCOBOL Studio using examples.

2.1 Creating a COBOL Program

This section explains the procedure for creating a COBOL program.

Point

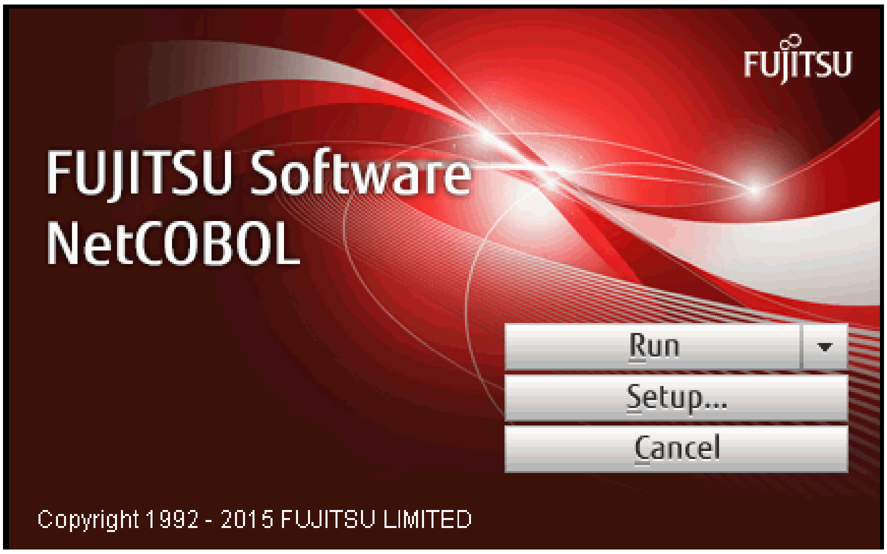
The "NetCOBOL Getting Started Guide", explains the procedures for importing sample projects to the workspace. Procedures to create a new project called SAMPLE1 are explained here.

1. Start NetCOBOL Studio

1. This section explains how to start NetCOBOL Studio. An executable file is created from a COBOL source program in the project in this example.

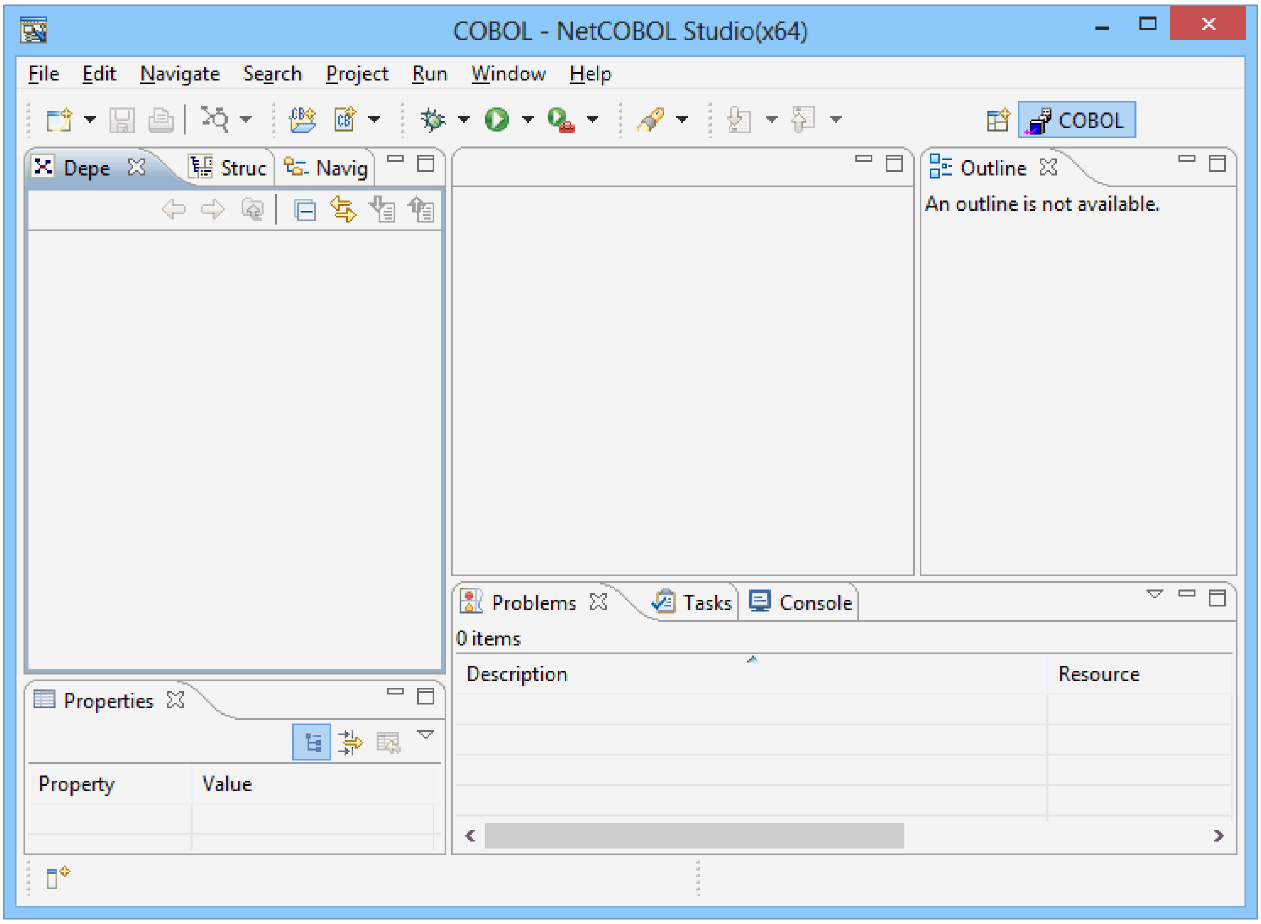
   Select the up arrow > **Apps** >**Fujitsu NetCOBOL V11(x64)** > **NetCOBOL Studio(x64)** from the **Start** menu.

   The start screen appears when NetCOBOL Studio is started.



2. Click the **Setup...** button to set the workspace. For details about setting a workspace, refer to "C.2.1 Setting workspace". When a workspace is not set, the default workspace is used. For details about the default workspace, refer to "C.1 Default workspace".

3. Click the **Run** button. The NetCOBOL Studio window is displayed.


COBOL - NetCOBOL Studio(x64)

An outline is not available.

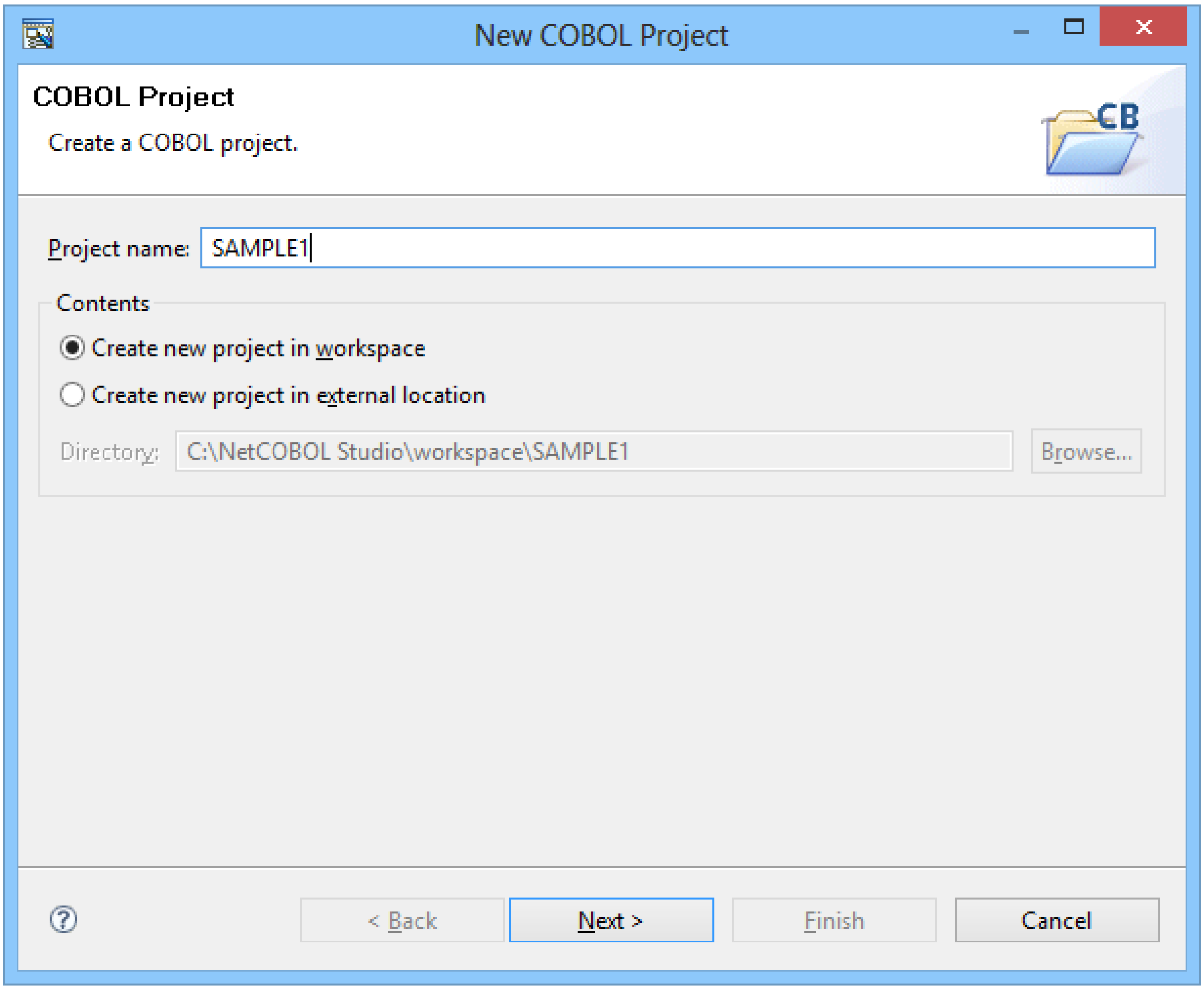
0 items

2. Create a project

1. To create a COBOL project:

   Select **File** > **New** >**COBOL Project** from the menu bar. The **New COBOL Project** wizard is displayed.

2. Specify a project name and a storage folder on the second page.



New COBOL Project

COBOL Project

Create a COBOL project.

Project name: SAMPLE1

Contents

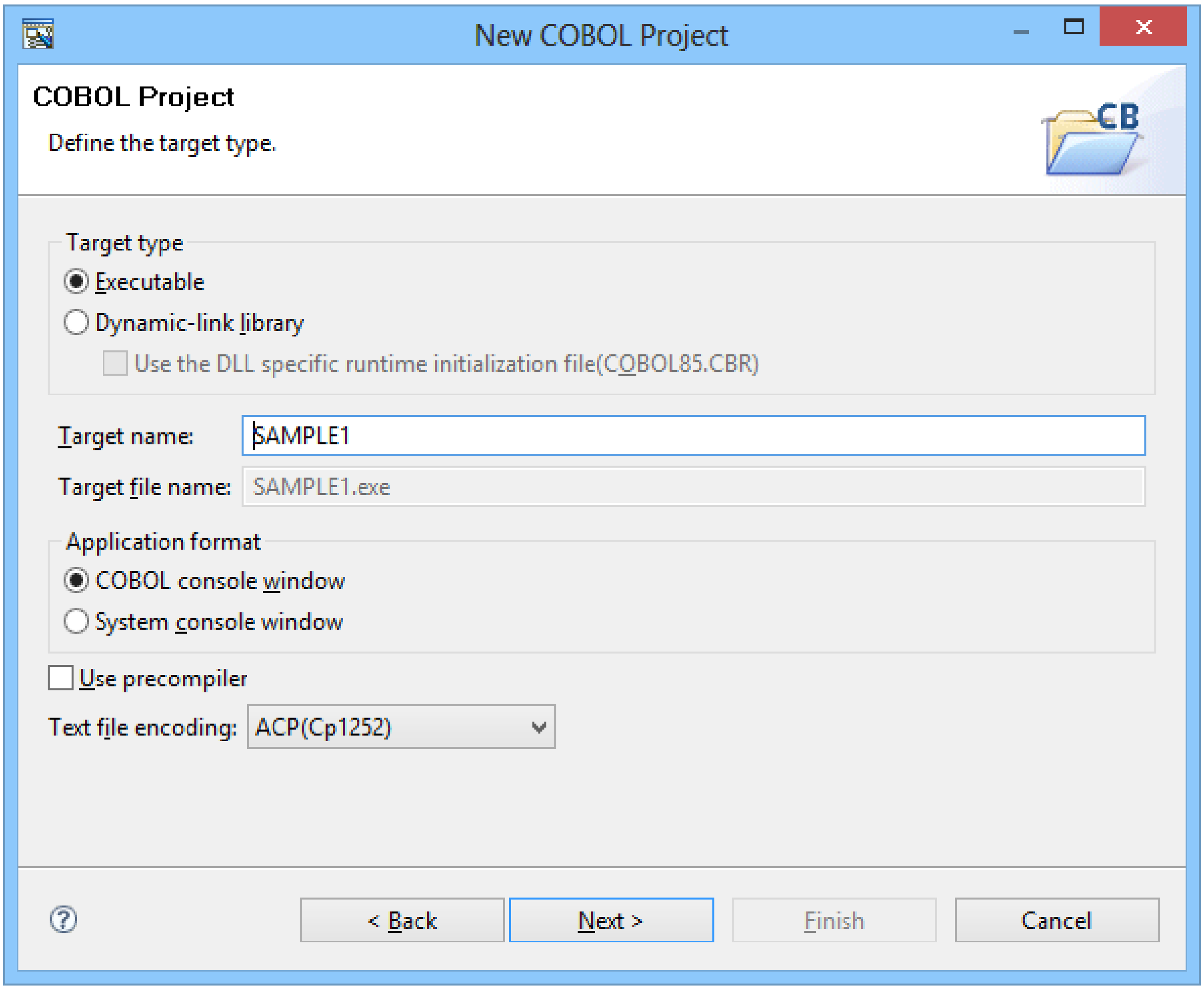
● Create new project in workspace

○ Create new project in external location

Directory: C:\NetCOBOL Studio\workspace\SAMPLE1

Field	Entry
Project name	SAMPLE1
Contents	Create new project in workspace

3. This page is used to specify information on the target. Leave the defaults and click the **Next** button.



New COBOL Project

COBOL Project

Define the target type.

Target type

● Executable

○ Dynamic-link library

□ Use the DLL specific runtime initialization file(COBOL85.CBR)

Target name: SAMPLE1

Target file name: SAMPLE1.exe

Application format

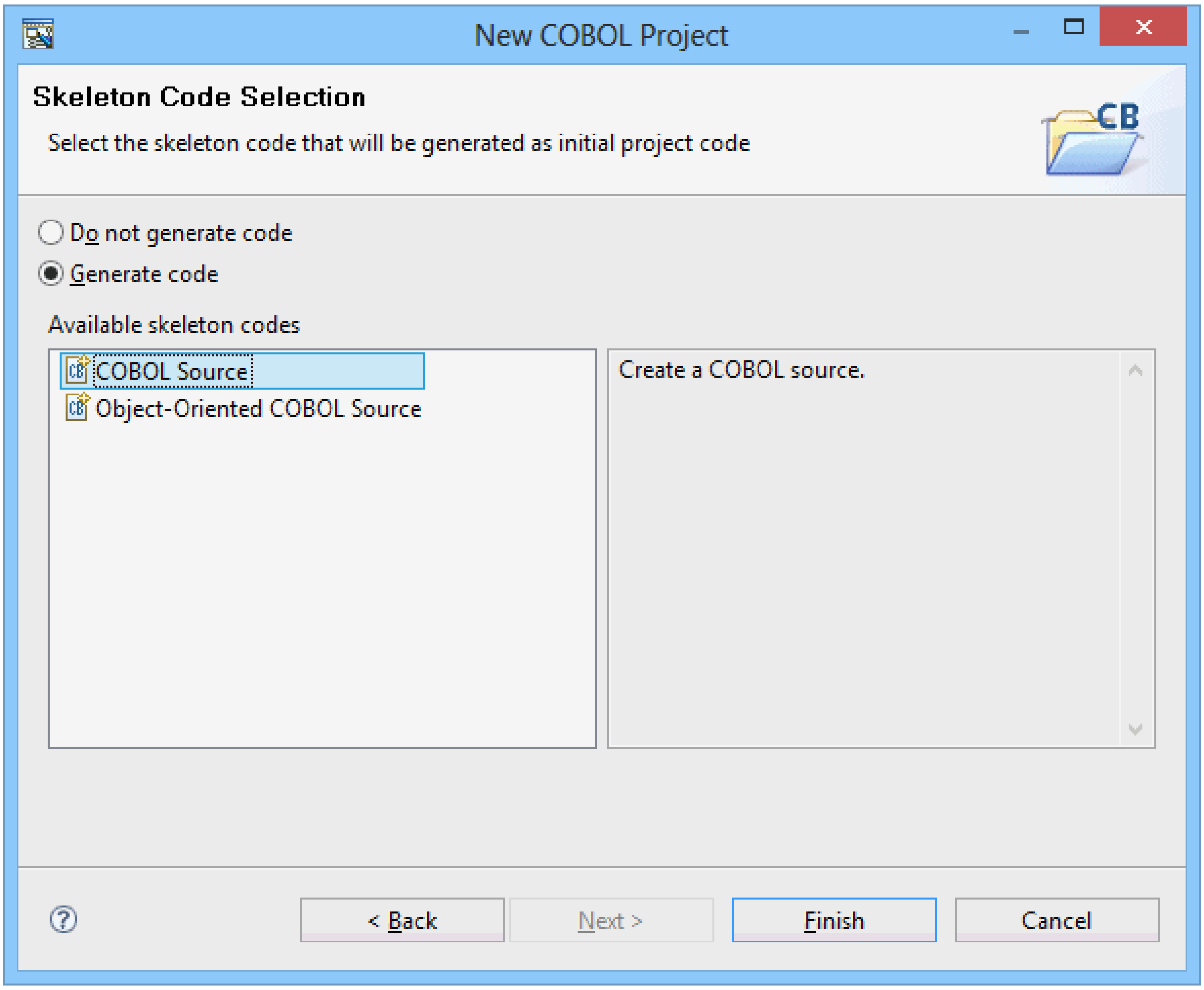
● COBOL console window

○ System console window

□ Use precompiler

Text file encoding: ACP(Cp1252)

4. This page is used to select the type of source code to be created. Select **COBOL Source** in the **Available skeleton codes** section, and then click on the **Finish** button.



New COBOL Project

Skeleton Code Selection

Select the skeleton code that will be generated as initial project code

○ Do not generate code

● Generate code

Available skeleton codes

- COBOL Source
- Object-Oriented COBOL Source

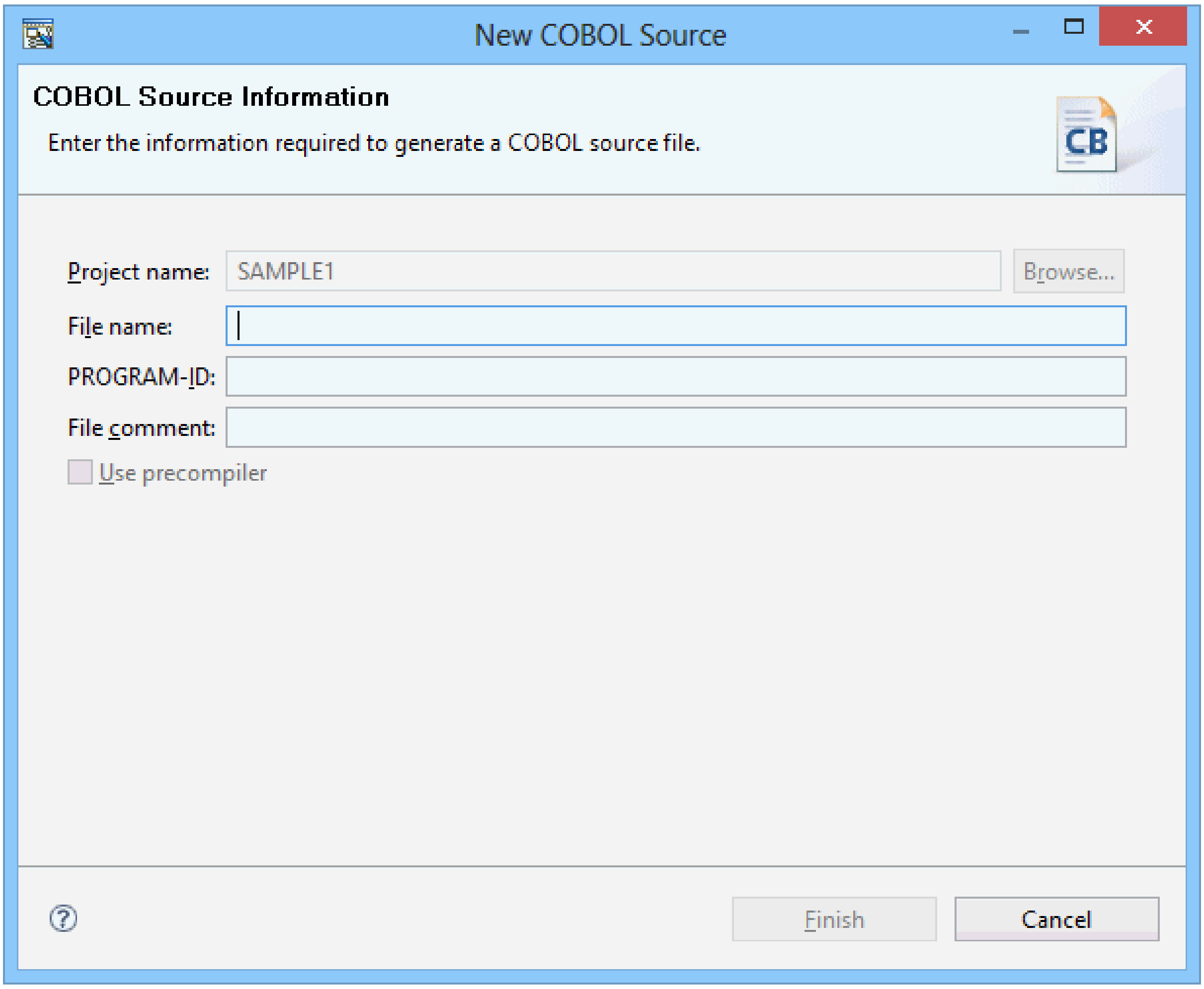
Create a COBOL source.

Note

Projects for 32-bit and 64-bit COBOL applications cannot be created in the same workspace. Separate workspace folders must be specified for 32-bit and 64-bit COBOL projects.

3. Create a template by using the COBOL source generation wizard

Using the **New COBOL Source** wizard, create a template of a COBOL source file.



COBOL Source Information

Enter the information required to generate a COBOL source file.

Project name: SAMPLE1

File name:

PROGRAM-ID:

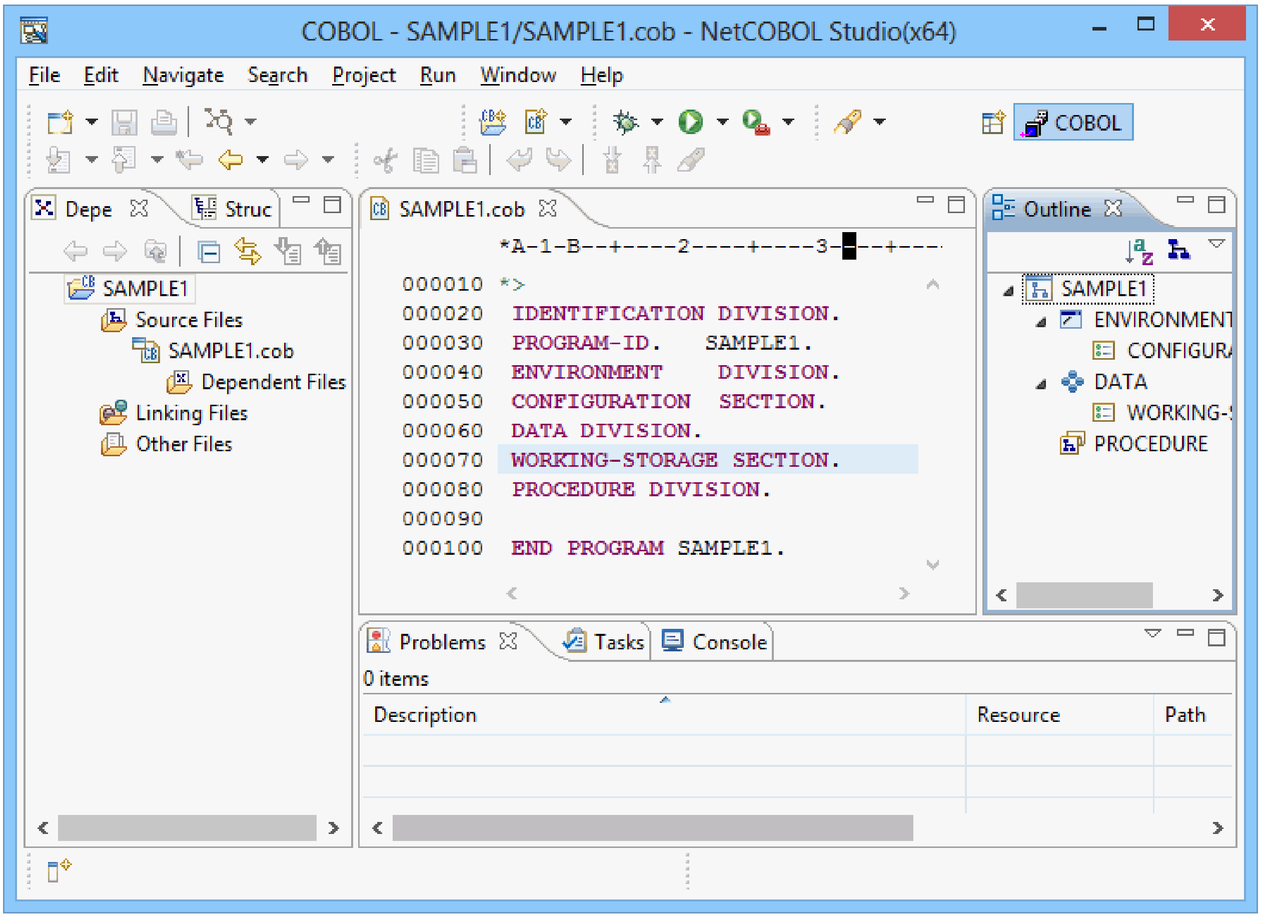
File comment:

□ Use precompiler

Make the following entries in the fields on this screen:

Field	Entry
Project name	SAMPLE1
File name	SAMPLE1
PROGRAM-ID	SAMPLE1
File comment	Optional: Enter a comment related to the COBOL source file.

Click on the **Finish** button. The SAMPLE1.cob file is created and opened in the COBOL editor.



```
 *A-1-B--+----2----+----3----+---
000010 *>
000020  IDENTIFICATION DIVISION.
000030  PROGRAM-ID.    SAMPLE1.
000040  ENVIRONMENT    DIVISION.
000050  CONFIGURATION  SECTION.
000060  DATA DIVISION.
000070  WORKING-STORAGE SECTION.
000080  PROCEDURE DIVISION.
000090
000100  END PROGRAM SAMPLE1.
```

4. Edit the program

Edit the COBOL source file that was created using the wizard. Add and modify data items and procedures as desired.

WORKIGN-STORAGE SECTION

```
 01  WORD-VALUES.
     02      PIC X(10) VALUE "apple".
     02      PIC X(10) VALUE "black".
     02      PIC X(10) VALUE "cobol".
     02      PIC X(10) VALUE "dog".
     02      PIC X(10) VALUE "eye".
     02      PIC X(10) VALUE "fault".
     02      PIC X(10) VALUE "good".
     02      PIC X(10) VALUE "high".
     02      PIC X(10) VALUE "idea".
     02      PIC X(10) VALUE "junior".
     02      PIC X(10) VALUE "king".
     02      PIC X(10) VALUE "love".
     02      PIC X(10) VALUE "medium".
     02      PIC X(10) VALUE "new".
     02      PIC X(10) VALUE "open".
     02      PIC X(10) VALUE "pig".
     02      PIC X(10) VALUE "queen".
     02      PIC X(10) VALUE "review".
     02      PIC X(10) VALUE "smile".
     02      PIC X(10) VALUE "tomorrow".
     02      PIC X(10) VALUE "understand".
     02      PIC X(10) VALUE "version".
     02      PIC X(10) VALUE "wood".
     02      PIC X(10) VALUE "xylophone".
```

```
      02     PIC X(10) VALUE "yesterday".
      02     PIC X(10) VALUE "zoo".
      02     PIC X(10) VALUE "**error**".
  01  WORD-TABLE REDEFINES  WORD-VALUES.
      02  WORD-ITEM OCCURS 27 TIMES.
         03  FIRST-CHARACTER PIC X.
         03                 PIC X(9).
  01  WORD-INDEX          PIC 9(3).
  01  INPUT-CHARACTER     PIC X.
  01  REQUEST-MESSAGE     PIC X(42)
       VALUE  "ENTER ONE CHARACTER OF ALPHABETIC-LOWER.=>".
```

PROCEDURE DIVISION

```
  DATA-INPUT  SECTION.
**(1)DISPLAY THE ABOVE MESSAGE
       DISPLAY REQUEST-MESSAGE WITH NO ADVANCING.
**(2)ACCEPT THE INPUT CHARACTER
       ACCEPT  INPUT-CHARACTER.
*
  SEARCH-WORD  SECTION.
**(3)WORDS ARE SEARCHED CORRESPONDING TO THE INPUT CHARACTER
       PERFORM TEST BEFORE
               VARYING WORD-INDEX FROM 1 BY 1
               UNTIL WORD-INDEX > 26
          IF INPUT-CHARACTER = FIRST-CHARACTER (WORD-INDEX)
            THEN EXIT PERFORM
           END-IF
        END-PERFORM.
*
  WORD-OUTPUT  SECTION.
**(4) THE RESULTING WORD MATCH IS DISPLAYED
       DISPLAY WORD-ITEM (WORD-INDEX).
*
       EXIT PROGRAM.
```

After completing the editing, select **Save** from the context menu in the COBOL editor.

5. Build the project

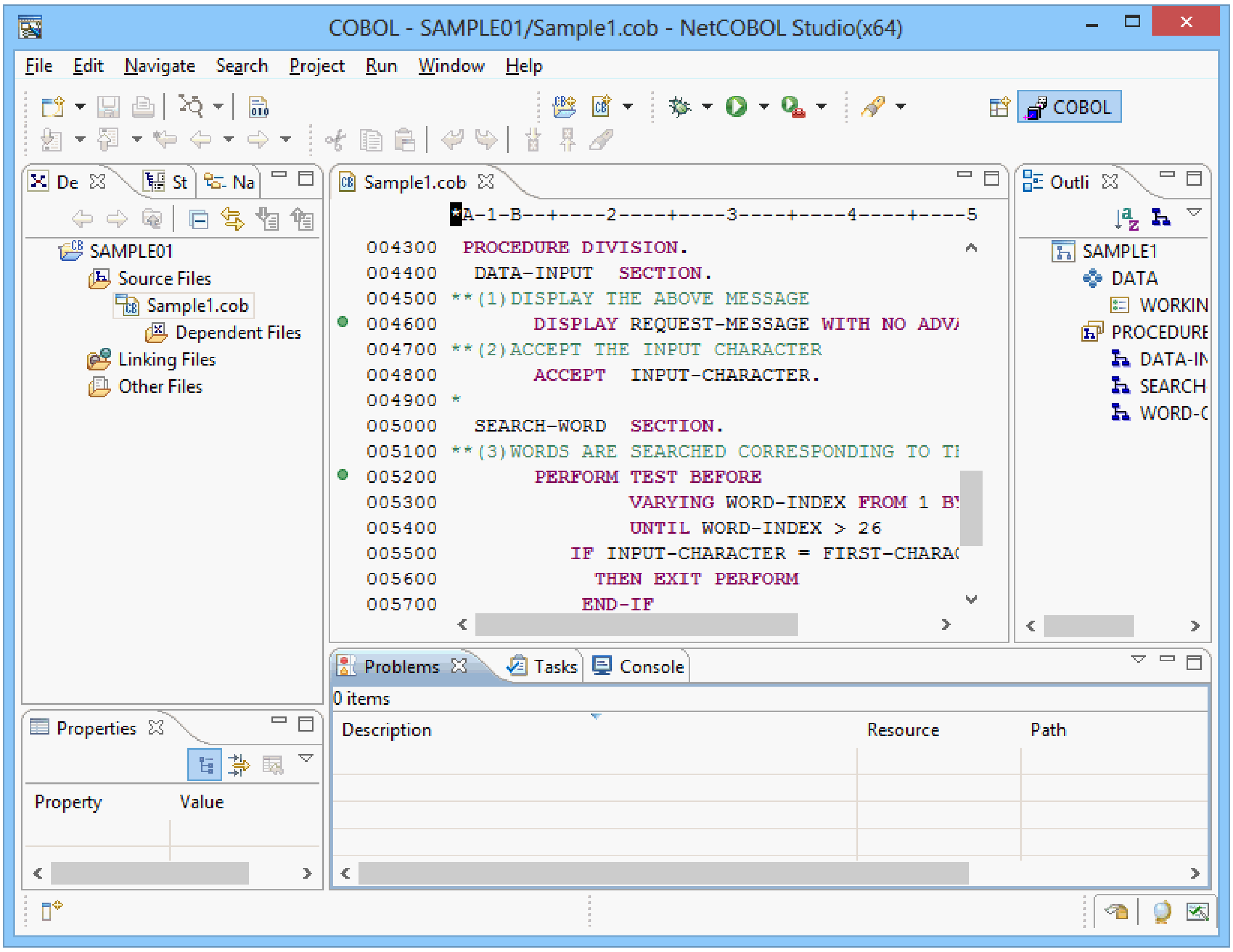
If **Project** > **Build Automatically** in the menu bar is enabled, the COBOL program will be built automatically when the source program is saved in the previous step. To manually execute the build, or to execute the build if the automatic build was cancelled, select **Project** > **Build Project** from the menu bar.

6. Start the debugger

Set Breakpoints

Before starting the debugger, set breakpoints as follows:

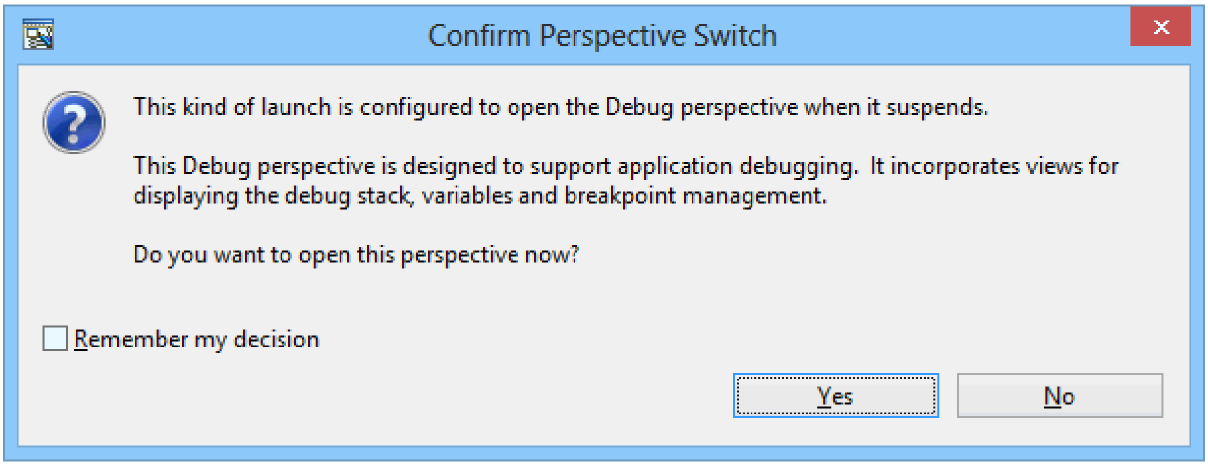
On the vertical ruler at the left edge of the COBOL editor window, position the cursor on the line where a breakpoint is to be set and double-click the left mouse button. A mark indicating a set breakpoint (●) is displayed on the vertical ruler.



```
       A-1-B--+----2----+----3----+----4----+----5
004300  PROCEDURE DIVISION.
004400   DATA-INPUT  SECTION.
004500 **(1)DISPLAY THE ABOVE MESSAGE
004600        DISPLAY REQUEST-MESSAGE WITH NO ADVA
004700 **(2)ACCEPT THE INPUT CHARACTER
004800        ACCEPT  INPUT-CHARACTER.
004900 *
005000   SEARCH-WORD  SECTION.
005100 **(3)WORDS ARE SEARCHED CORRESPONDING TO TI
005200        PERFORM TEST BEFORE
005300                VARYING WORD-INDEX FROM 1 BY
005400                UNTIL WORD-INDEX > 26
005500           IF INPUT-CHARACTER = FIRST-CHARAC
005600             THEN EXIT PERFORM
005700            END-IF
```

Start the debugger

1. After completing the setting of breakpoints, start the debugger:
   Select **Run** > **Debug As** > **COBOL Application** from the menu bar.
2. A message is displayed to confirm switching to the Debug perspective when the first breakpoint is reached. Click the **Yes** button.



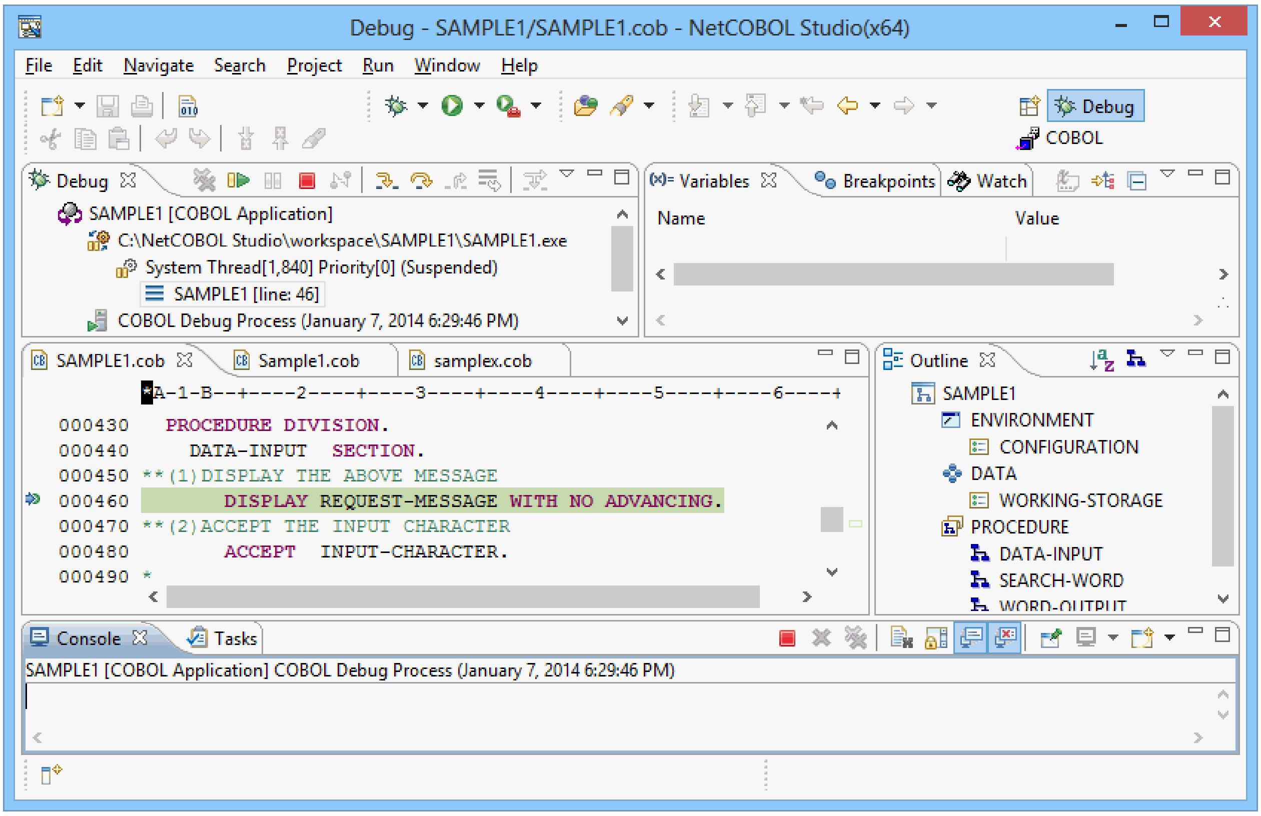
This kind of launch is configured to open the Debug perspective when it suspends.

This Debug perspective is designed to support application debugging. It incorporates views for displaying the debug stack, variables and breakpoint management.

Do you want to open this perspective now?

□ Remember my decision

3. The Debug perspective is displayed, and processing is stopped at the first breakpoint.



SAMPLE1 [COBOL Application]
C:\NetCOBOL Studio\workspace\SAMPLE1\SAMPLE1.exe
System Thread[1,840] Priority[0] (Suspended)
SAMPLE1 [line: 46]
COBOL Debug Process (January 7, 2014 6:29:46 PM)

```
       *A-1-B--+----2----+----3----+----4----+----5----+----6----+
000430  PROCEDURE DIVISION.
000440    DATA-INPUT  SECTION.
000450**(1)DISPLAY THE ABOVE MESSAGE
000460       DISPLAY REQUEST-MESSAGE WITH NO ADVANCING.
000470**(2)ACCEPT THE INPUT CHARACTER
000480       ACCEPT  INPUT-CHARACTER.
000490*
```

SAMPLE1 [COBOL Application] COBOL Debug Process (January 7, 2014 6:29:46 PM)

Change of value of data item

1. Change a value of the "INPUT-CHARACTER" data item.
   Select the "INPUT-CHARACTER" data item in the COBOL editor, and select **Add to Watch View** from the context menu. "INPUT-CHARACTER" is added into the **Watch view**.

2. Select the "INPUT-CHARACTER" data item in the **Watch view**, and select **Change Value** from the context menu. The **Set Value** dialog box appears.

3. Change a value in the **Set Value** dialog box appears. Click the **OK** button.

Execution restart

Click either of the following buttons from the **Debug view** and the application is executed.

- Resume
- Step Into
- Step Over

For additional information, refer to "7.2.1 Debug view".

Point

After completing debugging, select **Window** > **Open Perspective** > **COBOL** from the menu bar to return to the COBOL perspective.

7. Execute the COBOL program

To execute the COBOL program you created, select **Run** > **Run As** > **COBOL Application** from the menu bar.

2.2 Creating a COBOL Program by Remote Development

This section explains the procedure for converting an existing COBOL project into a project under remote development, and then creating a COBOL program for a server by remote development.

1. Make server environment settings

2. Set server information

3. Set server information for the project

4. Create a makefile

5. Build the project on the server

6. Debug the COBOL program

1. Make server environment settings

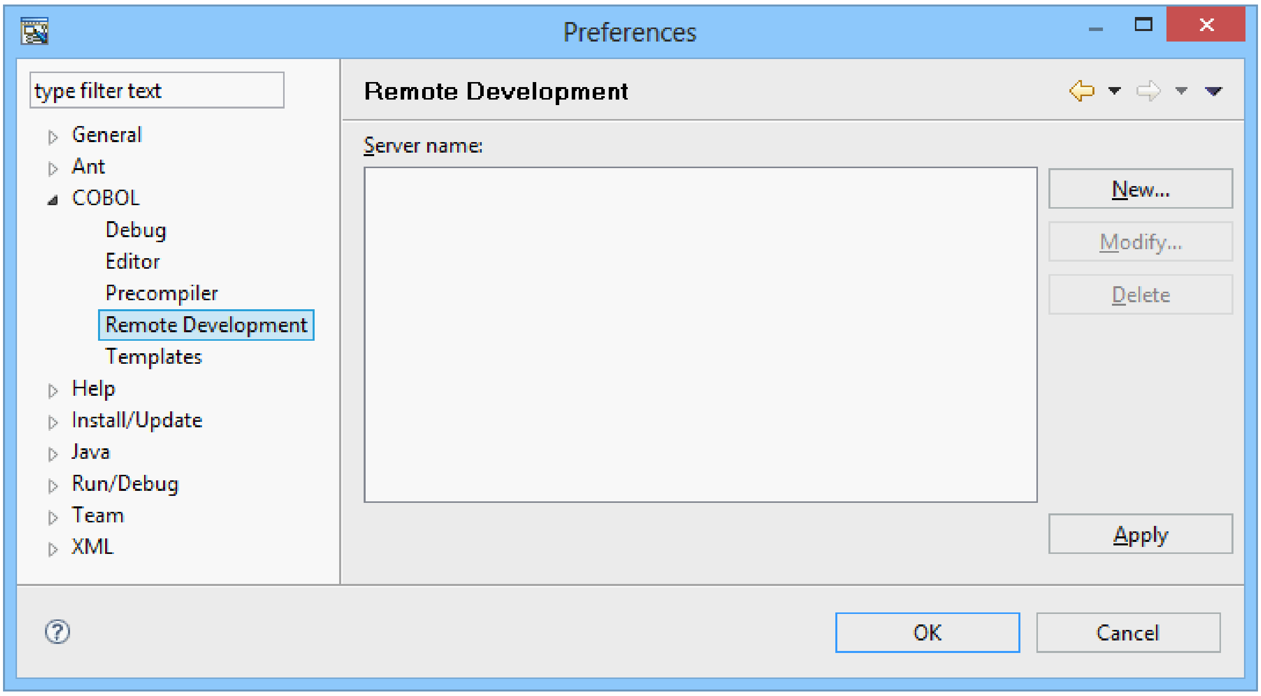
Make environment settings for the server to be used for remote development. Check with your server administrator. For details on environment settings of a server, see "9.2.1.1 NetCOBOL Remote Development Service" and "9.2.2 Setting up the user environment on the server."

2. Set server information

Set information for linkage with the server used for remote development.

1. Select **Window** > **Preferences** from the menu bar. When the **Preferences** dialog box appears, select **COBOL** > **Remote Development**. The **Remote Development** page is displayed.

2. Select the **New** button on the **Remote Development** page.



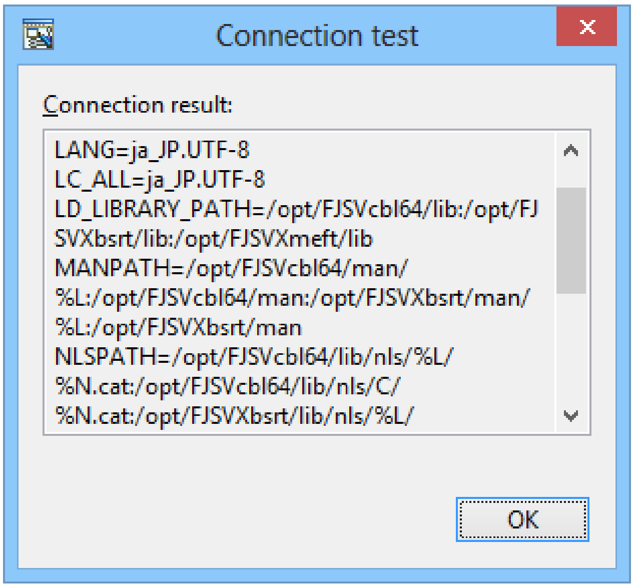
The **New Server Information** dialog box appears.

3. Set the information required for linking to the server as follows:

Field	Entry
Server name	Enter an arbitrary name that will be used to manage server information.

Field	Entry
Server OS	Select the operating system of the server used for remote development.
Server address	Enter a name (FQDN: Fully Qualified Domain Name) or IP address that identifies the server in the network.
Save user name and password	Check this checkbox.
User name	Enter the user name of the account used to access the server.
Password	Enter the password assigned to the user.
Character-code conversion	This section is for the specification of text file code conversion information.
COBOL Source character-code on the server side	Select the character-code of the COBOL source file transferred to the server side of remote development.
Convert character set in the server	Select **Convert character set in the server**, since code conversion processing is executed on the server. This field is disabled if "Windows(x64)" is selected at **Server OS** list.
Convert character set in the client	
Information for UNIX Server system	This information should be configured if Solaris, Linux(x86), Linux(Itanium), or Linux(x64) is specified at **Server OS** list**.**
FTP is used for remote development	If this field is selected, use the ftpd/rexec service as the remote development server-side service.
Use PASV mode for file transfer(FTP)	Check this box only if file transfer will be in PASV mode. This field is enabled if remote development uses the server-side ftpd/rexec service.
NetCOBOL Remote Development Service on server side	This information is for the server-side NetCOBOL remote development service.
Port number	Specify the TCP/IP port number of the NetCOBOL remote development service.
SSH Port Forwarding	The connection to NetCOBOL remote development services are encrypted. When "Linux(x64)" is selected at **Server OS** list, this item is enabled.
SSH Port number	Specify the SSH Port number. The default value is 22. When "SSH Port Forwarding" is selected, this item is enabled.
SSH Server FingerPrint	Specify the FingerPrint by hexadecimal digit string. Ask the server administrator about the value. When "SSH Port Forwarding" is selected, this item is enabled.
Algorithm	Specify the algorithm of SSH Server FingerPrint (either sha1 or md5). Ask the server administrator about the value. The default value is md5. When "SSH Port Forwarding" is selected, this item is enabled.

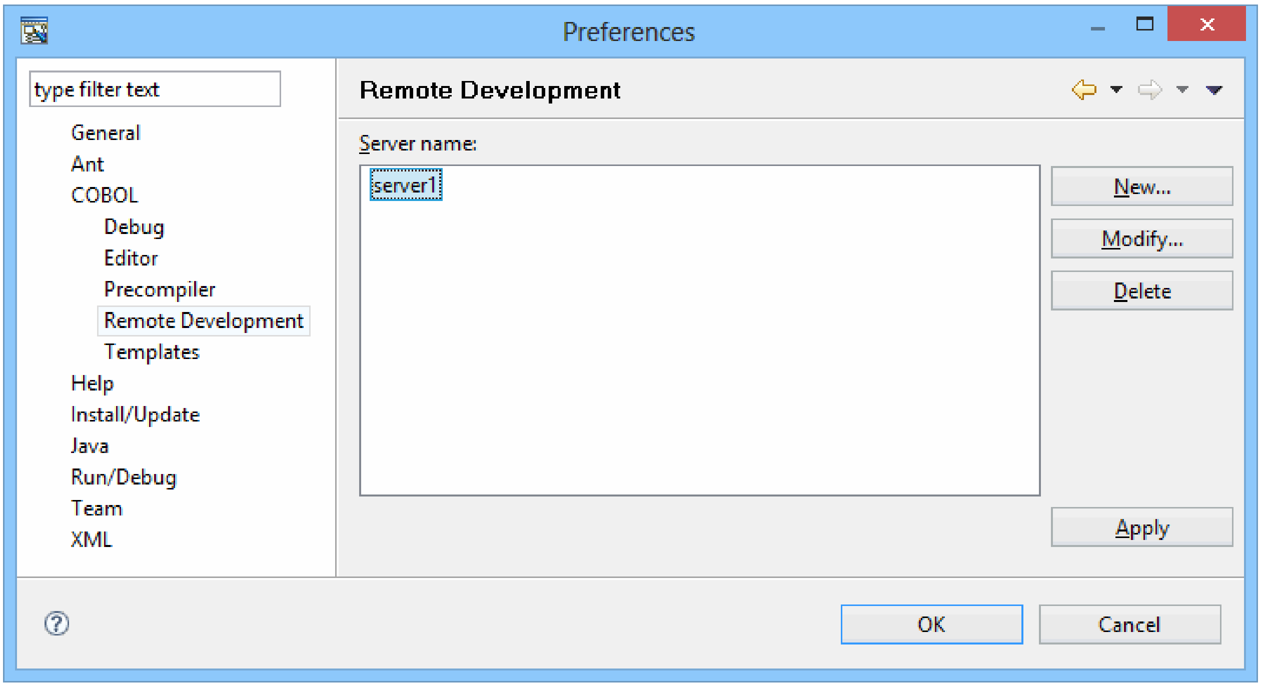
4. After setting the information in the **New Server Information** dialog box, click the **Connection test** button. When the information is correct, environmental variable information for the server is displayed.



```
LANG=ja_JP.UTF-8
LC_ALL=ja_JP.UTF-8
LD_LIBRARY_PATH=/opt/FJSVcbl64/lib:/opt/FJ
SVXbsrt/lib:/opt/FJSVXmeft/lib
MANPATH=/opt/FJSVcbl64/man/
%L:/opt/FJSVcbl64/man:/opt/FJSVXbsrt/man/
%L:/opt/FJSVXbsrt/man
NLSPATH=/opt/FJSVcbl64/lib/nls/%L/
%N.cat:/opt/FJSVcbl64/lib/nls/C/
%N.cat:/opt/FJSVXbsrt/lib/nls/%L/
```

Click the **OK** button in the **Connection test** dialog box to return to the **New Server Information** dialog box.

5. Click the **OK** button in the **New Server Information** dialog box. The **Preferences** dialog box appears. The server name entered in the **New Server Information** dialog box is displayed in *Server name* on the **Remote Development** page.



Server name: server1

Click the **OK** button to close the **Preferences** dialog box.

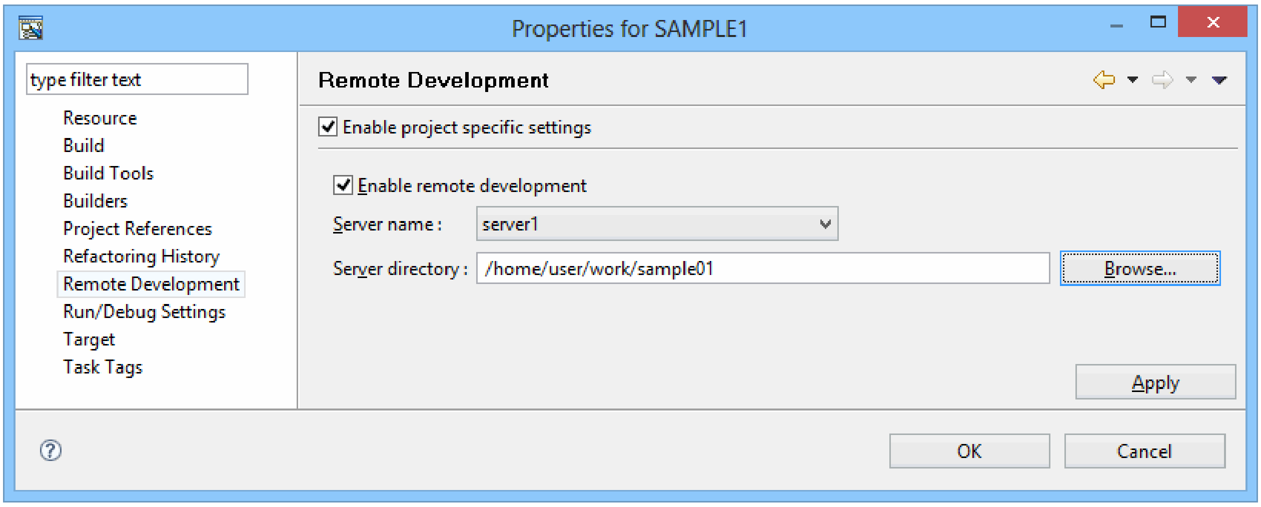
Point

Since the server information set in this step is shared among workspaces, the information can be used from another workspace.

3. Set server information for the project

To convert a COBOL project into one under remote development, set the server information for the project.

1. Select the project in the **Dependency view** or **Structure view**, and select **Property** from the context menu. The property dialog box appears.
2. Select **Remote Development** in the left pane of the property dialog box. The **Remote Development** page is displayed.
3. Set the server information for the project as follows:



☑ Enable project specific settings

☑ Enable remote development

Server name : server1

Server directory : /home/user/work/sample01

Field	Entry
Enable project specific setting	This checkbox must be checked.
Enable remote development	This checkbox must be checked.
Server name	Select the server name that was entered in the "Set server information".
Server directory	Specify the full path name of the storage directory of the resources used for remote development. The server directory can be selected by clicking the **Browse** button. The makefile creation function and remote build function use this directory as the current directory for processing.

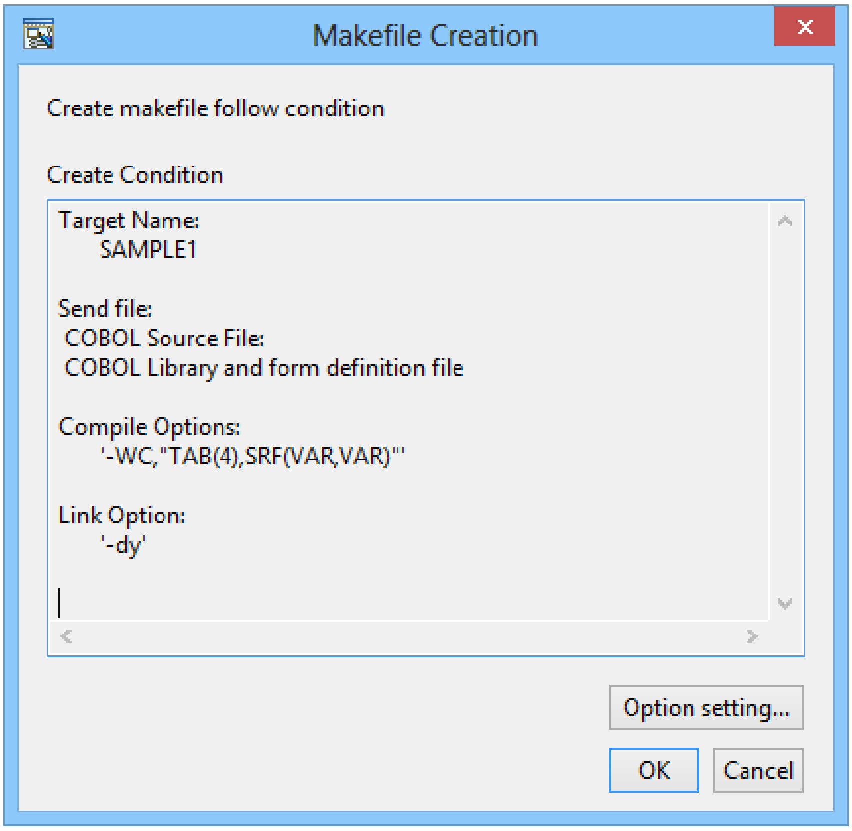
Click the **OK** button to close the property dialog box.

4. Create a makefile

Create a makefile that will be used to build a COBOL program on the server.

1. Select **Project** > **Remote Development** > **Makefile Creation...** from the menu bar. The **Makefile Creation** dialog box appears.

2. Leave the defaults and click the **OK** button.



Makefile Creation

Create makefile follow condition

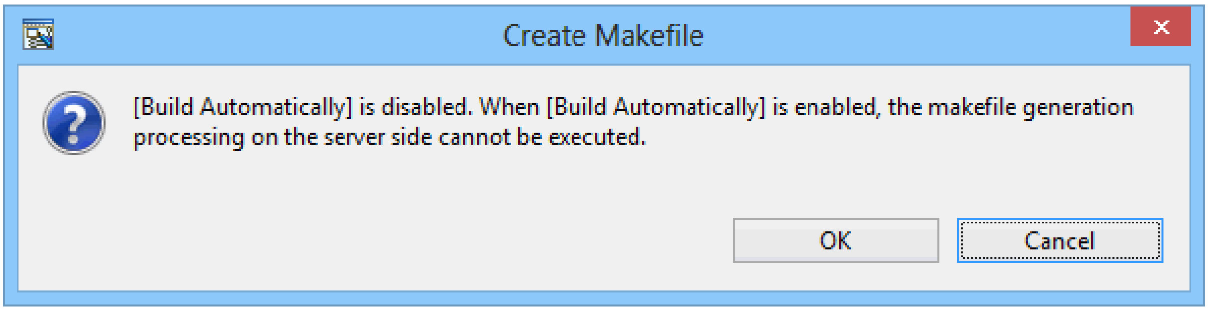
Create Condition

```
Target Name:
      SAMPLE1

Send file:
 COBOL Source File:
 COBOL Library and form definition file

Compile Options:
      '-WC,"TAB(4),SRF(VAR,VAR)"'

Link Option:
      '-dy'
```

The resources that are required for creating the makefile are sent to the server, and the makefile is created. The created makefile is named "Makefile" and registered in the **Other Files** folder. To view the contents of the makefile, select the makefile and select **Open** from the context menu.

Note

The following confirmation message may be displayed.



Create Makefile

[Build Automatically] is disabled. When [Build Automatically] is enabled, the makefile generation processing on the server side cannot be executed.

Click the **OK** button to disable the automatic build.

Point

The makefile creation results on the server can be checked by selecting **COBOL Remote** from the **Open Console** icon on the toolbar in the **Console view.**

5. Build the project on the server

To build the COBOL program on the server, select **Project** > **Remote Development** > **Build** from the menu bar.

Point

- Compilation errors are displayed in the **Problems view**. Select a compilation error in the **Problem view**, and select **Go To** from the context menu. The COBOL editor opens the COBOL source file and displays it with the current line at the error location.
- The build results on the server can be checked by selecting **COBOL Remote** from the **Open Console** icon on the toolbar in the **Console view**.

6. Debug the COBOL program

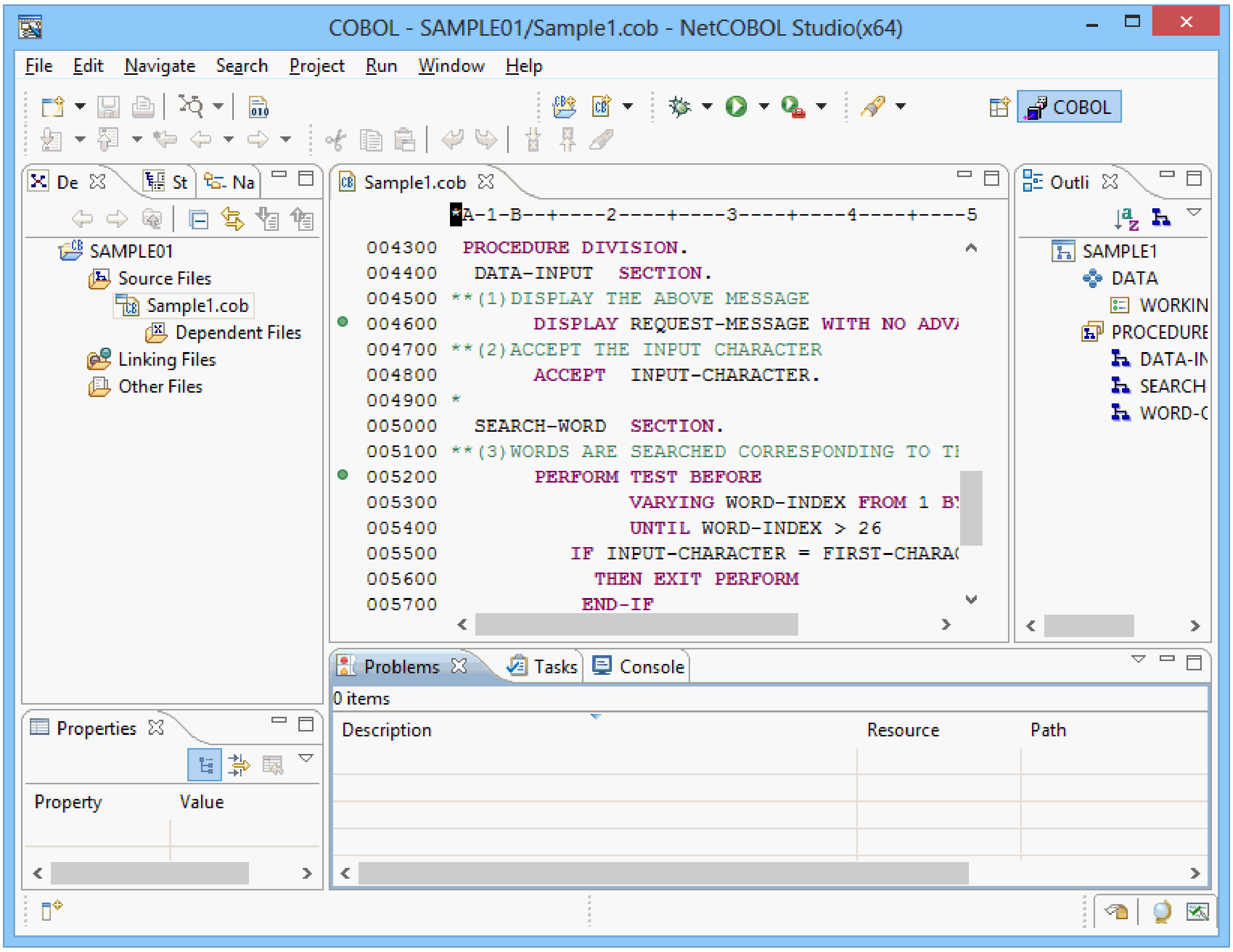
To debug the COBOL program running on the server, launch the NetCOBOL interactive remote debugger and start debugging. Use the remote debugger connector, which is a tool for connecting a client to the server.

1. To start remote debugging, execute one of the commands listed below on the server to start the remote debugger connector.

Server	Start command
Windows(x64)	cobrds64
Solaris	svdrds
Linux(Itanium)	
Linux(64)	
Linux	

2. Before starting the debugger, set breakpoints as follows:

   On the vertical ruler at the left edge of the COBOL editor window, position the cursor on the line where a breakpoint is to be set and double-click the left mouse button. A mark indicating a set breakpoint (●) is displayed on the vertical ruler.



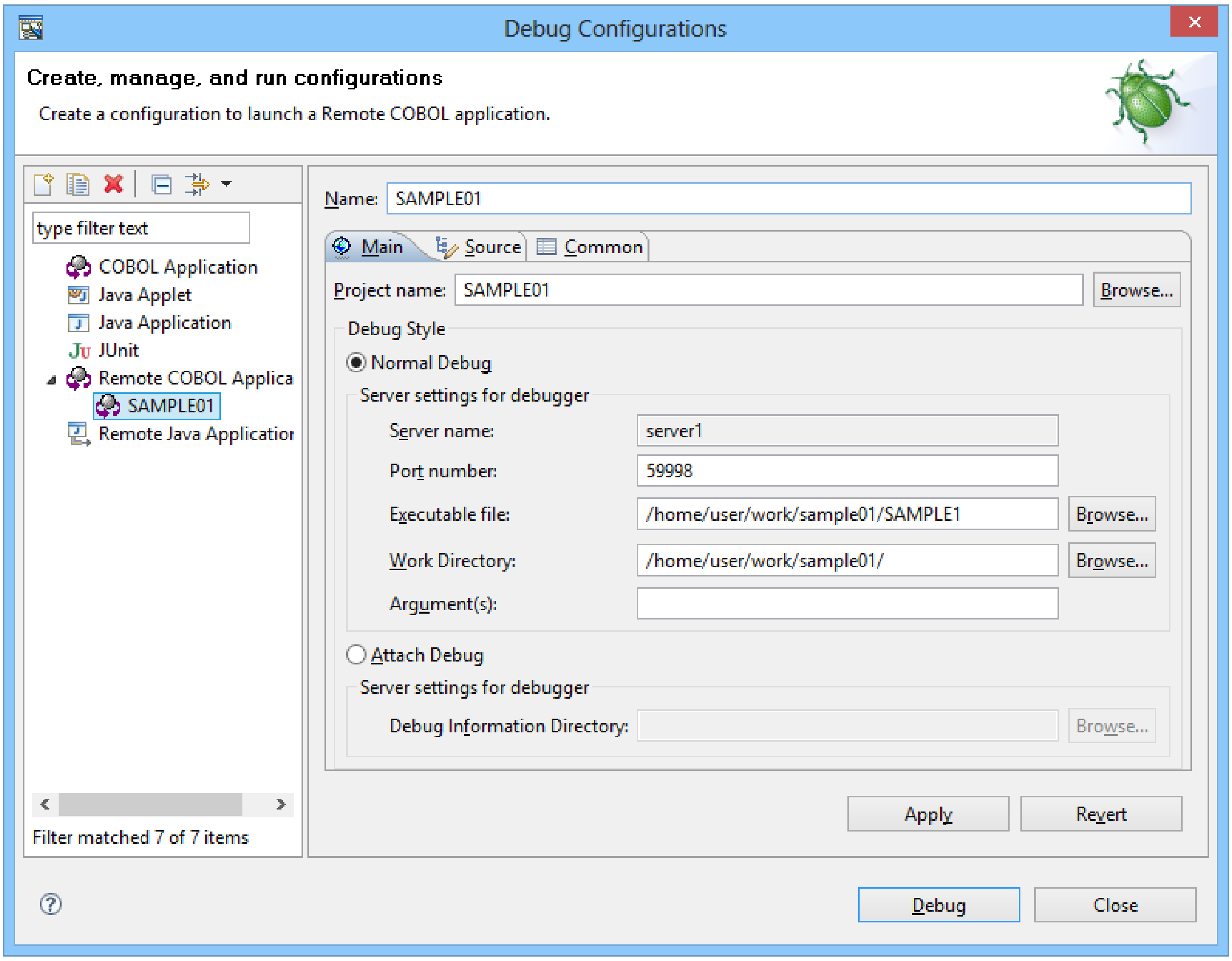
```
      *A-1-B--+----2----+----3----+----4----+----5
004300 PROCEDURE DIVISION.
004400  DATA-INPUT   SECTION.
004500**(1)DISPLAY THE ABOVE MESSAGE
004600       DISPLAY REQUEST-MESSAGE WITH NO ADV/
004700**(2)ACCEPT THE INPUT CHARACTER
004800       ACCEPT  INPUT-CHARACTER.
004900*
005000  SEARCH-WORD  SECTION.
005100**(3)WORDS ARE SEARCHED CORRESPONDING TO TI
005200       PERFORM TEST BEFORE
005300               VARYING WORD-INDEX FROM 1 BY
005400               UNTIL WORD-INDEX > 26
005500          IF INPUT-CHARACTER = FIRST-CHARA(
005600            THEN EXIT PERFORM
005700           END-IF
```

After completing the setting of breakpoints, start the debugger:

Select **Run** > **Debug Configurations...** from the menu bar. The **Debug Configurations** dialog box appears.

Select **Remote COBOL Application** from the left pane and press the **New** button to create a configuration.

Select **Normal Debug** for **Debug Style**, and select **Debug** button to start debugging.



Debug Configurations

Create, manage, and run configurations

Create a configuration to launch a Remote COBOL application.

Name: SAMPLE01

Project name: SAMPLE01

Debug Style

● Normal Debug

Server settings for debugger

Server name: server1

Port number: 59998

Executable file: /home/user/work/sample01/SAMPLE1

Work Directory: /home/user/work/sample01/

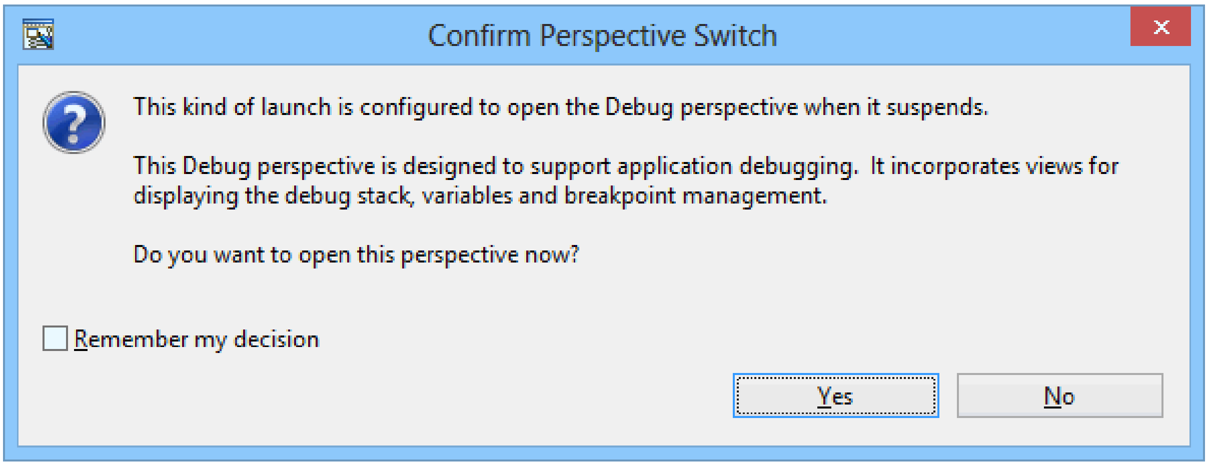
Argument(s):

○ Attach Debug

Server settings for debugger

Debug Information Directory:

Filter matched 7 of 7 items



Confirm Perspective Switch

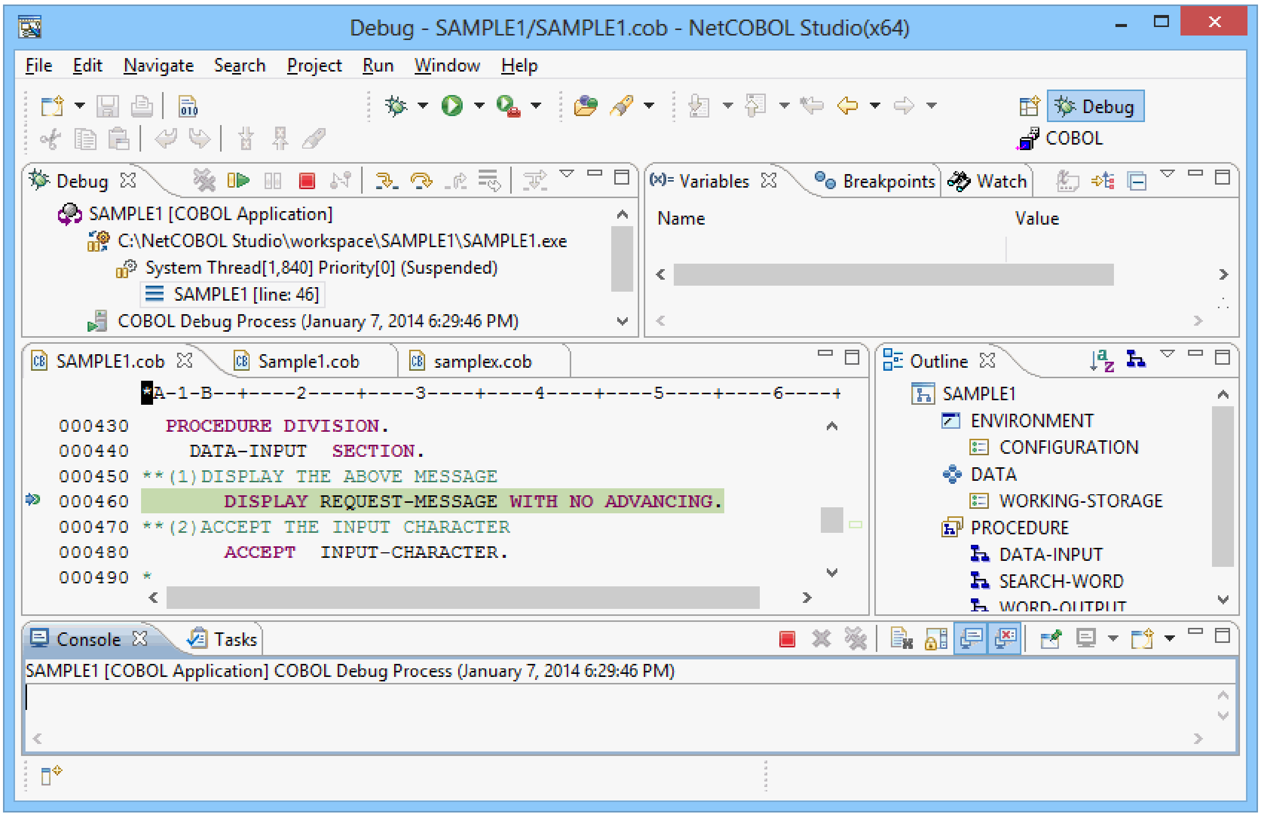
This kind of launch is configured to open the Debug perspective when it suspends.

This Debug perspective is designed to support application debugging. It incorporates views for displaying the debug stack, variables and breakpoint management.

Do you want to open this perspective now?

□ Remember my decision

A message is displayed to confirm switching to the Debug perspective when the first breakpoint is reached. Click the **Yes** button.



```
SAMPLE1 [COBOL Application]
  C:\NetCOBOL Studio\workspace\SAMPLE1\SAMPLE1.exe
    System Thread[1,840] Priority[0] (Suspended)
      SAMPLE1 [line: 46]
  COBOL Debug Process (January 7, 2014 6:29:46 PM)
```

```
       *A-1-B--+----2----+----3----+----4----+----5----+----6----+
000430   PROCEDURE DIVISION.
000440     DATA-INPUT   SECTION.
000450 **(1)DISPLAY THE ABOVE MESSAGE
000460        DISPLAY REQUEST-MESSAGE WITH NO ADVANCING.
000470 **(2)ACCEPT THE INPUT CHARACTER
000480        ACCEPT  INPUT-CHARACTER.
000490 *
```

```
SAMPLE1 [COBOL Application] COBOL Debug Process (January 7, 2014 6:29:46 PM)
```

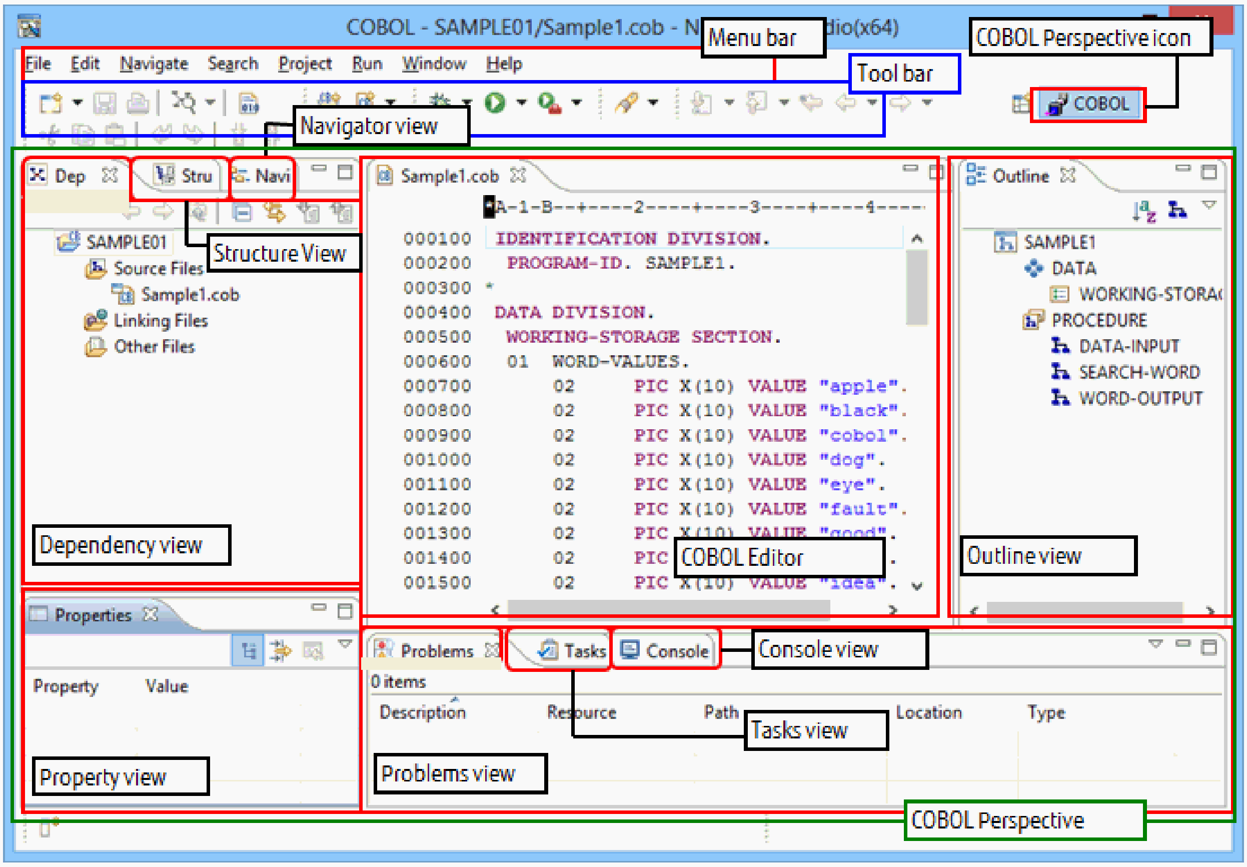
The Debug perspective is displayed, and processing is stopped at the first breakpoint. To perform debugging, select the target menu item from **Run** in the menu bar.

Point

After completing debugging, select **Window** > **Open Perspective** > **COBOL** from the menu bar to return to COBOL perspective.

Chapter 3 COBOL Perspective

The NetCOBOL Studio window consists of the editor area and information display windows (views). The types of views displayed and the layout of the views are managed as "perspectives". The COBOL perspective, a perspective designed for COBOL program development, is applied when a COBOL project is created.



```
   A-1-B--+----2----+----3----+----4----
000100 IDENTIFICATION DIVISION.
000200  PROGRAM-ID. SAMPLE1.
000300 *
000400 DATA DIVISION.
000500  WORKING-STORAGE SECTION.
000600  01 WORD-VALUES.
000700      02     PIC X(10) VALUE "apple".
000800      02     PIC X(10) VALUE "black".
000900      02     PIC X(10) VALUE "cobol".
001000      02     PIC X(10) VALUE "dog".
001100      02     PIC X(10) VALUE "eye".
001200      02     PIC X(10) VALUE "fault".
001300      02     PIC X(10) VALUE "good".
001400      02     PIC
001500      02     PIC X(10) VALUE "idea".
```

The COBOL perspective consists of the COBOL editor and the following views:

- **Dependency view**(*)
- **Structure view**(*)
- **Navigator view**(*)
- **Properties view**
- **Outline view**
- **Problems view**
- **Tasks view**
- **Console view**
- **Templates view**

*: In the properties dialog box, COBOL uses the following pages:

- Info page
- Target page
- Build page

- Build Tools page
- Project References page
- Remote Development page

3.1 Dependency View

The **Dependency view** displays a tree showing dependencies between the files to be compiled and files to be linked when a project coded in COBOL is built. The **Dependency view** displays the following three types of subfolders:

- **Source Files** folder

  The **Source Files** folder specifies the COBOL source files, precompiler input source files and resource files (.rc) to be compiled. Only files that have been added to this folder are compiled by the corresponding compilers.

- **Linking Files** folder

  The files to be linked are displayed in the **Linking Files** folder. The folder specifies the library files (.lib) and object files (.obj) to be linked, in addition to files to be compiled.

- **Other Files** folder

  Files that are in the project but are registered in neither the **Source Files** folder nor the **Linking Files** folder are displayed in the **Other Files** folder.

3.1.1 The role of the Dependency view

The **Dependency view** displays dependencies within a project.

Note

The structure of COBOL source files and the class structure of a COBOL class repository are displayed in the **Structure view**.

The following folders are created in the **Dependency view** when a COBOL project is created:

- **Source Files** folder
- **Linking Files** folder
- **Other Files** folder

Source Files folder

The **Source Files** folder specifies the files to be compiled in the project. The files displayed in this folder are compiled and linked at program build time. Files with .cob, .cbl, and .cobol extensions are treated as COBOL source files and compiled by a COBOL compiler. Files registered as precompiler input source are called by the precompiler command before compilation. Files with the .rc extension are compiled only when the resource compiler has been added as a build tool to the project.

The files in the **Source Files** folder are compiled in the order displayed, with the dependencies of the files taken into consideration. To change the position of a file in the compilation order, select the file, and use ↑ or ↓ on the toolbar to move the file up or down, respectively.

For COBOL source files in the **Source Files** folder, the following folders are displayed according to the contents of the files or specified options:

- **Target Repository** folder

  The repository file that is generated when COBOL source files are compiled are automatically analyzed and displayed when the COBOL source files are saved. The generated repository file corresponds to the class defined in the COBOL source files. The name of the generated repository file is "*class-name*.rep". The repository file displayed in this folder is used to automatically determine the compilation order of the COBOL source files. It is also used to search for files that must be compiled when they are updated.

- **Dependent Files** folder

  The **Dependent Files** folder specifies the files on which COBOL source files depend. It also specifies the libraries referenced by the COBOL source files or the repository files of classes referenced by the COBOL source files. The files specified with this folder are used to automatically determine the compilation order of the COBOL source files. They are also used to search for files that must be compiled when they are updated. To add files, select **Add File...** from the context menu. Executing **Analyze Dependency** from the context menu enables the COBOL source files to be automatically analyzed, and files on which they depend to be added.

- **Target Object Files** folder

  The **Target Object Files** folder is displayed when **Specify Target Object Files** is selected from the context menu. The object files to be generated from COBOL source files are displayed. The object files displayed in this folder are to be linked. The displayed object files are updated when the COBOL source files are saved.

These folders are used to automatically determine the compilation order of the COBOL source files.

Note

- To specify the name of a COBOL source file to register, specify a name that is different from the name of any other COBOL source file. If a COBOL source file with the same file name but a different extension is registered in the **Source Files** folder, an error occurs during build.
- Files registered in the **Source Files** folder with the extensions ".cobol" and ".cob" are treated as COBOL source files. Files with the extension ".cbl" are treated as library files.
- Of the COBOL library and descriptor files registered in the **Dependent Files** folder, only those within the project are transferred to the server during remote development. Files in other projects, and COBOL library and descriptor files managed in other folders, are not transferred. The only library files transferred to the server are those with the extension ".cbl".

Linking Files folder

The **Linking Files** folder specifies the library files (.lib) and object files (.obj) to be linked at build time. The object files to be generated from the COBOL source files specified with the **Source Files** folder need not be specified.

Other Files folder

Files that are in the project but registered in neither the **Source Files** folder nor the **Linking Files** folder are displayed in the **Other Files** folder.

Point

Operations involving files in the **Dependency view** are reflected in the **Structure view**.

3.1.1.1 Adding a COBOL source file in the Dependency view

For a COBOL source file in a project

Follow the steps below to add a COBOL source file to the project.

1. Select the **Source Files** folder in the **Dependency view**.
2. Select **Add File...** from the context menu.
3. The **Add Source Files** dialog box for the selected project appears.
4. Select the file to be added to the **Source Files** folder.
5. Click the **OK** button. The selected COBOL source file is added to the **Source Files** folder in the **Dependency view**. The addition of that added file is reflected in the **Structure view**.

For a COBOL source file that is not in a project

Either of the following methods can be used to add a COBOL source file that is not in the project:

- Using Explorer in the Windows system, select the COBOL source file to be registered, and then select **Copy** from the context menu. Select the **Source Files** folder, and select **Paste** from the context menu.

- Using Explorer in the Windows system, drag and drop the COBOL source file into the **Source Files** folder.

3.1.1.2 Adding a link file in the Dependency view

To add a link file:

1. Select the **Linking Files** folder to which the file is to be added.
2. Select **Add File...** from the context menu. The **Open** dialog box appears.
3. Select the file to be added to the **Linking Files** folder.
4. Click the **Open** button. The selected file is added to the **Linking Files** folder in the **Dependency view**. The addition of that added file is reflected in the **Structure view**.

3.1.1.3 Adding a dependent file

To add a dependent file:

1. Select the **Dependent Files** folder to which the file is to be added.
2. Select **Add File...** from the context menu. The **Open** dialog box appears.
3. Select the file to be added to the **Dependent Files** folder.
4. Click the **Open** button. The selected file is added to the **Dependent Files** folder.

3.1.1.4 Deleting a dependent file

To delete a dependent file:

1. Select the file to be deleted from the **Dependent Files** folder.
2. Select **Delete** from the context menu.
3. Click the **Yes** button when the file deletion confirmation message is displayed. The file is deleted from the **Dependent Files** folder.

Note

The file is deleted only from the **Dependent Files** folder. Even though the file is deleted from the **Dependent Files** folder, it is not deleted from the disk.

3.1.2 Context menu of the Dependency view

The following table outlines the context menu items that are unique to the **Dependency view**.

3.1.2.1 COBOL project

Element														Menu		Description
Project	Folder							File								
	Source Files	Target Repository	Dependent Files	Target Project File	Link Files	Other Files	Folder under [Other Files]	Source Files	Target Repository	Dependent Files	Target Object Files	Linking Files	Other Files			
						S	S						S	New file		New file
	S							S						new	COBOL source	Create a new Cobol source file in **Source Files** folder
	S							S							Object-Oriented COBOL Source	Create a new object-oriented source file in **Source Files** folder
	S		S		S			S		S		S		Add File		Add an existing file to the selected element. - If the **Source Files** folder or the file under the **Source Files** folder is selected, it is added to the **Source Files** folder. For details, refer to "3.1.1.1 Adding a COBOL source file in the Dependency view". - If the **Dependent Files** or the file under the **Dependent Files** folder is selected, dependent files such as libraries and repository files are added to the **Dependent Files** folder. For details, see "3.1.1.3 Adding a dependent file ". - If the **Linking Files** folder or the file under the **Linking Files** folder is selected, it is added to the **Linking Files** folder. For details, see "3.1.1.2 Adding a link file in the Dependency view".
S	S							S						Jump into		Makes the selected project, folder or file become the root of the displayed hierarchy.
S	S							S						Jump	Back	Returns to the previous hierarchy.
S	S							S							Forward	Proceeds to the former display hierarchy, when the **Jump** > **Back** menu item is selected.
S	S							S							Up One Level	Proceeds to the former display hierarchy, when the **Jump** > **Back** menu item is selected.
								S	S	S		S	S	Open		Opens the selected file.

Element															Menu	Description
Project	Folder							File								
	Source Files	Target Repository	Dependent Files	Target Project File	Link Files	Other Files	Folder under [Other Files]	Source Files	Target Repository	Dependent Files	Target Object Files	Linking Files	Other Files			
								S	S	S		S	S		Open From Application	Selects an editor (COBOL editor, text editor, system editor, default editor) to open the selected file.
										C D					Inheritance	Not to be used.
S							S	S					S		Copy	Copies the selected project, folder or file.
C D	S	C D	C D	C D	C D	S	S	S	C D	C D	C D	C D	S		Paste	Pastes the copied project, folder or file. Menu can be used when a project is copied. - When a project is copied, it will be pasted into the workspace. - When a folder is copied, if the selected element is the **Other Files** folder or the subordinate's element, this menu can be used. - When a file is copied, if the selected element is folder, it will be pasted into the folder. If the selected element is file, it will be pasted into the folder of the selected file.
S				S			S	S		S		S	S		Delete	Deletes the selected elements. - If the selected element is project, the selected project is deleted from the workspace. You can decide whether to delete the selected project from the disk. - If the selected element is a file in the **Dependent Files** folder, the selected file will be deleted from **Dependent Files** folder and moved into the **Other Files** folder. The selected file is not deleted from the disk. For details, see "3.1.1.4 Deleting a dependent file". - If the "Target Object Files" folder is selected, "specify target object file" will be unchecked and the "Target Object Files" folder is deleted. Files on disk are not deleted.

Element														Menu	Description
Project	Folder							File							
	Source Files	Target Repository	Dependent Files	Target Project File	Link Files	Other Files	Folder under [Other Files]	Source Files	Target Repository	Dependent Files	Target Object Files	Linking Files	Other Files		
															- If the selected element is a file in the **Linking Files** folder, the selected file is not deleted from the disk. - If the selected element is a file or a folder other than the above, it will be deleted from the disk.
							S	S					S	Rename	Renames the selected folder or file.
													S	Add To Source	Adds the selected file to the **Source Files** folder. When the file is added to the **Source Files** folder, it will be removed from the **Other Files** folder. If the file is in a folder other than the project storage location, it is copied to the project storage location.
								S						Remove from Source Folder	Deletes the selected COBOL source files from the **Source Files** folder and moves them to the **Other Files** folder. The files are not deleted from the disk.
								C D						Main program	Sets the selected source program as the main program of this project. This menu item displays when a COBOL source file, which is not the main program, is selected. It is only possible to select it when the target of the project is an executable file. Only one main program can be set for each project. For details, see "6.1.2 Setting the main program".
								C D						Reset Main Program	Releases the setting of main program from the selected COBOL source file. This menu item is only displayed when the COBOL source file, which is the main program, is selected. For details, see "6.1.2 Setting the main program".
								S						Specify Target Object Files	This menu item is selected for COBOL source files containing multiple compilation units (external programs or external classes). The files for which this menu item is selected (checked) are compiled after the NAME compile option is added to the other compile options of the files. The "Target Object Files" folder is added right under COBOL source file of the **Dependency view** and the

Element														Menu			Description
Project	Folder							File									
	Source Files	Target Repository	Dependent Files	Target Project File	Link Files	Other Files	Folder under [Other Files]	Source Files	Target Repository	Dependent Files	Target Object Files	Linking Files	Other Files				
																	object file generated from the COBOL source file is added to the "Target Object Files" folder. Analyzes all COBOL source files in the **Source Files** folder and extracts the target repositories and dependent files. Adds the target repository files to the **Dependent Files** folder. These target repository files are generated from library files, screen form descriptor files, dependent repository files to which COBOL source files refer, and COBOL source files.
S	S							S						Analyze dependency			Analyzes all COBOL source files in the **Source Files** folder and extracts the target repositories and dependent files.
S	S							S						Analyze dependency	All		Adds target repository files to the **Dependent Files** folder. These target repository files are generated from library files, screen form descriptor files, dependent repository files to which COBOL source files refer, and COBOL source files.
S	S							S							Library		Adds library files and screen form descriptor files to which COBOL source files refer, to the **Dependent Files** folder.
S	S							S							Repository	All	Adds dependent repository files and target repository files to which the COBOL source files refer, to **Dependent Files** folder.
S	S							S								Target	Adds target repository files generated from COBOL source files to the **Dependent Files** folder.
S	S							S								Depends	Adds dependent repository files to which the COBOL source files refer, to the **Dependent Files** folder.
S	S							S							Target object		Adds the name of object file generated from the COBOL source file to the "Target Object Files" folder.
S	S							S							Clear		Deletes all the files in the **Dependent Files** folder and clears their dependencies.
			S											Clear			Deletes all the files in the **Dependent Files** folder and clears their dependencies.

Element														Menu		Description
Project	Folder							File								
	Source Files	Target Repository	Dependent Files	Target Project File	Link Files	Other Files	Folder under [Other Files]	Source Files	Target Repository	Dependent Files	Target Object Files	Linking Files	Other Files			
CD	CD							CD						Build project		Builds a project. For details, see "Chapter 6 Build Function". This menu item is displayed only when "Build Automatically" from the "Project" menu is not checked.
S	S							S						Rebuild project		Rebuilds a project. For details, see "Chapter 6 Build Function".
S														Remote development	Build Debug Mode	When the remote development function is used, specifies whether the project is to be built in release mode or in debug mode for the remote project build. For remote project build details, see "9.5 Remote Build".
S															Build	If the remote development function is used, builds the project remotely. For remote project build details, see "9.5 Remote Build".
S															Rebuild	If the remote development function is used, rebuilds the project remotely. For remote project build details, see "9.5 Remote Build".
S															Makefile Creation	If the remote development function is used, generates the makefile for remote development. For remote project build details, see "9.5 Remote Build".
S															Debugger	Under the conditions that remote development functions are available and the server OS is the following, the remote debugger connector is started. - Solaris
S															Remote Debugger Connector	Starts the remote debugger connector. Before performing attach debugging, it can be used to set which computers are permitted to connect to the client computer and change the port number for remote debugging. For remote debug connector details, see "9.6.2.2 Remote debugger connector".
NS															Transfer	Cannot be used for COBOL projects.
								S		CD			S	Compile File		Compiles the selected file.
S	S	NS	S	S	S	S	S	S	S	S	S	S	S	Refresh		Refreshes the selected element and its subordinate.

Element														Menu	Description
Project	Folder							File							
	Source Files	Target Repository	Dependent Files	Target Project File	Link Files	Other Files	Folder under [Other Files]	Source Files	Target Repository	Dependent Files	Target Object Files	Linking Files	Other Files		
S							S	S	S	S		S	S	Property	Displays the property of the selected elements.

Meaning of signs

S: Menu can be selected

NS: Menu cannot be selected

CD: Menu can be displayed under certain conditions

Space: Menu cannot be displayed.

For the menu below, refer to the "3.3.3 Navigator view Context menu ".

- Team
- Compare With
- Replace
- Restore from Local History
- Source

3.1.2.2 COBOL Resource Project

Element			Menu		Description
Project	File	Folder			
S	S	S	New File		Creates a new file.
S			Jump into		Makes the selected project, folder or file become the root of the displayed hierarchy.
S			Jump	Back	Returns to the previous hierarchy.
S				Forward	Proceeds to the former display hierarchy, when the **Jump** > **Back** menu item is selected.
S				Up One Level	Changes the display hierarchy to one up from the level of the present display hierarchy.

Element			Menu		Description
Project	File	Folder			
	S		Open		Opens the selected file.
	S		Open from Application		Selects an editor (COBOL Editor, Text Editor, System Editor or Default Editor) to open the selected file.
S	S	S	Delete		Deletes the selected project, folder or file. - When a project is selected, the COBOL Resource project will be deleted. You can choose whether to delete from disk. - When a folder or file is selected, it will be deleted from the disk.
	S	S	Rename		Changes the name of the selected folder or file.
NS			Remote Development	Build Debug mode	Cannot be used for COBOL Resource projects.
				Build	
				Rebuild	
				Makefile Creation	
				Debugger	
				Remote Debugger Connector	
S				Transfer	Transfers the resource of the project to a remote server.
S	S	S	Refresh		Refreshes the selected element and its subordinate.
S	S	S	Property		Displays the property of the selected element.

Meaning of signs:

S: Menu can be selected

NS: Menu cannot be selected

Space: Menu cannot be displayed.

For the following menu, see "3.3.3 Navigator view Context menu ".

- Team
- Compare With
- Replace With
- Restore from Local History
- Source

3.1.2.3 COBOL Solution Project

This section explains the context menu that displays when a COBOL solution project is selected. For information about the context menu that displays when a project added to COBOL solution project is selected, see "3.1.2.1 COBOL project" and "3.1.2.2 COBOL Resource Project".

menu		Description
Jump into		Makes the selected project, folder or file become the root of the displayed hierarchy.
Jump	Back	Returns to the previous hierarchy.
	Forward	Proceeds to the former display hierarchy, when the **Jump** > **Back** menu item is selected.
	Up One Level	Changes the display hierarchy to one level up from the present display hierarchy.
Delete		Deletes the selected project from the workspace. You can choose whether to delete it from the disk. Projects added to the selected COBOL solution project are not deleted.
Project Management		Displays the **Project Management** dialog box. Projects can be added to or removed from the selected COBOL solution project. For the addition and the removal of projects, see "4.3.1 Manage COBOL project.
Analyze Dependency		Performs Dependency Analysis for COBOL projects added to the selected COBOL solution project.
Build Project		Builds COBOL projects added to the selected COBOL solution project. For project build details, see "Chapter 6 Build Function". This menu is displayed only when "Build Automatically" from the "Project" menu is not checked.
Rebuild Project		Rebuilds COBOL projects added to the selected COBOL solution project. For project build details, see "Chapter 6 Build Function".
Remote Development	Build Debug Mode	Changes the build mode of COBOL projects added to the selected project. Specifies whether projects are to be built in release mode or in debug mode for remote project build. For remote project build details, see "9.5 Remote Build".
	Build	Builds COBOL projects added to the selected COBOL solution project. For remote project build details, see "9.5 Remote Build".
	Rebuild	Rebuilds COBOL projects added to the selected COBOL solution project. For remote project build details, see "9.5 Remote Build".
	Makefile Creation	Cannot be used for COBOL solution projects.
	Debugger	
	Remote Debugger Connector	
	Transfer	
Refresh		Refreshes the selected element and its subordinate.
Property		Displays the property of the selected element.

For the following menu, see "3.3.3 Navigator view Context menu ".

- Team
- Compare With
- Restore from Local History

3.1.3 Analyze Dependency

Analyze Dependency is the menu command for adding the following files that the COBOL source files associated with the "Dependent File".

- Library file
- Screen form descriptor file

Execute the **Analyze Dependency** in any one of the following conditions.

a. The COPY statement is added to or deleted from the COBOL source file.

   Execute the **Analyze Dependency** regarding the modified COBOL source files as target.

   1. Select the modified COBOL source file in the **Dependency view**.

   2. Select **Analyze Dependency** > **All** from the context menu.

b. When one of the following happens with the workspace:

   - The library file used for projects (COBOL project or COBOL resource project) are added to or deleted from the workspace.

   - The dependent file is added to or deleted from the library file used for projects (COBOL project or COBOL resource project).

   - Under the setting of translation option, the reserved folder appointed library file in the translation option LIB or the reserved folder appointed descriptor file in the translation option FORMLIB, is modified.

   Execute the **Analyze Dependency** regarding the project that dependency happens in workspace as target in above conditions.

   1. Select the COBOL project in the **Dependency view**.

   2. From the context menu, select **Analyze Dependency** > **All**.

c. Compiler option.

Point

Following are some merits when executing **Analyze Dependency**.

- The dependent file can be added to the **Dependent Files** folder automatically.

- The dependent file can be modified (updated, saved) from the COBOL project that the COBOL source files of the reference is registered in.

Note

The screen form descriptor that is added as a dependent file can be updated in PowerFORM.

The form descriptor (.SMU/.PMU) used for UTF-32, which is produced with the Descriptor transformation command used for UTF-32, is not the target of **Analyze Dependency**. The screen form descriptor (.SMU/.PMU) used for UTF-32 cannot be added to dependent file.

3.2 Structure View

The **Structure view** hierarchically displays the internal program structure of COBOL source files. The **Structure view** has the following three types of subfolders:

- **Source Files** folder

- **Linking Files** folder

- **Other Files** folder

3.2.1 The role of the Structure view

Source Files folder

COBOL source files to be compiled are displayed in the **Source Files** folder. Its contents are equivalent to those of the **Source Files** folder in the **Dependency view**. The internal structure of each COBOL source file is hierarchically displayed under the COBOL source file. The structure displayed under each COBOL source file is as follows:

- For an ordinary COBOL source file

  - PROGRAM-ID

    - Environment division and section-name
    - Data division and section-name
    - Procedure division, section-name, and paragraph-name
- For an object-oriented COBOL source file
    - CLASS-ID
    - FACTORY
    - OBJECT
    - METHOD-ID
    - Environment division and section-name
    - Data division and section-name
    - Procedure division, section-name, and paragraph-name

Linking Files folder

Files to be linked but not compiled are displayed in the **Linking Files** folder. Its contents are equivalent to those of the **Linking Files** folder in the **Dependency view**.

Other Files folder

Files that are to be neither compiled nor linked are displayed in the **Other Files** folder. Its contents are equivalent to those of the **Other Files** folder in the **Dependency view**.

3.2.2 Context menu of the Structure view

The following table outlines the context menu items that are unique to the **Structure view**:

3.2.2.1 COBOL Project

Element								Menu		Description
Project	Folder				File					
	Source file	Link file	Other file	Other file inner	Source file	Link file	Other file			
			S	S			S	New file		A New file is created in the selected folder or the folder that the selected file belongs to.
	S				S			new	COBOL source	Adds the new COBOL source file to the **Source Files** folder.
	S				S				Object-Oriented COBOL Source	Adds the new Object-Oriented COBOL source file to the **Source Files** folder.
	S	S			S	S		Add File		Adds an existing file to the selected folder or to the folder that the selected file belongs to.
S	S							Jump into		Changes the display hierarchy in order that the selected project or folder converts to root.

Element								Menu		Description
	Folder				File					
Project	Source file	Link file	Other file	Other file inner	Source file	Link file	Other file			
S	S							Jump	back	Goes back to the former hierarchy from the current display hierarchy.
S	S								forward	Moves forward to the next hierarchy if the Jump Back option was used.
S	S								Up One Level	Changes the display hierarchy to up one level from the present display hierarchy.
					S	S	S	Open		Opens the selected file.
					S	S	S	Open From Application		Selects an editor (COBOL editor, text editor, system editor or default editor) to open the selected file.
S				S	S		S	copy		Copies the selected project, folder or file.
C D	S	C D	S	S	S	C D	S	paste		Pastes the copied project, folder or file. This menu item can be used when a project is copied. - When the project is copied, it will be pasted into the workspace. - When the folder is copied, if the selected element is the **Other Files** folder or the subordinate's element, this menu can be used. - When the file is copied, if the selected element is folder, it will be pasted into the folder. If the selected element is file, it will be pasted into the folder of the selected file.
S				S	S	S	S	delete		Deletes the selected elements. - If the selected element is project, the selected project is deleted from the workspace. You can decide whether to delete the selected project from the disk. - If the selected element is a file in the **Link Files** folder, the selected file is not deleted from the disk. - If the selected file is other than a file in the **Link Files** folder, the selected file or folder is deleted from the disk.
				S	S		S	rename		Renames the selected folder or file.
							S	Add To Source		Adds the selected file to the **Source Files** folder. When the file is added to the **Source Files** folder, it will be removed from the **Other Files** folder. If the file is in a folder other than the project storage location, it is copied to the project storage location.
					S			delete from Source file		Deletes the selected COBOL source files from **Source Files** folder and moves them to the **Other Files** folder. The files are not deleted from the disk.
					C D			Main program		Sets the selected source program as the main program of this project.

Element								Menu		Description
Project	Folder				File					
	Source file	Link file	Other file	Other file inner	Source file	Link file	Other file			
										This menu item is displayed when the selected COBOL source is not the main program. When the target of the project is an executable file, it is possible to select it. Only one main program can be set for each project. For details, refer to "6.1.2 Setting the main program".
					C D			Reset Main Program		Releases the setting of main program from the selected COBOL source file. This menu item is displayed when the selected COBOL source file is the main program. For details, refer to "6.1.2 Setting the main program".
					S			Specify Target Object Files		Select this menu item for COBOL source files containing multiple compilation units (external programs or external classes). The NAME compile option is added to the other compile options and then the checked files are built.
C D	C D				C D			Build project		Builds the selected project. For details, refer to "Chapter 6 Build Function". This menu is displayed only when "Build Automatically" in the "Project" menu is not checked.
S	S				S			Rebuild project		Rebuilds the selected project. For details, refer to "Chapter 6 Build Function".
S								Remote development	Build Debug Mode	When remote development function is used, this specifies whether the project is to be built in release mode or in debug mode for a remote project build. For remote project build details, see "9.5 Remote Build".
S									Build	If the remote development function is used, this builds the project remotely. For remote project build details, see "9.5 Remote Build".
S									Rebuild	If the remote development function is used, this rebuilds the project remotely. For remote project build, see "9.5 Remote Build".
S									Makefile Creation	If the remote development function is used, the makefile for remote development is generated. For remote project build details, refer to "9.5 Remote Build".
S									Debugger	Under the conditions that remote development functions are available and the server OS is the following, the remote debugger connector is started. - Solaris
S									Remote Debugger Connector	Starts the remote debugger connector. Before performing attach debugging, use the remote debug connector when you want to change the restriction and the port number of the computer that performs the remote debug. For remote debug connector details, refer to "9.6.2.2 Remote debugger connector".

Element								Menu	Description
	Folder				File				
Project	Source file	Link file	Other file	Other file inner	Source file	Link file	Other file		
NS								Transfer	Cannot be used for COBOL projects.
					S		S	Compile File	Compiles the selected file.
S	S	S	S	S	S	S	S	Refresh	Refreshes the selected element and its subordinate.
S				S	S	S	S	Property	Displays the property of the selected elements.

Meaning of signs

S: Menu can be selected

NS: Menu cannot be selected

CD: Menu can be displayed under certain conditions

Space: Menu cannot be displayed.

For the menu below, refer to the "3.3.3 Navigator view Context menu ".

- Team
- Compare With
- Replace
- Restore from Local History
- Source

3.2.2.2 COBOL resource project

The context menu of selected COBOL resource project is the same as **Dependency view** context menu. Refer to "3.1.2.2 COBOL Resource Project" under "Dependency view".

3.2.2.3 COBOL solution project

The context menu of selected COBOL solution project is the same as the **Dependency view** context menu except that **Analyze Dependency** is not displayed. Refer to "3.1.2.3 COBOL Solution Project" under "Dependency View".

3.3 Navigator View

The **Navigator view** hierarchically displays resources in a workspace. This view displays all the files and folders in each project. A file can be opened by double-clicking it. A project can be created, opened, or closed from the context menu of the **Navigator view**.

3.3.1 The role of the Navigator view

The **Navigator view** displays files in existing subfolders and folders of a project. The following operations can be performed from the **Navigator view**:

- Creating a project, file, or folder

- Opening a project, file, or folder
- Closing a project
- Deleting a project, file, or folder
- Renaming a project, file, or folder
- Setting properties for a project
- Applying a filter
- Sharing files in the Concurrent Versions System (CVS)

3.3.1.1 Applying a filter to the view

Follow the procedure below to apply a filter to the view.

1. Click ▽ in the **Navigator view**.
2. Select **Filters...** from the option list. The **Navigator Filters** dialog box appears.
3. Select one or more filters to be applied from the option list.
4. To select all filters, click the **Select All** button. To clear the filter, click the **Deselect All** button.
5. To apply all selected filters to the **Navigator view**, click the **OK** button. The dialog box closes. To close the dialog box without applying the filters to the **Navigator view**, click the **Cancel** button.

3.3.1.2 Sharing files in the Concurrent Versions System (CVS)

In the team programming environment of the Concurrent Versions System (CVS), each team member performs work independently from the other members. A CVS repository can be shared with other team members. Only a project or the files in a project can be shared. If a project is shared, only the files in the project are displayed in the repository. Other team members can access only the files that are explicitly shared.

Sharing an existing CVS repository location and project

To share an existing CVS repository location and project:

1. Select the project from the **Navigator view**, and select **Team** > **Share Project** from the context menu.
2. The **Share Project** dialog box appears.
3. Select **Use existing repository location** to use an existing repository.
4. Select the location from the displayed list.
5. Click the **Finish** button. The project is shared at the selected CVS repository location.

Sharing a new CVS repository location and project

To share a new CVS repository location and project:

1. Select the project from the **Navigator view**, and select **Team** > **Share Project** from the context menu.
2. The **Share Project** dialog box appears.
3. Specify the following on the **Enter Repository Location Information** page.

Field	Description
Host	Select or enter the host address.
Repository path	Select or enter the repository path to the host.
User	Enter the user name for a connection to the host.
Password	Enter the password for host access.

Field	Description
Connection type	Select the type of CVS connection to the repository.
Use Default Port	Select a port for the connection.
Use Port	Select a port from the list box.
Save password	When checked, the password is saved to the computer.

4. Click the **Next** button.
5. Specify the options on the **Enter Module Name** page as follows:

Field	Description
Use project name as module name	Specify this option to use the project name as a module name.
Use specified module name	Check this item to not use the project name as a module name, and enter a module name in the text box.

6. Click the **Finish** button. The project is shared at the new CVS repository location.

Sharing a file

To share a file:

1. Select the file from a project folder.
2. Select **Team** > **Add to Version Contro**l from the context menu.
3. Select **Commit** from the context menu of the file or project.
4. The file is shared at the CVS repository location, and other users can access the file.

Accessing a project or file in a CVS repository

To access a project or file in a CVS repository:

1. Select **Windows** > **Open Perspective** > **Other...** > **CVS Repository Exploring** from the menu bar. The **CVS Repository** perspective opens.
2. Expand the host that has shared projects and files.
3. Select the project for checkout processing.
4. Select **Check Out As** from the context menu. The **Check Out As** dialog box appears. Select **Check out as a project in the workspace** and enter a project name in the *Project Name* field.
5. Click the **Finish** button. The display returns to the COBOL perspective, where the **Navigator view** displays the project and the files under the project.

Comparing versions of a source file

To compare the latest version of a COBOL source file with an older version:

1. In the **Navigator view**, select the version of the COBOL source file whose contents are to be compared with the current version.
2. Select **Compare With** > **Local History** from the context menu. The **History** dialog box appears.
3. Select a version from the **Revision Time** list.
4. The latest version of the COBOL source file is displayed in the Workspace File area of the Text Compare editor, and the selected version is displayed in the Local History area. Modifications in the text in each file are highlighted.
5. To move to the previous or next modification, select the **Compare with Each Other** button.
6. To close the dialog box, click the **OK** button.

Replacing a current file with a previous version

To replace the current COBOL source file with a previous version:

1. In the **Navigator view**, select the COBOL source file to be replaced.
2. Select **Replace With** > **Local History** from the context menu. The **Compare** dialog box appears.
3. Select a version from the Local History list.
4. The current version of the COBOL source file is displayed in the right pane of the Text Compare editor, and the selected version is displayed in the Local History area in the left pane. Modifications in each file are highlighted.
5. To replace the file of the current version with that of the selected version, click the **Replace** button. The dialog box closes. To close the dialog box without replacing the file, click the **Cancel** button.

Note

Replace With > **Previous from Local History** is selected; the current version of the file is replaced with that of the last stored version.

To perform a Replace With operation for an open file, first save the replacement file.

3.3.1.3 Property options

The properties of the selected resource are displayed when File is used, or when Properties in the context menu is used. The items displayed in the properties page depend on the selected resource. For example, if a file is selected, only specific properties are displayed; if a project is selected, all setting options are displayed.
Property options are displayed by selecting **File** > **Properties** from the menu bar. Options can be displayed, or their settings can be changed using the properties dialog box.

Displaying and setting file properties

To change the read-only status of a file:

1. Select **Resource** in the left pane of the properties dialog box.
2. The details of **Path**, **Type**, **Location**, and **Last modifie**d are displayed on the Info page.
3. The file properties can be changed by selecting **Read only** or canceling the selection.

Displaying and setting project reference properties

To enable a resource to reference another project in a workspace:

1. Select **Project References** in the properties dialog box.
2. The **Project References** page is displayed.
3. To enable the resource to reference another project in the workspace, select the other project from the list.

3.3.2 Navigator view Tools

Tools of the **Navigator view** can be used for various operations.

Icon	Description
⇦	Returns to the previous hierarchical view.
⇨	Proceeds to the next hierarchical view.
📁	Displays the hierarchy at the next higher level.
⊟	Reduces the tree of all resources in the expanded display.

<table>
<tr><th>Icon</th><th colspan="3">Description</th></tr>
<tr><td>⇆</td><td colspan="3">Toggles enabling the link between the <b>Navigator view</b> and the active editor.</td></tr>
<tr><td rowspan="9">▽</td><td colspan="3">Displays options such as Sort and Filters in the menu.</td></tr>
<tr><th colspan="2">Option</th><th>Description</th></tr>
<tr><td colspan="2">Select Working Set</td><td>Selects a working set.</td></tr>
<tr><td colspan="2">Deselect Working Set</td><td>Cancels a working set.</td></tr>
<tr><td colspan="2">Edit Active WorkingSet</td><td>Used to edit a working set.</td></tr>
<tr><td rowspan="2">Sort</td><td>by Name</td><td>Sorts files in alphabetical order.</td></tr>
<tr><td>by Type</td><td>Sorts file types in alphabetical order.</td></tr>
<tr><td colspan="2">Filters</td><td>Filters the resources to be displayed.</td></tr>
<tr><td colspan="2">Link with Editor</td><td>Toggles enabling the link between the <b>Navigator view</b> and the active editor.</td></tr>
</table>

3.3.3 Navigator view Context menu

The following table outlines the context menu items for the **Navigator view**:

Element			Menu	Description
Project	Folder	File		
S	S	S	New	Creates a project, file, or folder.
S	S		Go Into	Displays subsets of the resource selected in the **Navigator view**. For example, if a project is selected and Go Into is then selected, the **Navigator view** displays only the files and folders under the project.
S	S		Open in New Window	Opens the selected resource.
			Open	Opens the selected file.
		S	Open With	Selects an editor (COBOL editor, text editor, system editor, or default editor) to open the selected resource.
		S	Copy	Copies a resource from a specific location.
S	S	S	Paste	Pastes the copied resource to the specific location.
S	S	S	Delete	Deletes the selected resource.
S	S	S	Move	Moves a resource to another location. The dialog box into which you can enter the move destination of the resource appears.
S	S	S	Rename	Renames a resource.
S	S	S	Import	Opens the **Import** wizard.
S	S	S	Export	Opens the **Export** wizard.
CD			Build Project	Builds the selected project. This menu item is displayed when the COBOL project or COBOL resource project is selected. And when "build automatically" is not checked.
CD			Close Project	Closes the selected project. This menu item is displayed when the selected project is opening.

Element			Menu		Description
Project	Folder	File			
CD			Open Project		Opens the selected project. This menu item is displayed when the selected project is closed.
CD			Close Unrelated Project		Closes project that is not the selected project. This menu item is displayed when the unselected project is opening.
S	S	S	Refresh		Updates the selected resource and view of subsets.
CD	CD	CD	Run As	COBOL Application	This menu item is displayed when a COBOL project, folder in COBOL project, or file in a COBOL project is selected.
S	S	S		Run Configurations	Opens the **Run Configurations** dialog. For details, refer to "Chapter 8 Execution Function".
CD	CD	CD	Debug As	COBOL Application	Start the debugger to debug the selected project. The project must have been built. This menu item is displayed when a COBOL project, folder in COBOL project, or file in a COBOL project is selected.
CD	CD	CD		Remote COBOL Application	When the remote development function of the selected project is effective, start debugger to debug the selected project remotely. This menu item is displayed when a COBOL project, folder in COBOL project, or file in COBOL project is selected.
S	S	S		Debug Configurations	Opens the **Debug Configurations** dialog. For details, refer to "7.1.2 The startup configuration", "9.6.1.2 Starting the remote debugger" or "9.6.2.1 Starting the remote debugger".
S	S	S	Team		Shares a project and displays the local history. (The sub-menu of the effective team function is displayed.) The sub-menu "apply to patch" cannot be used.
S	S	S	Compare with		Compares the selected resource. (Possible comparison targets are displayed as submenu items).
		S	Replace With		Replaces the selected resource. (Possible replacements are displayed as submenu items).
S	S		Restore from Local History		Restores a deleted resource from the local history.
S	S		Source		This menu item cannot be used.
S	S	S	Properties		Displays the resource properties.
NS	NS	S	Compile File		Compiles the file.

Meaning of signs

S: Menu can be selected

NE: Menu cannot be selected

CD: Menu can be displayed under certain conditions

Space: Menu cannot be displayed.

Point

The functions of the **Navigator view** context menu, except Move and Replace With, can be used in the **Dependency view** and **Structure view**.

3.4 Property View

The **Property view** displays the properties of resources in the COBOL project selected in the Dependency, Structure, or **Navigator view**. Buttons are provided in the toolbar to toggle to display properties by category, to filter advanced properties, and to restore a selected property to its default value. For more information regarding a resource, right-click the resource name in one of the navigation views and select Properties from the pop-up menu.

3.5 Outline View

The **Outline view** displays a structural outline of the COBOL source file that is currently active in the COBOL editor. The contents of the **Outline view** vary depending on the COBOL program type.

The tree structure displayed in the **Outline view** is as follows:

- For an ordinary COBOL source file
    - PROGRAM-ID
    - Environment division and section-name
    - Data division and section-name
    - Procedure division, section-name, and paragraph-name
- For an object-oriented COBOL source file
    - CLASS-ID
    - FACTORY
    - OBJECT
    - METHOD-ID
    - Environment division and section-name
    - Data division and section-name
    - Procedure division, section-name, and paragraph-name

Function of the Outline view

The **Outline view** has the following function:

- Move to an element
  The cursor in the editor is moved to the location of the selected element, such as a program name or method name.

3.5.1 Toolbar

Operation	Icon	Description
Sort	↓az	Toggles the listing order of the section-name, paragraph-name, or METHOD-ID elements between alphabetical order and the order specified in a COBOL source file. The listing order of divisions and FACTORY OBJECTs cannot be changed.
Procedure division		Toggles between displaying the contents of all divisions (procedure, environment, and data) and displaying the contents of only the procedure division.

3.6 Problems View

The **Problems view** displays error messages, warning messages, etc. for problems that occur during compilation. Each message displayed in the **Problems view** is associated with the corresponding COBOL source file or project. For each displayed COBOL compiler error or warning message, the corresponding COBOL source file and line number are indicated.

3.6.1 Problems view display

The information displayed in the **Problems view** is as follows

Item	Description
Severity	Displays an icon representing a severity level (error, warning, or information).
Description	Provides an explanation of the problem.
Resource	Displays the names of resources associated with the problem. For global problems, the relevant project name is displayed.
Path	Displays the folder containing the resources associated with the problem.
Location	If the problem is in a file, this field displays the line numbers indicating where the problem occurred.
Type	Displays the type of problem.

3.6.2 Using the Problems view to locate an error

If an error occurs during a build for a project, the **Problems view** displays the error. To locate an error:

1. Errors are indicated as follows:

    - The **Problems view** displays a list of errors.

    - The lines containing errors are highlighted in the editor.

2. Double-click on an error in the **Problems view**, or select an error and then select **Go To** from the context menu. The file containing the detected error is opened in the editor, and the relevant location is highlighted.

3. Use the editor to modify erroneous coding and re-build the project. When the problem is resolved, the **Problems view** no longer displays the error.

Note

If the folder path is not displayed in the **Problems view** in the Path item, the error position cannot be identified, even **Go to** is selected from the context menu. This applies if the compile error is in a library external to the project.

3.7 Tasks View

The **Tasks view** can be used to record an issue as a task for future consideration. Tasks can be associated with COBOL source files. For additional information regarding the **Tasks view**, see the "Workbench User's Guide" in the Help information.

3.7.1 Tasks view display

The **Tasks view** displays the status, priority, and explanation of each task. Tasks that are associated with COBOL source files are displayed together with relevant file names, folders, and line numbers. The information displayed in the **Tasks view** is as follows:

Item	Description
Completed	Indicates whether a task has been completed. Each completed task has a checkmark. Checkmarks for completed tasks can be also added manually.

Item	Description
Priority	Indicates the priority of a task (high, middle, or low). The priority of a task can be changed using the combo box in this column.
Description	Provides an explanation of a task. To edit the explanation provided by a user for a task, select this column.
Resource	Displays the names of resources involved in a task. This field is blank for global tasks.
In Folder	Displays the folder containing the resources involved in a task. This field is blank for global tasks.
Location	Displays the line numbers associated with a task.
Type	Displays the type of a task.

3.7.2 Associating a task with a COBOL source file

Associating a task with a COBOL source file designates the sections to be displayed in a task list. To associate a task with a COBOL source file:

1. Use the COBOL editor to display the COBOL source file.
2. Using the vertical ruler on the left side of the COBOL editor, go to the line in the source file where the new task is to be recorded and select **Add Task** from the context menu.
3. Set the contents and priority of the task in the **New Task** dialog box, and click the **OK** button. The task is added to the **Tasks view**, and an icon representing the task is displayed to the left of the line on which the task was added.
4. The task in the COBOL source file can be displayed with the COBOL editor by double-clicking on the task in the **Tasks view**.
5. To delete the task, select **Remove Task** from the context menu of the task icon, or select the task in the **Tasks view** and press the **Delete** key.

3.8 Console View

The following functions can be used in the **Console view**:

- COBOL remote

  In remote development, the results of makefile creation and the results of a build on a server can be displayed.

- Build console

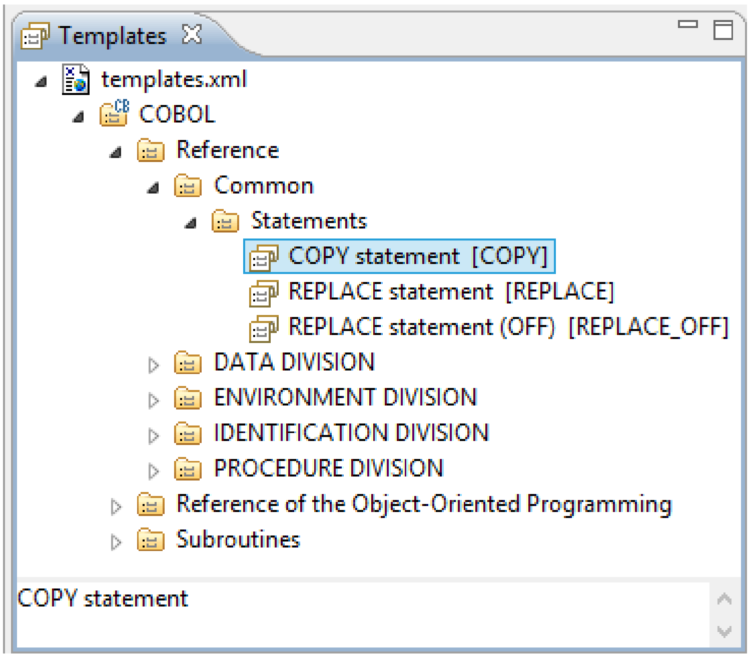
  The results of a COBOL program build on a local personal computer can be displayed.

3.9 Templates view

Use the following procedures to display the **Templates view**.

1. Select **Window** > **Show View** > **Other** from the menu bar. The **Show View** dialog box is displayed.

2. Select **COBOL** > **Templates** and the **Templates view** is displayed.



In the **Templates view**, root category, category and template are displayed as hierarchy.

The root category is "COBOL".

The category is "Reference", "Statement" and so on displayed in icon .

Template is "COPY statement [COPY]", "REPLACE statement [REPLACE]" and so on displayed in icon .

The content of the displayed template in the **Templates view** is different based on the extension of opening file in the active editor.

For example, the defined [COBOL] root category is associated with the following file extensions

- .cbl
- .cobol
- .cob

When opening a file that has an that is not associated with the defined root category in the editor, or the file cannot be opened in the editor, only the main file ("templates.xml") without the root category is displayed in **Templates view**.

Note

"COBOL" is the defined root category. The defined root category cannot be deleted or renamed.

3.9.1 The role of Templates View

In the **Templates view**, the following functions can be used.

- Insert template pattern (*1) into source file.
- Edit, add and delete template.
- Import and export template.
- Activate and invalidate template

*1: Also can call Model code which is inserted with using template "template pattern".

Templates view is useful for when inserting the syntax of the COBOL language into the source file, it.

3.9.1.1 Insert template patterns into source file

The example below illustrates insetting a PERFORM statement from a template into the procedure division of a COBOL source file.

```
           PERFORM WITH TEST BEFORE
                    VARYING repeat count FROM 1 BY 1
                    UNTIL repeat count > 26
```

1. Opens the edited COBOL source file in the COBOL editor.
2. PERFORM statement is inserted into position at which the cursor is positioned.
3. Displays the inserted template pattern in the **Templates view**.

Point

The content of displayed template in **Templates view** is different according to the extension of opening file in active editor.

Displays the root category used for COBOL when opening COBOL source file (the file whose extension is .cob, .cobol or .cbl).

4. Selects the inserted template pattern, and then select **Insert with Parameter Value...** or **Insert with Multiple Parameter Values...** from context menu, **Input Assistance** dialog box displays.
5. Input identifier, initial values, incremental value of PERFORM statement and then click **OK** button.

Table 3.1 Input Assistance dialog box

Item	Description
Pattern preview	Displays the pattern of template.
Description	Displays the descriptions of template.
Enter parameter value	In condition that pattern of template includes argument, inputs value to **Value** field. This graph cannot be displayed when the pattern of template without input argument.

PERFORM statement is inserted into COBOL source file.

The condition that template pattern without arguments.

In the condition that template pattern without arguments, inserts as the following procedures.

1. Displays the **Templates view**.
2. Opens the source file in COBOL editor.

   The content of displayed template in **Templates view** is different according to the extension of opening file in active editor.

   Displays the root category used for COBOL when opening the COBOL source file (the file whose extension is .cob, .cobol or .cbl).
3. In COBOL editor, puts the cursor into the position at which template pattern intends to insert.
4. From **Templates view**, selects used template and then selects [insert] from context menu. Template pattern is inserted in COBOL editor.

3.9.1.2 Adding, editing, and deleting a template

The following can be executed.

- Add root category
- Add category
- Add template

- Edit root category
- Edit category
- Edit template
- Delete root category
- Delete category
- Delete template

Create root category

Use the following procedures to create a root category.

1. Select "templates.xml" in the **Templates view**, and then select **Add** > **Root Category...** from the context menu. Or, select **templates.xml** from **Categorization of the template** in the **Templates** page, and then click the **New...** button.
   The **Add Root Category** dialog box appears.

   Table 3.2 Add Root Category dialog box

Item	Description
Name	Root category name
Descriptions	Description of the root category.
File extensions	The summary of the file extent associated with the root category is displayed.
Add...	Click the **Add...** button to add the file extent into the "File extensions". The **Select File Extensions** dialog box will display and then you can select an extent to add.
Remove	Click the **Remove** button to remove the file extent from the "File extensions".

2. Input the above items and then click the **OK** button.

Create category

Use the following procedures to create a category.

1. Select the parental category or root category from the **Templates view** and then select **Add** > **Category** from the context menu. Or, select the parental category or root category from **Categorization of template** in the **Templates** page and then click the **New...** button.

   The **Add Category** dialog box appears (**Add Category/Template** dialog box appears when creating from **Templates** page.)

   Table 3.3 Add Category dialog box (Add Category/Template dialog box when creating from Templates page.)

Item	Description
Parent Name	Displays the selected category or root category as parent.
Category	Check the "Category". Note: When **Add** > **Category** is selected from the **Templates view**, the "Templates" is unavailable.
Template	
Name	Category name. Category name must specify the identified name in parent category.
Keyword	Category cannot be inputted when it is checked.
Descriptions	Category description.

Item	Description
Pattern	Category cannot be inputted when it is checked.

2. Input the above items, and then click the **OK** button.

Create template

Use the following procedures to create a template.

1. Select the parental category or root category from the **Templates view** and then select **Add** > **Category** from the context menu. Or, select the parental category or root category from "Categorization of template" in the **Templates** page and then click the **New...** button.

   Add template dialog box appears (**Add Category/Template** dialog box appears when creating from **Templates** page.)

   Table 3.4 Add template dialog box (Add Category/Template dialog box when creating from template page.)

Item	Description
Parent name	Displays the selected category or root category as parent.
Category	Check the "Category". Note: When **Add** > **Templates** is selected from the **Templates view**, the "Category" is unavailable.
Template	
Name	Template name.
Keyword	Template key word.
Descriptions	Template description.
Pattern	Template pattern.

2. Input the above items, and then click the **OK** button.

Note

- The Template cannot be displayed in the Content Assist list without inputting the "Keyword".

- It is necessary to encompass the variable names with "${"and"}" when the variable will be used in "Pattern".

```
For example:
COPY$[text-name]
```

- Variable names that can be used in "Pattern" apply the rule of user-defined words used in COBOL. For user-defined words used in COBOL, refer to "COBOL Language Reference".

Edit root category

Use the following procedures to edit the name of the root category, descriptions and associated file extent. A system-defined Root category can be edited in the same way as a user-defined root category. However, a predefined root category name cannot be edited.

1. From the **Templates view**, select the root category to be edited, and then select **Edit...** from the context menu. Or, select the root category to be edited from "Categorization of template" in **Templates** page and then click the **Edit...** button.

   The **Edit Root Category** dialog box appears.

   Table 3.5 Edit Root Category dialog box

Item	**Descriptions**
Name	Root category name.
Description	Descriptions of root category.

Item	**Descriptions**
File extensions	Displays the file extent associated with the root category.
Add...	Click the **Add...** button to add a file extent into the "File extensions". The **Select File Extensions** dialog box will display, and then you can select extent to be added.
Remove	Click the **Remove** button to remove the file extent from the "File extensions".

2. Make any desired changes and then click the **OK** button.

Edit category

Use the following procedures to edit the category.

1. From the **Templates view**, select the category to be edited and then select **Edit...** from the context menu. Or, select the edited category from the "Categorization of the template" in **Templates** page, and then click the **Edit...** button.

   The **Edit Category** dialog box is displayed.

Table 3.6 Edit Category dialog box

Item	Description
Parent name	Displays the selected category or root category as parent.
Category	"Category" is checked.
Template	
Name	Category name. Category name must specify the identified name in parent category.
Keyword	Cannot be inputted when the "Category" is checked.
Description	Category descriptions.
Pattern	Cannot be inputted when the "Category" is checked.

2. Make any desired changes, and then click the **OK** button.

Edit template

Use the following procedures to edit the existing system-defined template and user-defined template.

1. From the **Templates view**, select the template to be edited and then select **Edit...** from the context menu. Or, selects the category to be edited from the "Categorization of the template" in **Templates** page, and then click the **Edit...** button.

   The **Edit Template** dialog box is displayed.

Table 3.7 Edit Template dialog box

Item	Description
Parent name	Displays the selected category or root category as parent.
Category	"Templates" is checked.
Template	
Name	Template name.
Keyword	Template keyword.
Description	Template description.
Pattern	Template pattern.

2. Make any desired changes, and then click the **OK** button.

Delete root category or category

Use the following procedures to delete a category or root category (except the predefined category). Both user-defined categories and system-defined categories can be deleted.

1. Select the category or root category that you want to delete from the **Templates view** and then select **Delete** from the context menu. Alternatively, select the category or root category that you want to delete from the "Categorization of the template" in **Templates** page, and then click the **Delete** button.

   The **Delete** dialog box is displayed.

2. Click the **Yes** button.

Delete template

Use the following procedures to delete a template. Both user-defined templates and system-defined templates can be deleted.

1. Select the template that you want to remove from the **Templates view**, and then select **Delete** from the context menu. Alternatively, select the template you want to delete from the "Categorization of the template" in **Templates** page, and then click the **Delete** button. The **Delete** dialog box is displayed.

2. Click the **Yes** button.

3.9.1.3 Import and export template

Import Template

It is possible to import a template or category into a category or root category. When importing the file that has a category or root category with the same name as the selected category or root category, the contents of category and root category will be merged.

When importing a root category and merging it into an existing root category, the file extent relationship of the existing root category will remain and not be changed.

Use the following procedures to import.

1. Display the **Templates** page.

2. Select the template or category that you want to import from the "Categorization of the template", and then click the **Import...** button**.** The **Importing Template** dialog box is displayed.

3. Select the path that is the import source of the template or category, and the file (XML file in valid form), and then click the "Open" button.
   The template or category is imported. An error message is displayed when an invalid XML file is selected.

When importing a template, the name of template can be used repeatedly.

Note

An XML file that only has a template or category (which does not have a route category) cannot be imported into the main root ("templates.xml").

Export template / category

Use the following steps to export the category or template.

1. Display the **Templates** page.

2. Select the template or category that you want to export from the "Categorization of the template", and then click the **Export...** button. The **Exporting Templates** dialog box is displayed.

3. Input the name of the XML file after selecting the path that is the export destination of the template or category, and then click the **save** button.
   The template or category is exported.

Export all

Use the following steps to export all templates, categories and root categories.

1. Display the **Templates** page.
2. Clicks the **Export All** button.

   The **Exporting Templates** dialog box is displayed.
3. Input the name of the XML file after selecting the path that is the export destination of the template or category, and then click the save button.

   The template or category is all exported.

3.9.1.4 Validation/Invalidation of template

Use the context menu for the **Templates view** to enable or disable the template.

- The validated template is displayed in the editor input support candidate list (Content Assist).
- The invalidated template cannot be displayed in the editor input support candidates list (Content Assist).

Select the template from the **Templates view**, and then select "Enable" or "Disable" from the context menu.

3.9.2 Context menu of Templates view

elements				menu		Description
Templates.xml	root category	category	template			
S				Add...	Root category	Adds the root category.
	S	S			Category	Adds the category.
	S	S	S		Template	Adds the template.
	S	S	S	Edit...		Edits the selected elements.
	S	S	S	Delete		Deletes the selected elements. The root category for COBOL cannot be deleted.
			S	Insert		Inserts the template pattern. This menu item can only be selected when the template is valid.
			S	Insert with Parameter Value...		Inserts the template pattern containing only one argument. This menu item can only be selected when the template is valid.
			S	Insert with Multiple Parameter Values...		Inserts the template pattern containing two or more arguments. This menu item can only be selected when the template is valid.
			S	Enable		Enables the template. This menu item can only be selected when the template is invalid.
			S	Disable		Disables the template. This menu item can only be selected when the template is valid.
			S	Copy		Copies the selected template pattern.

elements				menu	Description
Templates.xml	root category	category	template		
					The copied template pattern can be pasted in any editors. This menu item can only be selected when the template is valid.
S	S	S	S	Refresh	Refreshes the **template** dialog.

Meaning of signs

S: Menu can be selected

Space: Menu cannot be displayed.

3.9.3 Templates page

Using the Template page, it is possible to create a new template or edit the configuration of an existing template.

Use the following procedures do display the **Templates** page.

1. From menu bar select **Window** > **Preferences**.
   The **Preferences** dialog box is displayed.
2. From the left pane select **COBOL** > **Templates**. The **Templates** page is displayed.

Table 3.8 Templates page

Items	Descriptions
Categorization of template	Template list appears. - When selecting root category Displays the category description in "Description", and then displays the file extent in "File extensions". - When selecting template Displays the template in "Descriptions", displays the keyword in "Keyword", and then displays the template pattern in "Pattern preview".
New...	Creates a root category, category or template.
Edit...	Edits the selected category and template.
Remove	Deletes the selected category and template.
Import...	Imports template.
Export...	Exports template.
Export All...	Exports all templates into the file system.

Note

When "Templates.xml" is the root node in "Categorization of template", removing and changing cannot be executed.

Chapter 4 Project

4.1 Overview

NetCOBOL Studio manages the resources and information necessary for program development in individual units called "projects".

The following projects are used for COBOL program development.

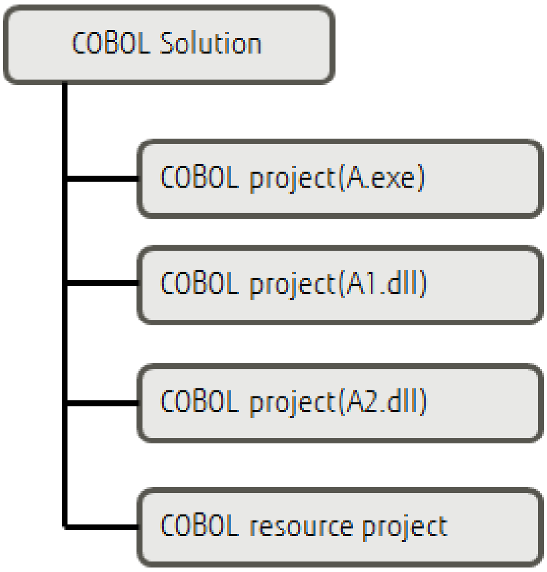
- COBOL Solution project
- COBOL project
- COBOL Resource project

4.1.1 COBOL Solution project

The COBOL solution project contains one or more projects (COBOL project, COBOL solution project).

The COBOL solution project can set the common options for the projects.

For example, when an application is created that consists of an executable file and multiple dynamic link libraries, the project that targets an executable file can be created. Also the multiple projects that target each dynamic link library can be created. The COBOL solution project can build and manage these projects.



4.1.2 COBOL project

The COBOL project is used for COBOL program development.

One COBOL project manages the resources of one target (executable program or library).

4.1.3 COBOL Resource project

COBOL resource projects can be used to effectively manage COBOL resources.

For example, a referenced Library-file from multiple COBOL projects can be registered to a COBOL resource project, and also can be referenced from multiple projects.

COBOL resource projects cannot be built.

COBOL resources that can be managed in a COBOL resource project are as follows.

File type	File extent (recommendation)
COBOL library file	*.CBL/*.COB/*.COBOL
Screen form descriptor file	*.SMD/*.PMD (*1)
Repository file	*.REP
Library file	*.LIB
Object file	*.OBJ

*1:

If the file is a form descriptor used for UTF-32, the file extension will be .SMU or .PMU. The form descriptor used for UTF-32 is created by the form descriptor used for the CNVMED2UTF32 command. The form descriptor used for UTF-32 (.SMD/.PMD) cannot be directly updated by using FORM and PowerFORM. When you want to update the selected form descriptor file from NetCOBOL Studio, create the form descriptor used for UTF-32 by the form descriptor used for the CNVMED2UTF32 command before building it.

Register the screen form descriptor before conversion (.SMD/.PMD) and the screen form descriptor after conversion (.SMU/.PMU) into a COBOL resource project.

For details about the form descriptor used for CNVMED2UTF32 command, refer to the "NetCOBOL User's Guide".

4.1.4 Notes for project management

- Projects for 32-bit and 64-bit COBOL applications cannot be created in the same workspace. Separate workspace folders must be specified for 32-bit and 64-bit COBOL projects.
- The following files that are created in the project folder must not be edited:
  - .Settings\org.eclipse.core.resources.prefs
  - .CobolOptions
  - .project
  - build.xml
- When using a workspace that was used in an old version of NetCOBOL Studio, some menu items do not appear. In this case, perform the following steps.
  1. Select **Window** > **Close All Perspectives** from the menu bar.
     All open perspectives will be all closed.
  2. Select **Window** > **Open Perspectives** > **Other...** from the menu bar.
     The **Open Perspective** dialog box is displayed.
  3. Select "COBOL (default)", and then click the **OK** button.

4.2 Wizards

To create a COBOL resource, the following wizards are available in NetCOBOL Studio.

Generated resource	Wizard
COBOL Solution project	COBOL Solution project generation wizard
COBOL project	COBOL project generation wizard
COBOL Resource project	COBOL Resource project generation wizard
COBOL Source file	COBOL Source generation wizard
COBOL Source file(Object-Oriented)	Object-Oriented COBOL Source generation wizard
COBOL library file	COBOL library generation wizard

Point

The encoding of the newly created COBOL source file is as followings.

- When starting the COBOL project generation wizard and creating a COBOL project, it will be the specified "Text file encoding" in the target definition page.
- When adding a new COBOL source file to an existing COBOL project, it will be the selected encode that is the "Text file encoding" in the "Resource" page of project property.

4.2.1 COBOL solution generation wizard

Use the following procedures to start the COBOL solution generation wizard.

1. Select **File** > **New** > **COBOL Solution Project** from the menu bar. The **New COBOL Solution Project** wizard is started.
2. Input the basic project information into the "COBOL Solution Project" page, and click the **Next** or **Finish** button.

Table 4.1 COBOL Solution Project information

Items		Descriptions
Project name		Project name.
Contents		Specify the location where the project resource should be saved.
	Create new project in workspace	Saves the project resource in the workspace folder.
	Create new project in external location	Saves the project source in the external workspace. When this item is selected, the **Browse** button is enabled, Click the **Browse** button and select the folder of the save location.

3. When the **Next** button is clicked, the **Project Management** page is displayed and projects can be added into the "Solution's projects". For details about the **Project Management** page, refer to "4.3.1 Manage COBOL project".

Information

Default values of COBOL solution project settings can be changed. For detail, refer to the "4.6 Default values of New COBOL Project settings"

4.2.2 COBOL Project generation Wizard

Wizards are used to create COBOL projects and COBOL source files. This section explains the COBOL project wizard used to create a COBOL project. The COBOL project generation wizard defines a target. To create a COBOL project:

1. Select **File** > **New** > **COBOL Project** from the menu bar.
2. Specify the project information, and click the **Next** button.

Table 4.2 Project information

Items		Descriptions
Project name		Specify a name for the project.
Contents		Specify the storage location of project resources.
	Create new project in workspace	If this is selected, project resources are stored in the workspace folder.
	Create new project in external location	If this is selected, project resources are stored in a folder other than the workspace folder. The storage folder can be selected by clicking the **Browse** button.

3. Define a target, and click the **Next** button.

   Define the target type and target file name of the COBOL program to be created.

Table 4.3 Defining the target

Field		Description
Target type		Specify the target type of the COBOL program to be created. When creating an executable file (exe), select "Executable". When creating a dynamic link library (dll), select "Dynamic-link library".
	Use the DLL specific runtime initialization file(COBOL85.CBR)	For a target type of Dynamic-link library, if this option is selected, an initialization file for DLL-specific execution is used.
Target name		Specify a file name for the target file (exe/dll) to be created.
Application format		Specifies the format of the created application.
	COBOL console window	Specifies that a console window created by a COBOL program is the input-output destination of ACCEPT and DISPLAY statements, and that a message box is the output destination of execution-time error messages.
	System console window	Specifies that the system console (command prompt window) is the input-output destination of ACCEPT and DISPLAY statements and execution-time error messages.
Use precompiler		Select this item to create a COBOL project that uses the precompiler.
Text file encoding		Select the text file encoding for the files to be created in the project.

Information

- After the project is created, text file encoding can be changed as follows:

  1. Select the project for which the Text file encoding is to be changed from the **Dependency view** or the **Structure view**.
  2. Select **Property** from the context menu. The Properties dialog box is displayed.
  3. Select Info from the left pane. The Info page is displayed.
  4. Select file encoding to be changed from "Text file encoding" in the Info page. Click **OK** in the Properties dialog box.

- Default values of COBOL project settings can be changed. For detail, refer to the "4.6 Default values of New COBOL Project settings".

Note

- If the Text file encoding is changed, file encoding of existing project files are not changed.
  When the Text file encoding is different from the file encoding of a project, a compile error or a runtime error might occur during the build or the execution. These encodings need to correspond.

- In a remote development, when you build COBOL source file of the text file encode "UTF-8", installs NetCOBOLV10.0.0 or later in the server side.

4. If "Use precompiler" is checked in the Define the target page, then specify the precompiler link information, and click the **Next** button.

Table 4.4 Precompiler link information

Field	Description
Precompiler command	Specifies the name of a command that starts the precompiler.
Precompiler parameters	Specifies the precompiler command parameters.

Field	Description
Precompiler source extension	Specifies the extension of an input source file for the precompiler. The following extensions cannot be specified: - cobol - cob - cbl - lcai
Precompiler output source extension	Select the file extension for the output source file from the precompiler.
Use original source line numbers for error messages	If this item is checked, COBOL compiler error messages are displayed using the line number in the original source, rather than the line number of the preprocessed source. (The INSDBINF (*1) command is invoked.) The default setting is unchecked.
INSDBINF parameters	This item sets parameters of the INSDBINF command, which develops line information for the precompiler input source in COBOL source files generated by precompilation. Note that input and output source file names cannot be specified because they are determined from precompiler input source file names.

*1: For the INSDBINF command, refer to "6.2.2 INSDBINF command".

Specify precompiler link information for the COBOL project to be created.

For more precompiler link information, see "6.2.1 Setting and changing initial values of precompiler link information".

5. Select "Generate code". Select either "COBOL Source" or "Object-Oriented COBOL Source" and click the **Finish** button. A COBOL project is generated, and the source generation wizard is displayed.

Table 4.5 Making selections

Item		Description
Do not generate code		A project is created, and no source code is generated with the project.
Generate code		A project is created using a wizard that generates a template of source code. Select a wizard from "Available skeleton codes".
Available skeleton codes		If "Generate code" is specified, select a source code generation wizard. Select either "COBOL Source" or "Object-Oriented COBOL Source"
	COBOL Source	
	Object-Oriented COBOL Source	

Specify whether to generate source code. To generate source code before creating the project, select the appropriate source code generation wizard from Available skeleton codes.

For details on the source generation wizard, see "4.2.4 COBOL source generation wizard."

Information

Default values of COBOL Solution Project settings are changed in the following way.

1. Select **Window** > **Preferences** from the menu bar. The Preferences dialog box appears.
2. Select **COBOL** > **Solution/Project** in the left pane. The **Solution/Project** page is displayed.
3. Change the values of settings on the **Solution/Project** page, and click the **OK** button.

4.2.3 COBOL resource generation wizard

Use the following procedures to start the COBOL resource generation wizard.

1. Select **File** > **New** > "COBOL Resource Project" from the menu bar. The "New COBOL Resource project" wizard is started.
2. Input the project information into the "COBOL Resource Project" page, and click the **Finish** button.

Table 4.6 Project information

Items	Descriptions
Project name	Project name.
Contents	The location where the project resource should be saved.
Create new project in workspace	Saves the project resource in the workspace folder.
Create new project in external location	Saves the project source in the external workspace. When this item is selected, the **Browse** button is enabled. Click the **Browse** button and selects the folder of the save location.

4.2.4 COBOL source generation wizard

To use the COBOL source generation wizard to create a COBOL source file:

1. Select **File** > **New** > **COBOL Source** from the menu bar.
2. Specify basic information for the COBOL source on the COBOL Source Information page, and click the **Finish** button.

Field	Description
Project name	Enter a name for the project in which the source file is created.
File name	Specify a name for the COBOL source file to be created.
PROGRAM-ID	Enter a PROGRAM-ID. The PROGRAM-ID is the same as the *File name* by default. To change the PROGRAM-ID, enter a new one in this field.
File comment	Optional: Enter a comment to be added at the beginning of the generated COBOL source code.
Use precompiler	Select this item to generate a precompiler input source. This item is enabled only if precompiler link information has been set for the workspace. For details on setting precompiler link information for a workspace, see "6.2.1 Setting and changing initial values of precompiler link information".

4.2.5 Object-oriented COBOL source generation wizard

To use the object-oriented COBOL source generation wizard to create an object-oriented COBOL source file:

1. Select **File** > **New** > **Object-Oriented COBOL Source** from the menu bar.
2. Specify basic information for the object-oriented COBOL source on the Object-Oriented COBOL Source Information page, and click the **Finish** button.

Field	Description
Project name	Enter a name for the project in which the source file is created.
File name	Specify a name for the COBOL source file to be created.
CLASS-ID	Enter a CLASS-ID. The CLASS-ID is the same as the *File name* by default. To change the CLASS-ID, enter a new one in this field.
Superclass	Optional: Specify the parent class name of the class to be generated. Specification of the parent class name can be omitted. In that case, FJBASE is assumed to be the parent class.

Field	Description
File comment	Optional: Enter a comment to be added at the beginning of the generated COBOL source code.
Use precompiler	Select this item to generate a precompiler input source. This item is accessible only if precompiler link information has been set for the workspace. For details on setting precompiler link information for a workspace, see "6.2.1 Setting and changing initial values of precompiler link information".

4.2.6 COBOL Library generation wizard

To use the COBOL Library generation wizard to create a COBOL library file:

1. Select **File** > **New** > **COBOL Library** from the menu bar.
2. Specify basic information for the COBOL library on the New COBOL Library Generation page, and click the **Finish** button.

Field	Description
COBOL Library File Name	Specify a name for the COBOL Library file to be created.
Project	Select if the destination of the COBOL Library file is an existing project.
Project name	Specify a name for the project in which the COBOL Library file is to be created.
External Folder	Select if the destination of the COBOL Library file is outside of the project.
Folder	Specify a name for the folder in which the COBOL Library file is to be created.

4.3 COBOL Solution Project

For the summary of COBOL solution, refer to "4.1.1 COBOL Solution project".

4.3.1 Manage COBOL project

4.3.1.1 Adding a project

The following projects can be added to the COBOL solution project:

- COBOL project
- COBOL resource project

Use the **Project Management** dialog box to add a project.

1. Select **Project Management** from the context menu of the COBOL solution project.
   The **Project Management** dialog box is displayed.
2. Select the project to be added and select a project from **Workspace's projects**, and then click the **Add-->** button.
   A project is added in **Solution's projects**.
3. Click the **Finish** button.

Note

- A project can only be added to one COBOL solution project. To add it in another COBOL solution project, close the COBOL solution project that already has it or remove it from the COBOL solution project. Close a project by selecting the project to be closed and then selecting "Close Project" from "Project" menu, or by selecting the project to be closed in the **Navigator view** and then selecting "Close Project" from context menu.

- When the COBOL solution project exists in workspace, if import the other COBOL solution project or update the multiple COBOL solution projects that are shared among projects, the project has been added to the multiple COBOL solutions. In this case, to make sure that the project has been added to one of the COBOL solution, close the COBOL solution project, or remove the project from the COBOL solution project.

4.3.1.2 Remove Project

When Remove Project is selected the target project is removed from the COBOL solution project.

Remove a project by using the **Project Management** dialog box.

1. Select the **Project Management** from the context menu of the COBOL solution project.
   The **Project Management** dialog box is displayed.
2. Select the project that you want to remove from the **Solution's projects** and then click the **<-- Remove** button.
   The selected project is deleted from the **Solution's projects** and added into the **Workspace's projects**.
3. Click the **Finish** button.

Note

A project that is in the following state is unable to be added into a **Workspace's projects** after being removed from a COBOL solution project.

- Project does not exist in the workspace
- The COBOL solution project in operation has been added to another COBOL solution project

4.3.2 Operate project together

It is possible to operate the projects together in the COBOL solution project using the COBOL solution project context menu that is available from the **Dependency view** or **Structure view**. The context menu is displayed when the COBOL solution project is selected and then right-clicked.

For help on the COBOL solution project context menu, refer to "3.1.2.3 COBOL Solution Project" for the **Dependency view**, and refer to "3.2.2.3 COBOL solution project" for the **Structure view**.

4.3.3 Setting common options for the projects

It is possible to set the common options for the projects in the COBOL solution project by using the COBOL solution property dialog.

- Build
- Build Tool (Pre-compiler)
- Build Tool (Resource Compiler)
- Remote Development

Moreover, to enable the build options that are set in the COBOL solution project, uncheck the "Enable project specific setting" check box in the **Build** page of each constitution project property.

Information

When the "Enable project specific setting" check box is unchecked, the options that were specified in each COBOL project are disabled.

4.4 COBOL Project

For the summary of COBOL solution, refer to "4.1.2 COBOL project".

4.4.1 Adding Existing COBOL Resources

This section explains how to add existing COBOL resources to a COBOL project.

Note

- Text file encoding

  When existing COBOL resources are added to a COBOL project, the project's text file encoding setting is applied for the added resources. The project's text file encoding is specified in the Info page of the Properties dialog box of the project. COBOL resources which have BOM (Byte Order Mark) are treated as UTF-8.

- Code conversion

  Although the text file encoding of existing COBOL resources are changed when they are added to a COBOL project, code conversion is not invoked. File encoding of the COBOL resources remain unchanged.

4.4.1.1 Adding a COBOL source file

COBOL source files registered in the Source Files folder of a COBOL project in the Dependency or **Structure view** are compilation processing targets. To register an existing COBOL source file in the Source Files folder:

- For a COBOL source file that is in the project

  1. Select the **Source Files** folder in the **Dependency view**.
  2. Select **Add File...** from the context menu.
  3. The **Add Source Files** dialog box corresponding to the selected project appears.
  4. Select the file to be added to the Source Files folder.
  5. Click the **OK** button. The selected source file is added to the Source Files folder in the **Dependency view**. The added file is reflected in the **Structure view**.

- For a COBOL source file that is not in the project, either of the following methods can be used:

  - Using Windows Explorer, select the COBOL source file to be registered and select "Copy" from the context menu. Select the Source Files folder, and then select "Paste" from the context menu.
  - Using Windows Explorer, drag and drop the COBOL source file into the Source Files folder.

4.4.1.2 Adding a link file

To add a link file:

1. Select the "Linking Files" folder of the relevant project.
2. Select **Add File...** from the context menu. The **Open** dialog box appears.
3. Select the file to be added to the "Linking Files" folder.
4. Click the **OK** button. The selected file is added to the "Linking FIles" folder in the **Dependency view**. The added file is reflected in the **Structure view**.

4.4.1.3 Adding a dependent file

To add a dependent file:

1. Select the "Dependent Files" folder to which the file is to be added.
2. Select **Add File...** from the context menu. The **Open** dialog box appears.
3. Select the file to be added to the "Dependent Files" folder.
4. Click the **OK** button. The selected file is added to the "Dependent Files" folder.

4.5 COBOL Resource Project

For the summary of COBOL resource project, refer to "4.1.3 COBOL Resource project".

4.5.1 Adding the existing COBOL resource

Adding a file in the project

Files in the project can be added as follows.

- Drag & drop files to the COBOL resource project from the registered file in the **Dependency view**.

Adding a file outside the project

Files outside the project, can be added in one of the following ways.

- Add the existing COBOL resource into a COBOL resource project by using the File system option for the import function. For details, refer to the file system of the Import Wizard.
- Drag & drop files to the COBOL resource project from Windows Explorer.

4.5.2 Take reference from other projects

A Resource that is managed by a COBOL resource project can be referenced in other projects.

The steps to reference a library-file of a COBOL resource project from a COBOL project are shown below.

It is assumed that the COBOL project and the COBOL resource project have already been registered in the workspace.

1. Display the "Compiler Options" page of the COBOL project.
   For details about the display method of the "Compiler Options" page, see "6.1.3 Setting compile options".
2. Click the **Add...** button.
   The **Add Compiler Options** dialog box is displayed.
3. Select the "LIB" from the "Compiler options" list, and then click the **Add...** button.
   The **LIB Compiler Option** dialog box is displayed.
4. Click the **Browse...** button.
   The **Folder type selection** dialog box is displayed.
5. Select "Specify from the project" from the "Folder type" list and then click the **OK** button.
   The **Selection from project** dialog box is displayed.
6. Select the COBOL resource project that contains the library-file you want to refer to on the tree, and then click the **OK** button.
   Click the **OK** button of the **LIB Compiler Option** dialog box.
   The LIB (the path of the selected COBOL resource project) is set in the "Compiler options" list.

4.6 Default values of New COBOL Project settings

Default values that are set when a new project is created from the COBOL solution generation wizard or the COBOL project generation wizard are changed in the following way.

1. Select **Window** > **Preferences** from the menu bar. The Preferences dialog box appears.
2. Select **COBOL** > **Solution/Project** in the left pane. The **Solution/Project** page is displayed.

Field	Description
Target type	Specifies the type of TARGET to be output. "Executable" or "Dynamic-link library" can be selected. The default is "Executable".
Application format	Specifies the type of a console window executable applications use.
Use Precompiler	Check if precompiler is used.

3. Change the values of settings on the **Solution/Project** page, and click the **OK** button.

Chapter 5 Editor

This chapter provides information regarding the COBOL editor, which is used to edit COBOL source files. To start the COBOL editor, select the file to be edited in the **Dependency view** or **Structure view**, and double-click the file or select **Open** from the context menu. An editing window is displayed when the COBOL editor is started, and the name of the file is displayed on the tab of the editing window. When the file has unsaved changes, an asterisk (*) is displayed after the file name.

5.1 Highlighting Key Words

Key words can be highlighted with colored characters or with a bold font style. The following table lists the items that can be highlighted.

Item	Description
Comment line	If "*" or "/" is in the indicator area at the beginning of a line, the entire line is commented out. If "D" or "d" is in the indicator area at the beginning of a line, the entire line is commented out, regardless of whether the line contains debugging information.
Comment in a line	If a line contains "*>", the part from that point to the end of the line is treated as a comment in the line.
Reserved word	A reserved word list is defined for each COBOL version. For reserved words, see the NetCOBOL Grammar. Reserved words are not case-sensitive.
Figurative constant	SPACE, SPACES, ZERO, ZEROS, ZEROES, HIGH-VALUE, HIGH-VALUES, LOW-VALUE, LOW-VALUES, QUOTE, QUOTES, ALL Figurative constants are not case-sensitive.
Special register	LINAGE-COUNTER, PROGRAM-STATUS, RETURN-CODE, SORT-STATUS, EDIT-MODE, EDIT-OPTION, EDIT-OPTION2, EDIT-OPTION3, EDIT-COLOR, EDIT-STATUS, EDIT-CURSOR, LINE-COUNTER, PAGE-COUNTER, SORT-CORE-SIZE Special registers are not case-sensitive.
Character string	A character string is defined with any of the following formats: " ",B" ",X" ",N" ",NC" ",NX" " Character strings are not case-sensitive. Single quotation marks or double quotation marks can be used.

Displaying the highlight setting

To display the highlight setting:

1. Select **Window** > **Preferences** from the menu bar. The **Preferences** dialog box appears.
2. Select **COBOL** > **Editor** in the left pane. The **Editor** page is displayed.
3. Click on the **Colors** tab. The Foreground, Color, Bold, and Preview areas are displayed.
4. Select an item in the Foreground area. The associated style is displayed in the Color and Bold areas. The current setting is displayed in the Preview area.

Note

By default, reserved words are displayed in bold characters, and other items are displayed in the normal style.

Changing the highlight setting

To change the highlight setting:

1. In the Foreground area, select the item whose highlight setting is to be changed.

2. Select **Color**. The **Color** dialog box appears.
3. Select the color to be applied from the "Basic colors" palette. A color can be created by selecting "Custom colors".
4. To apply the selected color and close the dialog box, click the **OK** button. The selected color is reflected in the displayed contents of the dialog box. To close the dialog box without applying the selected color, click the **Cancel** button.
5. To display text in bold characters, select **Bold** check box

5.2 Font Setting

The font used in the COBOL editor can be changed in the **Preferences** dialog box.

Changing the font setting

To change the font setting.

1. Select **Windows** > **Preferences** from the menu bar. The **Preferences** dialog box appears.
2. Select **COBOL** > **Editor** in the left pane. The **Editor** page is displayed.
3. Click on the **General** tab.
4. Select **Change...**. The **Font** dialog box appears.
5. Change the values of Font, Font style, and Size as desired.

5.3 Changing the width of the tab character

To change the width of the tab character:

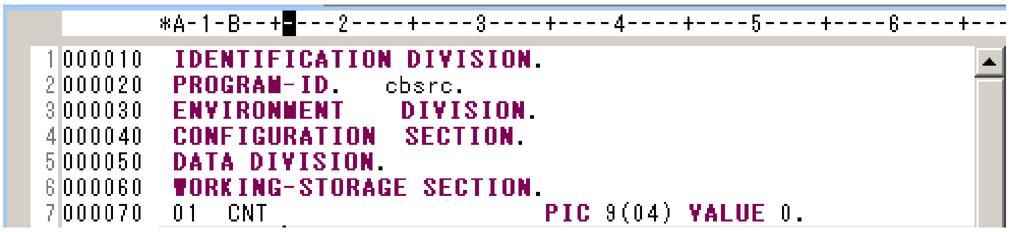
1. Select **Windows** > **Preferences** from the menu bar. The **Preferences** dialog box appears.
2. Select **COBOL** > **Editor** in the left pane. The **Editor** page is displayed.
3. Click the **Reference Format** tab.
4. Specify 4 or 8 as the tab width value for **Tab width**.

Note

By default, 4 is set for the tab width value.

5.4 Displaying Line-Numbers

Line numbers can be displayed alongside the vertical ruler of the COBOL editor.



```
         *A-1-B--+----2----+----3----+----4----+----5----+----6----+---
1 000010  IDENTIFICATION DIVISION.
2 000020  PROGRAM-ID.   cbsrc.
3 000030  ENVIRONMENT    DIVISION.
4 000040  CONFIGURATION  SECTION.
5 000050  DATA DIVISION.
6 000060  WORKING-STORAGE SECTION.
7 000070  01   CNT                      PIC 9(04) VALUE 0.
```

Displaying line-numbers

To display line-numbers:

1. Select **Windows** > **Preferences** from the menu bar. The **Preferences** dialog box appears.
2. Select **General** > **Editors** > **Text Editors** in the left pane. The **Text Editors** page is displayed.

3. Select the **Show line numbers** check box to display line numbers.

Note

Setting information is shared among editors; therefore, changing the settings in the NetCOBOL Studio Editor changes those settings in the other editors also.

5.5 Sequence Numbers

The COBOL editor supports automatic numbering of sequence numbers. The COBOL editor supports the following two patterns of sequence numbers:

Pattern A

Every line in a file has a six-digit sequence number, and the lines are sorted in ascending order.

Pattern B

Patterns other than pattern A.

Sequence numbers can be edited. The following table describes the operation of the COBOL editor when lines are numbered manually.

Adding a new line at the end of a file	In pattern A, the sequence number for a new line is the sequence number of the last line plus the increment value. If the calculated value is greater than 999999, 1 is set as the increment value. If a line is being added after the line whose sequence number is 999999, the numbering is switched to pattern B. In numbering in pattern B, a six-byte space is assigned for the sequence number area of the newly line.
Inserting a new line	In pattern A, the sequence number for a new line is the sequence number of the previous line plus the increment value. If the calculated value is the sequence number of the next line or greater, 1 is set as the increment value. However, if adding 1 to the sequence number of the previous line results in a value equal to or greater than the sequence number of the next line, the next and subsequent lines are renumbered. In pattern B, a six-byte space is assigned for the sequence number area of the new line.
Deleting a line	When a line is deleted, lines are not renumbered.
Pasting a line	The processing is the same as for inserting a new line.

Note

Sequence numbers are not targeted in revision history comparisons. Therefore, simply reassigning sequence numbers does not correct them. However, when replacing from a revision history, the sequence numbers are also replaced.

5.5.1 Renumbering Lines

The initial value and increment value for numbering can be changed, and lines can be renumbered. The maximum increment value is 999999.

Renumbering lines

To renumber lines:

1. Select **Edit** > **Renumber** from the menu bar, or press the "Ctrl + R" keys. The **Renumber** dialog box appears.
2. Specify values for Initial Value and Increment Value.
3. Click the **Renumber** button to renumber the lines and close the dialog box. To close the dialog box without renumbering the lines, click the **Cancel** button.

Note

By default, 10 is set for Initial Value, and 10 is set for Increment Value. If the values set for Initial Value and Increment Value would result in a line number equal to or greater than 999999 in a file, an error message is displayed in the **Renumber** dialog box.

5.5.2 Changing the Initial Value and Increment Value for Sequence Numbers

The initial value and increment value for sequence numbers can be changed.

Changing the initial value and increment value for sequence numbers

To change the initial value and increment value for sequence numbers:

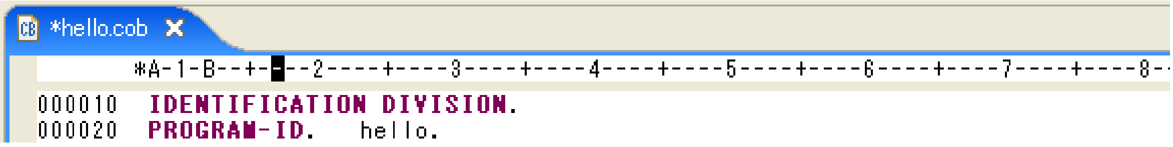
1. Select **Windows** > **Preferences** from the menu bar. The **Preferences** dialog box appears.
2. Select **COBOL** > **Editor** in the left pane. The **Editor** page is displayed.
3. Click on the **Reference Format** tab. The **Reference Format** page is displayed.
4. Specify values for "Initial value" and "Increment value" of "Initial and incremental values for sequence number".

Note

By default, 10 is set for the Initial value and the Increment value.

5.6 Display of the Horizontal Ruler

The horizontal ruler is displayed in the upper part of the editor window. This section describes the features of the horizontal ruler.



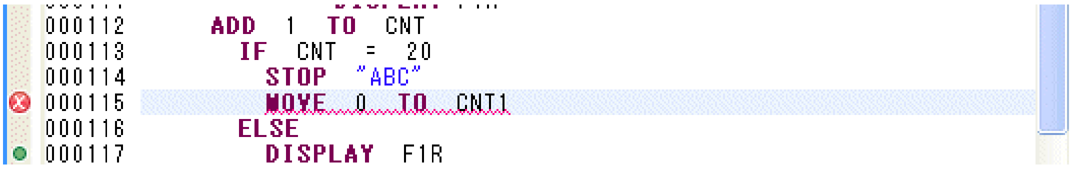
```
       *A-1-B--+---2----+----3----+----4----+----5----+----6----+----7----+----8--
000010  IDENTIFICATION DIVISION.
000020  PROGRAM-ID.    hello.
```

- Column numbers are displayed in every tenth column. On the ruler, "1" indicates column 10, "+" indicates column 15, "2" indicates column 20, and so on.
- For a fixed format file:
  - The ruler begins with "*" at column 7.
  - "A" in column 8 indicates that the A area begins from this column. The A area is from column 8 to column 11.
  - "B" in column 12 indicates that the B area begins from this column. The B area is from column 12 to column 72.
  - "I" in column 73 indicates that the program identification area begins from this column. The program identification area is column 73 and subsequent columns.
  - The ruler display ends at column 80.
- For a variable format file:
  - The ruler begins with "*" at column 7.
  - "A" in column 8 indicates that the A area begins from this column. The A area is from column 8 to column 11.
  - "B" in column 12 indicates that the B area begins from this column. The B area is from column 12 to column 251.
  - The ruler display ends at column 251.
- For a free format file:
  - The ruler begins at column 1.

A value can be selected for the width of the tab character, which is inserted by pressing the **Tab** key. Either 4 or 8 can be selected.

5.7 Vertical Ruler

Icons that represent set breakpoints, errors that have occurred during build, and other items are displayed on the vertical ruler in the left part of the COBOL editor window.



```
000112      ADD  1  TO  CNT
000113        IF  CNT  =  20
000114          STOP  "ABC"
000115          MOVE  0  TO  CNT1
000116        ELSE
000117          DISPLAY  F1R
```

The COBOL editor supports the following display items:

- Errors
- Tasks
- Bookmarks
- Breakpoints
- Warnings
- Search results
- Debugging of instruction points
- Debugging of invocation stacks

Changing the display settings

To change the display settings of the vertical ruler:

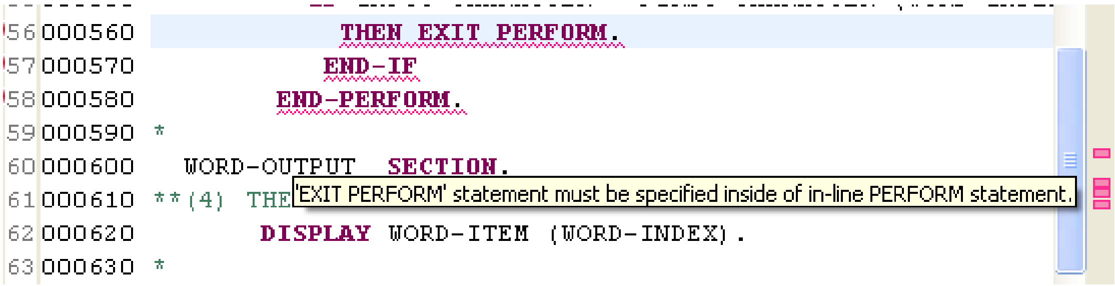
1. Select **Windows** > **Preferences** from the menu bar. The **Preferences** dialog box appears.
2. Select **General** > **Editor** > **Text Editors** > **Annotations** in the left pane. The **Annotations** page is displayed.
3. Select a setting item from "Annotation types". The setting indicating whether to display the selected item on the vertical ruler can be changed, as well as the color used to display it.

Note

Setting information is shared among editors; therefore, changing the settings in the NetCOBOL Studio Editor changes those settings in the other editors also.

5.8 Display of the Overview Ruler

On the overview ruler in the right part of the editor window, the locations of errors such as those that have occurred during build are indicated by rectangles in different colors representing error categories. Selecting a rectangle on the ruler displays the relevant source line.



```
000560              THEN  EXIT  PERFORM.
000570            END-IF
000580         END-PERFORM.
000590 *
000600   WORD-OUTPUT   SECTION.
000610 **(4)  THE
000620          DISPLAY  WORD-ITEM  (WORD-INDEX).
000630 *
```

'EXIT PERFORM' statement must be specified inside of in-line PERFORM statement.

The COBOL editor supports the following display items:

- Errors
- Tasks
- Bookmarks
- Breakpoints
- Warnings
- Search results
- Debugging of instruction points
- Debugging of invocation stacks

Changing the display settings

To change the display settings of the overview ruler:

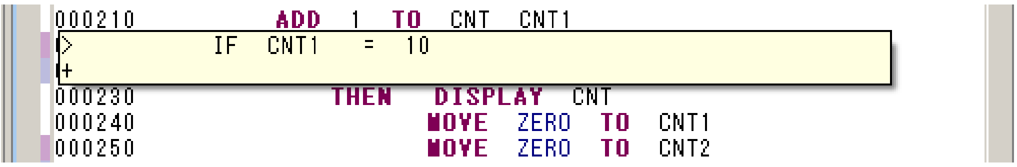
1. Select **Windows** > **Preferences** from the menu bar. The **Preferences** dialog box appears.
2. Select **General** > **Editor** > **Text Editors** > **Annotations** in the left pane. The **Annotations** page is displayed.
3. Select a setting item from "Annotation types". The setting indicating whether to display the selected item on the overview ruler can be changed, as well as the color used to display it.

Note

Setting information is shared among editors; therefore, changing the settings in the NetCOBOL Studio Editor changes those settings in the other editors also.

5.9 Quick Diff Display

Differences between the source code currently being edited and corresponding stored source code can be indicated by different colored blocks alongside the vertical ruler of the editor window.



```
000210          ADD   1  TO  CNT  CNT1
>           IF   CNT1   =   10
+
000230              THEN    DISPLAY  CNT
000240                     MOVE  ZERO  TO  CNT1
000250                     MOVE  ZERO  TO  CNT2
```

Displaying Quick Diff

To change the Quick Diff display settings:

1. Select **Windows** > **Preferences** from the menu bar. The **Preferences** dialog box appears.
2. Select **General** > **Editor** > **Text Editors** > **Quick Diff** in the left pane. The **Quick Diff** page is displayed.
3. Change the Quick Diff settings as desired.

5.10 Reference Formats

When the COBOL editor is used to create or edit a COBOL source file, the file format conforms to the rules specified under a reference format. The COBOL editor supports the following two reference formats:

- Fixed format
- Variable format

- Free format

Specifying a reference format

To specify a reference format:

1. Select **Windows** > **Preferences** from the menu bar. The **Preferences** dialog box appears.
2. Select **COBOL** > **Editor** in the left pane. The **Editor** page is displayed.
3. Click the **Reference Format** tab. The **Reference Format** page is displayed.
4. Select "Fixed format ", "Variable format" or "Free format" for the "Reference Format."

"SRF and TAB compile option setting to be consistent with the applicable editor setting" checkbox

If it is checked, the value of SRF is consistent with **Reference Format** setting, and the value of TAB is consistent with "Tab width" setting. And if it is not checked, these changing settings do not occur.

COBOL editor operation for the fixed format

In the fixed format:

- Columns 1 to 6 in the COBOL editor are used for line numbers, which are called sequence numbers.
- Column 7 is used as the indicator column.
- "/", "D", "d", or "*" in the indicator area indicates that the line is a comment line.
  "D" or "d" in the indicator area indicates that the subsequent text is debugging information. This option is valid for builds that are executed in debug mode.
- Columns 8 to 11 are used as the A area, and columns 12 to 72 are used as the B area.
- Column 73 and subsequent columns are used as the program identification area.

COBOL editor operation for the variable format

In the variable format:

- Columns 1 to 6 in the COBOL editor are used for line numbers, which are called sequence numbers.
- Column 7 is used as the indicator column.
- "/", "D", "d", or "*" in the indicator column indicates that the line is a comment line.
  "D" or "d" in the indicator area indicates that the subsequent text is debugging information. This option is valid for builds that are executed in debug mode.
- Columns 8 to 11 are used as the A area, and columns 12 to 251 are used as the B area.
- The text in column 252 and subsequent columns is treated as a comment.

COBOL editor operation for the free format

In the free format:

- You do not need to distinguish among the sequence number area, the indicator area, the A area, and the B area. You can start the source program at any positions in each line.
- If a line contains "*>", the part from that point to the end of the line is treated as a comment.
- Combining three COBOL characters ">>D" immediately followed by a single-byte space indicates a debugging indicator. A line with zero or one or more blanks before the debugging indicator is called "debugging line". The debugging line leaves debugging information in the source program.
- The text in column 252 and subsequent columns is treated as a comment.

Note

A reference format can be changed. If changes have been made in a file but not yet saved, a message prompting the user to save the changes in the file is displayed before the reference format is changed.

To save the changes made in the file and then change the reference format, click the **Yes** button. To change the reference format without saving the changes, click the **No** button.

5.11 Code Formatter

The following table outlines the automatic indentation supported by the COBOL editor.

When the ENTER key is pressed	All of the space and tab characters at or after the cursor position are moved to the next line together with the text, and the cursor is moved to the beginning of the line.
When multiple lines of text are pasted	The text is pasted at the current cursor position, and then is pasted in subsequent lines without placing any extra space or tab characters at the beginning of each subsequent line.
When the text selected from the input support candidate list is inserted	All of the space and tab characters at the beginning of the current line are also pasted to each new line.

5.12 Comment Styles

The following table lists comment styles supported by the COBOL editor.

Style	Description
Style A	If "*", "/", "D", or "d" is in the indicator area (column 7) of a line, the entire line is treated as a comment line.
Style B	If a line contains "*>", the part from that point to the end of the line is treated as a comment.
Style C	Characters described in the program identification area (column 73 and subsequent columns) are also treated as a comment.

Note

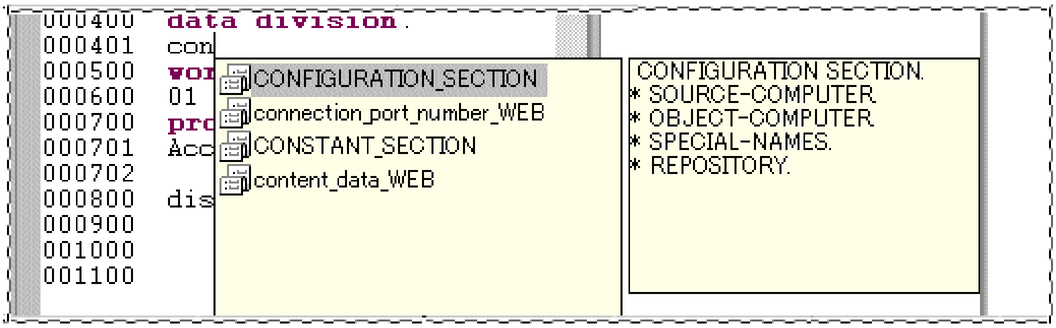
The fixed format supports all of the above comment styles.

The variable format supports style A and style B. In the variable format, column 252 and subsequent columns are treated as a comment.

The free format supports style B.

5.13 Input Support Candidate List (Contents Assist)

The input support candidate list displays key words of the COBOL language syntax, class names, method names (factory method names or object method names), program names, and subprogram names.



```
000400  data division.
000401  con
000500  wor
000600  01
000700  pro
000701  Acc
000702
000800  dis
000900
001000
001100
```

```
CONFIGURATION SECTION.
* SOURCE-COMPUTER.
* OBJECT-COMPUTER.
* SPECIAL-NAMES.
* REPOSITORY.
```

Input support candidates are displayed according to the following conditions:

- When the Ctrl + Space keys are pressed in a file, the candidate list displays class names and template key words. Example: MOVE<Ctrl + Space>

  If no space character has been entered after the last input characters when the Ctrl + Space keys are pressed, the list displays the class names and template key words that begin with the last input characters. In the following example, the list displays the class names and template key words that begin with "DA".

  Example: MOVE DA<Ctrl + Space>.

- When the Ctrl + Space keys are pressed with the cursor to the right of a space character following the key word INVOKE, the candidate list displays class names, method names, and template key words. Example: INVOKE <Ctrl + Space>

  If no space character has been entered after the last input characters when the Ctrl + Space keys are pressed, the list displays the class names, Method names, and template key words that begin with the last input characters. In the following example, the list displays the class names, method names, and template key words that begin with "DA".

  Example: INVOKE DA<Ctrl + Space>

- When the Ctrl + Space keys are pressed with the cursor to the right of a space character that is entered two characters after the key word [CALL], the candidate list displays program names, subprogram names, and template key words.

  Example: CALL <Ctrl + Space>

  If no space character has been entered after the last input characters when the Ctrl + Space keys are pressed, the list displays the program names and template key words that begin with the last input characters. In the following example, the list displays the program names and template key words that begin with "DA".

  Example: CALL DA<Ctrl + Space>

When a template key word is selected, an auxiliary window that displays template patterns is displayed.

5.14 Insert and Overwrite Modes

Text can be inserted and overwritten in the COBOL editor. The insert/overwrite mode can be toggled by pressing the "Insert" key. The current mode is indicated in the status bar.

For example, if "1234567890" is displayed in the text editor with the cursor to the left of the "1" and you type "abcde":

In Insert mode, the result will be: abcde1234567890

In Overwrite mode, the result will be: abcde67890

5.15 Select All

All of the text in the active file of the editor can be selected by choosing **Select All** from the Edit menu or by pressing the Ctrl + A keys.

Note: Sequence numbers in the sequence number area of fixed and variable format files cannot be selected.

5.16 Undo and Redo

Editing operations can be undone and redone.

Undo

Up to 25 previous editing operations can be undone by selecting **Undo** from the Edit menu, or by pressing the Ctrl + Z keys, or by clicking ↩ in the toolbar.

Redo

Editing operations that have been undone can be redone by selecting **Redo** from the Edit menu, or by pressing the Ctrl + Y keys, or by clicking ↪ in the toolbar.

5.17 Shift Left/Right

The selected text in the COBOL editor window can be shifted left and shifted right.

Shift left

The selected text can be shifted left in the COBOL editor window by selecting **Shift Left** from the Edit menu, or by pressing the Shift + Tab keys. The length of the text shift in units of columns is determined by the value set for *Tab width* on the **COBOL** > **Editor** page of the **Preferences** dialog box.

Shift right

The selected text can be shifted right in the COBOL editor window by selecting **Shift Right** from the Edit menu, or pressing the Tab key. The length of the text shift in units of columns is determined by the value set for *Tab width* on the **COBOL** > **Editor** page of the **Preferences** dialog box.

Note

These operations cannot be performed in the sequence number area.

5.18 Cut, Copy, and Paste

The selected text in the editor window can be cut, copied, and pasted.

Cut

The selected text can be cut by selecting **Cut** from the Edit menu or context menu, or by pressing the Ctrl + X keys, or by clicking ✂ in the toolbar.

Copy

The selected text can be copied by selecting Copy from the Edit menu or context menu, or by pressing the Ctrl + C keys, or by clicking 📄 on the toolbar.

Paste

The cut or copied text can be pasted by selecting **Paste** from the Edit menu or context menu, or by pressing the Ctrl + V keys, or by clicking 📋 on the toolbar.

5.19 Find and Replace

A key word can be found and/or replaced by selecting **Find /Replace** from the Edit menu, or by pressing the Ctrl + F keys, or by clicking 🔦 on the toolbar.

Finding a key word

To find a key word:

1. Select **Edit** > **Find/Replace...** from the menu bar. The **Find/Replace** dialog box appears.
2. Enter the key word.
3. Click on the **Find** button. If the key word is found, it is highlighted in the editor window.

Replacing a character string

To replace a character string:

1. Select **Edit** > **Find/Replace** from the menu bar. The **Find/Replace** dialog box appears.
2. Enter the search target character string in the **Find** field, and the replacement character string in the **Replace with** field.
3. Click the **Replace** button to replace text that has been found.

The items in the **Find/Replace** dialog box are as follows:

Item	Description
Forward	A downward (forward) search is performed starting from the current cursor position.
Backward	An upward (backward) search is performed starting from the current cursor position.
All	The entire file is searched.
Select lines	The selected range in the file is searched.
Case sensitive	Uppercase and lowercase letters are treated as different characters in a key word search.
Wrap search	The search operation is performed until it reaches the end of the file, and then it continues at the start of the file. The search operation is repeated cyclically until it is canceled.
Whole word	The search target is a word that exactly matches the key word.
Incremental	The cursor is positioned at a matching character string in the file as soon as each character in a character string search target is entered.
Regular expressions	The specified key word is assumed to be a regular expression.

- When the **Find /Replace** button is clicked, the character string that was found to match is replaced, and the cursor is moved to the next matching character string.
- When the **Replace All** button is clicked, all matching character strings are replaced.
- If the Incremental option is selected, the search is conducted as soon as the search string is input, without the use of the **Find** button.

5.20 Jumping to a Specified Line or Sequence Number

By selecting Go to Line from the Navigate menu or pressing the Ctrl + L keys, you can move to a specified line number or sequence number.

To jump to a specified line number or sequence number in a fixed or variable format file:

1. Select **Navigate** > **Go to Line...** from the menu bar. The **Go to Line** dialog box appears.
2. Select **Line Number** check box to enter a line number in the **Enter line number** field, or select **Sequence Number** check box to enter a sequence number in the **Enter Sequence number** field. The default selection is Sequence Number.
3. If the sequence number is represented by zeros followed by integers, you can jump to the line by entering just the integer part of the number, without the leading zeros. For example, if the sequence number is 000100, you can enter 100 instead of 000100.

4. After selecting to move to the specified line, click the **OK** button to close the dialog box. The cursor moves to the beginning of the specified line. Select the **Cancel** button to close the dialog box without moving to the specified line.

Note

The range of sequence numbers that can be specified for a fixed or variable format file is displayed in the header of the **Enter Sequence number** field.

5.21 Bookmarks

A bookmark can be added to a source code line by selecting "Add Bookmark" from the **Edit** menu, or by using the context menu of the vertical ruler of the editor.

Note

To display your bookmark list, select **Windows** > **Show View** > **Other...** from the menu bar. Select **General** > **Bookmarks** in the **Show View** dialog box.

5.22 Tasks

Task settings can be made by selecting **Add Task...** from the **Edit** menu or context menu of the editor. A task can be used, for example, as a memorandum for a file. The tasks that have been set are displayed in the Task view, which is used to manage task completion. For details on tasks, see the "Workbench User's Guide" in the Help information.

To insert a task in source code:

1. Select **Add Task...** from the context menu of the vertical ruler.
2. Enter an explanation of the task in the **Description** field of the **Properties** dialog box.
3. Select a priority and click the **Completed** check box when the task has been completed.
4. Default values are displayed for "On element", "In folder", and "Location".

Chapter 6 Build Function

Build is a process that uses a set of build tools according to definitions in a project. COBOL program build tool settings are made when a project is created. During a project build, build tools are used in the order specified. Build tool settings and options can be made by project. If a build contains errors, they are displayed in the **Problems view**. A detailed build log can be checked in the **Console view**. To display the build log, select "Build Console" from the "Open Console" icon in the toolbar of the **Console view**.

Point

Error information detected by the precompiler is not displayed in the **Problems view**. The error information is displayed in the **Console view** under Build Console.

Note

- The build.xml file that is created in each project folder is used for build. This file must not be edited, and an Ant build (execution of an Ant script) must not be executed for the file.
- When you build COBOL source file of the text file encode "UTF-8", installs NetCOBOLV10.0.0 or later.

Build Tools

NetCOBOL Studio combines build tools to execute build processing. When creating a new COBOL project, the COBOL compiler and linker are set as build tools. If you specify to use the precompiler in creating the project, the Precompiler build tool is also set. The resource compiler can be added to existing COBOL projects as a build tool.

6.1 COBOL compiler

A project can be compiled using the COBOL compiler.

6.1.1 Files associated with compilation

The following table lists the files associated with the COBOL compiler.

Files used by the COBOL compiler
Source files (*.cob, *.cobol, *.cbl)
Library files (*.cbl)
Screen form descriptor file (*.smd, *.pmd, *.smu, *.pmu)
Repository files (*.rep)

Source files displayed in the Source Files folder of the **Dependency view** are the objects of compilation. The following table lists the files created by the COBOL compiler.

Files created by the COBOL compiler
Object files (*.obj)
Repository files (*.rep)
Debugging information files (*.svd)
Compilation list files (*.lst)
Source Analysis Information (*.sai)
Executable files (*.exe)
Dynamic link libraries (*.dll)

The following table lists details of the files used for compilation.

File contents	File name format	I/O (*1)	Use and creation conditions	Relevant compilation options	Relevant environment variables(*2)
Source files	*arbitrary-file-name*.cob *arbitrary-file-name*.cobol *arbitrary-file-name*.cbl	I	Required	-	-
Library files	*arbitrary-file-name*.cbl	I	For compiling a source program that uses the COPY statement	LIB	COB_COBCOPY COB_LIBSUFFIX COB_*library-name*
Screen form descriptor file	*screen form descriptor*.smd *screen form descriptor*.pmd(*3)	I	For compiling a source program that has the screen form descriptor.	FORMLIB	SMED_SUFFIX
Object files	*source-file-name*.obj	O	Created when compilation is performed correctly	-	-
Repository files	*class-name*.rep	I	For compiling a source program that has the REPOSITORY paragraph	REPIN REP	COB_REPIN
		O	Created when a class definition is compiled correctly		
Debugging information files (for compilation in debug mode)	*source-file-name*.svd *source-file-name*.pdb	O	- Used when debug mode is set as the build mode - When the TEST compiler option is specified	-	-
Source analysis information file	*source-file-name*.sai	O	When the SAI compiler option is specified	SAI	-
Compilation list files	*.lst	O	For output of a compilation list	PRINT	-

*1: I and O are as follows:

- I: source file for compilation
- O: output file as a result of compilation

*2: For detail, refer to the "NetCOBOL User's Guide".

*3: When the encoding of the form data is UTF-32, the UTF-32 form descriptor is created using "Form descriptor conversion command for UTF-32 (CNVMED2UTF32)". The extension of the created descriptor is .smu or .pmu.

For details of CNVMED2UTF32, refer to the "NetCOBOL User's Guide".

The UTF-32 form descriptor is used when the encoding of the national item is UTF-32. In this case, the search order of the files is .pmu and .smu.

Note

The form descriptor for UTF-32 (*.pmu, *smu) cannot be updated by using FORM and PowerFORM. In this case, you update the original form descriptor (*.pmd/*.smd), and convert it by using CNVMED2UTF32.

In the resource management of NetCOBOL Studio, an original descriptor (*.smd/*.pmd) is registered.

6.1.2 Setting the main program

Before performing a build of a COBOL project with an executable file as the target type, specify one of the source files of the project as the main program. If the main program is not specified, a link error will occur. The two types of main programs, determined according to types of programs created, are described below. The main program type is specified with a wizard when a COBOL project is created.

- Application that uses the COBOL console

  Specifies that a console window created by a COBOL program is the input-output destination of ACCEPT and DISPLAY statements, and that a message box is the output destination of execution-time error messages.

- Application that uses the system console

  Specifies that the system console (command prompt window) is the input-output destination of ACCEPT and DISPLAY statements and execution-time error messages.

Point

If the COBOL console window is specified as the main program, and the application is executed from NetCOBOL Studio, the **Console view** is used instead of the system console.

Specifying the main program

To specify the main program:

1. Select a source file in the **Dependency view** or **Structure view**.
2. Select **Main Program** from the context menu.
3. The selected COBOL source file is set as the main program, and the icon of the file is changed.

Changing the main program type

To change the main program type:

1. Select the appropriate project from the **Dependency view** or **Structure view**.
2. Select **File** > **Properties** from the menu bar, or select **Property** from the context menu. The Properties dialog box appears.
3. Select **Target** in the left pane. The **Target** page is displayed.
4. Select "COBOL console window" or "System console window" from "Application format", and click **OK** button.

6.1.3 Setting compile options

This section explains how to set the compile options that are required for building a project.

See

For details of the compile options, refer to the "Appendix A Compile Options".

Compile options are set for each project. The set options are sent to the COBOL compiler as parameters.

To display the Compiler Options page and set compile options:

1. Select a COBOL project in the **Dependency view** or **Structure view**.
2. Select **File** > **Property...** from the menu bar or **Property...** from the context menu. The Property dialog box appears.
3. Select **Build** in the left pane. The **Build** page is displayed.
4. Select the **Compiler Options** tab. The **Compiler Options** page is displayed.

Table 6.1 Compiler Options page

Item	Description
Compiler options	Displays the compile options to be sent to the COBOL compiler.
Add	Adds the compile options. To add compile options, select the compile options in **Compiler Options** in the **Add Compiler Options** dialog box, and then click the **Add** button. For details on compile options, see "A.1 Compile option details."
Change	Changes the compile options selected in Compiler options.
Remove	Deletes the compile options selected in Compiler options.
The CHECK(ALL) compile option is automatically added when the build mode is Debug.	Specifies whether the compiler option CHECK(ALL) is added automatically or not when the build mode is debug. If checked, compiler option CHECK(ALL) is added. To specify CHECK option except for ALL, deselect the CHECK option and then select the CHECK option again, and select the other Check items in the CHECK Compiler Option dialog.
Other compiler options	Specifies the compile options that cannot be added from the **Add Compiler Options** dialog box.

Note

- Compiler option specifies file extension

  The following compile options are not displayed in the **Add Compiler Options** dialog box:

  - FORMEXT
  - LIBEXT

  These compiler options can be specified in the environment variables. For details, see the "NetCOBOL User's Guide."

- Compiler option specifies text file encoding of COBOL source files

  The compiler option SCS that specifies the text file encoding of COBOL source files is not displayed in the **Add Compiler Options** dialog. The text file encoding is determined from the file property Text file encoding. When the Text file encoding of the file property is "UTF-8", SCS(UTF8) is set for compiler option SCS. If the Text file encoding is other than "UTF-8", SCS(ASCII) is set.

- The following compiler options cannot be specified for NetCOBOL for Windows(x64) edition

  - AIMLIB compiler option
  - FILELIB compiler option
  - GEN compiler option
  - CHECK(LINKAGE) compiler option

6.1.4 Use the Compiler Option File

The compiler option file has the following information on the compiler option.

- Project manager (for Windows(x86) NetCOBOL)
- WINCOB (for Windows(x86) NetCOBOL)

Windows(x64) NetCOBOL cannot use the compiler option file. Set the content of the compiler option file by the following.

- The **Compiler Options** page of the property dialog box.
- The environment variable

For details of the environment variable, refer to the "NetCOBOL User's Guide".

6.1.5 Setting Library names

To build a source COBOL program that includes the COPY statement specifying library names, use the Library Name option to specify the library names and the library file storage folder.

Adding a library name

To add a library name:

1. Select a COBOL project in the **Dependency view** or **Structure view**.
2. Select **File** > **Properties** from the menu bar, or **Property...** from the context menu. The property dialog box appears.
3. Select **Build** in the left pane. The **Build** page is displayed.
4. Select the "Library Names" tab. The "Library names" page is displayed.

Item	Description
Add	Adds a library option. Click the **Add** button. The "Add Library Name" dialog box appears. Specify a library name in "Library name". Specify the library file storage folder in "Select folder". The folder can also be selected by clicking the **Browse** button.
Change	Changes the selected library folder. Click the **Change** button. The "Modify Library Name" dialog box appears. Change the folder. (Library names cannot be changed in the "Modify Library Name" dialog box.)
Remove	Deletes the selected library option.

Note

- In compilation with the ALPHAL(ALL) or ALPHAL(WORD) option, and a library name described in lowercase letters in the source program, the library name is treated as being coded in uppercase letters. If the library name specified in the source program is coded in lowercase letters, the intended library file cannot be read. To perform compilation with the ALPHAL(ALL) or ALPHAL(WORD) option, specify the library name in uppercase letters as described above.
- When compiling the source program including the COPY statement without specifying the COBOL library name, specify the COBOL library folder using the compiler option LIB.

6.1.6 Setting the file folder option

When you set the file folder option (LIB, FORMLIB, REPIN, REP, etc.), specify the folder using the **Select** dialog box.

In this example, another project folder is specified for the LIB compiler option.

1. Select a COBOL project in the **Dependency view** or **Structure view**.
2. Select **File** > **Property** from the menu bar, or **Property** from the context menu. The Property dialog box appears.
3. Select **Build** in the left pane. The **Build** page is displayed.
4. Select the **Compiler Options** tab, and click the **Add...** button. The **Add Compiler Options** dialog box appears.
5. Select the "LIB" option, and click the **Add...** button. The **LIB Compiler Option** dialog box appears.
6. Click the **Browse**... button to specify the library File folder. The **Folder type selection** dialog box appears.
7. In this example, you specify another project folder in the same workspace. Select "Specify from project", and click the **OK** button. The **Selection from project** dialog box appears.

Table 6.2 Folder type selection dialog box

select item	Description
${NETCOBOL}	Refer to the folder based on the NetCOBOL installation folder.

select item	Description
Specify from project	Refer to the project under the workspace.
Specify absolute path	Refer to the folder.

8. Select the project to which the library file is saved, and click the **OK** button. The path of the library file is displayed in the **LIB Compiler Option** dialog box.

9. Click the **OK** button in the **LIB Compiler Option** dialog box.

10. Click the **Done** button in the **Add Compiler Options** dialog box.

11. Select the **Compiler Options** tab, and click the **OK** button in the **Build** page of Property dialog box. The following message boxes appear.

```
Build options has been changed. To enable the changes, it is necessary to clean the project. Do
you want to clean the project now? [Yes] [No] [Cancel]
```

12. Click the **Yes** button.

6.2 Precompiler

COBOL programs, including input sources for the precompiler, can be developed with the precompiler link function.

6.2.1 Setting and changing initial values of precompiler link information

The initial values used when a precompiler command is invoked can be set and changed. The initial values for setting the precompiler link information are shared by the COBOL projects in a workspace.

To set or change the initial values of the precompiler link information.

1. Select **Window**> **Preferences** from the menu bar. The **Preferences** dialog box appears.

2. Select **COBOL** > **Precompiler** in the left pane of the **Preferences** dialog box, and set or change initial values on the displayed **Precompiler** page.

Item	Description
Use Precompiler	Select this item to use a precompiler for a COBOL project in the current workspace.
Precompiler command	Specifies the name of a command that starts the precompiler.
Precompiler parameters	Specifies precompiler command parameters.
Precompiler source extension	Specifies the extension of an input source file for the precompiler. The following extensions cannot be specified: - cobol - cob - cbl - lcai
Precompiler output source extension	Selects the file extension for the output source file from the precompiler.
Use original source line numbers for error messages.	If this item is checked, COBOL compiler error messages are displayed using the line number in the original source, rather than the line number of the preprocessed source. (The INSDBINF command is invoked.) The default setting is unchecked.
INSDBINF parameters	This item sets parameters of the INSDBINF command, which develops line information for the precompiler input source in COBOL source files generated by precompilation. Note that input and output source file names

Item	Description
	cannot be specified because they are determined from precompiler input source file names.

Note

- When Symfoware Server is used for the pre-compiler, the sqlpcob command must be specified for the pre-compiler command. If the sqlcobol command is specified, a compilation error may be generated.
- Setting the pre-compiler is also required for remote builds. For details, see "9.4.4.2 Precompiler tab "

Method for specifying file names for precompiler command parameters

If file names must be specified for precompiler command parameters, use the macros listed below. File names are automatically generated when the precompiler is invoked.

Macro name	Description of macro
%INFILE%	Name of the source file that is input to the precompiler
%INFILE_BASE%	Name of the source file that is input to the precompiler without an extension.
%OUTFILE%	Name of the source file that is output from the precompiler.

Note

Use the following macro for the file specified for the command parameter in the precompiler.

- Input file of the precompiler: %INFILE%
- Output file of the precompiler: %OUTFILE%

6.2.2 INSDBINF command

If an Oracle link is used, the INSDBINF command solves the following problems in linking between the COBOL compiler and the precompiler:

- The compiler outputs line numbers where compilation errors have been detected, but these line numbers belong to lines in an intermediate file (source file after precompilation). To modify the original source program (COBOL source program with embedded SQL statements before precompilation), the user must reference the intermediate file.
- A line with an error in the original source program cannot be correctly located using **Go To** in the **Problems view**.
- The original source program cannot be debugged.

The INSDBINF command generates an intermediate file that contains embedded line number control information and file name control information. Line number control information (#LINE information) is used to report the line numbers of lines in the original source program, before it is precompiled, to the compiler or precompiler. File name control information (# FILE information) is used to report original source program file names or the names of files included by the precompiler.

An intermediate file generated by the INSDBINF command can be input to the COBOL compiler. This allows referencing the line number control information and file name control information. From that information, a modified original source program, in which line numbers in the original source files correspond to line numbers in precompiled source files, and file names of include files, can be obtained. Thus, problems that occur in linking between the COBOL compiler and precompiler can be resolved.

INSDBINF command parameters

Different INSDBINF command parameters are used for the different operating systems of the servers that perform remote development. For details, see the "NetCOBOL User's Guide".

6.2.3 Creating a COBOL program by using the precompiler

To develop COBOL programs including input sources for the precompiler from creating new project, the "Build Tools" settings can be set automatically if Use precompiler is checked in the New COBOL Project wizard.

To specify the Build Tools for the precompiler to the existing COBOL project:

1. Adding the precompiler to "Build Tools".
2. Setting precompiler link information.
3. Creating and adding precompiler input sources.

6.2.3.1 Build tool settings

To invoke the precompiler during build for a COBOL project, Precompiler must be set for Build Tools.

Adding Precompiler to build tools

If Precompiler is not specified for the Build Tools, follow the procedure below, and select Precompiler.

1. Select the project in the **Dependency view** or **Structure view**.
2. Select **Property...** from the context menu. The property dialog box appears.
3. Select **Build Tools** in the left pane. The **Build Tools** page is displayed.
4. Click the **Add Build Tool...** button. The **Add Build Tool** dialog box appears.
5. Select **Precompiler**, and then click the **OK** button.

Confirm that Precompiler has been added to the "List of build tools associated with COBOL Builder" on the **Build Tools** page, and then click the **OK** button in the property dialog box.

Deleting Precompiler from build tools

To delete Precompiler from a COBOL project:

1. Select the project from the **Dependency view** or **Structure view**.
2. Select **Property...** from the context menu. The property dialog box appears.
3. Select **Build Tools** in the left pane. The **Build Tools** page is displayed.
4. Select "Precompiler "from "List of build tools associated with COBOL builder" on the **Build Tools** page, and click the "Remove" button.
5. Confirm that Precompiler has been deleted from "List of build tools associated with COBOL builder ", and click the **OK** button in the property dialog box.

Note

If a precompiler input source is registered in the **Source Files** folder, Precompiler cannot be deleted from **Build Tools**. Delete the precompiler input source from the **Source Files** folder before deleting Precompiler from **Build Tools**.

6.2.3.2 Setting and changing precompiler link information

Information can be set and changed for the precompiler command used to build COBOL projects. Existing precompiler link settings are used as the defaults.

Setting and changing precompiler link information

Follow the procedure below to set or change precompiler information for a COBOL project.

1. Select the project in the **Dependency view** or **Structure view**.
2. Select **Property...** from the context menu. The property dialog box appears.

3. Select **Build Tools** > **Precompiler** in the left pane. The **Precompiler** page is displayed.

Example

Examples of settings for the Oracle precompiler

```
INAME=%INFILE% ONAME=%OUTFILE%Description
```

For details on the **Precompiler** page, see "6.2.1 Setting and changing initial values of precompiler link information".

Note

If a precompiler input source is registered in the **Source Files** folder, no change can be made in "Precompiler source extension" on the **Precompiler** page. Delete the precompiler input source from the **Source Files** folder before making a change in "Precompiler source extension".

6.2.3.3 Creating and adding a precompiler input source

To add a precompiler input source to a project, precompiler link information must be set in the project in advance. The precompiler input source must be registered in the Source Files folder of the COBOL project. The following two methods can be used to register a precompiler input source in the Source Files folder:

- Create a precompiler input source and registering the new source
- Register an existing precompiler input source

Creating a precompiler input source and registering the new source in the Source Files folder

To create a precompiler input source and register the new source in the **Source Files** folder, use the COBOL Source generation wizard or the Object-Oriented COBOL Source generation wizard. Check the "Use precompiler" check box on the first page of the wizard. Then, the created COBOL source or object-oriented COBOL source is registered as a precompiler input source in the **Source Files** folder.

To start the wizard:

1. In the **Dependency view** or **Structure view**, select the Source Files folder of the COBOL project.
2. Select **File** > **New** > "COBOL Source" or "Object-Oriented COBOL Source" from the menu bar, or select **New** > "COBOL Source" or "Object-Oriented COBOL Source" from the context menu. The wizard is started.

Note

If no precompiler link information is set for the project, the check box for "Use precompiler" is disabled. Set the precompiler link information, and then restart the wizard.

Registering an existing precompiler input source in the Source Files folder

An existing precompiler input source can be registered in the **Source Files** folder of a COBOL project.

- For a precompiler input source inside a project:

  To add a precompiler input source that is inside a project:

  1. Select the **Source Files** folder in the **Dependency view** or **Structure view**.
  2. Select "Add File..." from the context menu. The "Add Source Files" dialog box corresponding to the selected project appears.
  3. Select the precompiler input source to be added to the **Source Files** folder.
  4. Click the **OK** button. The selected precompiler input source is registered in the **Source Files** folder.

- For a precompiler input source outside a project:

  Use either of the following methods to add a precompiler input source that is outside a project:

  - Using Windows Explorer, select the precompiler input source to be registered, and then select "Copy" from the context menu. Select the **Source Files** folder, and then select "Paste" from the context menu.
  - Using Windows Explorer, drag and drop the precompiler input source onto the **Source Files** folder.

Note

If no precompiler link information is registered for the project concerned, an error occurs. The file is registered in the Source Files folder, but it is not registered as a precompiler input source. In this state, even if precompiler link information is set for the project, the file added by the above operation is not automatically registered as a precompiler input source. Move it from the Source Files folder to the Other Files folder then register the file in the Source Files folder again.

6.2.3.4 Editing a precompiler input source

To edit the contents of a precompiler input source with the COBOL editor, the extension of the precompiler input source file must be associated with the Content Types and COBOL editor. The association is invoked automatically when the Precompiler setting is set. If the precompiler input file is not opened by the COBOL editor, follow the procedure below and associate the file extension with the Content Types and COBOL editor.

1. Select **Window**> **Preferences** from the menu bar. The **Preferences** dialog box appears.
2. In the left pane of the **Preferences** dialog box, select **General** > **Content Types**. This displays the **Content Types** page.
3. In the **Content Types** page, select **Text** > "COBOL Source File".
4. Click the **Add** button, The **Add Content Type Association** dialog box displays.
5. In the **Add Content Type Association** dialog box, enter the extension of the precompiler input source file at "Content type" and click the **OK** button.
6. In the **Content Types** page, check that the extension of the precompiler input source file is displayed at **File Associations**, and click the **OK** button.
7. Select **General** > **Editors** > **File Associations** in the left pane of the **Preferences** dialog box. The **File Associations** page is displayed.
8. Click the **Add...** button to the right of File types on the **File Associations** page. The **Add File Type** dialog box appears.
9. Enter the extension of the precompiler input source file in "File type″in the **Add File Type** dialog box, and click the **OK** button. The specified extension is added to "File types″on the **File Associations** page.
10. Select the added extension from "File types" on the **File Associations** page, and click the **Add...** button to the right of "Associated editors". The **Editor Selection** dialog box appears.
11. Confirm that Internal Editors is selected in the **Editor Selection** dialog box, and select "COBOL Editor".
12. "COBOL Editor" is added to "Associated editors″on the **File Associations** page.

6.3 Linker

Executable files or dynamic link libraries can be created by linking object files or libraries.

6.3.1 Setting link options

To display the Linker Options page and set link options:

1. Select a COBOL project in the **Dependency view** or **Structure view**.
2. Select **File** > **Properties** from the menu bar or **Property...** from the context menu. The property dialog box appears.
3. Select **Build** in the left pane. The **Build** page is displayed.

4. Select the **Linker Options** tab. The **Linker Options** page is displayed.

Item	Description
Linker options	Displays a list of the set libraries and object files.
Add	Adds libraries and object files to be linked. Click the **Add** button to display the Add Linker Option dialog box so that libraries and object files can be selected. Click the **Browse** button to display the Folder type selection dialog box so that files can be selected. One or more libraries and object files can be added.
Change	Changes the selected library or object file.
Remove	Deletes the selected library or object file.
Remove All	Deletes all the libraries and object files displayed in *Linker options*.
Up	Changes the linking order of the selected library file.
Down	Changes the linking order of the selected library file.
Use C function	Used to select whether the TARGET type is a dynamic link library and whether the TARGET uses the C functions. This item defaults to unchecked.
C Run-time Library Name	Specifies the filename of the C run-time library to be linked during linkage. If the C run-time library name is omitted, "LIBCMT.lib" is linked. In the following cases, specify "MSVCRT.lib." - When two or more DLLs execute in one process. - The called C program is created with Microsoft Visual C++ and compiled with the /MD option. For details of the C run-time libraries linkage, refer to "NetCOBOL User's Guide."
DLL Entry Object	Specifies whether only object files created in COBOL are used to create the dynamic link library, or whether object files created in other languages are also used.
For COBOL program only	Creates a dynamic link library using only object files created in COBOL.
For linking with non-COBOL Program	Creates a dynamic link library using object files created in other languages as well as COBOL.
Output debugging information	Used to select whether to output debug information to the program data base file(PDB). This item defaults to check.
Other options	Used to enter additional link options. To enter multiple options, insert space characters as delimiters.

Note

When specifying the project from Selection from the project dialog in selecting Link options, the path separator backslash is displayed as a forward slash. The forward slash is treated as a backslash.

The following table lists link options that can specified for Other options. For details on the link options, see the NetCOBOL User's Guide.

Specification format	Description
/EXPORT:*external-reference-name*	Creates external reference information.
/OUT:*filename*	Specifies the name of the main output file.

Specification format	Description
/STACK:*stack-size*	Specifies a change in the stack size. The default is 1 MB. Specify a stack size in units of bytes.
/MAP:*filename*	Specified this when creating a map file.

6.3.2 Setting TARGET options

The following TARGET options can be set for each project:

- TARGET name
- TARGET type
- Build mode

To display the Target page and specify TARGET options:

1. Select a COBOL project in the **Dependency view** or **Structure view**.
2. Select **File** > **Properties** from the menu bar, or **Property...** from the context menu. The property dialog box appears.
3. Select **Target** in the left pane. The **Target** page is displayed.

Item	Description
Target name	Specifies a TARGET name. The current project name is set as the initial value.
Target type	Specifies the type of TARGET to be output. "Executable" or "Dynamic-link library" can be selected. The default is "Executable".
Use the DLL specific runtime initialization file (COBOL85.CBR)	For a target type of Dynamic-link library, if this option is selected, an initialization file for DLL-specific execution is used.
Application format	Specifies the format of the created application.
COBOL console window	Specifies that a console window created by a COBOL program is the input-output destination of ACCEPT and DISPLAY statements, and that a message box is the output destination of execution-time error messages.
System console window	Specifies that the system console (command prompt window) is the input-output destination of ACCEPT and DISPLAY statements and execution-time error messages.
Build mode	Specifies "Release" or "Debug" as the build mode. The default is "Debug".

Note

When you specify Debug for a build mode, the "CHECK(ALL)" compiler option is added.

6.4 Resource compiler

The resource compiler is a build tool used to add version information to a program or set icons for a program. This build tool is valid when resource definitions have been described to a resource definition file (.rc file) and this resource definition file has been added to the Source Files folder. For details on the specification format and description method of resource definition files, see the Microsoft(R) Resource Compiler manual.

6.4.1 Build Tool Settings

To invoke the resource compiler during a build for a COBOL project, the Resource compiler must be set for Build Tools. When creating a new COBOL project, the Resource compiler is not set for Build tools.

Adding the Resource compiler to the build tools

To add the Resource compiler to a COBOL project:

1. Select the project in the **Dependency view** or **Structure view**.
2. Select **Property...** from the context menu. The property dialog box appears.
3. Select **Build Tools** in the left pane. The **Build Tools** page is displayed.
4. Click the **Add Build Tool...** button. The **Add Build Tool** dialog box appears.
5. Select **Resource compiler**, and then click the **OK** button.
6. Confirm that **Resource compiler** has been added to the "List of build tools associated with COBOL builder" on the **Build Tools** page, and click the **OK** button in the property dialog box.

Deleting the Resource compiler from the build tools

To delete the Resource compiler from a COBOL project:

1. Select the project in the **Dependency view** or **Structure view**.
2. Select **Property...** from the context menu. The property dialog box appears.
3. Select **Build Tools** in the left pane. The **Build Tools** page is displayed.
4. Select **Resource compiler** from the "List of build tools associated with COBOL builder" on the **Build Tools** page, and then click the "Remove" button.
5. Confirm that the **Resource compiler** has been deleted from the "List of build tools associated with COBOL builder", and click the **OK** button in the property dialog box.

6.4.2 Setting the Resource compiler options

To display the Resource compiler page, specify the Resource compiler options as follows:

1. Select the COBOL project from **Dependency view** or **Structure view**.
2. Select **File** > **Properties** from the menu bar, or select **Property...** from the context menu. The properties dialog box appears.
3. Select **Build Tools** > **Resource compiler** in the left pane, the **Resource compiler** page is displayed.

6.5 Building a COBOL Program

Several methods can be used to build a COBOL program. The range of build targets can be one or more selected projects, or an entire workspace.

6.5.1 Manual build

If **Project** > **Build Automatically** on the menu bar is checked, build is automatically executed when a resource is saved. If the developer needs to control execution of build, the checkmark can be cleared from **Build Automatically**, and build can be executed manually.

Build Project

To execute a build for one or more selected projects, select the projects in the **Dependency view** or **Structure view**, and then select **Project** > **Build Project** from the menu bar or **Build Project** from the context menu.

A Build is executed only for the resources that have been updated since the last build.

Build All

To build all of the projects in a workspace, select **Project** > **Build All** from the menu bar.

A Build is executed only for the resources that have been updated since the last build.

Rebuild Project

To execute a build for all project resources including those not updated since the last build, select each target project for the build in the **Dependency view** or **Structure view**, and then select either **Project** > **Clean...** from the menu bar or "Rebuild Project" from the context menu.

Build Working set

The working set is a grouping of the project and the resource. Create the working set, and divide the build targets into groups. A Build is executed only for the projects that have been updated since the last build.

Creating Working set

1. Select **Project** > **Build Working Set** > **Select Working Set** from the menu bar. The **Select Working Set** dialog box appears.
2. Click the **New...** button. The **New Working Set** wizard appears.
3. Select the "Resource" in the "Working set type", and click the **Next** button.
4. Input an arbitrary name to the "Working set name", and select the project of working set target. Click the **Finish** button.

Build

1. Select **Project** > **Build Working Set** > **Select Working Set** from the menu bar. The **Select Working Set** dialog box appears.
2. Select the working set, and click the **OK** button. The build of the project selected in the working set is executed.

After the build, the selected working set name is added to the submenu of **Project** > **Build Working Set**. A build of the working set can be executed by the selection of this menu.

Note

When the file not included in the project is referred by the compiler option, even if the reference files that are not included in the project are changed, changing the resource for the project will not be recognized. For this reason, even if **Build Project** is executed or **Build Automatically** is checked, the target file is not updated.

When the files that are not included in the project are changed, execute the "Rebuild project" for the project that is using the files.

Point

Follow the procedure below to automatically save modified resources before manual build execution.

1. Select **Window**> **Preferences** from the menu bar. The **Preferences** dialog box appears.
2. Select **General** > **Workspace** in the left pane. The **Workspace** page is displayed.
3. Select "Save automatically before build" on the **Workspace** page.
4. Click the **OK** button to close the **Preferences** page.

6.5.2 Automatic build

If **Project** > **Build Automatically** on the menu bar is checked, the build is automatically executed when the following things occur.

- The project has been imported.
- The resource was saved.
- The build option was changed.

6.5.3 Analyzing dependency of repository file

To build a project including object-oriented COBOL source files, the dependency between the COBOL source files must be set before the build. By setting that dependency, the COBOL source files are built in the correct sequence. If the dependency is not set correctly, a compile error may occur during the build. When a project is created or COBOL source files are added to the Source Files folder, the added source files are analyzed and the dependency is set automatically. However, when the REPOSITORY paragraph of a COBOL source program has been edited, the dependency must be analyzed manually before a build is executed.

To analyze the dependency, select a project in the **Dependency view** of NetCOBOL Studio, and then select **Analyze Dependency** > **All** from the context menu.

Note

The message "abc.rep: dose not exist" is displayed in the **Console view** when the dependency is analyzed. Repository files (*.rep) for the object-oriented COBOL source files in a project are created only after the source files are compiled. Therefore, if the dependency is analyzed before the build, that message is displayed. When the dependency analysis itself has been correctly performed, the project can be built without any special measure taken.

6.6 Correcting Compilation Errors

Compilation errors during build are displayed in the **Problems view**. To edit a COBOL source file containing a compilation error, double-click the information on the compilation error in the **Problems view**, or select **Go To** from the context menu. The COBOL source file is opened with the editor, and the current line is the one that contains the compilation error. If the file is already open in the COBOL editor, the line containing the compilation error becomes the current line.

Chapter 7 Debugging Function

This chapter describes the debugger, which is used to detect logic errors in programs. The debugger can be used to set breakpoints to halt execution of a program and to verify execution of a program by confirming data item values. The debugger supports debugging of multithreaded programs; however it does not support multiple debug sessions.

7.1 Debugging an Application

The program to be debugged is executed at the same time that the debugger is started. The startup configuration of the application is used at debugger startup.

7.1.1 Creating the debugging information file

In the debugging of the COBOL program, the debugging information file is prepared. When the build is executed by the following settings, the debugging information file is created.

1. Display the property dialog box of the COBOL project.
2. Select the **Target** in the left pane. The **Target** page is displayed.
3. Check the **Build mode** of **Target** page. When **Debug** has been selected in the **Build mode**, the debugging information file is created at build.

The setting of the build mode is effective only for the specified project.

Point

The COBOL compiler creates the SVD File of each COBOL source file. At debugging, store the SVD file in the same folder as the executable file. In remote debugging, when a local machine is selected as a server, you copy the debugging information File with the execution file.

7.1.2 The startup configuration

The startup of the debugger uses the startup composition. NetCOBOL Studio supports the local debug and the remote debug.

- Local debug
  Use the COBOL application startup configuration.

  For details, refer to "7.1.3 Starting debugging"
- Remote debug

  Use the Remote COBOL application startup configuration.
  - Normal Debug: "9.6.1.2 Starting the remote debugger"
  - Attach Debug: "9.6.2.1 Starting the remote debugger".

Note

When debugging it by the same startup composition, select **Run** > **Debug History** from the menu bar. Alternatively, click ▾ in 🐞▾ on the toolbar, and select the Name. The Debugger is started.

7.1.3 Starting debugging

This section explains the procedure for debugging a program.

1. Select a COBOL project in the **Dependency view** or **Structure view**.

2. Select **Run** > **Debug Configurations...** from the menu bar. Alternatively, click ▾ in 🐞▾ on the toolbar, and select **Debug Configurations...**. The **Debug Configurations** dialog box appears.

3. Double-click **COBOL Application** in the left pane. The startup configuration setting page displays in the right pane.

4. The default startup configuration name is displayed in the *Name* field. The startup configuration name can be changed.

5. Check the contents of the **Main** tab, and add or modify the contents as necessary.

   a. ″Project name" is the COBOL project name selected in step 1.

   b. "Executable Fil*e*″ is the program to be executed first at the start of debugging. If the specified TARGET of the project is an executable file, it cannot be changed. If the specified TARGET is a dynamic link library, specify the relevant file.

   c. Specify the current folder at execution time in the **Working folder** field. If the dynamic link libraries required for execution are stored in this folder, they can be loaded during execution, even if other settings such as path settings have not been made.

   d. For "Program arguments″, enter the arguments specified for Executable File in the format used for the command line interface.

6. Select Debug to start debugging.

Point

Debugging can be started with the default settings by selecting a project in the **Dependency view** or **Structure view** and then selecting **Run** > **Debug As** > **COBOL Application** from the menu bar.

Note

Other debuggers (e.g. COBOL, Java) cannot be started while debugging a COBOL program.

The same workspace folder cannot be used from more than one NetCOBOL Studio.

7.1.3.1 Setting items of the COBOL application startup configuration

This section explains setting items of the COBOL application startup configuration.

Main tab

item	Description
Project name	The select project name is displayed.
Executable File	This file is executed first at the debugging. When the target is an execution file, cannot be changed. When the target is a dynamic link library, specify an appropriate file.
Working folder	Specify the current folder at execution time. The dynamic library stored in this folder is loaded without other path settings.
Program arguments	The argument of the program is input.

Source tab

This tab specifies search paths for COBOL source files and COBOL library files. Searches for sources are performed in the order displayed in "Source lookup path". Projects and folders can be added to the search paths. The order of the added search paths can be changed using "Up" or "Down".

Note

If an external folder is specified in a search path, the following functions do not work correctly for the COBOL source programs in the external folder:

- Breakpoint operations

- Run to Line of the context menu of the COBOL editor

To use either of the above functions, Fujitsu recommends specifying the corresponding project in the source search paths. When debugging a COBOL source file in an external folder, debug each line using **Step Into** or **Step Over**.

Environment tab

This tab sets information used by a COBOL runtime module. Before the start of its processing, the COBOL runtime module uses this information to obtain environment-specific information.

To add the environment variable information set here to existing system environment variables or user environment variables, select **Append environment to native environment**. To replace the existing environment, select "Replace native environment with specified environment".

Note

If the same setting in an environment variable is made both on the **Environment** tab and in the runtime initialization file, the information set in the runtime initialization file is valid. For details on runtime environment information, see the "NetCOBOL User's Guide".

Common tab

This tab specifies how to save the startup configuration, which perspective to open after the start of execution or debugging, and other settings. For details, see the description of the **Common** tab in "8.2 Executing a COBOL Program".

7.1.4 Debug perspective

A COBOL program is debugged in the Debug perspective. The Debug perspective consists of views that are suitable for debugging, and facilitate debugging operations. For details on the debug perspective, see "7.2 Debug Perspective". Debugging can also be performed in the COBOL perspective.

7.1.5 Outline of the debugging functions

The debugger has the following functions:

- Setting breakpoints
- Unconditional execution
- Single-step execution
- Execution to the specified line
- Referencing a data item
- Changing the value of a data item
- Monitoring data items

For details on each debugging function, see "7.3 Debugger Functions."

7.1.6 Exiting debugging

Use any of the following methods to exit debugging:

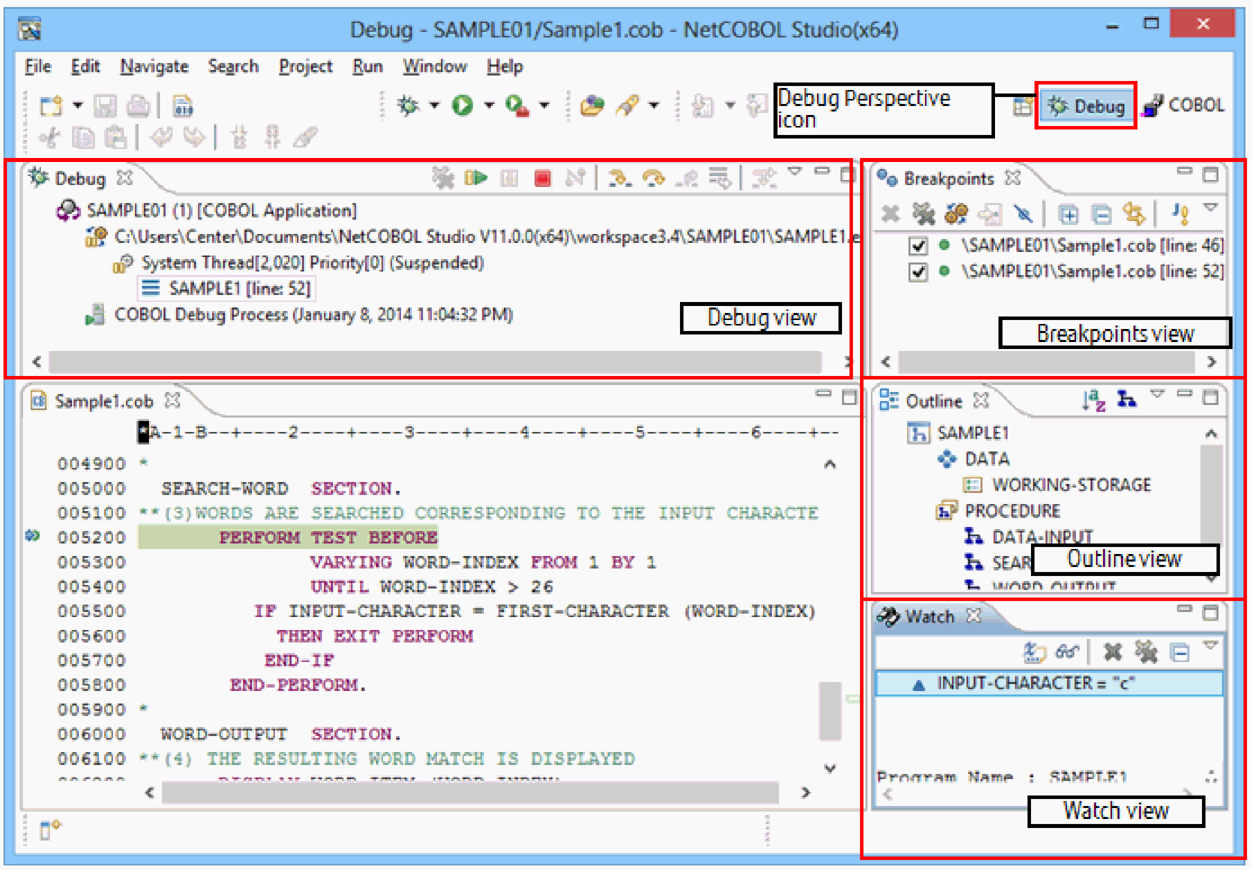
- Click the ■ toolbar button in the **Debug view**.
- Select **Terminate** from the context menu of the **Debug view**.
- Select **Run** > **Terminate** from the menu bar.

7.1.7 Notes

To invoke a dynamic link library of another project linked through the dynamic link structure from the project to be debugged, the storage location of the dynamic link library must be set using either of the following methods:

- Specify the project of the dynamic link library for the "Source lookup path″on the **Source** tab of Debug.
- Specify the dynamic link library by adding a PATH variable to "Environment variables to set″and specify the folder containing the dynamic link library in the **Environment** tab of Debug.

7.2 Debug Perspective



```
SAMPLE01 (1) [COBOL Application]
    C:\Users\Center\Documents\NetCOBOL Studio V11.0.0(x64)\workspace3.4\SAMPLE01\SAMPLE1.e
        System Thread[2,020] Priority[0] (Suspended)
            SAMPLE1 [line: 52]
    COBOL Debug Process (January 8, 2014 11:04:32 PM)
```

☑ ● \SAMPLE01\Sample1.cob [line: 46]

☑ ● \SAMPLE01\Sample1.cob [line: 52]

```
       A-1-B--+----2----+----3----+----4----+----5----+----6----+--
004900 *
005000  SEARCH-WORD  SECTION.
005100 **(3)WORDS ARE SEARCHED CORRESPONDING TO THE INPUT CHARACTE
005200        PERFORM TEST BEFORE
005300                VARYING WORD-INDEX FROM 1 BY 1
005400                UNTIL WORD-INDEX > 26
005500           IF INPUT-CHARACTER = FIRST-CHARACTER (WORD-INDEX)
005600             THEN EXIT PERFORM
005700            END-IF
005800         END-PERFORM.
005900 *
006000  WORD-OUTPUT  SECTION.
006100 **(4) THE RESULTING WORD MATCH IS DISPLAYED
```

```
SAMPLE1
  DATA
    WORKING-STORAGE
  PROCEDURE
    DATA-INPUT
    SEAR
```

▲ INPUT-CHARACTER = "c"

Program Name : SAMPLE1

The Debug perspective consists of the COBOL editor and the following views that facilitate debugging operations:

- **Debug view**
- **Breakpoints view**
- **Watch view**
- **Outline view**

Note

In NetCOBOL Studio, the **Watch view** is used to reference and set data items. The **Variables view** cannot be used in NetCOBOL Studio.

7.2.1 Debug view

The **Debug view** displays a project name, the name of the program being executed, and other information in a tree structure, enabling the user to check the program execution status (status of threads and stacks), calling paths, and other information. Buttons and context menus can be used to perform debugging operations such as starting, executing, or exiting a program.

7.2.1.1 Context menu of the Debug view

The following table lists the context menu items used in the **Debug view**.

Menu item	Button	Description
Step Into	⤵	Executes one statement in a program. If the executed statement invokes another process such as a CALL statement, control jumps to the called process, and execution is halted. When Step Into is executed just before a COPY statement, execution is halted at the first statement in the COBOL Library.
Step Over	↷	Executes one statement in a program. If the executed statement invokes another process such as a CALL statement, the entire called process is executed, and execution is halted at the next statement following the calling statement. When Step Over is executed just before a COPY statement, execution is halted at the first statement in the COBOL Library.
Step Return	↱	Executes a program until control returns to the calling process. When Step Return is executed in a COBOL library, the program is executed until returning to the call process of the program in which the COBOL library is specified.
Resume	▶	Restarts the halted execution of a program.
Suspend	⏸	Halts execution of a program.
Terminate	■	Exits debugging.
Terminate and Relaunch	-	Exits the current debugging and starts debugging of the same target again.

7.2.2 Breakpoints view

The **Breakpoints view** displays all of the breakpoints set for a project. When a breakpoint is double-clicked, the editor displays the breakpoint locations. Breakpoints can be enabled, disabled, and deleted in this view.

7.2.2.1 Context menu of the Breakpoints view

The following table lists the context menu items used in the **Breakpoints view**.

Menu item	Button	Description
Go to File	⇨	Displays in the COBOL editor the location at which the selected breakpoint is set.
Hit Count...	-	Displays the **Set Breakpoint Hit Count** dialog box.
Enable	-	Enables a disabled breakpoint.
Disable	-	Disables an enabled breakpoint.
Remove	✖	Deletes the selected breakpoint.
Remove ALL	✖	Deletes all the set breakpoints.

7.2.2.2 Breakpoint properties

A breakpoint can be enabled or disabled by using these properties. The properties can also be used to set, enable, and disable the hit count of a breakpoint.

1. Set a breakpoint using the COBOL editor.
2. Select the breakpoint from the **Breakpoints view**, and then select **Properties...** from the context menu.
3. Enable or disable the breakpoint in the **COBOL Line Breakpoint Properties** dialog box. This dialog box is also used to set the hit count of a breakpoint, and enable or disable the hit count of the breakpoint.

7.2.3 Watch view

The **Watch view** is used to perform debugging operations for data items. This view enables the user to add and delete data items, and change their values.

7.2.3.1 Context menus of the Watch view

The following table lists the context menu items used in the **Watch view**.

Menu item	Button	Description
Add Data Item...	👓	Displays the **Add Data Item** dialog box used to add data items to the **Watch view**.
Change Value	-	Displays the **Set Value** dialog box used to change the values of the selected data items.
Change Hex Value	-	Displays the **Set Value** dialog box used to change the values of the selected data items, where the values are specified in hexadecimal format.
Suspend by Value Change	-	Specifies whether to halt execution of the program when the values of the selected data items are changed.
Remove	✖	Deletes the selected data item from the **Watch view**.
Remove All	✖	Deletes all the data items from the **Watch view**.
Show Data Type	-	Specifies whether to display data types in the **Watch view**.

7.2.3.2 Adding data items to the Watch view

Data items can be added to the **Watch view** by selecting Add Data Item from the context menu or toolbar.

1. Select **Add Data Item** from the context menu or toolbar. The **Add Data Item** dialog box appears.
2. Specify information such as data item names, a program name, and whether to halt execution when values are changed, and add the data items to the **Watch view**.
3. To add data items defined in methods of a class, specify "*class-name*:*method-name*" as the program name. A data item can be added to the **Watch view** by double-clicking the data item in the COBOL editor and then selecting **Add to Watch View** from the context menu.

Table 7.1 Add Data Item dialog box

item	Description
Data name	When adding data items for monitoring to the **Watch view**, specify identifier name as data name. One of the following is the identifier name. - Identifier - Condition-name - Index Name - Special Register

item	Description
	- Symbolic-constant - Named Literal - Function For details, refer to "D.1 Identifier Name".
Program name	Specify the program name. For details, "D.2 Program name".
Suspend by Value Change	If this item is checked, when the value of the data changes, the program execution is stopped.

Note

Data items can be added to the **Watch view** during debugging. First, start debugging of the program. Then, when execution of the program is halted, such as at a breakpoint, add data items to the **Watch view**.

7.2.3.3 Display formats of the values in the Watch view

Hexadecimal format or ASCII character format can be used as the data display format of the **Watch view**, or both formats can be used for data display as an added format.

To add or change a display format:

1. Select **Window**> **Preferences** from the menu bar. The **Preferences** dialog box appears.

2. Select **COBOL** > **Debug** in the left pane.

3. Select a display format from **Watch view** display options in the right pane, and then click the **OK** or **Apply** button.

7.2.4 Outline view

The **Outline view** displays a structural outline of the COBOL source file that is currently the active one in the COBOL editor. For details, see "Outline View."

7.3 Debugger Functions

This section describes the main debugger functions.

7.3.1 Breakpoints

A breakpoint is represented by a mark indicating a specified point in a program at which the debugger can be stopped. This section shows how to add and delete breakpoints, as well as how to use breakpoints.

7.3.1.1 Adding a breakpoint

To add a breakpoint:

1. Use the COBOL editor to open the file to which to add a breakpoint.

2. Immediately to the left of the line to which to add the breakpoint, select **Add Breakpoint** from the context menu of the vertical ruler.

3. The breakpoint mark is displayed on the vertical ruler. The breakpoint is displayed in the list of the **Breakpoints view**.

Point

A breakpoint can also be set by double-clicking on the vertical ruler.

Note

If a breakpoint is set in a COBOL library, and that library is used by multiple COPY statements, the breakpoint will be hit in each of those cases and the program will be interrupted.

7.3.1.2 Deleting a breakpoint

To delete a breakpoint:

1. Select the breakpoint to be deleted, from the **Breakpoints view**.
2. Select **Remove Breakpoint** from the context menu.

Point

A breakpoint can also be deleted by double-clicking on it in the marker bar.

7.3.1.3 Using breakpoints

When control reaches a breakpoint during execution of a program, the debugger halts execution. The program calling path and data item values can be referenced at this point in time. The breakpoint can be enabled or disabled, and its hit count can be specified using breakpoint properties. The breakpoint properties can be displayed using the context menu of the vertical ruler or the context menu of the **Breakpoints view**.

Once a breakpoint is set, it is maintained across multiple sessions until it is explicitly deleted. Breakpoints are saved even when their file, which is open in the COBOL editor, is closed. The set breakpoints are displayed the next time the file is opened.

7.3.1.4 Hit count of a breakpoint

Execution of a program can be halted by using the hit count of a breakpoint. Execution is halted when the line at which the breakpoint is set has been executed the number of times specified for the hit count.

1. Set a breakpoint using the COBOL editor.
2. Select the breakpoint from the **Breakpoints view**, and then select **Hit Count...** from the context menu.
3. Set the hit count of the breakpoint, and enable or disable the hit count of the breakpoint in the **Set Breakpoint Hit Count** dialog box.

Note

The hit count of the breakpoint in the COBOL library is the total execution count of the breakpoint, even though the COBOL library may be used by several COPY statements.

7.3.2 Execution

The debugger can execute a series of statements up to the next breakpoint or the specified statement all at once, or it can execute it statement by statement for checking of the behavior of each statement in the execution path. However, if a breakpoint is detected during execution of the program, execution is halted at this breakpoint even when it has not reached any specified statement.

Note

If the path of the executable program to be debugged is not displayed for the selected tree element in the **Debug view**, the execution operation of the debugger is disabled. If the correct tree element has not been selected, select a tree element for which the path of an executable program is displayed, and run the debugger.

7.3.2.1 Unconditional execution

In unconditional execution, a program is executed without any halt statement specified. The program is executed until either the next breakpoint is reached or the program exits.

Use any of the following methods to perform unconditional execution:

- Select the toolbar button of the **Debug view**.
- Select **Resume** from the context menu of the **Debug view**.
- Select **Run** > **Resume** from the menu bar.
- Press the **F8** shortcut key.

7.3.2.2 Step Into

Step-into executes only one statement in a program. If the executed statement invokes another process such as a CALL statement, control jumps to the called process, and execution is halted.

Use any of the following methods to perform Step Into:

- Select the toolbar button of the **Debug view**.
- Select **Step Into** from the context menu of the **Debug view**.
- Select **Run** > **Step Into** from the menu bar.
- Press the **F5** shortcut key.

Note

When Step Into is executed just before a COPY statement, execution is halted at the first statement in the COBOL Library.

7.3.2.3 Step Over

Step-over executes only one statement in a program. If the executed statement invokes another process such as a CALL statement, execution is not halted in the called process. Instead, all of the called process is executed, and execution is halted at the next statement following the calling statement.

Use any of the following methods to perform Step Over:

- Select the toolbar button of the **Debug view**.
- Select **Step Over** from the context menu of the **Debug view**.
- Select **Run** > **Step Over** from the menu bar.
- Press the **F6** shortcut key.

Note

When Step Over is executed just before a COPY statement, execution is halted at the first statement in the COBOL Library.

7.3.2.4 Execution until control returns to the calling process

When execution has been halted within a subprogram or method, this function performs execution until control returns to the calling process.

Use any of the following methods to perform execution until control returns to the calling process:

- Select the toolbar button of the **Debug view**.

- Select **Step Return** from the context menu of the **Debug view**.
- Select **Run** > **Step Return** from the menu bar.
- Press the **F7** shortcut key.

Note

When Step Return is executed in a COBOL library, the program is executed until returning to the call process of the program in which the COBOL library is specified.

7.3.2.5 Execution until control reaches the specified statement

The program is executed from the current halt statement to the statement at which the cursor is positioned in the COBOL editor.

To execute the program to the specified statement:

- Select **Run to Line** from the context menu of the COBOL editor.

Note

Run to Line cannot be executed from a COBOL library. In order to execute to the specified statement, use one of the operations below.

- Set the breakpoint to the target line and use Unconditional execution.
- Repeat Step Into or Step Over until the control reaches the target line.

7.3.3 Debugging functions for data items

Data item values can be checked and changed with the debugger. This section explains debugging functions for data items.

Note

When a REPLACE statement or a COPY statement with a REPLACING phrase, DISJOINING phrase, or JOINING phrase is described in the source program, the source program before string replacement is displayed. In order to reference, change, or monitor the value of the data item to be replaced, specify the data item name after replacing.

7.3.3.1 Referencing the value of a data item

This section explains methods that can be used to reference a data item value.

Displaying the value in a tooltip

When the mouse cursor is positioned on a data item displayed in the COBOL editor, a tooltip appears with the value of the data item.

Using the Watch view

The value of a data item and more detailed information on it can be referenced by adding the data item to the **Watch view**. For details on the **Watch view**, see "7.2.3 Watch view."

7.3.3.2 Changing the value of a data item

This section explains the methods that can be used to change the value of a data item added to the **Watch view**.

Changing a data item in automatic format

To change the value of a data item in automatic format:

1. In the **Watch view**, select the data item whose value is to be changed.

2. Select **Change Value** from the context menu.

3. Enter the replacement value for the current value in the **Set Value** dialog box, and click the **OK** button.

Changing a data item in hexadecimal format

Follow the procedure below to change the value of a data item in hexadecimal format.

1. In the **Watch view**, select the data item whose value is to be changed.

2. Select **Change Hex Value** from the context menu.

3. Enter the replacement value for the current value in the **Set Value** dialog box, and click the **OK** button.

7.3.3.3 Monitoring changes in the values of data items

Data items can be monitored so that execution of the program can be halted when the value of a data item is changed.

Use either of the following methods to specify whether to halt execution of the program when such a value is changed:

- During registration in the **Watch view**:

  Check the **Suspend by Value Change** checkbox in the **Add Data Item** dialog box.

- For a data item registered in the **Watch view**:

  Select **Suspend by Value Change** from the context menu of the **Watch view**.

7.3.3.4 Notes

- The following alphanumeric data items are considered as Int type binary integer data items.

int type binary integer data item	Treatment in Debug	
BINARY-CHAR	PIC X(1)	Alphanumeric data item for 1 byte
BINARY-SHORT	PIC X(2)	Alphanumeric data item for 2 bytes
BINARY-LONG	PIC X(4)	Alphanumeric data item for 4 bytes
BINARY-DOUBLE	PIC X(8)	Alphanumeric data item for 8 bytes

  In addition, set the hexadecimal format if you want to set the value in int type binary integer data item.

- When using the debug function to handle the data (value reference, value change, and interruption in changing value), if the area length is specified in an extremely large data item, the debugging operation may be very slow. Although it is not possible to say unconditionally that the limit of the data length depends on the ability and environment of the debugging machine, pay attention to the data size in the debugging operation.
  When considering the slow operation during debugging due to using a large area length data item, specify the reference modification and limit the scope of the data area that is taken as the handle target.

7.3.4 Adding data items to the Watch View from the COBOL editor

1. Data items can be added to the **Watch view** from the COBOL editor as the following steps when execution of the program is halted, such as at a breakpoint.

2. Drag the data item to be added to the **Watch view** in the COBOL editor.

3. Select **Add to Watch View** from the context menu.

Note

Data items of the COBOL library files in the COBOL editor cannot be added to the **Watch view** from the context menu. Use "Add Data Item" in the **Watch view**. For details, see "7.2.3.2 Adding data items to the Watch view".

7.3.5 Changing the beginning of the execution

The point where a program is stopped can be changed to the point where the cursor is set as the following steps. And then, the point after changed gets to be the beginning of the execution.

1. Move the cursor to the line to be changed in the COBOL editor.
2. Select **Run from Line** from the context menu.

Chapter 8 Execution Function

A new program can be executed after it has been built. This chapter describes the procedure for executing a COBOL program.

8.1 Runtime Environment Information

The information necessary for executing a COBOL program is called runtime environment information. The two types of runtime environment information are environment variable information and entry information. The COBOL runtime uses environment variable information to obtain environment-specific information at the start of processing. Entry information is used to specify information on the dynamic program structure.

In NetCOBOL Studio, the Runtime Environment Setup Tool is used to set runtime environment information. The Run-time Environment Setup Tool can be started by double-clicking the runtime initialization file (COBOL85.CBR) that is generated when a project is created in the "Dependency" or "Structure" view. Environment variable information can be specified from the "Run Configurations" dialog box.

For details on runtime environment information, see the NetCOBOL User's Guide.

Note

In NetCOBOL that operates on Windows(x64), neither the Run-time Environment Setup Tool nor the files are related by initialization file for execution (COBOL85.CBR). Please select the "Apps" > "Fujitsu NetCOBOL V11(x64)" > "Run-time Environment Setup Tool" from the start menu. The Run-time Environment Setup Tool is started. Select "File" > "Open" from the Run-time Setup Tool menu bar, and select the initialization file in the "Select Rum-time Initialization file" dialog box.

8.2 Executing a COBOL Program

This section explains the procedure for executing a COBOL program.

Executing a COBOL program

To execute a COBOL program:

1. Select the COBOL project in the "Dependency" or "Structure" view.
2. Select "Run" > "Run Configurations..." from the menu bar. The "Run Configurations" dialog box appears.
3. Double-click "COBOL Application" in the left pane. The startup configuration setting page displays in the right pane.
4. The project name is the initial value displayed in the "Name" field. The startup configuration name can be changed.
5. Click the "Main" tab.
   - Enter the COBOL project name in the "Project name" field. Alternatively, click the "Browse..." button and select the COBOL project.
   - If the TARGET of the specified project is a dynamic link library, specify the "Executable File".
   - Specify the current folder at the execution time in the "Working folder" field. If the dynamic link libraries required for execution are stored in this folder, they can be loaded during execution time.
   - For "Program arguments", enter parameters in the format used for the command line interface.
6. Click the "Run" button to execute the program.

Once the COBOL program has been executed using the steps detailed above, it can also be executed using the following steps:

1. From the menu bar, select "Run" > "Run history".
2. Select the startup configuration name that was specified in "Name" field in Procedure 4 above.

If the COBOL program is executed using any method other than those described here, the values set in the Run dialog box do not take effect.

Note

For example, a user may execute a COBOL program by double-clicking an executable file from a view. In this case, the work directory settings specified in the Run dialog box are not reflected. Problems may occur, if for example, the relative path name specified in the runtime initialization file (COBOL85.CBR) is not the intended path.

Setting the COBOL application startup configuration

Setting items of the COBOL application startup configuration.

Main tab

item	Description
Project name	The select project name is displayed.
Executable File	It is a program executed first at the debugging. When the target is an execution file, cannot be changed. When the target is a dynamic link library, specify an appropriate File.
Working folder	Specify the current folder at execution time. When you store the dynamic link library in this folder, the load of dynamic link library can be done at execution time even if other paths are not set.
Program arguments	The argument of the program is input.

Source tab

item	Description
Source lookup path	This page contains the paths used to search source files in debugging. The information in the "Source" tab is not used at execution time

Environment tab

item		Description
Environment variables to set		This item contains information that is used by the COBOL runtime module to obtain environment-specific information at the start of processing. For details on environment-specific information, see the "NetCOBOL User's Guide".
	Variable	
	Value	
Append environment to native environment		To add the environment variable information set here to existing system environment variables or user environment variables.
Replace native environment with specified environment		To replace the existing environment.

Note

If the same environment variable is set in both the "Environment" tab and the runtime initialization file, the information in the runtime initialization file is used.

Common tab

The "Common" tab specifies how to save startup configuration information, the perspective to open after the start of execution or debugging, and other related settings.

item		Description
Save as		Either Local file or Shared file is selected for the startup configuration type. If "Local file" is selected, startup configuration information is saved locally in workspace metadata. If "Shared file" is selected, startup configuration information is saved at the specified location, and can be shared.
	Local file	
	Shared file	
Display in favorites menu		To add Run or Debug to the favorites menu, select "Run" or "Debug". These items are not selected by default.
	Debug	
	Run	
Console Encoding		Default is "Cp1252".
Standard Input and Output		In the COBOL application startup configuration, any changes made to the standard input-output settings are ignored.
	Allocate Console	
	File	
	Append	
Launch in background		To start COBOL applications in the background, select Launch in back ground. Launch in background is selected by default.

Chapter 9 Remote Development Function

This chapter explains NetCOBOL remote development.

9.1 Overview of Remote Development

9.1.1 What is Remote Development?

Using the remote development support functions of NetCOBOL Studio on the client side and NetCOBOL on the server side, you can develop COBOL applications efficiently with a variety of Windows systems.

Server Side

NetCOBOL must be installed on the server.

Operating system	NetCOBOL Product
Windows(64)	Windows 64bit NetCOBOL
Solaris	Solaris 32bit NetCOBOL
Linux(64)	Linux 64bit NetCOBOL
Linux(Itanium)	Linux(Itanium) 64bit NetCOBOL

Client Side

NetCOBOL and NetCOBOL Studio must be installed on the client system in order to enable NetCOBOL Studio to perform development tasks. NetCOBOL Studio can be connected to a server to perform tasks such as compilation, with the results displaying in NetCOBOL Studio.

9.1.2 Advantages of Remote Development

Many COBOL applications run on expensive server machines. Developing a COBOL application on such a machine causes the following problems:

- You must use a command line interface because many of these systems do not enable a GUI-based environment.
- The server side system must be shared among the developers.

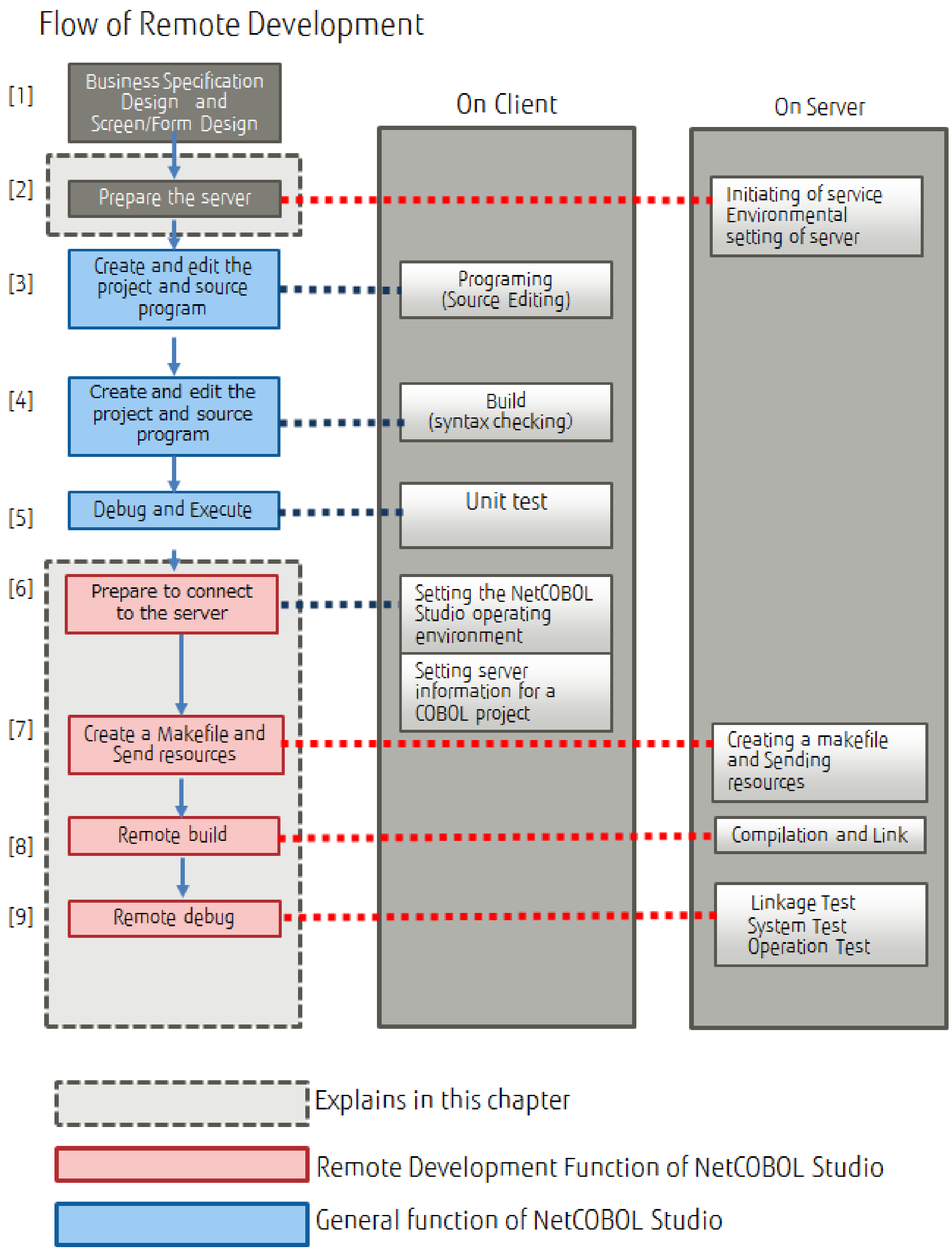
On the other hand, a Windows system is easily available as a personal machine and the developer can occupy its GUI-based environment exclusively.

Remote development solves the above problems by performing development tasks on a Windows system as far as possible.

- You can develop a COBOL application efficiently using a GUI-based development environment.
- You can reduce the load on the server side system by performing development tasks such as source editing on the client side as far as possible.

9.1.3 Flow of Remote Development

The flow of remote development is as follows:


Flow of Remote Development
On Client
On Server
[1]
Business Specification Design and Screen/Form Design
[2]
Prepare the server
Initiating of service Environmental setting of server
[3]
Create and edit the project and source program
Programing (Source Editing)
[4]
Create and edit the project and source program
Build (syntax checking)
[5]
Debug and Execute
Unit test
[6]
Prepare to connect to the server
Setting the NetCOBOL Studio operating environment
Setting server information for a COBOL project
[7]
Create a Makefile and Send resources
Creating a makefile and Sending resources
[8]
Remote build
Compilation and Link
[9]
Remote debug
Linkage Test
System Test
Operation Test
Explains in this chapter
Remote Development Function of NetCOBOL Studio
General function of NetCOBOL Studio

[1] Business Specification Design and Screen/Form Design

Design the application.

[2] Prepare the server

Set the environment for remote development on the server side. Start the remote service, and start the service for the remote development function.

Information

"9.2 Work on server side"

[3] Create and edit the project and source program

COBOL sources and other program resources are created or updated on a local personal computer.

- COBOL source programs
- COBOL library text (COPY clause)
- Screen form descriptors
- Overlay patterns

The program resources that are created or updated at this time are registered in a COBOL project in NetCOBOL Studio.

[4] Build the project (Compilation and linkage - check Syntax)

NetCOBOL Studio is used to compile and link created or updated program resources. This work is performed for the following purposes:

- To confirm that the created program resources do not contain errors or discrepancies
- To check the dependency among the program resources registered in NetCOBOL Studio
- To create an executable program for unit testing

[5] Debug and Execute

A program that has been compiled and linked on a local personal computer is used to test functions within the closed scope in the program. Errors in the program can be detected on the local personal computer by using compile options of the debugging functions provided by NetCOBOL (CHECK, COUNT, and TRACE), and using the COBOL debugger of NetCOBOL Studio.

[6] Prepare to connect to the server

To connect to the server from NetCOBOL Studio, the operating environment of NetCOBOL Studio and the operating environment of the project are set.

Information

"9.3 Preparation to connect to the server"

[7] Create a Makefile and Send resources

Resources on the local personal computer are sent to the server, and a makefile required for a build on the server is created.

Information

"9.4 Creating a makefile and Sending resources"

[8] Remote build

Compiling and linking are performed on the server according to the information defined in the project (e.g., compile options, link options).

Information

"9.5 Remote Build"

[9] Remote debug

The remote debug function is used to debug the COBOL program on the server.

See

"9.6 Remote Debugging"

Note

An executable program on a server cannot be started with NetCOBOL Studio. The executable program must be started directly on the server.

9.1.4 Notes on Remote Development

Generally, COBOL programs have high portability and, in many cases, you can perform tasks from the creation of your program, to unit testing on the client side.

However, if your program contains server-specific features or platform dependent features, you must test your program on the server side.

Following items are examples of platform dependent features:

Literals or hexadecimal literals

If the client side and the server side use different character encoding, a value that can be compiled on one side may cause a compilation error on the other side.

Key sequence or sort key sequence for character comparison or indexed files

The result of descending order of characters may differ based on the difference between the runtime character encoding on the client side and the server side.

Class conditions

If the runtime character encoding on the client side and the server side are different, the identification results may also differ.

Printing

Available characters and fonts differ according to the target platform

Database functions

The execution results may also differ based on the differences in the database products being used.

Web linkage function

The available functions differ depending on the target platform. In addition, Windows and UNIX systems have different file and path name rules.

Note

This system does not support the Web linkage function.

9.1.5 Server side and Client side Combinations

Refer to the "NetCOBOL Software Relese Guide" for the supported combinations of the server side and the client side NetCOBOL products.

9.2 Work on server side

9.2.1 Start Service

To use remote development, the services required by this function must be operating at the server system side. Accordingly, these services must be installed and started.

The services that must be installed at the server side are determined by the server OS and NetCOBOL product version.

Server OS	NetCOBOL Product on Server	Service to start
Windows(64)	Windows 64bit NetCOBOL	NetCOBOL Remote Development Service (*1)
Solaris	Solaris 32bit NetCOBOL V10 or earlier	rtpd/rexec service telnetd(*2)
Linux(Itanium)	Linux(Itanium) 64bit NetCOBOL	rtpd/rexec service telnetd(*2)

Linux(64)	Linux 64bit NetCOBOL	NetCOBOL Remote Development Service(*1)
		telnetd(*2)
		sshd(*3)

*1: The remote development service is a service that NetCOBOL provides. Refer to the following for the service startup procedure.

- For a Windows server

  Refer to "9.2.1.1 NetCOBOL Remote Development Service".

- For a Linux(64) server

  Refer to "NetCOBOL User's Guide" of Linux 64bit NetCOBOL.

*2: This service is required in order to use the remote debug function.

*3: This service is required in order to encode communication contents by SSH port forwarding.

9.2.1.1 NetCOBOL Remote Development Service

The NetCOBOL Remote Development Service (referred as "Remote Development Service" after in this chapter) must be run on the server side system to make use of the remote development functions from NetCOBOL Studio on the client side. The Remote Development Service accepts a request from NetCOBOL Studio, logs on the server side system with the specified account and performs tasks on the server side system with the account.

For security reasons, the Remote Development Service is not configured to start automatically in its installation. You must start the Remote Development Service before you use remote development functions.

Note

Security Notes

For security reasons, the Remote Development Service is not configured to start automatically in its installation.

In order to maintain security, make sure to disclose the Remote Development Service only for the limited period required. When the release of the Remote Development Service is stopped, restore the changes made in the firewall settings or Remote Development Service "Startup type".

Make sure to use remote development functions with the Remote Development Service only inside a safe network such as an intranet with security that is appropriately managed.

When you connect to the Linux(64) server to use Remote Development Service, you can encode communication contents by SSH port forwarding.

In this case, SSH must be running at the server system side, and you need to notify the NetCOBOL Studio user of the public key fingerprint used on the SSH server.

Information

Generally, execute the following command to get the public key fingerprint (typical md5).

```
ssh-keygen -lf /etc/ssh/ssh_host_rsa_key.pub
```

For details, refer to the manual for ssh-keygen command.

9.2.1.1.1 Starting and Stopping the Remote Development Service

This section explains how to start and stop the Remote Development Service.

Starting the Remote Development Service

To start the Remote Development Service, log on the server side system with an administrator account and take the following steps:

1. From the Windows start menu, select **Apps** > **Control Panel** > **System and Security** > **Administrative Tools** > **Services**.
2. From the list of services, select **NetCOBOL Remote Development Services**.
3. In the window menu, select **Action** > **Properties**, and open the **NetCOBOL Remote Development Services Properties** dialog.
4. In the **NetCOBOL Remote Development Services Properties** dialog, select the **General** tab.
5. Click the **Start** button.
6. If you want the Remote Development Service to start automatically when the system is started, change the "Startup type" to "Automatic".

The Remote Development Service uses port 61999 by default, to disclose its services. If port 61999 is already being used in the system, the port number must be changed. Refer to the description of port setting in "9.2.1.1.4 Configuring the Remote Development Service" for details.

If the Windows Firewall or other firewall software is running on the server, you must configure it in a way that it does not block the port which is used by the Remote Development Service. The configuration method for the firewall differs according to the type of firewall software being used. Refer to the document on each firewall software.

Before starting the Remote Development Service, read "Security Notes "under "9.2.1.1 NetCOBOL Remote Development Service".

9.2.1.1.2 Stopping the Remote Development Service

To stop the Remote Development Service, log on the server side system with an administrator account and take the following steps:

1. From the Windows start menu, select **Apps** > **Control Panel** > **System and Security** > **Administrative Tools** > **Services**.
2. From the list of services, select "NetCOBOL Remote Development Services".
3. In the window menu, select **Action** > **Properties**, and open the **NetCOBOL Remote Development Services Properties** dialog.
4. In the **NetCOBOL Remote Development Services Properties** dialog, select the **General** tab.
5. Click the **Stop** button.
6. If you do not want the Remote Development Service to start automatically when the system is started, change the "Startup type" to "Manual".

When Remote Development Service no longer needs to be disclosed, restore the settings changes that were made in the firewall software or so.

9.2.1.1.3 Log files of the Remote Development Service

This section explains the log files output by the remote development service.

Contents of the Log file

The following information is recorded in the log file:

- Connection start date and time
- Client IP address
- Account name

- Whether or not log-on was successful
- Connection end date and time

The commands executed under each user account after logon are not recorded.

The time output to the log file is Universal Time and Coordinated (UTC): coordinated universal time.

The path of the Log file

The path of the log file is as follows:

```
%ProgramData%\Fujitsu\NetCOBOL\RDS\Log\rds.log
```

%ProgramData% is a common application data folder for Windows.

You can change the folder to output the log file by configuration of the Remote Development Service. See the explanation for the logdir setting in "9.2.1.1.4 Configuring the Remote Development Service" for details.

Generations of the Log file

When the size of a log file reaches the maximum size, a backup is made for that log file and a new log file is created.

You can change the maximum size of log files by configuration of the Remote Development Service. Refer to the explanation for the maxlogsize setting in "9.2.1.1.4 Configuring the Remote Development Service" for details.

Backup files are located in the same folder as the log file output folder and have the below name.

```
rds.<sequence number>.log
```

Where <sequence number> is a number starting at 001 and has a maximum value of 999. When new backup file is created, a new sequence number is allocated for it. This sequence number is the next sequence number of the backup file with the most recently updated time amongst the backup files in the same folder. If there is no backup file in the same folder, the sequence number 001 is allocated. 001 is also regarded as the next number of 999.

When a new backup file with sequence number n is created, only the backup files which have sequence number between (n - the number of backup generation + 1) and n are retained. All other backup files are deleted. For example, if the new backup file is rds.007.log and the number of backup generations is 3, all backup files are deleted except for rds.005.log, rds.006.log, and rds.007.log.

You can change the number of backup generation by configuration of the Remote Development Service. See the explanation for the maxloggen setting in "9.2.1.1.4 Configuring the Remote Development Service" for details.

9.2.1.1.4 Configuring the Remote Development Service

This section describes the Remote Development Service settings that can be changed and explains how to set them.

The following settings can be changed:

- Port number to be used
- The output folder, maximum size, and number of backup generations for log files

Specify the settings that you want to change as service start parameters. The Remote Development Service must then be restarted for the changes to take effect.

Changing setting of the Remote Development Service

To change setting of the Remote Development Service, log on the server side system with an administrator account and perform the following steps:

1. From the Windows start menu, select **Apps** > **Control Panel** > **System and Security** > **Administrative Tools** > **Services**.
2. From the list of services, select "NetCOBOL Remote Development Services".
3. In the window menu, select **Action** > **Properties**, and open the **NetCOBOL Remote Development Services Properties** dialog.
4. In the **NetCOBOL Remote Development Services Properties** dialog, select the **General** tab.
5. Enter the settings in the "Start parameters".

Note

- To enter more than one setting in the "Start parameters", separate the settings with spaces.
- To enter a setting that contains a space, enclose it in quotation marks (").

```
[Example] /port:61999 "/logdir:C:\log data" /maxlogsize:128 /maxloggen:2
```

- The contents of "Start parameters" are not saved. If you configure the Remote Development Service to start automatically when the system is started, the Remote Development Service starts with default setting when the system is started next time.

Refer to the list below for the settings that can be entered in the "Start parameters".

Setting list of the Remote Development Service

The following table shows the settings that can be changed.

Setting name	Form to specify in start parameters	Default value	Explanation
port	/port:<number>	61999	Specify the TCP/IP port number used by the Remote Development Service. In <number>, specify the port number in decimal form.
logdir	/logdir:<path>	Windows shared application data folder (*)	Specify the folder to which remote development service log files are output. In <path>, specify the folder path. To use a path that contains spaces, enclose the start parameter in quotation marks (").
maxlogsize	/maxlogsize:<size>	128	Specify the maximum size of the Remote Development Service log file. In <size>, specify the maximum size in decimal form. Its unit is kilobytes. If 0 is specified, the Remote Development Service does not output log file
maxloggen	/maxloggen:<number of generations>	2	Specify the number of generations of Remote Development Service log file backups to be retained. In <number of generations>, specify the number of generations in decimal form. If n is specified, n backups (rds.xxx.log) are retained. If a number greater than 999 is specified, 999 is considered to be specified. If 0 is specified, no backup is retained.

*: This depends on the version of Windows. Refer to "The path of the log file" under "9.2.1.1.3 Log files of the Remote Development Service".

9.2.1.2 ftpd/rexec services

This section explains how to install and start the ftpd and rexec services at the server.

9.2.1.2.1 On a Solaris server

The ftpd services are installed with their default settings when the operating system is installed on the Solaris server, and always start automatically.

Note

No start setting for the rexec service are already configured when the Solaris operating system is installed. The following operations must be performed:

- Checking the status

  Execute the following command to check the status:

```
# svcs -a | grep rexec
disabled   18:28:10 svc:/network/rexec:default
```

If "disabled" is displayed at the beginning of the execution results, start the rexec service.

- Starting the rexec service

  Execute the following command to start the rexec service:

```
# svcadm enable svc:/network/rexec:default
```

In most cases, to change the system settings of the Solaris server involves checking the contents of the setting file, and modifying them as necessary.

1. Check the service status.

   inetd (Internet daemon) invokes the ftpd and rexec services on the Solaris server. Therefore, the contents of the following files should be checked:

   - /etc/services
   - /etc/inetd.conf

   If valid descriptions of the ftpd and rexec services are found in /etc/services and /etc/inetd.conf, skip the remaining steps in this section. See the examples below. If the line in which ftpd or rexec or both are set is commented out ("#" is added at the beginning of the relevant line), then the remaining steps must be performed.

   Example for /etc/services

```
#
# Network services, Internet style
#
:
ftp 21/tcp
:
## UNIX specific services
## these are NOT officially assigned
#
```

   Example for /etc/inetd.conf

```

:
# FTPD - FTP server daemon
ftp stream tcp6 nowait root /usr/sbin/in.ftpd in.ftpd -a
:
# REXECD - rexec daemon (BSD protocols)
exec stream tcp nowait root /usr/sbin/in.rexecd in.rexecd
exec stream tcp6 nowait root /usr/sbin/in.rexecd in.rexecd
:
```

2. Change the settings of the services.

   Modify /etc/services and /etc/inetd.conf.

3. Start the services.

   Restart inetd because the ftpd and rexec services are started under the control of inetd.

   Execute either the following command:

```
# kill -HUP `cat /var/run/inetd.pid`
```

   Or execute the following:

```
# ps -ea | grep inetd
inetd process id is displayed.
# kill -HUP inetd process id D
```

9.2.1.2.2 On a Linux(Itanium) server

The ftpd and rexec services may not be installed along with the operating system on the Linux server. Therefore, the first step is to determine if their packages are installed. A Linux system may provide tools with a GUI that can be used to configure system settings. Since there are considerable differences between the versions and individual system settings in differing GUI tools, a method for using commands to perform operations is presented below.

1. Check the packages.

   Execute the rpm command in the following format to determine if the packages are installed:

   ```
   # rpm -query package-name
   ```

   Enter the following package names for ftpd and rexec:

   - ftpd : vsftpd
   - rexec : rsh and rsh-server

   If the following information is displayed, the packages have been installed, and you can proceed to checking the service status.

   ```
   # rpm -query vsftpd
   vsftpd-2.0.1-5
   # rpm -query rsh
   rsh-0.17-17
   # rpm -query rsh-server
   rsh-server0.17-17
   ```

   If the package information is not displayed, one or more packages must be installed.

   Note

   The type of ftpd package used for the Linux system depends on the version and distribution. The following may be used:

   - wu-ftpd
   - proftpd

2. Install packages.

   Execute the rpm command to install each package.

   ```
   rpm -Uvh package-name
   ```

3. Check the service status.

   Execute the /sbin/chkconfig command in the following format to determine if the services are configured to start when the system starts.

   ```
   /sbin/chkconfig --list service-name
   ```

   For example, if the following results are obtained, startup of ftpd (vsftpd) and rexec when system starts has been disabled using their respective settings.

   ```
   # /sbin/chkconfig --list vsftpd
   vsftpd 0:off 1:off 2:off 3:off 4:off 5:off 6:off
   # /sbin/chkconfig --list rexec
   rexec off
   ```

4. Change the setting for a service as necessary.

   Execute the /sbin/chkconfig command in the following format to confirm that the service will start when the system starts:

```
/sbin/chkconfig [--level run-level] service-name on
```

In the following example, the setting is changed so that the service is started when the system starts, and the changed setting is checked:

```
# /sbin/chkconfig --level 5 vsftpd on
# /sbin/chkconfig --list vsftpd
vsftpd 0:off 1:off 2:off 3:off 4:off 5:on 6:off
# /sbin/chkconfig rexec on
# /sbin/chkconfig --list rexec
rexec on
```

5. Start the services.

   Execute the /sbin/service command in the format shown below to start the services. Since the rexec service is started under the control of xinetd, restart xinetd.

```
# /sbin/service vsftpd start
vsftpd start: [ OK ]
# /sbin/service xinetd stop
xinetd stop: [ OK ]
# /sbin/service xinetd start
xinetd start: [ OK ]
```

9.2.2 Setting up the user environment on the server

9.2.2.1 On a Solaris server, Linux(64) or Linux(Itanium) server

For code conversion

When the code conversion is done with Interstage Charset Manager on the server side in the sending and receiving of the resource, the following environment variables are set.

- LD_LIBRARY_PATH

  This must be set to use Interstage Charset Manager on the server for the code conversion required for transmitting and receiving program resources.

```
${CHARSET_BASED}/lib
```

{CHARSET_BASED}: Interstage Charset Manager installation folder.

By setting the following environment variable on a Linux(64) server where NetCOBOL version is V11.0.0 or later, the code conversion is done with Interstage Charset Manager on the server side in the sending and receiving of the resource code conversion.

- COBOL_REMOTE_CONVERT_CHARACTER

  For details about environment variables, refer to the "NetCOBOL User's Guide" of Linux 64bit NetCOBOL.

For build

- PATH

  Add the following to the PATH environment variable to specify the storage path of the COBOL compile command:

```
${COB_BASED}/bin
```

- LD_LIBRARY_PATH

  Add the following to the LD_LIBRARY_PATH environment variable to specify the storage path of the COBOL runtime shared library:

```
${COB_BASED}/lib
```

- NLSPATH

  Add the following to the NLSPATH environment variable to specify the storage locations of the messages output at compile time or execution time of COBOL programs:

```
${COB_BASED}/lib/nls/%L/%N.cat:{COB_BASED}/lib/nls/C/%N.cat
```

- LANG

  Specify each character code set that is used by a COBOL program. This specification is used to determine whether COBOL sources contain national characters, and which character sets are used in COBOL sources at compile time.

For the Solaris and Linux (Itanium) operating systems, C is specified for the LANG environment variable, which identifies the Unicode (UTF8) character code set en_US.UTF-8 used by COBOL programs.

Dedicated shell scripts provided in the NetCOBOL product on each UNIX system is used to set individual environment variables except LANG. The following table outlines shell scripts for setting the required environment variables for compilation and linkage.

NetCOBOL product	Storage location	File name	Remarks
Solaris 32bit NetCOBOL	/opt/FJSVcbl/config	cobol.csh	For csh
Linux(Itanium) 64bit NetCOBOL	/opt/FJSVcbl/config	cobol.sh	For sh/bash
		cobol.csh	For csh/tcsh
Linux 64bit NetCOBOL	/opt/FJSVcbl64/config	cobol.sh	For sh/bash
		cobol.csh	For csh/tcsh

Specify the following other environment settings as necessary:

- COBOLOPTS

  Use this environment variable to commonly and independently specify some compile options from individual programs under development. This is useful for specifying the following:

  - Options for COBOL debugging functions
  - Options for compilation lists

- COBCOPY/FORMLIB/FILELIB

  Specify the storage directories of COBOL libraries, screen form descriptors, and file descriptors to be shared by multiple developers.

- Library names

  Set the library file storage directories in the environment variables whose names are library names specified with IN/OF.

Note

If the LC_ALL environment variable is required, use the same value as that used in the LANG environment variable.

Examples of shell scripts for setting environment variables

In the examples shown below, the environment variables are assumed to be set as follows:

- Environment variable related to transfer of resources

  A setting for using Interstage Charset Manager on the server side is added to the LD_LIBRARY_PATH environment variable.

- Environment variables related to build

  - Environment variables required for compilation and linkage of a COBOL program are set using shell scripts provided by NetCOBOL.
  - UTF-8 is the character code set used by programs under development. It is specified in the LANG environment variable.

- The compilation list of COBOL source programs is stored in a common directory.
- The common storage directory of libraries referenced by developers is specified.

On a Solaris server

For remote development using a Solaris server, csh must be used as the login shell. Edit .cshrc that is found in the home directory used by each developer, and add the text shown below.

In the following example, .cshrc is modified for a Solaris server.

```
## COBOL environment setting
source /opt/FJSVcbl/config/cobol.csh
## Environment setting for Interstage Charset Manager
if(${?LD_LIBRARY_PATH}) then
setenv LD_LIBRARY_PATH /opt/FSUNiconv/lib:${LD_LIBRARY_PATH}
else
setenv LD_LIBRARY_PATH /opt/FSUNiconv/lib
endif
## Setting for compilation and link that is common to developers
setenv COBOLOPTS "-dp ../list"
setenv COBCOPY ../COPYLIB:${COBCOPY}
## Character code set used by the program to be developed
setenv LANG en_US.UTF-8
```

On a Linux(Itanium) or Linux(64) server

For remote development using a Linux server, csh or bash can be used as the login shell.

To use csh as the login shell, edit .cshrc that is found in the home directory used by each developer, and add the text shown below.

```
## COBOL environment setting
source /opt/FJSVcbl/config/cobol.csh
## Environment setting for Interstage Charset Manager
if(${?LD_LIBRARY_PATH}) then
setenv LD_LIBRARY_PATH /opt/FSUNiconv/lib:${LD_LIBRARY_PATH}
else
setenv LD_LIBRARY_PATH /opt/FSUNiconv/lib
endif
## Setting for compilation and link that is common to developers
setenv COBOLOPTS "-dp ../list"
setenv COBCOPY ../COPYLIB:${COBCOPY}
## Character code set used by the program to be developed
setenv LANG en_US.UTF-8
```

To use bash as the login shell, edit .bashrc that is found in the home directory used by each developer, and add the text shown below.

```
## COBOL environment setting
source /opt/FJSVcbl/config/cobol.sh
## Environment setting for Interstage Charset Manager
if  ${LD_LIBRARY_PATH:-""} = ""  ; then
LD_LIBRARY_PATH=/opt/FSUNiconv/lib; export LD_LIBRARY_PATH
else
LD_LIBRARY_PATH=/opt/FSUNiconv/lib:${LD_LIBRARY_PATH};export LD_LIBRARY_PATH
fi
## Setting for compilation and link that is common to developers
COBOLOPTS="-dp ../list";export COBOLOPTS
COBCOPY=../COPYLIB:${COBCOPY}; export COBCOPY
## Character code set used by the program to be developed
LANG=en_US.UTF-8; export LANG
```

9.2.2.2 On a Windows server

For remote development using a Windows server, a user account for the server environment is used to perform development work. Unless remote development work is performed with an existing user account, a new user account for remote development work must be created in the server environment. When creating a user account, consult the server administrator.

Generally, to add a local user account to the server machine, log on to the server with an administrator account. Then, select **Apps** > **Control Panel** > **System and Security** > **Administrative Tools** > **Computer Management** from the Start menu to display **Computer Management** window, and use "Local Users and Groups" .If an additional environment setting must be made for remote development work, add the setting to the environment of the relevant user account.

Note

When registering a user account, register it as a member of a user group. Its password cannot be changed while the associated local personal computer is connected to the server. Therefore, when making settings for the user account, do not set a password change while the local personal computer is connected to the server.

Note

There are two types of environment variables: user environment variables and system environment variables. The user environment variables are generally used . However, for PATH environment variables, the system environment variables are used.

When giving priority to an original path, the COB_RDENV_X64 environment variable is used.

When a file is specified for the COB_RDENV_X64 environment variable, the file is considered to be a batch file.

A NetCOBOL remote build executes the file first by the CALL batch command.

```
COB_RDENV_X64=C:\MyTools\myenv.bat
```

`Content of"C:\MyTools\myenv.bat"`

```
path C:\MyCommand;%PATH%
```

As a result, "C:\MyCommand" is given a higher priority than the path of the system environment variable in a remote build.

Windows(64)

The environment variable "PATH" will be set to the folder name of the user account at the installation destination of NetCOBOL. Remote Build is executed in the same environment as the "NetCOBOL command prompt" on the NetCOBOL server side.

9.3 Preparation to connect to the server

To connect to the server, set NetCOBOL Studio as follows.

- Set the server information
- Set the Project setting for remote development

When the ftpd/rexec service is used without using the NetCOBOL remote development service on the server side, a Windows firewall exception message will display.

9.3.1 Setting the NetCOBOL Studio operating environment

The information that is used by each developer's NetCOBOL Studio for link with the server must be set. These settings are shared among workspaces.

Setting server information

To display the **New Server Information** dialog box and set server information.

1. Select **Window** > **Preferences** from the menu bar. The **Preferences** dialog box appears.

2. Select **COBOL** > **Remote Development** in the left pane of the **Preferences** dialog box. The **Remote Development** page is displayed.

3. Click the **New** button on the **Remote Development** page. The **New Server Information** dialog box appears.

Table 9.1 New Server Information dialog box

Item		Description
Server name		Specify an arbitrary name, which will be displayed in Server name on the **Remote Development** page of the **Preferences** dialog box. The server name is used to specify the target server of a COBOL project. A server name that is already defined cannot be specified.
Server OS		Select the operating system of the server.
Server address		Specify a name (FQDN: Fully Qualified Domain Name) or IP address that identifies the server in the network.
Save user name and password		Specify whether the user name and password are used to connect the server. If this item is checked, the user name and password specified in this dialog box will be used. If the item is unchecked, the dialog box for Inputting user name and password for connecting to the server appears when the server is initially connected after NetCOBOL Studio is launched. The default setting is unchecked.
	User name	Specify the user name of an account used to access the server. If Save user name and password is checked, a "User name" must be specified.
	Password	Specify the password assigned to the user. If Save user name and password is checked, a "Password" must be specified.
Character-code conversion		Code conversion information for files is transmitted to and received from the server. Code conversion processing is usually executed by the code conversion function provided in the system. However, if ADJUST (only for Windows) or Interstage Charset Manager is installed, it is used for code conversion.
	COBOL Source character-code on the server side	Select the character-code of the COBOL source file transferred to the server side of remote development. The following character-codes can be selected. - UTF-8 - C
	Convert character set in the server	Specify whether the server or the local PC converts code for sending or receiving a text file. The default setting is with "Convert character set in the server" checked. If "Windows(x64)" is selected at "Server OS", this item is disabled.
	Convert character set in the client	
Information for UNIX Server system		This information is specified if the selection at "Server OS" is Solaris or Linux(Itanium).
	FTP is used for remote development	When this option is selected, the ftpd/rexec services are used as the remote development server-side services.

Item		Description
	Use PASV mode for file transfer(FTP)	If this item is checked, files are transferred in PASV mode. The server may not be connected when files are transferred to or from an FTP server outside the firewall. For such cases, this item must be checked. The default setting is unchecked. This option is enabled for remote development that uses the server-side ftpd/rexec services.
NetCOBOL Remote Development Service on server side		This information is for the server-side NetCOBOL remote development service.
	Port number	Specify the TCP/IP port number of the NetCOBOL remote development services. If the port number of the NetCOBOL remote development services has been changed on the server side, adjust this value to the changed port number. The default value is 61999. This option is enabled if remote development uses the server-side NetCOBOL remote development service.
	SSH Port Forwarding	The connection to NetCOBOL remote development services are encrypted. When "Linux(x64)" is selected at "Server OS", this item is enabled.
	SSH Port number	Specify the SSH Port number. The default value is 22. When "SSH Port Forwarding" is selected, this item is enabled.
	SSH Server FingerPrint	Specify the FingerPrint by hexadecimal digit string. Ask the server administrator about the value. When "SSH Port Forwarding" is selected, this item is enabled.
	Algorithm	Specify the algorithm of SSH Server FingerPrint (either sha1 or md5). Ask the server administrator about the value. The default value is md5. When "SSH Port Forwarding" is selected, this item is enabled.
Connection test		The server is connected using the current set values when this button is clicked, and the connection results are displayed in the Connection test dialog box.

Verifying a connection to the server

The server is connected using the values specified in the **New Server Information** or **Modify Server Information** dialog box of server information when the **Connection test** button is clicked. The results are displayed in the **Connection test** dialog box. When a connection is successfully established, the information set for the server environment variables is displayed. If the connection attempt fails, error information is displayed.

Inputting user name and password for connecting to the server

If Save user name and password is not checked in the **New Server Information** or **Modify Server Information** dialog box of server information, the dialog box for specifying a user name and password appears when the server is initially connected after NetCOBOL Studio is launched. The user name and password are used to connect the server.

Changing server information

To display the **Modify Server Information** dialog box and change server information:

1. Select **Window** > **Preferences** from the menu bar. The **Preferences** dialog box appears.

2. Select **COBOL** > **Remote Development** in the left pane of the **Preferences** dialog box. The **Remote Development** page is displayed.

3. In the **Remote Development** page, select the name of the server whose information is to be changed from the **Server name** list, and click the **Modify** button. The **Modify Server Informatio**n dialog box appears.

Deleting server information

To delete server information:

1. Select **Window** > **Preferences** from the menu bar. The **Preferences** dialog box appears.
2. Select **COBOL** > **Remote Development** in the left pane of the **Preferences** dialog box. The **Remote Development** page is displayed.
3. In the **Remote Development** page, select the name of the server whose information is to be deleted from the **Server name** list, and click the **Delete** button. The server information is deleted.

9.3.2 Setting server information for a COBOL project

No server information for remote development is set for newly created COBOL projects, so the remote development function cannot be used.

To set information for remote development of the COBOL project:

1. Select the project in the **Dependency view** or **Structure view**.
2. Select **Property...** from the context menu. The property dialog box appears.
3. Select **Remote Development** in the left pane of the property dialog box. The **Remote Development** page is displayed.

Table 9.2 Remote Development page

Item	Description
Enable project specific setting	When the "Enable project specific setting" checkbox is unchecked, the options that were specified in each COBOL project are disabled.
Enable remote development	If this project is for remote development, this item must be checked. If no server information is set, this item is disabled. The default setting is unchecked.
Server name	Specify the name of the server that is the target of this project. A list of server names can be displayed by selecting **COBOL** > **Remote Development** in the **Preferences** dialog box. If "Enable remote development" is unchecked or no server name is defined, this item is disabled.
Server directory	Specify the full path name of the folder of server resources. The makefile creation function and build function use this folder as the current folder to execute their processing. - If Solaris, Linux(Itanium) or Linux(64) is the operating system of the server, the root folder cannot be specified. - If Windows is the operating system of the server, the location immediately under a drive name cannot be specified. The server folder can be referenced by clicking the **Browse...** button. If "Enable remote development" is unchecked or no server name is defined, this item is disabled.

Note

If the directory that is specified for "Server directory″is not found on the server, it is created at the makefile creation time.

The directory that is specified for "Server directory″must be unique to each COBOL project on a local personal computer. If multiple COBOL projects share one such directory on the server, makefiles is not executed correctly.

9.3.3 Exception registration of Windows firewall

If Windows Firewall, the remote development function cannot be used. To prevent this problem, change settings so that Windows Firewall does not check the program outlined in the following table.

Program name	Storage folder	Remarks
COBRDC32.exe	NetCOBOL installation folder	Remote debugger connector

To register COBRDC32.exe as an exception:

1. Click the **Control Panel** > **System and Security** > **Allow an app through Windows Firewall**.
2. Click the **Change settings** button, and then click the **Allow another app** button in the **Allowed apps** window.
3. Click the **Browse** button in the **Add an app** dialog box.
4. Select **COBRDC32.exe** and click the **Open** button in the NetCOBOL installation folder in the **Browse** dialog box.
5. Click the **Add** button in the **Add an app** dialog box, and then click the **OK** button in the **Allowed apps** window.

Changing the scope

To change the scope:

1. In the Windows Firewall setting window, click the **Advanced settings** tab.
2. Click the **Inbound Rules** button in **Windows Firewall with Advanced Security** dialog box. Select the program whose scope is changed, and then select **Properties** menu from the context menu.
3. In the **Scope** tab of properties dialog box, select the "These IP addresses", and then click the **Add** button.
4. Enter the IP address, and then click the **OK** button in the **IP address** dialog box.
5. Check the IP address is displayed in the **These IP addresses** window, and then click the **OK** button in the properties dialog box.

On a Linux (Itanium) server

In the event that there is a connection to a Linux server and the Windows firewall is enabled, and the ftpd/rexec service is utilized instead of the server-side NetCOBOL remote development service, the linkage to the Linux server might exhibit extremely slow processing time. To alleviate this, follow the procedure below to register port 113 as an exception.

1. In the Windows Firewall setting window, click the **Advanced settings** tab.
2. Click the **Inbound Rules** button in **Windows Firewall with Advanced Security** dialog box. Click the "New Rule" in the **Actions** menu list.
3. Select the "Port", and then click the **Next** button in the **New Inbound Rule Wizard**.
4. In the **Protocol and Ports** page, select the "TCP" and the "Specific local ports", and then enter "113". Click the **Next** button.
5. Select the "Allow the connection", and click the **Next** button. Set the remaining items following instructions of the wizard.

9.4 Creating a makefile and Sending resources

A makefile is required for build processing executed on the server in remote build. This makefile can be created using the makefile creation function.

9.4.1 Creating a makefile

To create a makefile.

1. Select the project in the **Dependency view** or **Structure view**.
2. Select **Project** > **Remote Development** > **Makefile Creation...** from the menu bar, or select **Remote Development** > **Makefile Creation...** from the context menu. The **Makefile Creation** dialog box appears.

3. Makefile creation conditions are displayed in "Create Condition ″in the **Makefile Creation** dialog box. To create a makefile, under "Create Condition" displayed, click the **OK** button. To change the "Create Condition" displayed, click the **Option setting...** button and change the contents of "Create Condition".

The created makefile is named "Makefile", and it is stored on both the server and the local personal computer. The makefile on the local personal computer is registered in the Other Files folder of the COBOL project. The makefile on the server is stored in the Server directory as specified on the **Remote Development** page of project properties.

Point

Makefile creation processing is executed on the server. The results of execution on the server can be reviewed by selecting "COBOL Remote" from an icon (**Open Console**) on the toolbar in the **Console view**.

Note

Makefile generation handles as COBOL source files any files with the extension ".cobol" or ".cob" that are directly under the directory specified on the project **Property** > **Remote Development** page at "Server directory".

9.4.2 Changing makefile creation conditions

The following makefile creation conditions, displayed in the **Makefile Creation** dialog box, can be changed:

- TARGET names
- Precompiler options
- Compile options
- Link options

To change these conditions, click the "Option" setting button in the **Makefile Creation** dialog box. The **Option setting** dialog box appears, and the creation conditions of the TARGET names, compile options, and link options at makefile creation can be changed.

9.4.3 Makefile Creation dialog box

- Target Name

  The name of an executable file or dynamic link library (shared library) that is a TARGET of the makefile is displayed.

- Send files

  A list of files to be transferred to the server when the makefile is created is displayed. Files are displayed by file type as follows:

  - COBOL source files
  - COBOL libraries and descriptor files
  - Precompiler input sources

- Compile Options

  Compile options that are used in the makefile at compile time of a COBOL source are displayed.

- Link options

  Link options that are used in the makefile during linkage of a COBOL source are displayed.

- Precompiler link

  If precompiler link is used, the following precompiler information is displayed:

  - Extensions of precompiler input sources
  - Extensions of precompiler output sources

    - Precompiler command name and parameters

    - Indication of whether to display error messages of the COBOL compiler with the line numbers of lines in precompiler input sources

Values in the Makefile Creation dialog box

Values displayed for Create Condition in the **Makefile Creation** dialog box vary according to whether the first makefile is being created or the second or subsequent makefile is being created:

First makefile is being created

The values set on the **Target**, **Build** and **Precompiler** page of project properties are referenced. However, the information set on the **Build** page will be partially modified. The tabs on the **Build** page are as follows:

- **Compiler Options** tab

    - A copy of the information is created for compile options that have formats that are common to the systems.

    - Specifications for compile options not supported by the operating system of the server are ignored.

- **Library Names** tab

  The values specified in the **Library Names** tab are not reflected in the makefile. Set the library file storage folder in the server-side environment variables whose names are library names specified with IN/OF.

  Note

  When the server side is Windows (x64), the library file storage folder is defined in the COB_library-name environment variable.

- **Linker Options** tab

    - The storage paths and names of the object files and library files provided by NetCOBOL are replaced with the storage paths and names of files provided by the server-side NetCOBOL.

    - The following applies to object and library files not provided by NetCOBOL:

        - If the server operating system is Windows(x64), a copy of the information is created.

        - If the server operating system is Solaris or Linux, information for the object files and library files is deleted.

Second or subsequent makefile is being created

The values used for the previous makefile are used for second and subsequent makefiles.

9.4.4 Option Setting dialog box

9.4.4.1 TARGET tab

TARGET names can be changed by clicking the **Target** tab in the **Option setting** dialog box.

Item	Description
Target Name	Specifies a TARGET name.
Initialize	Changes the values to the ones specified on the **Target** page of project properties.

Note

The types of executable files and dynamic link libraries (shared libraries) become the values selected for *Target type* on the **Target** page of project properties. To change TARGET types, change the values selected for *Target type* on the **Target** page of project properties.

9.4.4.2 Precompiler tab

Precompiler settings can be changed by clicking the **Precompiler** tab in the **Option setting** dialog box.

Item	Description
Use Precompiler	Select this item to create precompiler link information in the makefile for the COBOL project. When using the precomplier, first add the precomplier build tool and set precomplier link information for the project properties. If the precomplier build tool is not suitable for the project, makefile could not be created correctly. If not selected, precompiler link information is not created in the makefile even though the information is set. For details about the precomplier setting method, refer to "6.2.3 Creating a COBOL program by using the precompiler".
Precompiler command	Specifies the name of the command that starts the precompiler.
Precompiler parameters	Specifies the precompiler command parameters.
Precompiler source extension	Specifies the extension of the input source file for the precompiler. The following extensions cannot be specified: - cobol - cob - cbl - lace
Precompiler output source extension	Specifies the extension of an output source file for the precompiler.
Use original source line numbers for error messages	If this item is checked, COBOL compiler error messages are displayed using the line number in the original source, rather than the line number of the preprocessed source. (The INSDBINF command is invoked.) The default setting is unchecked.
INSDBINF parameters	This item sets the parameters of the INSDBINF command (*), which develops line information for the precompiler input source in COBOL source files generated by precompilation. Note that input and output source file names cannot be specified because they are determined from precompiler input source file names.
Initialize	Changes the values to those specified in the precompiler link information on the preferences **Precompiler** page.

*: For the INSDBINF command, refer to "6.2.2 INSDBINF command".

For more information about precompiler link information, see "6.2.1 Setting and changing initial values of precompiler link information

Note

".cobol" cannot be specified as the extension of the precompiler output source if the server side NetCOBOL is listed below.

- Solaris 32bit NetCOBOL or Linux(Itanium) 64bit NetCOBOL

  V9.1 or before

9.4.4.3 Compiler Options tab

Compile options can be changed by clicking the **Compiler Options** tab in the **Option setting** dialog box.

Item	Description
Compiler Options	Displays the compile options used in the makefile at COBOL source compile time.
Add	Adds the compile options. To add compile options, select the compile options to add in **Compiler Options** in the **Add Compiler Options** dialog box, and click the **Add** button.
Change	Changes the compile options selected in **Compiler Options**.
Remove	Deletes the compile options selected in **Compiler Options**.
Initialize	Changes the values to the ones specified on the **Compiler Options** tab on the project properties **Build** page.
Specify other compiler options	Specifies the compile options that cannot be added from the **Compiler options** dialog box.

Compile options that cannot be used during remote development

When the server side is Solaris, the following compiler option cannot be used.

- ASCOMP5 compiler option

  When the server side is the following, the following compiler option can be used. When the server side is Solaris the following compiler option cannot be used.

  - Windows 64bit NetCOBOL V11.0 or later

- ENCODE compiler option

  When the server side is the following, this compiler option can be used.

  - Windows 64bit NetCOBOL V11.0 or later
  - Linux 64bit NetCOBOL V11.0 or later

- CONVCHAR compiler option

  When the server side is the following, this compiler option can be used.

  - Linux 64bit NetCOBOL V11.0 or later

Compile options specific to remote development

Some compile options specific to remote development are accessed in the **Option setting** dialog box **Compiler Options** tab.

The following table lists the compile options that are specific to remote development and specifies whether or not these compile options can be used with particular target operating systems.

Compile options	Solaris	Linux		Windows
		Itanium	64	64
CODECHK	Y	Y	N	N
KANA	Y	Y	N	N
LALIGN	Y	Y	Y	N

Y: Can be used.

N: Cannot be used.

Details of the compile options specific to remote development are as follows:

CODECHK compile option

This option specifies whether to perform a compile-time check of the national language code system at execution time (CODECHK) or not (NOCODECHK). When creating a program that does not depend on the national language code system (program common to ASCII, EUC, and Unicode), specify NOCODECHK.

Item		Description
National language code system check at execution time		Specifies whether to perform a compile-time check of the national language code system at execution time. The default is CODECHK.
	CODECHK	Performs a compile-time check of the national language code system at execution time.
	NOCODECHK	Does not perform a compile-time check of the national language code system at execution time.

KANA compile option

This option specifies the code sets for kana characters in non-numeric literals, alphabetic and alphanumeric data items.

Item		Description
KANA character code processing		Specifies how to process the kana character code set. The default is KANA(EUC).
	KANA(EUC)	Two-byte codes (EUC) are used for kana character codes.
	KANA(JIS8)	One-byte codes (JIS) are used for kana character codes.

LALIGN compile option

This option specifies whether to generate objects based on an assumption that data is aligned on an eight-byte boundary (LALIGN) or not (NOLALIGN) when data declared in the linkage section is referenced. The data processing speed is faster when objects are generated based on an assumption that data is aligned on an eight-byte boundary.

Item		Description
Processing of data declared in LINKAGE SECTION		Specifies how to process data declaration in the linkage section. The default is NOLALIGN.
	NOLALIGN	Does not assume that data is aligned on an eight-byte boundary.
	LALIGN	Assumes that data is aligned on an eight-byte boundary.

Compile option with different specification format during remote development

- RCS compile option

When the server side is Solaris, this compiler option cannot be used.

When the server side is Linux (itanium), this compiler option has different specification formats for the local personal computer environment.

When the server side is Linux(x64) or Windows(x64), this compiler option can be used in the same specification formats as the local personal computer environment.

RCS compile option

The RCS compile option has different specification formats for the local personal computer environment and the Linux (Itanium) environment. The following information pertains to the dialog box that appears when the server operating system is Linux (Itanium). The encoding form of national items in a Unicode environment is UTF-16.

Item	Description
Encoding of national items with Unicode environment	Specifies an endian of UTF-16. The default is LE (little-endian).

Item		Description
	BE	Specifies big-endian as the endian of UTF-16.
	LE	Specifies little-endian as the endian of UTF-16.

9.4.4.4 Library Name tab

The values specified in the **Library Name** tab of the **Build** page of project properties can be referenced by selecting the **Library Name** tab in the **Option setting** dialog box. The values specified on the **Library Name** tab are not reflected in the makefile. Specify the library file storage folder in the server-side environment variables with library names specified with IN/OF.

Note

When the server side is Windows (x64), the library file storage folder is defined in the COB_library-name environment variable.

9.4.4.5 Linker Options1 tab, Linker Options2 tab

Link options are configured on the two tabs in the **Option setting** dialog box as follows:

- **Linker Options1** tab

  Libraries and object files can be specified to be linked with COBOL programs on the server side.

- **Linker Options2** tab

  Link options specific to the operating system of the server can be specified.

Changing the libraries and object files to be linked

The libraries and object files to be linked on the server side can be changed by selecting the **Linker Options1** tab in the **Option setting** dialog box.

Item		Description
Add		Adds libraries or object files to those to be linked with a COBOL program. The **Add Linker Option** dialog box appears when the **Add** button is clicked. The added libraries and object files are displayed in "Select library or object". One or more libraries or object files can be added.
Change		Changes the specification of the library or object file selected in "Select library or object". The **Modify Linker Option** dialog box appears when the **Change** button is clicked.
Remove		Deletes the libraries and object files selected in "Linker options".
Remove All		Deletes all of the libraries and object files in Linker options.
C Run-time Library Name		Specify the file name of the C run-time library to be linked. This option is enabled if the server OS is Windows(x64). If the C run-time library name is omitted, "LIBCMT.lib" is linked. In the following cases, specify "MSVCRT.lib." - When two or more DLLs execute in one process. - The called C program is created with Microsoft Visual C++ and compiled with /MD option. For details of the C run-time libraries linkage, refer to "NetCOBOL User's Guide."
COBOL Entry Object		Specify whether only object files created in COBOL are used to create the dynamic link library, or whether object files created in other languages are also used to create it. This option is enabled if the server OS is Windows(x64).
	Link COBOL program only	The dynamic link library is created using only object files created in COBOL.

Item	Description
Other	Object files created in other languages are also used to create the dynamic link library.
Initialize	The item is initialized by the value specified on the build page of the property of the project.

Adding and changing libraries and object files

The **Add Linker Option** dialog box appears when the **Add** or **Change** button on the **Linker Options1** tab in the **Option setting** dialog box is clicked. A library or object file can be added or changed from this dialog box. "Linker options″specifies the name of the library or object file to be added or changed. The absolute path name or relative path name of the library or object file is specified. The file name itself cannot be specified alone. The dialog box for referencing server-side files appears when the **Browse** button is clicked, and a library or object file can then be selected.

Changing link options specific to the operating system of the server

Link options specific to the operating system of the server can be changed by selecting the **Linker Options2** tab in the **Option setting** dialog box. For details on server-side link options, see the NetCOBOL User's Guide on the server side.

Item	Description
Linkage mode	Specify a linkage mode. The default is Dynamic linkage[-dy]. This option is valid only if the server operating system is Solaris OS, Linux(64) or Linux(Itanium).
Dynamic linkage[-dy]	Creates a COBOL program through dynamic linkage.
Static linkage[-dn]	Creates a COBOL program through static linkage.
Use screen form descriptor	Used to select whether the linked program uses a screen form descriptor. This item defaults to unchecked. This option is valid only if the server operating system is Solaris.
Use screen handling function	Used to select whether the linked program performs screen handling. This item defaults to unchecked. This option is valid only if the server operating system is Solaris.
Use C-ISAM	Used to select whether the linked program uses C-ISAM. This item defaults to unchecked. This option is valid only if the server operating system is Solaris.
Program invoked from C	Used to select whether the linked program is invoked by coding in the C language. This item defaults to unchecked. This option is valid only if the server operating system is Solaris.
Use C functions	Used to select whether the TARGET type is a dynamic link library and the TARGET uses the C functions. This item defaults to unchecked. This option is valid only if the server operating system is Windows(x64).
Output debugging information	Used to select whether to output debug information to the program data base file (PDB). This item defaults to check. This option is valid only if the server operating system is Windows(x64).
Link Option[-Wl]	Specifies the link options used by the ld command. This option is valid only if the server operating system is Solaris, Linux(64) or Linux(Itanium)

Note

If "THREAD (MULTI)" is specified on the project properties **Build** page **Compiler Options** tab, the -Tm option, which is used for linking a multithread model program, is automatically set.

9.4.5 Editing a makefile

A created makefile can usually be used for a build on the server side without any modification. However, the makefile can be edited.

- Editing a makefile on the local personal computer:

  The edited makefile is automatically sent to the server before it is used for a Build or a Rebuild,

- Editing a makefile on the server:

  During creation of another makefile, a confirmation message asks if you want to replace the edited makefile on the server with the makefile being created. If "No", makefile creation is canceled.

9.4.6 Creating another makefile

Even if the organization of a COBOL project is changed, another makefile is not automatically created. When performing any of the following operations and changing components of a COBOL project, create another makefile:

- Adding or renaming a COBOL source file registered in the **Source Files** folder, or deleting such a file from the folder.
- Adding or renaming a library file, descriptor file, or repository file registered in the Dependent Files folder, or deleting such a file from the folder.
- Adding a file to the Linking Files folder, deleting one from the folder, or renaming one in the folder.
- Adding or changing precompiler link information.

Follow the procedure below to create another makefile.

1. Select a project in the **Dependency view** or **Structure view**.
2. Select **Project** > **Remote Development** > **Makefile Creation...** from the menu bar, or select **Remote Development** > **Makefile Creation...** from the context menu.

The same values used for TARGET names, precompiler link information, compile options, and link options when the last makefile was created are used to create another makefile. To change the values for the current makefile, create another makefile and then change those values.

9.4.7 Sending resources

9.4.7.1 COBOL project

To create a makefile, files that are managed in the following folders of a COBOL project are required on the server side:

- COBOL source files and precompiler input sources registered in **Source Files** folder
- COBOL library files and descriptor files registered in "Dependent Files" folder

These files are automatically sent to the server when a makefile is created. The names of the files sent to the server when a makefile is created are displayed in "Create Condition" in the **Makefile Creation** dialog box.

Note

The files that are registered in the **Source Files** folder and have the extension .cbl are not sent to the server. Of the COBOL library files and descriptor files registered in the "Dependent Files" folder, only those in the project are sent. COBOL library files and descriptor files that are in other projects or managed in other folders are not sent. Also, only the library files with a .cbl extension are sent to the server.

9.4.7.2 COBOL resource project

Use the following procedures to transfer the resource included in the COBOL resource project.

1. Select the COBOL resource project in **Dependency view** or **Structure view**.

Select **Project** > **Remote Development** > **Transfer...** from the menu bar, or select **Remote Development** > **Transfer...** from the context menu. The server information and files to be transferred are displayed,

Click the **OK** button. The file is transferred.

9.5 Remote Build

A COBOL program that operates on the server can be created by executing a remote build for compilation and linkage on the server. Use a makefile created by the makefile creation function for compilation and linkage.

9.5.1 Executing a build

To execute a remote build:

1. Select a project in the **Dependency view** or **Structure view**.
2. Select **Project** > **Remote Development** > **Build** from the menu bar, or select **Remote Development** > **Build** from the context menu.

In remote build, only those resources that have been changed since the last build are the target. To build all resources regardless of whether changes have been made since the last build, use the following method:

1. To execute a rebuild for a project, select the project in the **Dependency view** or **Structure view**.
2. Select **Project** > **Remote Development** > **Rebuild** from the menu bar, or select **Remote Development** > **Rebuild** from the context menu.

Point

Compilation errors are displayed in the **Problems view**. The execution results of a remote build on the server can be viewed by clicking the **Open Console** icon on the toolbar in the **Console view**, and selecting "COBOL Remote".

9.5.2 Setting a build mode

A COBOL program that is created with remote target can be built as a release version or as a debug version by setting the proper build mode.

To change a build mode:

1. Select the project in the **Dependency view** or **Structure view**.
2. Select **Project** > **Remote Development** > **Build Debug mode** from the menu bar, or select **Remote Development** > **Build Debug mode** from the context menu. If "Build Debug" mode is selected, a debug version is created. Otherwise, a release version is created.

The release or debug state of each program that has been built is stored in the project.

Note

Even if "Do not generate DEBUG information" is specified for the TEST compile option to build the target as a debug version, the build is executed as though "Generate WINSVD DEBUG information" were specified.

9.5.3 Transfer of resources

Any files that are managed in the following folders of a COBOL project and that have been updated since the last remote build are automatically sent to the server prior to the next remote build:

- COBOL source files and precompiler input sources registered in the **Source Files** folder
- COBOL library files and descriptor files registered in the Dependent Files folder

Note

Files that are registered in the **Source Files** folder and have the extension .cbl are not sent to the server. Of the COBOL library files and descriptor files registered in the Dependent Files folder, only those in the project are sent. COBOL library files and descriptor files that are in other projects or managed in other folders are not sent. Only library files with the extension .cbl are sent to the server.

9.5.4 Correcting compilation errors

Compilation error information for a remote build is displayed in the **Problems view**. To edit COBOL source files containing compilation errors on the local personal computer side, double-click the compilation error information in the **Problems view** or select **Go To** from the context menu. The relevant COBOL source file is opened with the editor, and the current line contains the compilation error. If the COBOL source file is already open in the editor, the line containing the compilation error becomes the current line. When modification of the errors is completed, execute a remote build. The corrected files are automatically sent to the server, and the build is executed.

Point

Error information detected by the precomplier is not displayed in the **Problems view**. Error information can be checked from the **Console view** toolbar icon (**Open Console**) by selecting "COBOL Remote".

9.6 Remote Debugging

A COBOL program that has been built on the server is debugged with the remote debug function. This section explains how to start remote debugging, and provides an overview of the remote debugger connector.

The following two methods can be used to start debugging:

- Normal Debug

  In this method, the remote debugger is called from the local personal computer to start debugging. The remote debugger connector must be started on the server in advance.

- Attach Debug

  In this method, the remote debugger is called from a COBOL program that runs on the server. This start method is used for debugging a COBOL application that runs in an Interstage Application Server environment, on a Web server, or in another environment.

Note

Other debuggers (e.g. COBOL, Java) cannot be started while debugging a COBOL program.

9.6.1 Normal Debug

This section explains the procedures for starting normal debugging, including starting the remote debugger connector on the server, and starting the remote debugger.

9.6.1.1 Starting the remote debugger connector on the server

To perform remote debugging, the remote debugger connector, which monitors instructions from the debugger on a client, must be started on the server first. Start the remote debugger connector from the server-side "NetCOBOL Command Prompt".

Use the following command format to start the remote debugger connector on the server.

Server	Start command
Windows (64)	cobrds64[[port-specification] [connection-limitation-specification]]
Solaris	svdrds[[port-specification] [connection-limitation-specification]]
Linux (Itanium)	
Linux (64)	

- Port Format:

```
-p port number
```

For the port number, specify the same value detailed in NetCOBOL Studio Starting the remote debugger.

Specify a port number in the range 1024 to 65535. If omitted, it defaults to 59998.

- Connect restriction format:

```
-h host name | -s connect restriction file name [ -e ]
```

-h host name

Specify a host name or an IP address to allow connection.

-s connect restriction file name

Specify the connect restriction file.

-e

The processing result of the contents of the specified connect restriction file are displayed.

- If -s and -h are both omitted, connection will be allowed from all computers.
- After the remote debugger connector is started, the IP address and port number are displayed, and orders from the client to start debugging are monitored.

Note

- If you have to specify an environment variable for the program you want to debug, set this before you start the remote debugger connector.
- If you have to start each program that you want to debug in a different environment, you have to run the remote debugger connector in each environment. Like the environment variable, the environment here determines the operation when the remote debugger connector is started, and its settings cannot be modified later.
- Since the remote debugger connector on the server does not exit automatically when the debugger exits, end the connection when remote debugging ends. To end the remote debugger connection on the server, press the **Ctrl + C** keys in the command input window from which the remote debugger connection was started.
- If you have to start each program that you want to debug in a different environment, or if several developers debug at the same time, specify -p when the remote debugger connector starts, and then use a different port.

Example

Examples of starting the Windows(64) server remote debugger connector

- Several developers debug at the same time

```
cobrds64  -p 10001  -h client-1
cobrds64  -p 10002  -h client-2
cobrds64  -p 10003  -h client-3
```

  - This is the case when three developers use the remote debugger at the same time.
  - Choose the port number to be used by each developer and specify this using -p.
  - Specify -h to accept only the specific computer's request for remote debugging and to reject incorrect connections by other developers.

Format of the connect restriction file

Specify the format of the connect restriction file as shown below:

```
 [ ALLOW={ ALL | connection-target [connection-target ...] } ]
[ DENY={ ALL | connection-target [connection-target ...] } ]
```

- This allows connection from the connection target specified in ALLOW. If this is omitted, it is assumed that ALL has been specified.
- This denies connection from the connection target specified in DENY. If this is omitted, it is assumed that ALL has been specified.
- If you specify ALL, connection is allowed or denied from all computers.
- If ALLOW and DENY are both specified, and ALLOW is described, ALLOW has priority.
- If this is register on several lines, write "\" at the end of the line.

Example

```
ALLOW=192.168.0.1 192.168.0.3 \
192.168.0.8-192.168.0.10
```

The connection target format is shown below.

Table 9.3 Connection target format (IPv4)

Format	Example	Example of IP address application range
IP address	192.168.0.1	192.168.0.1
Range	192.168.0.1-192.168.0.10	192.168.0.1 to 192.168.0.10
Wildcard (*1)	192.168.0.*	192.168.0.0 to 192.168.0.255
Host name (*2)	Hostname	192.168.0.1

1: You can specify "" instead of the decimal parts (8-bit) delimited by "."

*2: This example assumes that a host name called "Hostname" has been assigned for an IP address of 192.168.0.1.

Table 9.4 Connection target format (IPv6))

Format	Example	Example of IP address application range
IP address	fe80::1:23:456:789a	fe80::1:23:456:789a
Range	fe80::1:23:456:1-fe80::1:23:456:789a	fe80::1:23:456:1 to fe80::1:23:456:789a
Wildcard (*3)	fe80::1:23:*:789a	fe80::1:23:0:789a to fe80::1:23:ffff:789a
Host name (*4)	Hostname	fe80::1:23:456:789a

3: You can specify "" instead of the hexadecimal parts (16-bit) delimited by ":"

*4: This example assumes that a host name called "Hostname" has been assigned for an IP address of fe80::1:23:456:789a.

Note

If there is an error in the contents of the connect restriction file, the specified details will be invalid, and connection will be allowed from all computers.

Windows firewall

If the server-side OS is Windows(x64), the remote debugger connector cannot be used if the "Windows firewall" is enabled. Therefore, use the"9.3.3 Exception registration of Windows firewalll" register "cobrds64.exe" as a Windows firewall check exception:

9.6.1.2 Starting the remote debugger

To start the remote debugger:

1. Select a COBOL project from the **Dependency view** or **Structure view**.

2. Select **Run** > **Debug Configurations...** from the menu bar, or click ▾ of 🐞▾ on the toolbar and select **Debug Configurations...**. The **Debug Configurations** dialog box appears.
3. Select **Remote COBOL Application** in the left pane.
4. Click 📄 at the top of the left pane. The **startup configuration settings** page is displayed in the right pane.
5. The default startup configuration name is displayed in the "Name" field. The startup configuration name can be changed.
6. Click the **Main** tab, check each setting item, and change it as necessary.

item	Description
Project name	The selected project name is displayed.
Debug Style	Select Normal Debug.
Server name	The server name that is set in the project properties is displayed.
Port number	Specify the port number used for communication with the server during debugging. Specify the same port number as that specified when the remote debugger connector started on the server. The initial setting value of the remote debugger connector on the server is 59998.
Executable file	Specify the executable file to be debugged. The initial setting value is either of the following, depending on whether there is a makefile: - With a created makefile: Target name specified in the makefile - Without a created makefile: Target name of the project If the debug target is a dynamic link library, specify the target file.
Workspace Directory	Specify the work directory of the application to be debugged. The initial setting value is the directory of the executable file.
Argument(s)	Specify program arguments in the format used for the command line interface.

7. Start debugging by selecting "Debug". Once debugging is started with a particular configuration, the startup configuration is registered as a shortcut in the toolbar and you can restart debugging by selecting **Run** > **Debug History** or using the shortcut.

Note

If no information for remote development is set in the project, "Normal Debug" cannot be selected for "Debug Style".

Point

The debugger can be started with the default settings by selecting the project in the **Dependency view** or **Structure view**, and then selecting **Run** > **Debug As** > **Remote COBOL Application** from the menu bar.

9.6.2 Attach Debug

The remote debugger can be used for attachment-type remote debugging. In the attach format, once the remote debugger is started on the local PC, it stays in the standby state until a COBOL application is executed on the server. Remote debugging is started when a COBOL application is executed and the environment variable for requesting the start of attachment debugging has been set on the server.

Note

When multiple NetCOBOL Studio is a standby state of attached debugging, attached debugging is not supported.

9.6.2.1 Starting the remote debugger

1. Select a COBOL project from the **Dependency view** or **Structure view**.

2. Select **Run** > **Debug Configurations...** from the menu bar, or click ▾ of ▾ on the toolbar and select **Debug Configurations...**. The **Debug Configurations** dialog box appears.

3. Select **Remote COBOL Application** in the left pane.

4. Click at the top of the left pane. The **startup configuration settings** page is displayed in the right pane.

5. The default startup configuration name is displayed in "Name". The startup configuration name can be changed.

6. Click the **Main** tab, check each setting item, and change it as necessary.

item	Description
Project name	The selected project name is displayed.
Debug Style	Select Attach Debug.
Debug Information Directory	If the debugging information file is placed in a directory that is different from the server directory (Server directory of **Remote Development** page of project property), specify the directory path. Multiple directories can be specified as follows: - For Solaris or Linux server, connect directory paths with ":" (colon). - For Windows server, connect directory paths with ";" (semicolon).

7. Start the debugger by selecting "Debug". The debugger stays in the standby state until an application to be debugged is reported to have started. Once debugging is started with a particular configuration, the startup configuration is registered as a shortcut in the toolbar and you can restart debugging by selecting **Run** > **Debug History** or using the shortcut.

Note

If the server operating system is Solaris or Linux and the debug target is a COBOL application that uses a dynamic link library stored in a directory other than the one containing the executable file, start debugging after copying the debugging information file (extension "svd") of the dynamic link library to the directory containing the executable file.

Point

The debugger can be started with the default settings by selecting the project in the **Dependency view** or **Structure view**, and then selecting **Run** > **Debug As** > **Remote COBOL Application** from the menu bar.

9.6.2.2 Remote debugger connector

After the remote debugger connector starts, the following icon is displayed in the task tray, and the settings are the same as those the previous time the remote debugger connector was started. The order from the server to start debugging is monitored.

Note

When the debugger ends, the remote debugger connector is not automatically closed. To close the remote debugger connector, right-click the task tray icon, and then select Exit from the displayed menu.

Remote Debugger Connector dialog box

Parameter		Description
Connect Information	Network Condition:	IP address of the machine or hostname where the remote debugger connector is activating, and the port number monitored by the remote debugger connector are displayed. When you select Change Port Number, Change Port Number dialog box opens.
Connect Restriction	Connect Allowed Target:	Set the connect-allowed targets. If you select Allow All the Connection, all computers are allowed to connect. Allow Target displays the list of the connect-allowed targets. Using each button, Add, Edit and Remove, you can modify the list of connect-allowed targets.
	Connect Denied Target:	Set the connect-denied target. If you select Deny All the Connection, all computers are denied to connect. Deny target displays the list of the connect-denied targets. Using each button, Add, Edit and Remove, you can modify the list of the connect-denied targets.

9.6.2.3 Restricting connection to computers that allow remote debugging

The way to restrict connections to computers that allow remote debugging is shown in table below. This table uses the IPv4 address. The IPv6 address can also be used.

In this table, the server remote debugger connector is expressed by "server", and the client remote debugger connector is expressed by "client".

The connect restriction file is used in the server. For details, refer to "Format of connect restriction file".

In the client, make the settings in the **Connect Restriction** page of the **Remote Debugger Connector** dialog box. For details, refer to "Remote Debugger Connector dialog box. "

Table 9.5 Typical examples of restricting connection

Format	Server	Client

This allows connection for only a specific target.
The IP address to be allowed connection is as follows:
192.168.0.1
	ALLOW=192.168.0.1
	DENY=ALL
		Setting for "Connect Allowed Target":
		- Select "Allow Target".
		- Set as follows in the edit box:
		`192.168.0.1`
		Setting for "Connect Denied Target":
		- Select "Deny All the Connection".

This allows connection for several specific targets.
The IP addresses to be allowed connection are as follows:
192.168.0.1
192.168.0.2
192.168.0.10
	ALLOW=192.168.0.1 \\
	192.168.0.2 192.168.0.10
	DENY=ALL
		Setting for "Connect Allowed Target":
		- Select "Allow Target".
		- Set as follows in the edit box:
		`192.168.0.1`
		`192.168.0.2`
		`192.168.0.10`

Format	Server	Client
		Setting for "Connect Denied Target": - Select "Deny All the Connection".

This allows connection for a specific range.
The range of IP addresses to be allowed connection is as follows:
Lower limit: 192.168.0.1
Upper limit: 192.168.0.10
	ALLOW=192.168.0.1-192.168.0.10
	DENY=ALL
		Setting for "Connect Allowed Target":
		- Select "Allow Target".
		- Set as follows in the edit box:
		`192.168.0.1-192.168.0.10`
		Setting for "Connect Denied Target":
		- Select "Deny All the Connection".

This allows connection in octet units.
The range of IP addresses to be allowed connection is as follows:
Lower limit: 192.168.0.0
Upper limit: 192.168.0.255
	ALLOW=192.168.*
	DENY=ALL
		Setting for "Connect Allowed Target":
		- Select "Allow Target".
		- Set as follows in the edit box:
		`192.168.0.*`
		Setting for "Connect Denied Target":
		- Select "Deny All the Connection".

This allows connection for all computers.	ALLOW=ALL
	DENY=ALL
		Setting for "Connect Allowed Target":
		- Select "Allow All the Connection".
		Setting for "Connect Denied Target":
		- Select "Deny All the Connection".

9.6.2.4 Executing an application on the server

On the server, set the environment variable required for reporting the start of execution of a debug target application to the remote debugger as shown in the table below. Specify the IP address or host name of the local PC for "connection-destination" and specify its port number. For details on how to set the environment variable, see the NetCOBOL User's Guide.

COBOL application

Server	Environment variable
Windows(x64)	@CBR_ATTACH_TOOL= *connection-destination*/STUDIO [path-list]
Solaris	CBR_ATTACH_TOOL= *connection-destination*/STUDIO [path-list]
Linux(Itanium)	
Linux(x64)	

- For "connection-destination", specify the client side port number for the remote debugger connector and the operating computer in the following format:

```
{IP address | host name}[:port number]
```

Specify the IP address in IPv4 or IPv6 format.

Specify a port number in the range 1024 to 65535. If the port number is omitted, it defaults to 59999.

The IPv6 format can specify the scope address. The scope address specifies the scope identifier after the address.

If a port number is specified in an IPv6 address, enclose the address part with "[ ]".

Example:

- Specification with IPv4 address (192.168.0.1) and port number (2000)

```
@CBR_ATTACH_TOOL=192.168.0.1:2000/STUDIO
```

- Specification with IPv6 address (fe80::1:23:456:789a) and port number (2000)

```
@CBR_ATTACH_TOOL=[fe80::1:23:456:789a]:2000/STUDIO
```

- Specification with IPv6 address (fe80::1:23:456:789a) and scope identifier (%eth0)

```
@CBR_ATTACH_TOOL= fe80::1:23:456:789a%eth0/STUDIO
```

- Specification with IPv6 address (fe80::1:23:456:789a), scope identifier (%5) and port number (2000)

```
@CBR_ATTACH_TOOL=[fe80::1:23:456:789a%5]:2000/STUDIO
```

- Specification with host name (client-1) and port number (2000)

```
@CBR_ATTACH_TOOL=client-1:2000/STUDIO
```

- Specification with port number omitted (IPv4 address)

```
@CBR_ATTACH_TOOL=192.168.0.1/STUDIO
```

- Specification with port number omitted (IPv6 address)

```
@CBR_ATTACH_TOOL=fe80::1:23:456:789a/STUDIO
```

An IPv6 address cannot be specified if the server OS is Linux(Itanium).

Note

For the Linux (Itanium) operating system, the IPv6 address cannot be specified.

Remote debugging corresponds to the IPv6 address within the following scopes.

Table 9.6 For IPv6 of remote debugging

item	Specification
Network	It corresponds to the following environments. - IPv4/IPv6 Dual stack - IPv6 only.
IPv6 address	The following IPv6 addresses are used. - Aggregatable Global Unicast Address (GUA) - Unique Local Unicast Addresses (ULA) - link local address (LLA)
IPv4/IPv address selection	In the IPv4/IPv6 dual stack environment, IPv6 is used by priority. The following are not supported - Use limited (IPv4 only or IPv6 only) - Switch of priority - When failing in the connection by IPv6, it connects it by IPv4. (fallback)
IP address notation	The following not supported 1. Specifying a port number ([]) 2. The notation hex.

item	Specification
	3. Recommended notation of IPv6 addresses (RFC5952) 4. Uppercase letters section 5. Lowercase letters section 6. Mixed-case notation of letters section In the following cases, use the IPv4 address notation - IPv4/IPv6 Dual stack operating system. - Using of IPv4 and IPv6 together

Note

Privacy Extensions are not supported. When an IP address is changed, it is necessary to restart a remote debugger connector and a remote debugger.

- For path-list, specify the directory that contains the debugging information file. The debugging information is retrieved in the following order, and is used for debugging.

    1. The specified order of the additional path list (delimit by semicolon ";" and enter the full path name when you specify two or more folders).
    2. The current folder when the COBOL program began to run.
    3. The folder that contains the COBOL program.

Note

2. and 3. do not need to be added to PATHLIST.

Canceling the standby state

To stop debugging with the remote debugger in the standby state after it was started in the attach format, click in the bottom right of the window, and then click ■ in the displayed **Progress view**.

Attached debugging for a COBOL application on the client

Attached debugging can also be performed for a COBOL application built on the client. To perform attachment debugging on the client, set an environment variable on the client. This variable is used to report the start of an application to the debugger.

To perform attachment debugging on the client:

1. Set the environment variable shown below on the local PC. This variable is used for reporting the start of an application to the debugger.

```
@CBR_ATTACH_TOOL=localhost/STUDIO
```

2. Select a COBOL project from the **Dependency view** or **Structure view**.
3. Select **Run** > **Debug Configurations...** from the menu bar, or click ▼ of ▼ on the toolbar and select **Debug Configurations...** . The **Debug Configurations** dialog box appears.
4. Select **Remote COBOL Application** in the left pane.
5. Click at the top of the left pane. The **startup configuration setting** page is displayed in the right pane.
6. The default startup configuration name is displayed in "Name". The startup configuration name can be changed.

7. Click the **Main** tab, check each setting item, and change it as necessary.

item	Description
Project name	Displays the selected project name.
Debug Style	Select "Attach Debug".
Debug Information Directory	Specify the storage folder for the debugging information file. When it is stored in the folder displayed by the Location on the **Resource** page (*), it is omissible. If you specify more than one directory, the folder is delimited by the semicolon (;).

*: The resource page is in the properties dialog box of the project.

8. Start the debugger by selecting "Debug". The debugger stays in the standby state until an application to be debugged is reported to have started.
9. Execute a COBOL application to start debugging.
10.

9.7 Notes on using the remote development function

Use the version of NetCOBOL Studio that corresponds to the version of the NetCOBOL product that is installed on the server.

If the version of NetCOBOL on the server is newer than the version of NetCOBOL Studio, some of the server side NetCOBOL functions might not be available for use. For example:

- Newly changed or added compiler options cannot be specified by the **Compiler Option** dialog.
- Newly added reserved words are not recognized as reserved words in the COBOL editor.
- When remotely debugging, the message "An error detected in the debugger." is displayed.

Chapter 10 Differences between Eclipse 3.4 and Eclipse 4.3

Some menu names and window names of NetCOBOL Studio on Eclipse4.3 are different from that on Eclipse3.4. See "10.1 Comparing table of display name in NetCOBOL Studio between Eclipse bases".

10.1 Comparing table of display name in NetCOBOL Studio between Eclipse bases

The place displaying (*1)	Display name in NetCOBOL Studio on Eclipse 4.3	Display name in NetCOBOL Studio on Eclipse 3.4
The page on the property dialog box(*1)	"Task Tags"	none
	"Builders"	none

*1: Select "Property" from the context menu in **Dependency view** or **Structure view**. The property dialog box appears.

Chapter 11 CORBA Development Support Function

The CORBA development support function is used when the plug-in (in the following text, it will be called COBOL plug-in) for Interstage Studio Standard-J Edition, which is included in the NetCOBOL development package, is inserted into the workbench (in the following text, will be called Interstage Studio workbench) in Eclipse 3.4 base, which is offered separately.

When reading this user's guide, refer to the following manuals about Interstage Studio and Interstage Application Server.

Interstage Studio Documents

- Interstage Studio General Description
- Interstage Studio User's Guide
- Installation Guide

Interstage Application Server Documents

- Interstage Application Server Installation Guide
- Interstage Application Server Operatior's Guide
- Interstage Application Server Tuning Guide
- Interstage Application Server Distributed Application Development Guide (CORBA)
- Interstage Application Server Reference Manual (Command Edition)
- Interstage Application Server Messages

11.1 Overview

This topic describes the functions that are provided as the CORBA development support function and the COBOL Plug-in.

11.1.1 What is the COBOL-Plug-in?

The COBOL plug-in is the Add-on component that provides the development environment functions of the COBOL applications and CORBA applications in the Interstage Studio Workbench. The COBOL plug-in can provide the following functions.

- COBOL Perspective
- Wizards for Creating COBOL Related Resources

The following table shows the resources that can be created and the relationship of the wizard.

Wizards	Generation Resources category	Generation Resource	**Descriptions**
COBOL project wizard	Project	COBOL project	Creates the COBOL project.
CORBA server project wizard		CORBA server project	Creates the CORBA server project.
COBOL source generation wizard	Source	COBOL source file	Creates the COBOL source.
COBOL library generation wizard		COBOL library file	Creates the COBOL library.
CORBA server application generation wizard		CORBA server application	Create the IDL files (Interface definition files) which are required to create the CORBA server application and the model of server application.
CORBA stub file wizard		CORBA stub file	Create the stub file using the IDL file created at the time of developing the CORBA server application, and then

Wizards	Generation Resources category	Generation Resource	**Descriptions**
			register into the project of the client application.
IDL file wizard		IDL file	Creates a client interface (IDL) file that describes the structure according to the object-oriented COBOL Mapping of CORBA / IDL
Object -oriented COBOL source generation wizard		Object-oriented COBOL source file	Creates the object-oriented COBOL source.

- Editor
    - COBOL Editor
    - IDL Editor
- View
    - **Outline view**
    - **Template view**
    - **Dependence view**
    - **Structure view**
    - **Watch view**
- COBOL Builder
- COBOL Debugger
- CDCORBA Class

In the CORBA client application development, provides the CDCORBA class in order to briefly describe the CORBA initialization process and the object search process.

Application that can support the development according to the COBOL Plug-in

Interstage Studio, which the COBOL plug-in has been inserted into, supports the development in GUI of the following applications.

- COBOL Applications

  Applications such as CORBA client according to the COBOL language description can be developed.

  For the development methods of COBOL client applications, refer to "11.6 Development of CORBA Client Application".

- CORBA Server Applications

  According to the object-oriented COBOL language, CORBA server applications can be developed.

  For the develop methods of CORBA server applications according to the object-oriented COBOL language, refer to "11.5 Development of CORBA Server Application".

11.1.2 What is CORBA application?

CORBA is a specification of the object-oriented technology provided by OMG (Object Management Group: a nonprofit organization established with the aim of the standardization and the spread of object-oriented technology in 1989). The following functions are being offered as CORBA specifications.

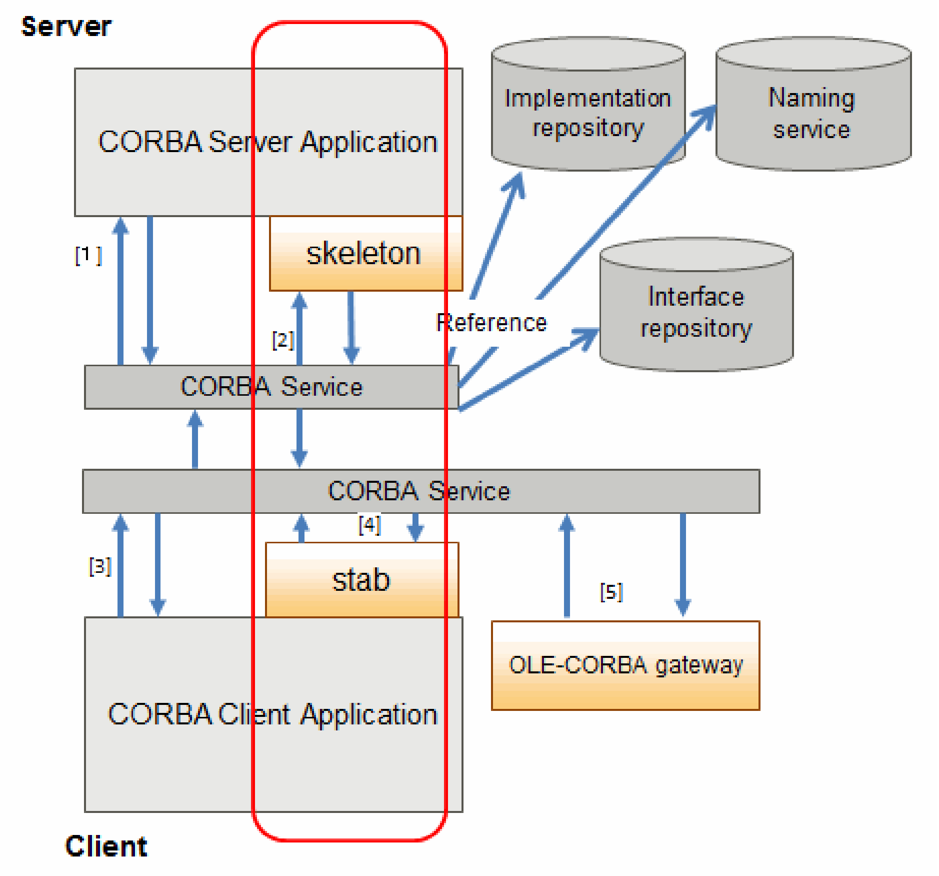
- The mechanism that generates the stub and the skeleton from IDL(Interface Definition Language). Stub and skeleton are the communication library between server/application.
- API used for communication between client / server application

- Protocol for communication between client/server application that operates on different kinds of machine (IIOP :Internet Inter-ORB Protocol).

Fujitsu is providing the distributed communication base and service in accordance with CORBA in Interstage Application Server. In Interstage Studio, the CORBA application means the application made by using Interstage Application Server. Moreover, both the CORBA server application and the CORBA client application can be made in the Interstage Studio workbench where the COBOL plug-in is built in.

CORBA application operation form

The figure below shows the operation form of the CORBA application. The operation form supported by the COBOL plug-in is shown with a red frame.



"Server application"

The server application can be roughly divided into the following two types according to the method of disclosing interface information to the client.

"1" Dynamic skeleton interface

"2" Static skeleton interface

"Client application"

The client application can be roughly divided into the following three types according to the method of calling the server application.

"3" Dynamic Invocation Interface

"4" Static Invocation Interface

"5" OLE-CORBA gateway

Feature and range of support for CORBA application

The feature of each CORBA application and the range of support in the COBOL plug-in are shown below.

Application type	Description	Support	
		Server	Client
Dynamic interface "1""3"	The stub file and the skeleton file are unnecessary, information is taken out of the interface repository, and the parameter is assembled in the program to call the method of the server.	not supported	not supported
Static interface "2""4"	Links the stub and the skeleton file made from the IDL file into the application. The stub and the skeleton convert the data used between the server and the client from CORBA communications infrastructure protocol (IIOP) into each language type.	support(*1)	support(*2)
OLE-CORBA gateway	Functions of the server application are called by the OLE access of the Windows system.	N/A	N/A

*1: The server application is made in object-oriented COBOL language.

*2: The client application is made in object-oriented COBOL language or COBOL language.

- Because the Dynamic Interface dynamically assembles the interface of the server application, it's not always necessary to change the program when the interface has changed slightly. From this respect, maintainability is good.
- Because the Static Interface accesses repository information on the server less frequent than the Dynamic Interface, the static interface has excellent performance.
- Because the OLE-CORBA gateway has the processing internally similar to the dynamic interface, the performance is the same as the dynamic interface. Also, very little coding is required.

11.2 Tutorial

This section explains how to develop CORBA server applications and CORBA client applications.

11.2.1 Environment Preparation

The following programs are necessary for developing CORBA applications.

- Interstage Studio Workbench (*1)
- COBOL Plug-in (*2)
- Interstage Application Server Function or Client Function (*3)
- NetCOBOL Development Package

*1: Program provided by Interstage Studio Standard-J Edition, which is sold separately.

*2: Point to the "COBOL plug-in for Interstage Studio" which is contained in the NetCOBOL development package.

*3: Program provided by Interstage Application Server Enterprise Edition, which is sold separately.

In addition, the following preparation is necessary.

- Confirmation of environment setting about Interstage Application Server
- Start of Interstage basis service

Refer to "11.3 Preparation for Developing CORBA Applications", and perform the preparation.

11.2.2 Developing CORBA Server Applications

This section explains how to develop CORBA server applications.

Contents of Program Created

In this tutorial, you will create the following type of program:

Binary program

This program performs arithmetic operations with two parameters.

- Methods named addop, subop, mltop, and divop are used for addition, subtraction, multiplication, and division, respectively.
- Without returning a return value, each method places the parameters in a structured format and sets result of the operation into the structure so that clients can refer to it. (The parameter mode must be inout because the parameters are specified by a client and each result of the operation is set by a server application.)
- Each of the multiplication and division methods checks the parameters and, if one of them is 0, returns an exception.

Developing procedure

This section explains how to develop CORBA server applications.

1. Starting the Interstage basis service

2. Creating a CORBA Server Project

3. Editing Programs

4. Building the Project

5. Running the Program

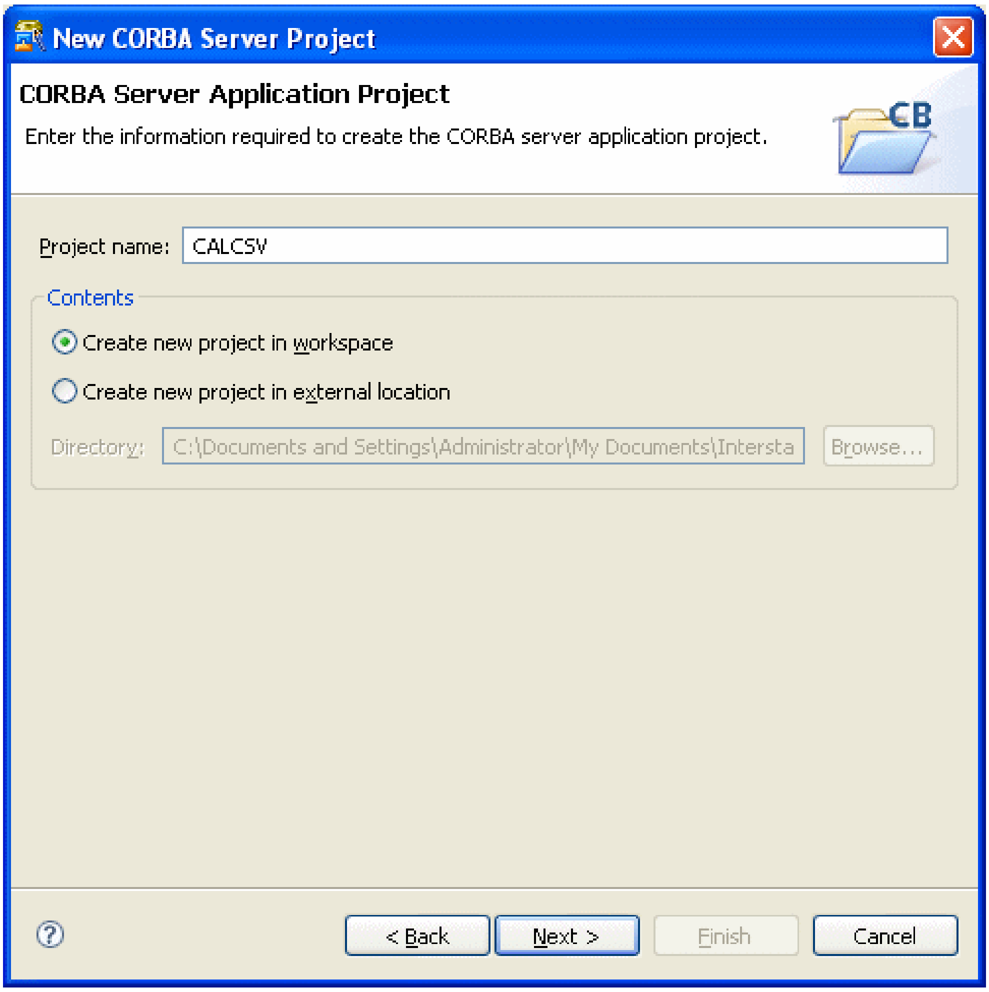
1. Starting the Interstage basis service

Before creating CORBA applications, it is necessary to start the J2EE execution environment service . Because the J2EE execution environment service is not started by default, use the Interstage basis service operation tool to start the service. Refer to "Checking the environment settings of Interstage Application Server".

2. Creating a CORBA Server Project

1. Start the Interstage Studio Workbench.
2. If the COBOL perspective is not displayed, then follow the procedure below to display it:
    a. Select **Window** > **Open Perspective** > **Other** from the menu bar. The **Open Perspective** dialog box is displayed.
    b. Select **COBOL**, then click the **OK** button.
3. Select **File** > **New** > **CORBA Server Project** from the workbench - a new CORBA Server project is displayed.

4. Enter the project name and the save folder.



New CORBA Server Project

CORBA Server Application Project

Enter the information required to create the CORBA server application project.

Project name: CALCSV

Contents

● Create new project in workspace

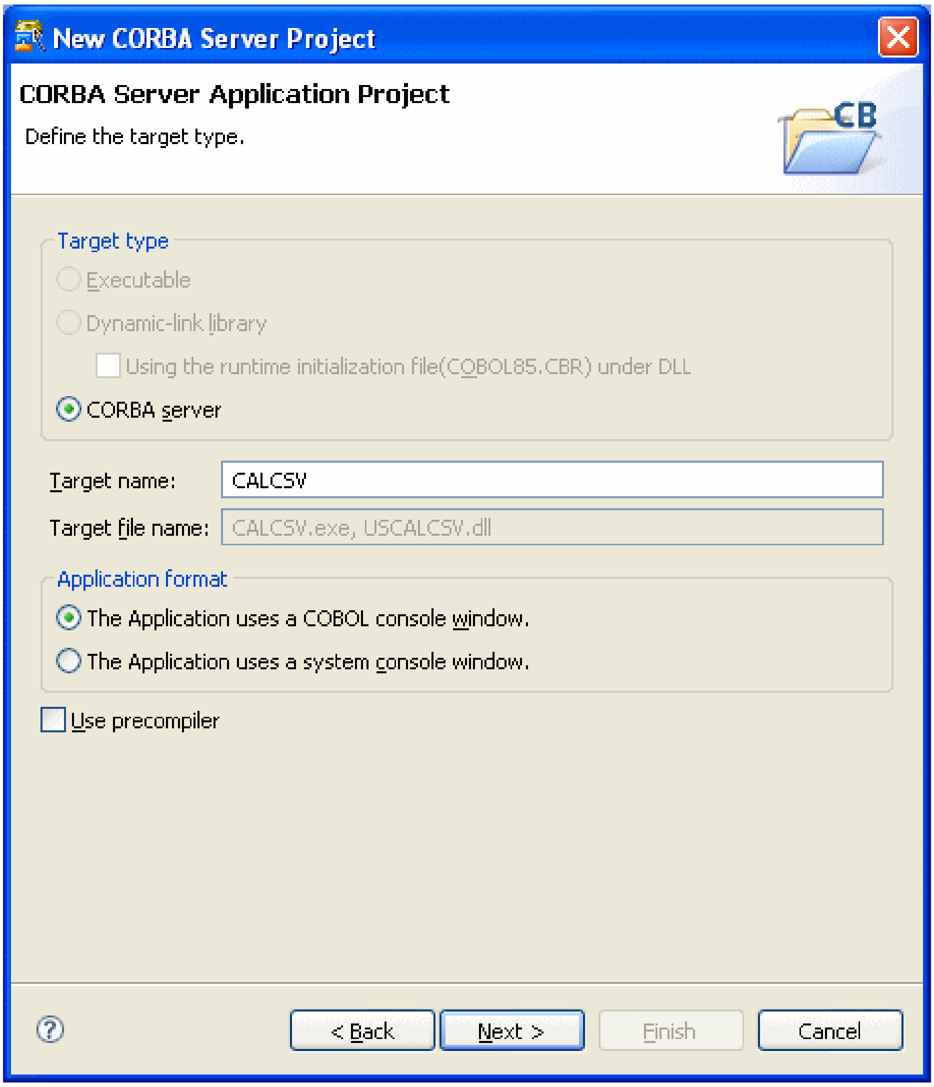
○ Create new project in external location

Directory: C:\Documents and Settings\Administrator\My Documents\Intersta

Setting item	Setting value
Project name	CALCSV
Contents	Select "Create new project in workspace"

Click the **Next** button.

5. You can specify the target information on this screen. Proceed without specifying any changes here.



New CORBA Server Project

CORBA Server Application Project

Define the target type.

Target type

○ Executable

○ Dynamic-link library

□ Using the runtime initialization file(COBOL85.CBR) under DLL

● CORBA server

Target name: CALCSV

Target file name: CALCSV.exe, USCALCSV.dll

Application format

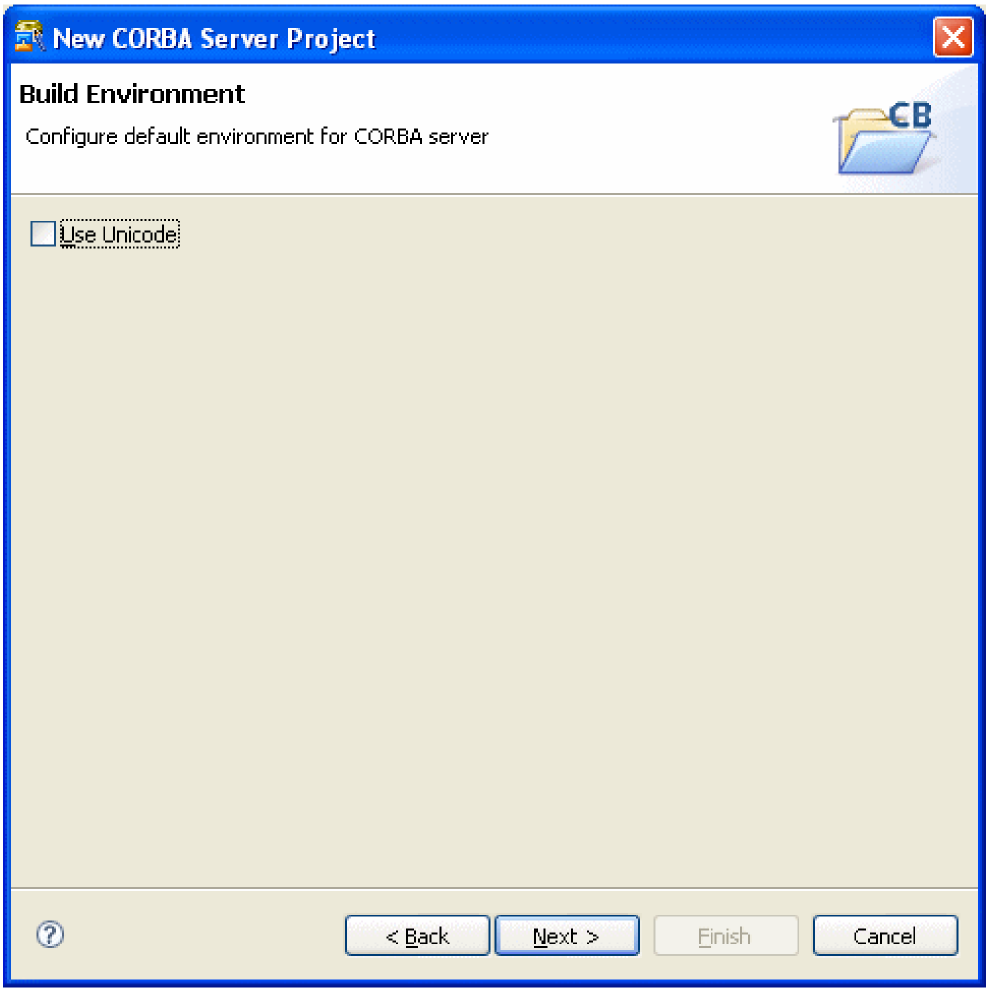
● The Application uses a COBOL console window.

○ The Application uses a system console window.

□ Use precompiler

Click the **Next** button.

6. You can specify the target information on this screen. Proceed without specifying any changes here.



New CORBA Server Project

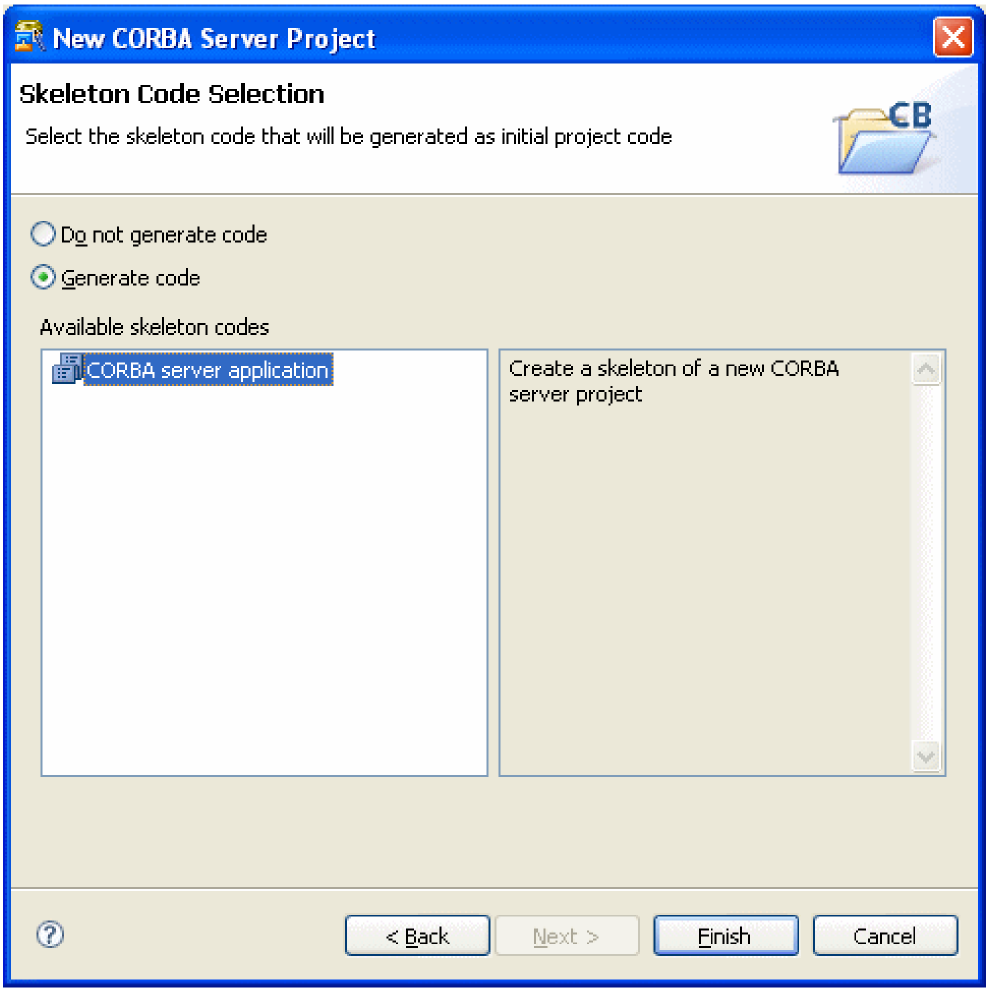
Build Environment

Configure default environment for CORBA server

□ Use Unicode

Click the **Next** button.

7. You can select the option "Generate code". Make sure that "CORBA server application" is selected as the code generation wizard, and click the **Finish** button



New CORBA Server Project

Skeleton Code Selection

Select the skeleton code that will be generated as initial project code

○ Do not generate code

● Generate code

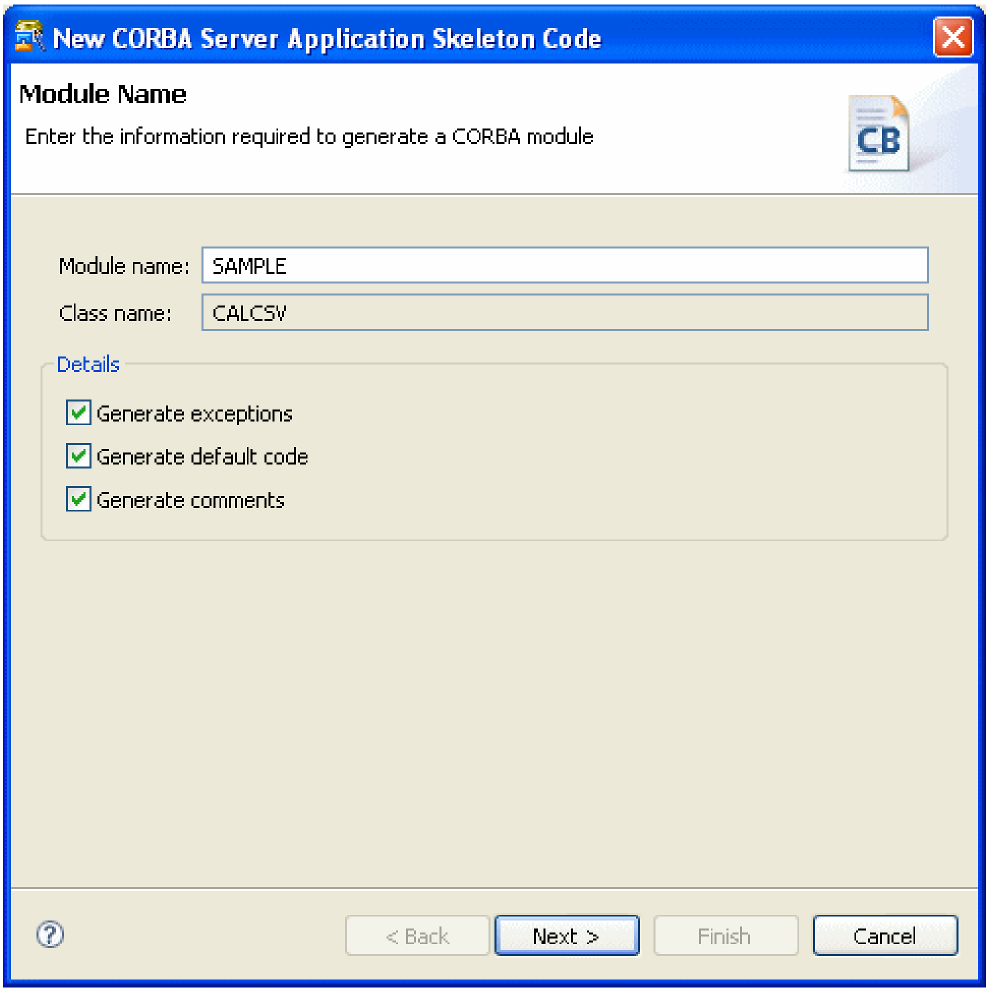
Available skeleton codes

CORBA server application

Create a skeleton of a new CORBA server project

The **New CORBA Server Application Skeleton Code wizard** is started after the project is created.

8. On this screen, specify the following settings. Click the **Next** button.



New CORBA Server Application Skeleton Code

Module Name

Enter the information required to generate a CORBA module

Module name: SAMPLE

Class name: CALCSV

Details

☑ Generate exceptions

☑ Generate default code

☑ Generate comments

Setting item	Setting value
Module name	SAMPLE(any)
Class name	CALCSV(project name, fixed)
Generate exceptions(*1)	Check
Generate default code (*2)	Check
Generate comments (*3)	Check

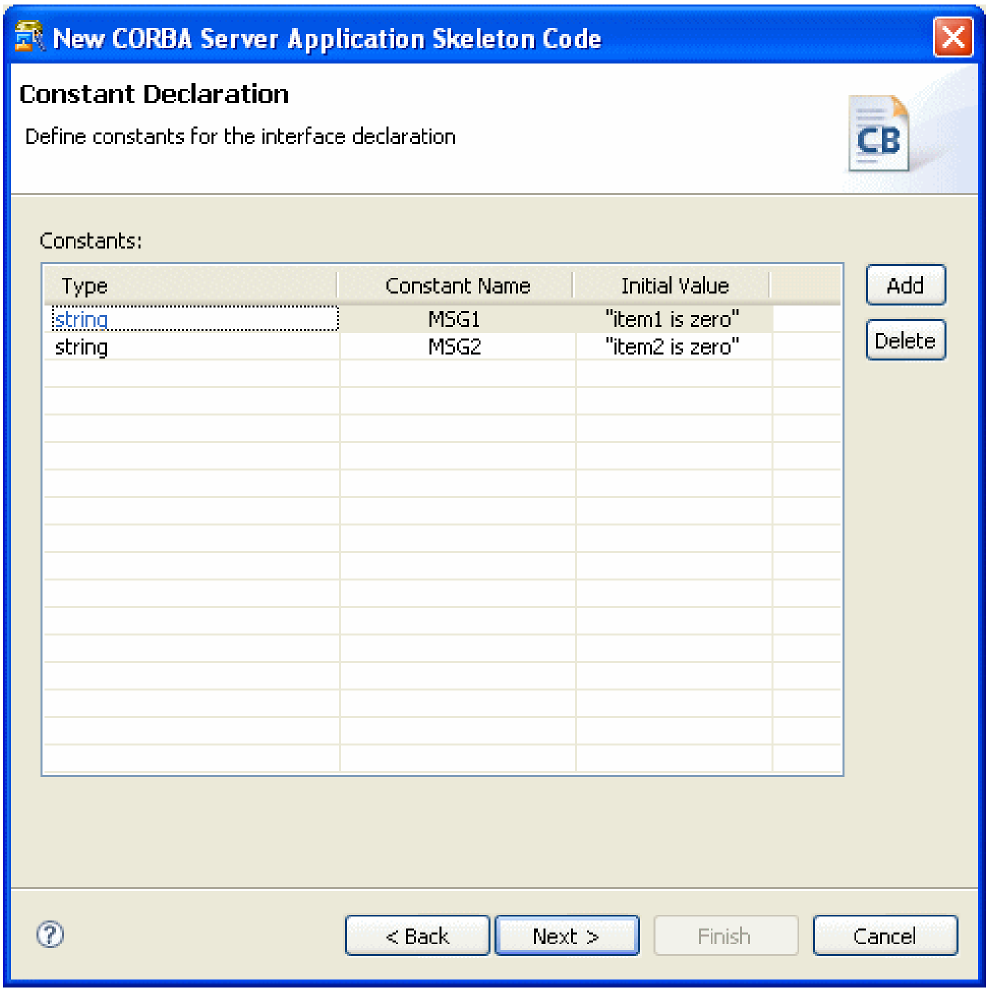
*1: For multiplication and division exception, check "Generate exceptions". If it is checked, the following exception declaration is generated in IDL.

```
exception CDException{
    string CDExceptionMsg;
    long CDExceptionCode;
};
```

*2: If "Generate default code" is checked, a definition of a method to be implemented as a server application and the code for that method is generated. Otherwise, only a method declaration is generated.

*3: If "Generate default code" is checked, a definition of a method to be implemented as a server application and the code for that method is generated. Otherwise, only a method declaration is generated. If "Generate comments" is checked, a comment is generated in the template source created.

9. On this second screen, you can declare constants. In this tutorial, define two constants in such a way that exceptions are generated and error messages are thrown. Click the **Next** button.



Constant Declaration

Define constants for the interface declaration

Constants:

Type	Constant Name	Initial Value
string	MSG1	"item1 is zero"
string	MSG2	"item2 is zero"

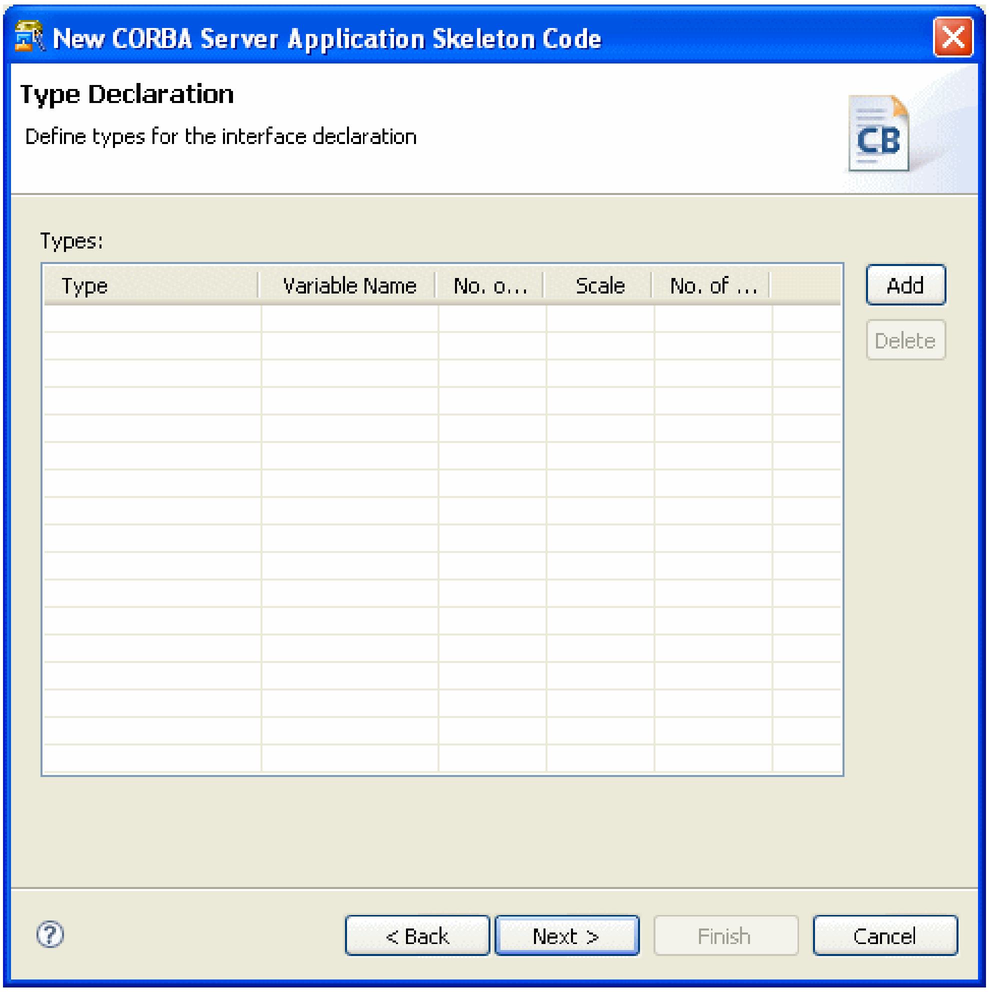
- Constant1

Setting item	Setting value
Type	string
Constant name	MSG1
Initial value	"item1 is zero"

- Constant2

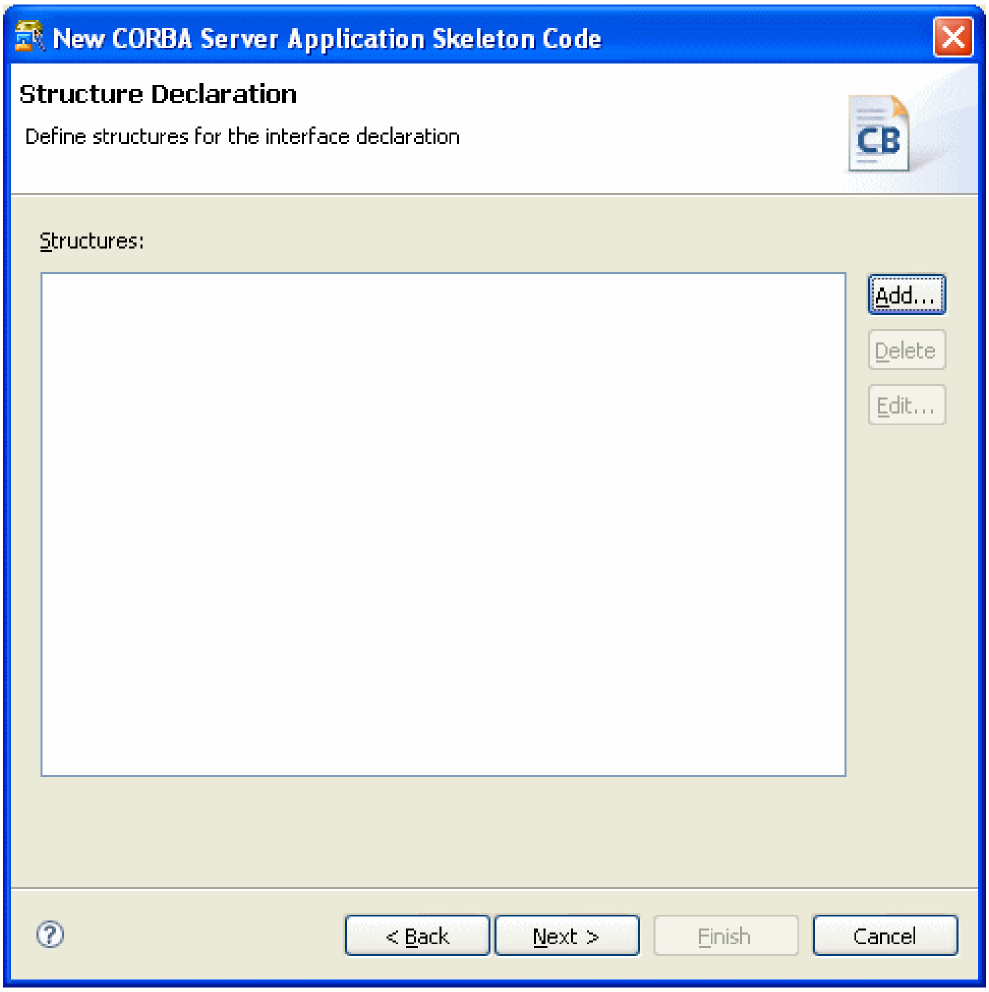
Setting item	Setting value
Type	string
Constant name	MSG2
Initial value	"item2 is zero"

10. On this screen make a type declaration. Here, you can define repetition items, but no such definition is required in this tutorial.



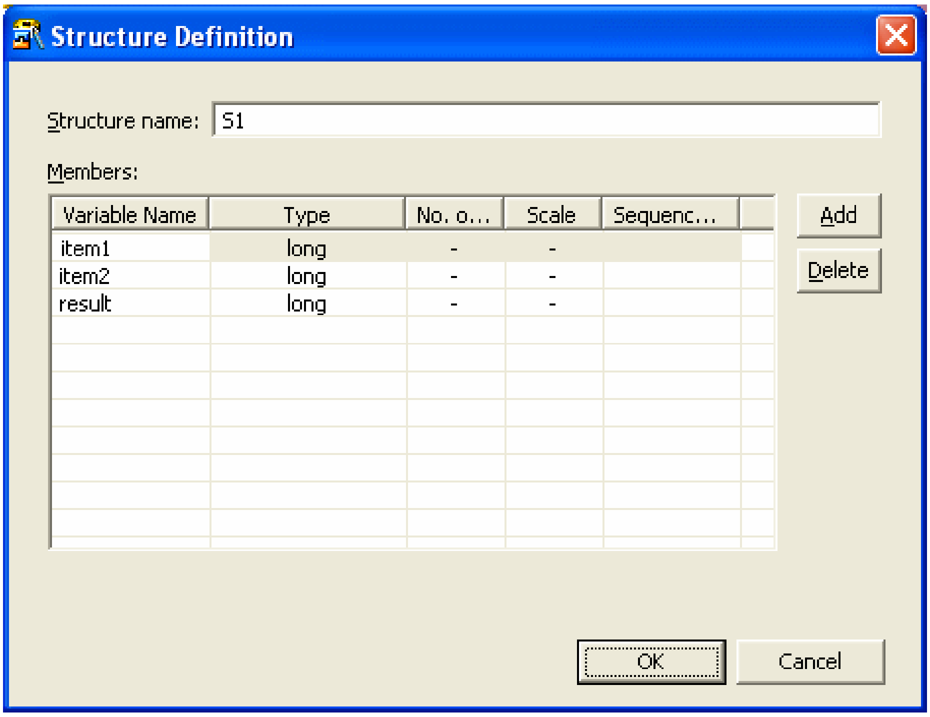
Proceed without making any changes on this screen. Click the **Next** button.

11. Here, you can declare a structure. This sample program requires two input areas from a client and one area for results of operation. Declare these areas as one structure.


New CORBA Server Application Skeleton Code
Structure Declaration
Define structures for the interface declaration
Structures:
Add...
Delete
Edit...
< Back
Next >
Finish
Cancel

On this screen, click the **Add** button to display the **Structure Definition** dialog, and declare the structure in this dialog.

12. Here, declare the structure "S1", which consists of the following members.



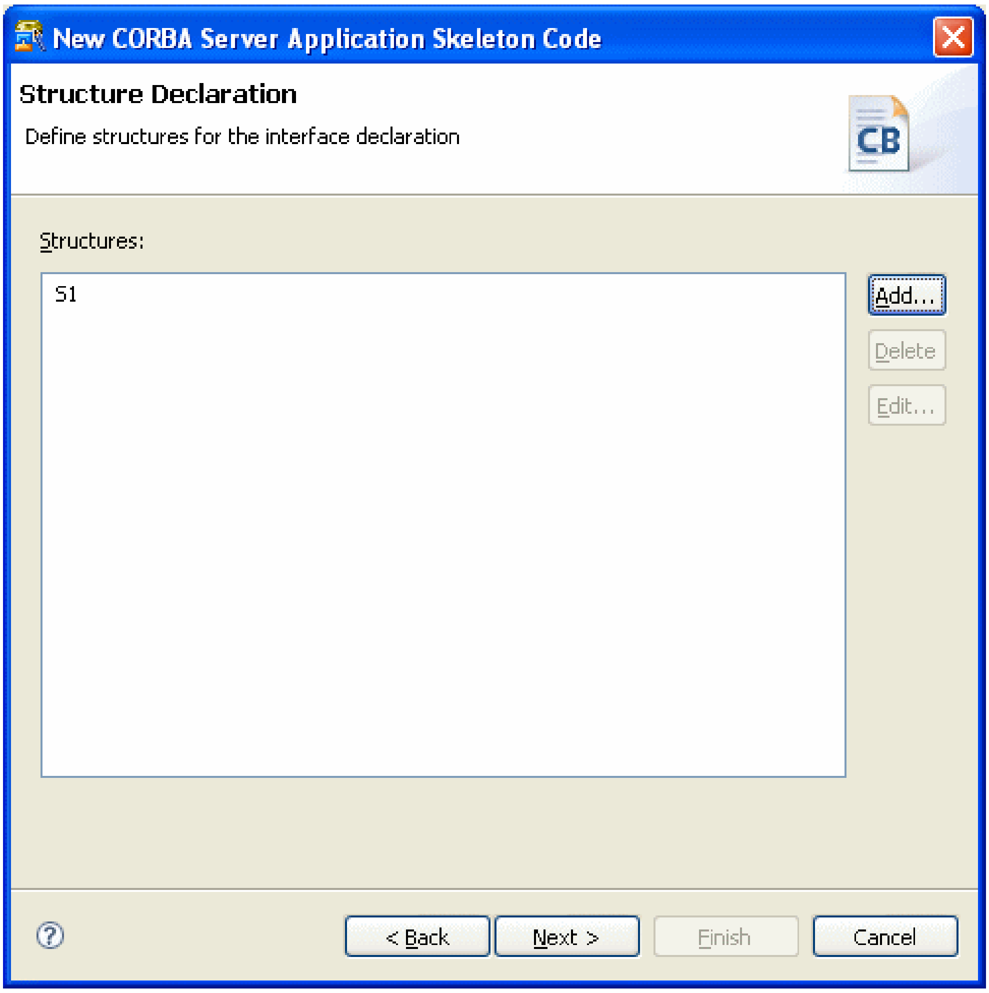
Structure name: S1

Members:

Variable Name	Type	No. o...	Scale	Sequenc...
item1	long	-	-	
item2	long	-	-	
result	long	-	-	

Variable name	Type	Usage
item1	long	Item used for a binary operation (its value is specified by a client)
item2	long	Item used for a binary operation (its value is specified by a client)
result	long	Result of a binary operation (its value is set by the server)

Click the **OK** button, "S1" is displayed on the **Structure Declaration** page.



Structure Declaration

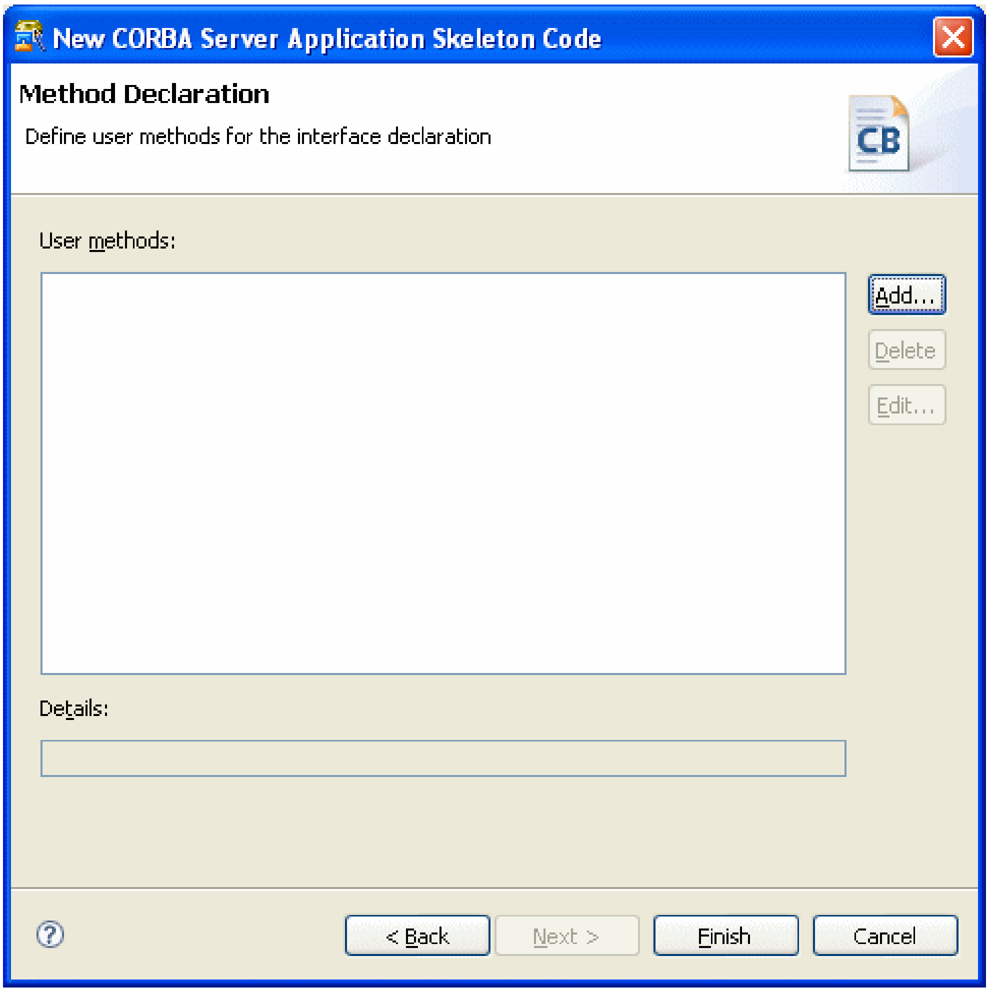
Define structures for the interface declaration

Structures:

S1

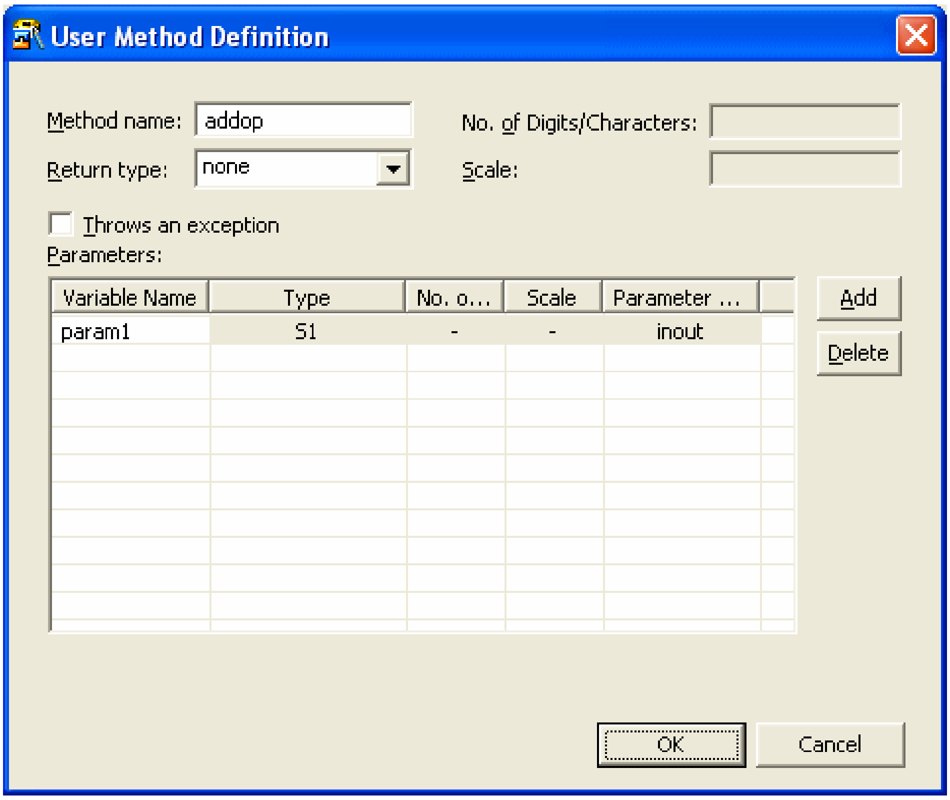
Click the **Next** button.

13. Next, declare business methods. Here, declare the addop, subop, mltop, and divop methods described in "Contents of Program Created".


New CORBA Server Application Skeleton Code
Method Declaration
Define user methods for the interface declaration
CB
User methods:
Add...
Delete
Edit...
Details:
< Back
Next >
Finish
Cancel

On this screen, click the **Add** button to display the **User Method Definition** dialog, and declare the methods in this dialog.

14. The addop method settings are specified in the dialog as shown below.



Method name: addop

Return type: none

No. of Digits/Characters:

Scale:

□ Throws an exception

Parameters:

Variable Name	Type	No. o...	Scale	Parameter ...
param1	S1	-	-	inout

The following are specified.

- addop method

Setting item	Setting value
Method name	addop
Return type	none(void)
Throws an exception	Uncheck
Parameters	Variable Name : param1 Type : S1 (*1) Parameter Ttype : inout (*2)

- subop method

Setting item	Setting value
Method name	subop
Return type	none(void)
Throws an exception	Uncheck
Parameters	Variable Name : param1 Type : S1 Parameter Ttype : inout

- mltop method

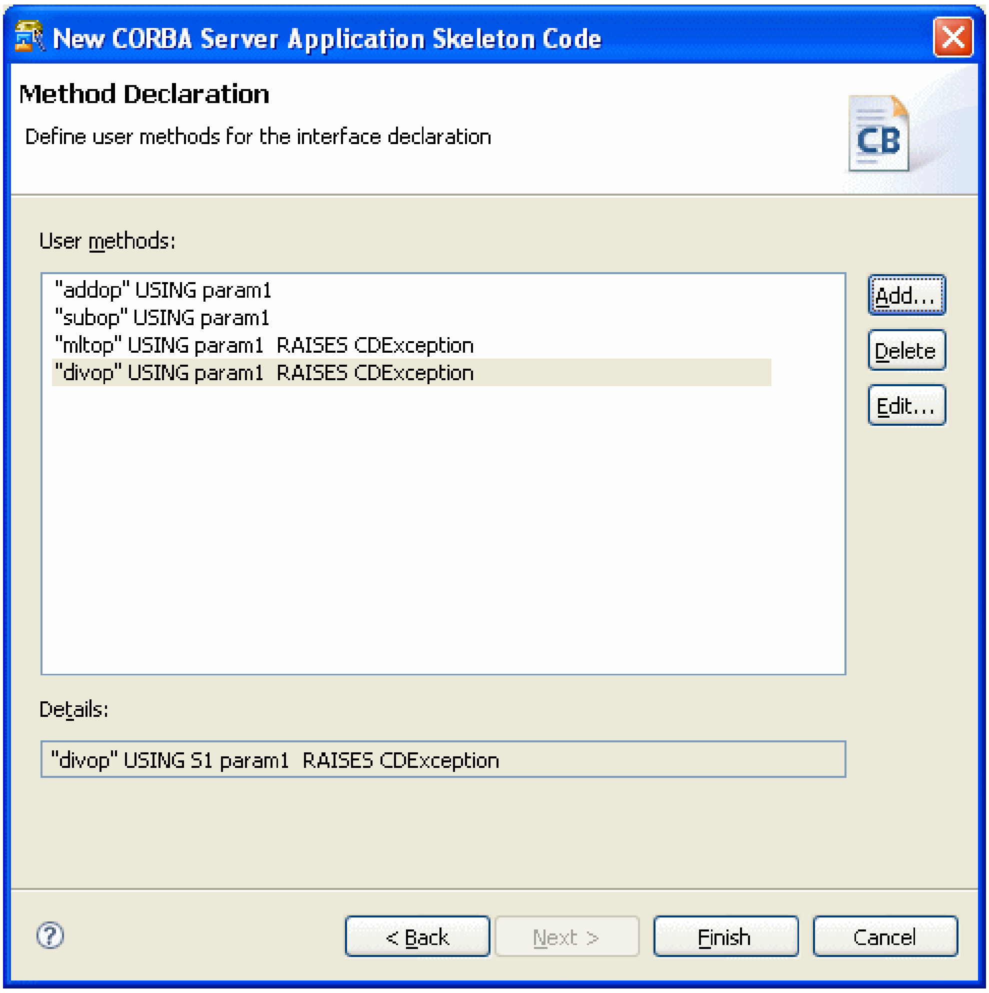
Setting item	Setting value
Method name	mltop
Return type	none(void)

Setting item	Setting value
Throws an exception	Check(*3)
Parameters	Variable Name : param1 Type : S1 Parameter Ttype : inout

- divop method

Setting item	Setting value
Method name	divop
Return type	none(void)
Throws an exception	Check (*3)
Parameters	Variable Name : param1 Type : S1 Parameter Ttype : inout

(*1) "S1" is the structure name declared in the structure definition.
(*2) Specify "inout" as the parameter type for the "S1" structure as values are set by both server and client applications.
(*3) Enable "Throws an exception" for multiplication and division methods.


New CORBA Server Application Skeleton Code
Method Declaration
Define user methods for the interface declaration
CB
User methods:
Add...
Delete
Edit...
Details:
< Back
Next >
Finish
Cancel

"addop" USING param1
"subop" USING param1
"mltop" USING param1 RAISES CDException
"divop" USING param1 RAISES CDException

"divop" USING S1 param1 RAISES CDException

15. Click the **Finish** button. When **Finish** button is clicked, template sources are created in the source file folder. In this tutorial, the following files are generated.

File name	Description
CALCSV.cob	Main source program (server application framework)
SAMPLE-CALCSV-IMPL.cob	Business method program
SAMPLE-CALCSV--INIT.cob	Server application registration program
USCALCSV.idl	Interface file (IDL file)
COBOL source file beginning with "SAMPLE"	Skeleton file created by IDL compiler

3. Editing Programs

IDL file (USCALCSV.idl)

```
// Module declaration
module SAMPLE {

  // Constant declaration
  const string MSG1 = "item1 is zero" ;
  const string MSG2 = "item2 is zero" ;

  // Type declaration

  // Structure declaration
  struct S1 {
    long item1;
    long item2;
    long result;
  };

  // Exception declaration
  exception CDException{
    string CDExceptionMsg;
    long CDExceptionCode;
  };
  // User interface declaration
  interface CALCSV {
    void addop(inout S1 param1);
    void subop(inout S1 param1);
    void mltop(inout S1 param1)
      raises (CDException);
    void divop(inout S1 param1)
      raises (CDException);
  };

};
```

You can edit the IDL file as per requirement, but no such editing is required in this tutorial.

* If interface information is modified, be sure to also modify the business method program source so that they are consistent with each other.

Main source program (CALCSV.cob)

```
000010 IDENTIFICATION DIVISION.
000020 PROGRAM-ID. CALCSV.
000030 ENVIRONMENT DIVISION.
000040 CONFIGURATION SECTION.
000050 REPOSITORY.
000060      COPY CORBA--REP.
000070      COPY USCALCSV--REP.
000080 .
000090 SPECIAL-NAMES.
```

```
000100      SYMBOLIC CONSTANT
000110      COPY CORBA--CONST.
000120  .
000130 DATA DIVISION.
000140 WORKING-STORAGE SECTION.
000150 COPY CORBA--COPY.
000160 COPY USCALCSV--COPY.
000170 01  API-NAME        PIC   X(50).
000180 01  APL-NAME        PIC   X(64)   VALUE     "CALCSV".
000190 01  ORB             USAGE OBJECT REFERENCE CORBA-ORB.
000200 01  BOA             USAGE OBJECT REFERENCE CORBA-BOA.
000210 01  IMPL-REP        USAGE OBJECT REFERENCE FJ-IMPLEMENTATIONREP.
000220 01  IMPL            USAGE OBJECT REFERENCE CORBA-IMPLEMENTATIONDEF.
000230 01  REP-ID          PIC   X(128) VALUE     "IDL:SAMPLE/CALCSV:1.0".(*1)
000240 01  OBJ             USAGE OBJECT REFERENCE CORBA-OBJECT.
000250 01  EXCEPT-ID       USAGE OBJECT REFERENCE CORBA-STRING.
000260 01  EXCEPT-ID-VALUE PIC   X(50).
000270 LINKAGE SECTION.
000280 PROCEDURE DIVISION
000290                   .
000300*
000310*  ORB initialization
000320*
000330      INVOKE CORBA "ORB_INIT" USING APL-NAME FJ-OM_ORBID RETURNING ORB.
000340*
000350*  BOA initialization
000360*
000370      INVOKE ORB "BOA_INIT" USING APL-NAME CORBA-BOA_OAID RETURNING BOA.
000380*
000390*  Obtain the implementation object repository.
000400*
000410      INVOKE ORB "RESOLVE_INITIAL_REFERENCES" USING
000411                    CORBA-OBJECTID_IMPLEMENTAT-001 RETURNING OBJ.
000420      INVOKE FJ-IMPLEMENTATIONREP "NARROW" USING OBJ RETURNING IMPL-REP.
000430*
000440*  Obtain implementation information.
000450*
000460      INVOKE IMPL-REP "LOOKUP_ID" USING REP-ID RETURNING OBJ.
000470      INVOKE CORBA-IMPLEMENTATIONDEF "NARROW" USING OBJ RETURNING IMPL.
000480*
000490      SET OBJ TO NULL.
000500*
000510*  Notify the OD of server activation.
000520*
000530      MOVE "CORBA::BOA::IMPL_IS_READY" TO API-NAME.
000540      INVOKE BOA "IMPL_IS_READY" USING IMPL.
000550*
000560      STOP RUN.
000570 END PROGRAM CALCSV.
```

(*1) Repository ID

Each CORBA server application is determined by a unique repository ID. The default ID is "IDL:module-name/interface-name:1.0". For details, refer to the Interstage Application Server manuals.

Server application registration program (SAMPLE-CALCSV--INIT.cob)

This program registers the server application so that it can be used as a CORBA server. No modification is required.

```
000010 IDENTIFICATION DIVISION.
000020 PROGRAM-ID. SAMPLE-CALCSV--INIT.
000030 ENVIRONMENT DIVISION.
000040 CONFIGURATION SECTION.
000050 REPOSITORY.
```

```
000060      CLASS SAMPLE-CALCSV
000070      CLASS SAMPLE-CALCSV-IMPL
000080 .
000090 DATA DIVISION.
000100 WORKING-STORAGE SECTION.
000110 LINKAGE SECTION.
000120 01 RET USAGE OBJECT REFERENCE SAMPLE-CALCSV.
000130 PROCEDURE DIVISION
000140         RETURNING RET
000150                   .
000160      INVOKE SAMPLE-CALCSV-impl "new" RETURNING RET.
000170      EXIT PROGRAM.
000180 END PROGRAM SAMPLE-CALCSV--INIT.
```

Implementing business methods (SAMPLE-CALCSV-IMPL.cob)

The addop, subop, mltop, and divop methods entered in the wizard are declared. Implement their processes.

Add the statements shown in blue.

```
000010 CLASS-ID. SAMPLE-CALCSV-IMPL AS "SAMPLE-CALCSV-IMPL" INHERITS
000020     SAMPLE-CALCSV
000030   .
000040 ENVIRONMENT DIVISION.
000050 CONFIGURATION SECTION.
000060 REPOSITORY.
000070      COPY CORBA--REP.
000080      COPY USCALCSV--REP.
000090 .
000100 SPECIAL-NAMES.
000110      SYMBOLIC CONSTANT
000120      COPY CORBA--CONST.
000121      COPY USCALCSV--CONST.
000130  .
000140 OBJECT.
000150 DATA DIVISION.
000160 WORKING-STORAGE SECTION.
000170 COPY CORBA--COPY.
000180 COPY USCALCSV--COPY.
000190 PROCEDURE DIVISION
000200                   .
000210 METHOD-ID. ADDOP AS "ADDOP" OVERRIDE.
000220* <IDL-INFO-START>
000230* void addop(inout S1 param1)
000240* <IDL-INFO-END>
000250 DATA DIVISION.
000260 WORKING-STORAGE SECTION.
000270 LINKAGE SECTION.
000280 01 PARAM1  TYPE SAMPLE-S1.
000290 PROCEDURE DIVISION
000300         USING
000310                   PARAM1
000320                   .
000321      COMPUTE result OF param1 = item1 OF param1 + item2 OF param1.
000330 END METHOD ADDOP.
000340 METHOD-ID. SUBOP AS "SUBOP" OVERRIDE.
000350* <IDL-INFO-START>
000360* void subop(inout S1 param1)
000370* <IDL-INFO-END>
000380 DATA DIVISION.
000390 WORKING-STORAGE SECTION.
000400 LINKAGE SECTION.
000410 01 PARAM1  TYPE SAMPLE-S1.
000420 PROCEDURE DIVISION
```

```
000430         USING
000440                   PARAM1
000450                   .
000451      COMPUTE result OF param1 = item1 OF param1 - item2 OF param1.
000460 END METHOD SUBOP.
000470 METHOD-ID. MLTOP AS "MLTOP" OVERRIDE.
000480* <IDL-INFO-START>
000490* void mltop(inout S1 param1)
000500* <IDL-INFO-END>
000510 DATA DIVISION.
000520 WORKING-STORAGE SECTION.
000521 01 W-EXCEPTION OBJECT REFERENCE SAMPLE-CDException.
000522 01 W-STRING OBJECT REFERENCE CORBA-STRING.
000530 LINKAGE SECTION.
000540 01 PARAM1  TYPE SAMPLE-S1.
000550 PROCEDURE DIVISION
000560         USING
000570                   PARAM1
000580         RAISING
000590                   SAMPLE-CDEXCEPTION
000600                   .
000601* Exception processing
000602       IF item1 OF param1 = 0
000603       THEN
000604          INVOKE SAMPLE-CDException "NEW" RETURNING W-EXCEPTION
000605          MOVE -1 TO CDEXCEPTIONCODE OF W-EXCEPTION
000606          INVOKE CORBA-STRING "NEW" RETURNING W-STRING
000607          INVOKE W-STRING "SET-VALUE" USING SAMPLE-MSG1
000608          SET CDEXCEPTIONMSG OF W-EXCEPTION TO W-STRING
000609          EXIT METHOD RAISING W-EXCEPTION
000610       END-IF
000611       IF item2 OF param1 = 0
000612       THEN
000613          INVOKE SAMPLE-CDException "NEW" RETURNING W-EXCEPTION
000614          MOVE -1 TO CDEXCEPTIONCODE OF W-EXCEPTION
000615          INVOKE CORBA-STRING "NEW" RETURNING W-STRING
000616          INVOKE W-STRING "SET-VALUE" USING SAMPLE-MSG2
000617          SET CDEXCEPTIONMSG OF W-EXCEPTION TO W-STRING
000618          EXIT METHOD RAISING W-EXCEPTION
000619       END-IF
000620       COMPUTE result OF param1 = item1 OF param1 * item2 OF param1.
000621
000622 END METHOD MLTOP.
000623 METHOD-ID. DIVOP AS "DIVOP" OVERRIDE.
000630* <IDL-INFO-START>
000640* void divop(inout S1 param1)
000650* <IDL-INFO-END>
000660 DATA DIVISION.
000670 WORKING-STORAGE SECTION.
000671 01 W-EXCEPTION OBJECT REFERENCE SAMPLE-CDException.
000672 01 W-STRING OBJECT REFERENCE CORBA-STRING.
000680 LINKAGE SECTION.
000690 01 PARAM1  TYPE SAMPLE-S1.
000700 PROCEDURE DIVISION
000710         USING
000720                   PARAM1
000730         RAISING
000740                   SAMPLE-CDEXCEPTION
000750                   .
000751* Exception processing
000752       IF item1 OF param1 = 0
000753       THEN
000754          INVOKE SAMPLE-CDException "NEW" RETURNING W-EXCEPTION
```

```
000755          MOVE -1 TO CDEXCEPTIONCODE OF W-EXCEPTION
000756          INVOKE CORBA-STRING "NEW" RETURNING W-STRING
000757          INVOKE W-STRING "SET-VALUE" USING SAMPLE-MSG1
000758          SET CDEXCEPTIONMSG OF W-EXCEPTION TO W-STRING
000759          EXIT METHOD RAISING W-EXCEPTION
000760       END-IF
000761       IF item2 OF param1 = 0
000762       THEN
000763          INVOKE SAMPLE-CDException "NEW" RETURNING W-EXCEPTION
000764          MOVE -1 TO CDEXCEPTIONCODE OF W-EXCEPTION
000765          INVOKE CORBA-STRING "NEW" RETURNING W-STRING
000766          INVOKE W-STRING "SET-VALUE" USING SAMPLE-MSG2
000767          SET CDEXCEPTIONMSG OF W-EXCEPTION TO W-STRING
000768          EXIT METHOD RAISING W-EXCEPTION
000769       END-IF
000770       COMPUTE result OF param1 = item1 OF param1 / item2 OF param1.
000771
000772 END METHOD DIVOP.
000773 END OBJECT.

000780 END CLASS SAMPLE-CALCSV-IMPL.
```

Point

- The "S1" structure is fixed in length and hence, is accompanied by a TYPE declaration.
- Elements of the "S1" structure are defined as group items and therefore can be used as group items.
- To report an exception, generate (new) an exception class instance, and specify a value in the member.

4. Building the Project

Select "Save" from the context menu of the edit window in USCALCSV.cob and the CORBA server application is built automatically. However, if **Project** > **Build Automatically** was not previously selected (the **Build Automatically** menu item will have a check if it was selected), the application is not built.

Select the CORBA server project in the **Dependency view** or **Structure view**, and select **Project** > **Project Build** from the menu bar to build the CORBA server project.

5. Running the Program

The CORBA server application runs on a WorkUnit. This section explains the general procedure for verifying the operation of CORBA server application.

Point

Note on MyCORBADebug

In the initial state, a WorkUnit named MyCORBADebug is created for debugging. This created MyCORBADebug can be used as the deployment destination for functional tests. For details, see "11.5.1.4.3 Creating the CORBA work Unit".

1. Copying runtime resources
   Copy the execution file and dynamic link library to the runtime environment, both of which have been generated during building. In this tutorial, copy the following files to the following destination:

   ```
   File: CALCSV.exe, USCALCSV.dll
   Copy destination folder: C:\Interstage\APS\var\CORBA_WU\MyCORBADebug
   ```

   The files can be copied to any folder. In this tutorial, however, they are copied to the default application folder of the WorkUnit to be created.

2. Making of application operation current directory

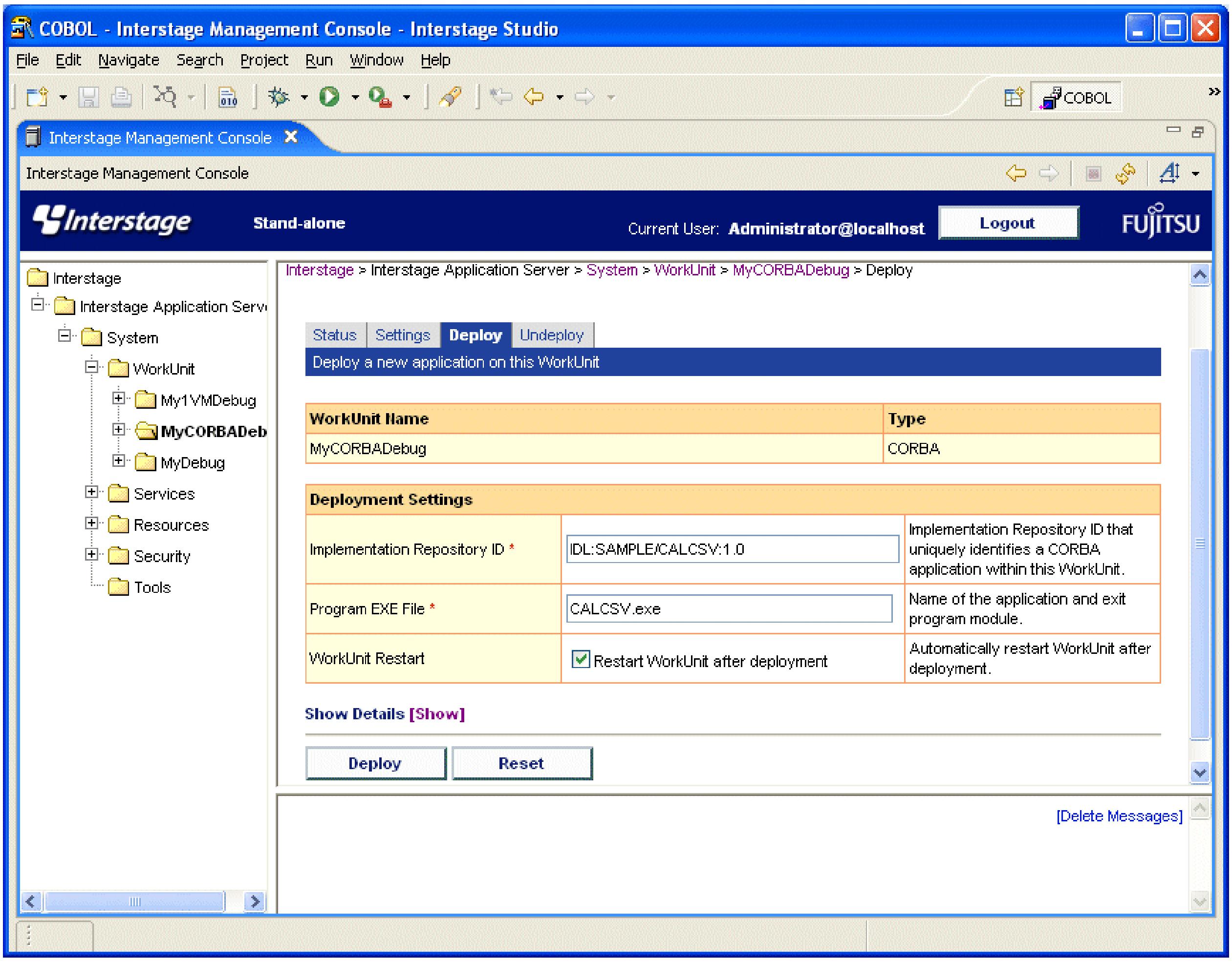
   Create the application operation current directory that is needed because of the start of the work unit.
   Because the default value is used in this tutorial when the work unit is made, it is necessary to prepare the following directories.

```
C:\Interstage\APS\var\CORBA_WU\MyCORBADebug\work
```

3. Deploying the CORBA server application

   Click right button on IJServer view, select "Interstage Management Console". When it is not connected to the Interstage Application Server, select **Connect/Login** from the context menu of the Interstage Application Server, connect or log on to the Interstage Application Server and then select "Interstage Management Console". Select the deployment destination WorkUnit from the left tree of the Management Console, and select the **Deploy** tab in the right side of the window.

   Specify an implementation repository ID and executable program file name.


COBOL - Interstage Management Console - Interstage Studio
File Edit Navigate Search Project Run Window Help
COBOL
Interstage Management Console
Interstage Management Console
Interstage
Stand-alone
Current User: Administrator@localhost
Logout
FUJITSU
Interstage
Interstage Application Servi
System
WorkUnit
My1VMDebug
MyCORBADeb
MyDebug
Services
Resources
Security
Tools
Status Settings Deploy Undeploy
Show Details [Show]
Deploy
Reset
[Delete Messages]

Interstage > Interstage Application Server > System > WorkUnit > MyCORBADebug > Deploy

Deploy a new application on this WorkUnit

WorkUnit Name	Type
MyCORBADebug	CORBA

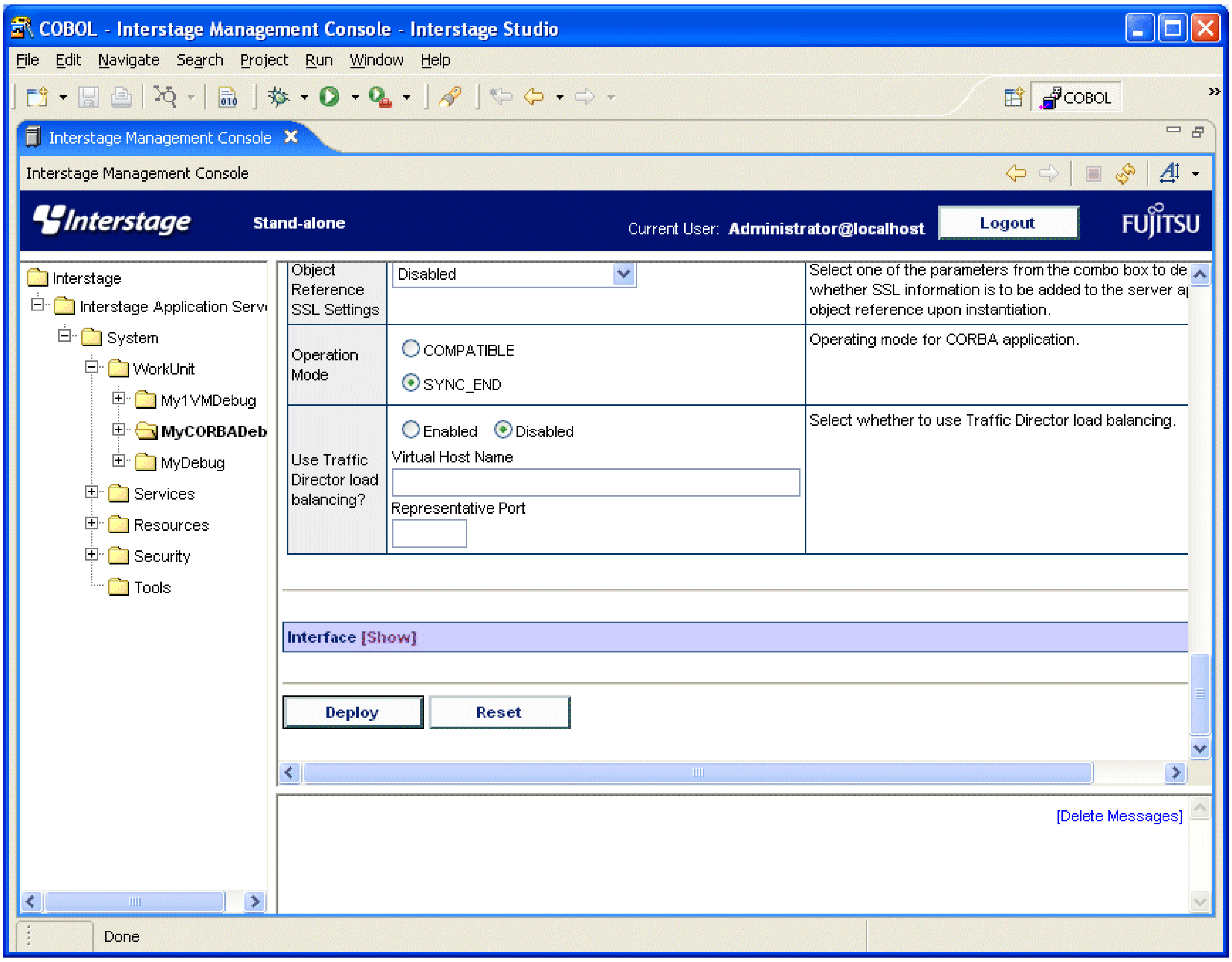
Deployment Settings		
Implementation Repository ID *	IDL:SAMPLE/CALCSV:1.0	Implementation Repository ID that uniquely identifies a CORBA application within this WorkUnit.
Program EXE File *	CALCSV.exe	Name of the application and exit program module.
WorkUnit Restart	☑ Restart WorkUnit after deployment	Automatically restart WorkUnit after deployment.

   Specify the implementation repository ID in the following format:

```
IDL:module-name/interface-name:1.0
```

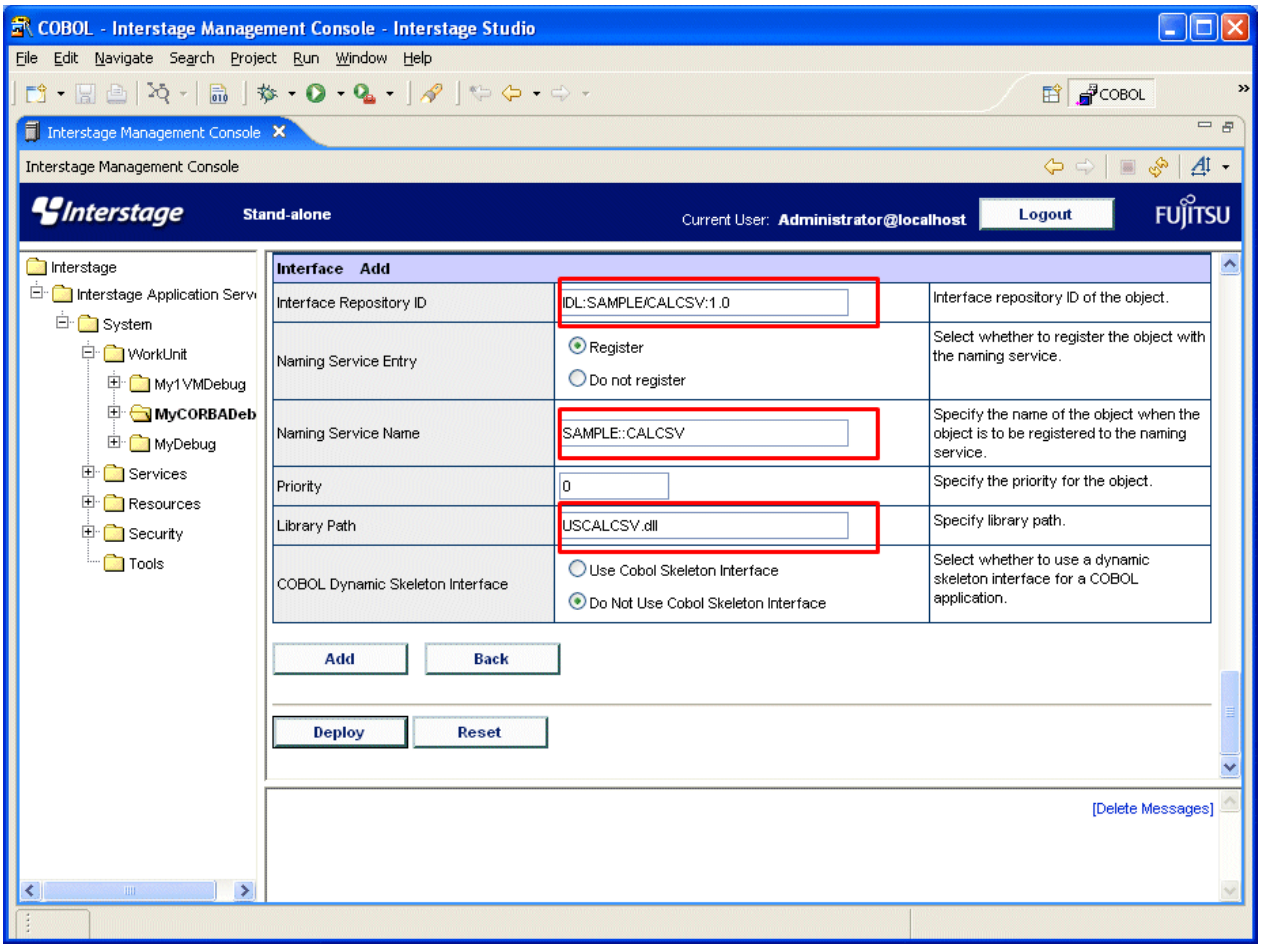
   For the executable program file name, specify the name of the EXE file that was copied.

Click "Show" of the Show Details, then click "Show" of the CORBA application. The Detailed Settings window is displayed. Select "SYNC_END" from the "Operation Mode".


COBOL - Interstage Management Console - Interstage Studio
File Edit Navigate Search Project Run Window Help
COBOL
Interstage Management Console
Interstage Management Console
Interstage
Stand-alone
Current User: Administrator@localhost
Logout
FUJITSU
Interstage
Interstage Application Serv
System
WorkUnit
My1VMDebug
MyCORBADeb
MyDebug
Services
Resources
Security
Tools
Interface [Show]
Deploy
Reset
[Delete Messages]
Done

Object Reference SSL Settings	Disabled	Select one of the parameters from the combo box to de whether SSL information is to be added to the server a object reference upon instantiation.
Operation Mode	○ COMPATIBLE ● SYNC_END	Operating mode for CORBA application.
Use Traffic Director load balancing?	○ Enabled ● Disabled Virtual Host Name Representative Port	Select whether to use Traffic Director load balancing.

Click "Show" of the Interface. Click the **Add Interface** button, and specify the interface repository ID, naming service registration name, and library name as shown below.


COBOL - Interstage Management Console - Interstage Studio
File Edit Navigate Search Project Run Window Help
COBOL
Interstage Management Console
Interstage Management Console
Interstage
Stand-alone
Current User: Administrator@localhost
Logout
FUJITSU
Interstage
Interstage Application Serv
System
WorkUnit
My1VMDebug
MyCORBADeb
MyDebug
Services
Resources
Security
Tools
Add
Back
Deploy
Reset
[Delete Messages]

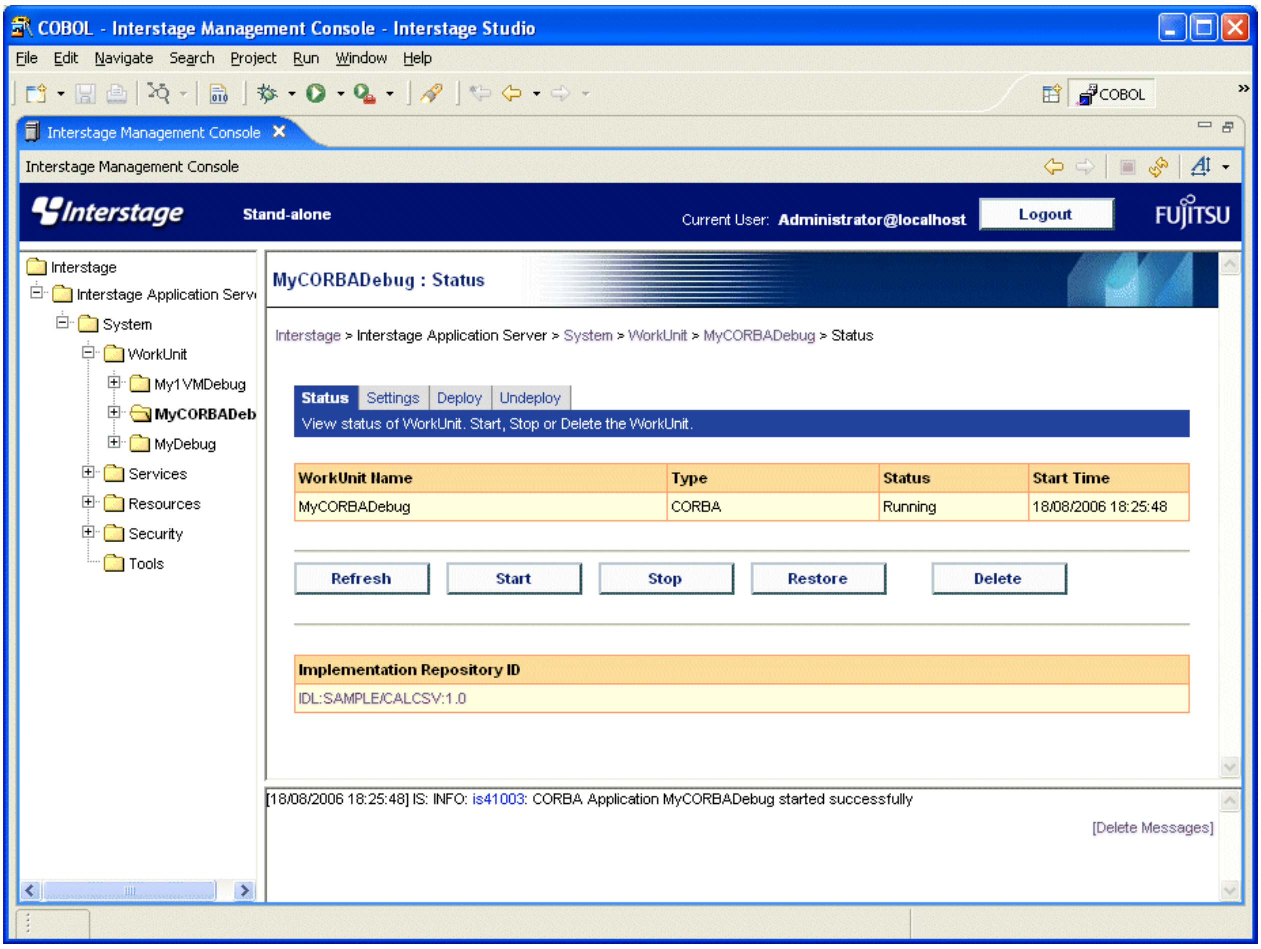
Interface Add		
Interface Repository ID	IDL:SAMPLE/CALCSV:1.0	Interface repository ID of the object.
Naming Service Entry	● Register ○ Do not register	Select whether to register the object with the naming service.
Naming Service Name	SAMPLE::CALCSV	Specify the name of the object when the object is to be registered to the naming service.
Priority	0	Specify the priority for the object.
Library Path	USCALCSV.dll	Specify library path.
COBOL Dynamic Skeleton Interface	○ Use Cobol Skeleton Interface ● Do Not Use Cobol Skeleton Interface	Select whether to use a dynamic skeleton interface for a COBOL application.

Click the **Add** button, then click the **Deploy** button to execute deployment.

4. Starting the WorkUnit

If "Restart WorkUnit after deployment" was not selected as the initial setting during deployment, start the created WorkUnit as follows.

Select the WorkUnit from the tree in the view on the left side of the Interstage Management Console, and click the **Status** tab in the right side of the window. Then, click the **Start** button.


COBOL - Interstage Management Console - Interstage Studio
File Edit Navigate Search Project Run Window Help
COBOL
Interstage Management Console
Interstage Management Console
Interstage
Stand-alone
Current User: Administrator@localhost
Logout
FUJITSU
Interstage
Interstage Application Serv
System
WorkUnit
My1VMDebug
MyCORBADeb
MyDebug
Services
Resources
Security
Tools
MyCORBADebug : Status
Interstage > Interstage Application Server > System > WorkUnit > MyCORBADebug > Status
Status
Settings
Deploy
Undeploy
View status of WorkUnit. Start, Stop or Delete the WorkUnit.
Refresh
Start
Stop
Restore
Delete
[Delete Messages]

WorkUnit Name	Type	Status	Start Time
MyCORBADebug	CORBA	Running	18/08/2006 18:25:48

Implementation Repository ID
IDL:SAMPLE/CALCSV:1.0

```
[18/08/2006 18:25:48] IS: INFO: is41003: CORBA Application MyCORBADebug started successfully
```

During the startup sequence, if there is no application operation current directory, an error occurs. You can check the application operation current directory by clicking the **Settings** tab of the WorkUnit.

5. Running a client application

   Run a client application that accesses the CORBA server application.

   For details on how to create and run a client application, refer to "11.2.3 Developing CORBA Client Applications".

11.2.3 Developing CORBA Client Applications

This section explains the procedure for creating a client application that calls the binary program of the tutorial on CORBA server applications.

1. Starting the Interstage basis service
2. Creating a Project
3. Creating a Template with COBOL Source Wizard
4. Editing the Program
5. Adding a Stub
6. Dependency Analysis
7. Building the Project
8. Verifying Operation

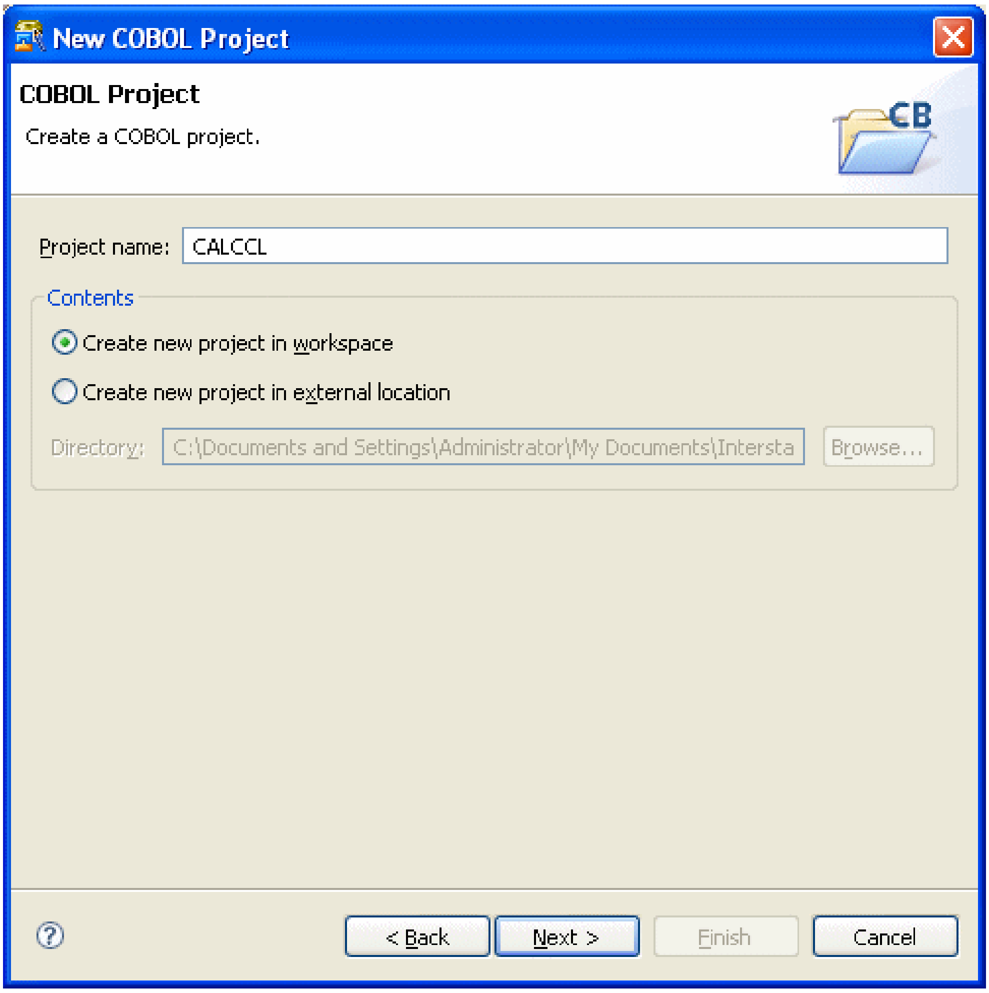
1. Starting the Interstage basis service

Before creating CORBA applications, it is necessary that the J2EE execution environment service is started. Because the J2EE execution environment service is not started by default, use the Interstage basis service operation tool to start the service. Refer to "Checking the environment settings of Interstage Application Server".

When the Interstage basis service has already been started by "11.2.2 Developing CORBA Server Applications", this operation is unnecessary.

2. Creating a Project

1. This section explains how to call a CORBA application using object-oriented COBOL. For details on programming using object-oriented COBOL, refer to the "Interstage Application Server Distributed Application Development Guide (CORBA Service Edition)".
2. If the COBOL perspective is not displayed, then follow the procedure below to display it:
   a. Select **Window** > **Open Perspective** > **Other** from the menu bar. The **Open Perspective** dialog box is displayed.
   b. Select **COBOL**, then click the **OK** button.
3. Select **File** > **New** > **COBOL Project** from the workbench - a new COBOL project is displayed.
4. Enter the project name and the save folder. Click the **Next** button.



New COBOL Project

COBOL Project

Create a COBOL project.

Project name: CALCCL

Contents

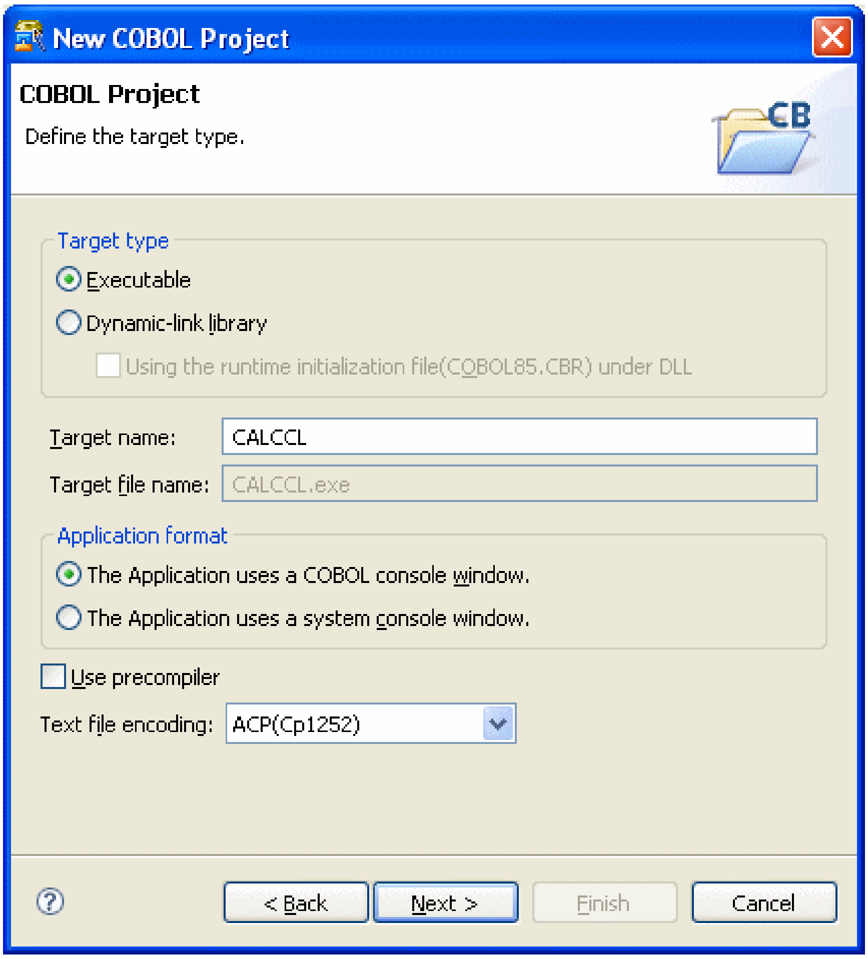
● Create new project in workspace

○ Create new project in external location

Directory: C:\Documents and Settings\Administrator\My Documents\Intersta

Setting item	Setting value
Project name	CALCCL
Contents	Select "Create new project in workspace"

5. You can specify target information on this screen. In this tutorial, proceed without specifying anything.



New COBOL Project

COBOL Project

Define the target type.

Target type

● Executable

○ Dynamic-link library

□ Using the runtime initialization file(COBOL85.CBR) under DLL

Target name: CALCCL

Target file name: CALCCL.exe

Application format

● The Application uses a COBOL console window.

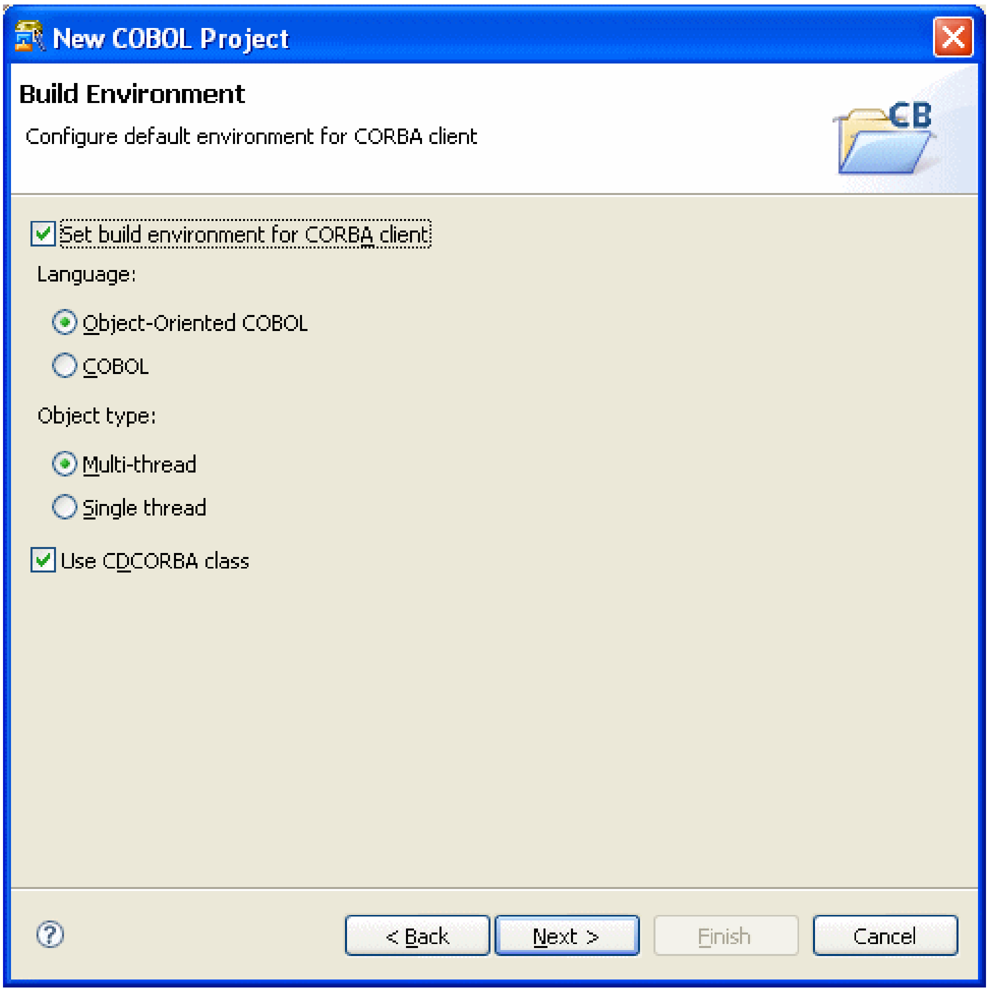
○ The Application uses a system console window.

□ Use precompiler

Text file encoding: ACP(Cp1252)

Click the **Next** button.

6. You can specify the build environment on this screen. Select "Set build environment for CORBA client". Leave the other settings at their default values. Click the **Next** button.



New COBOL Project

Build Environment

Configure default environment for CORBA client

☑ Set build environment for CORBA client

Language:

● Object-Oriented COBOL

○ COBOL

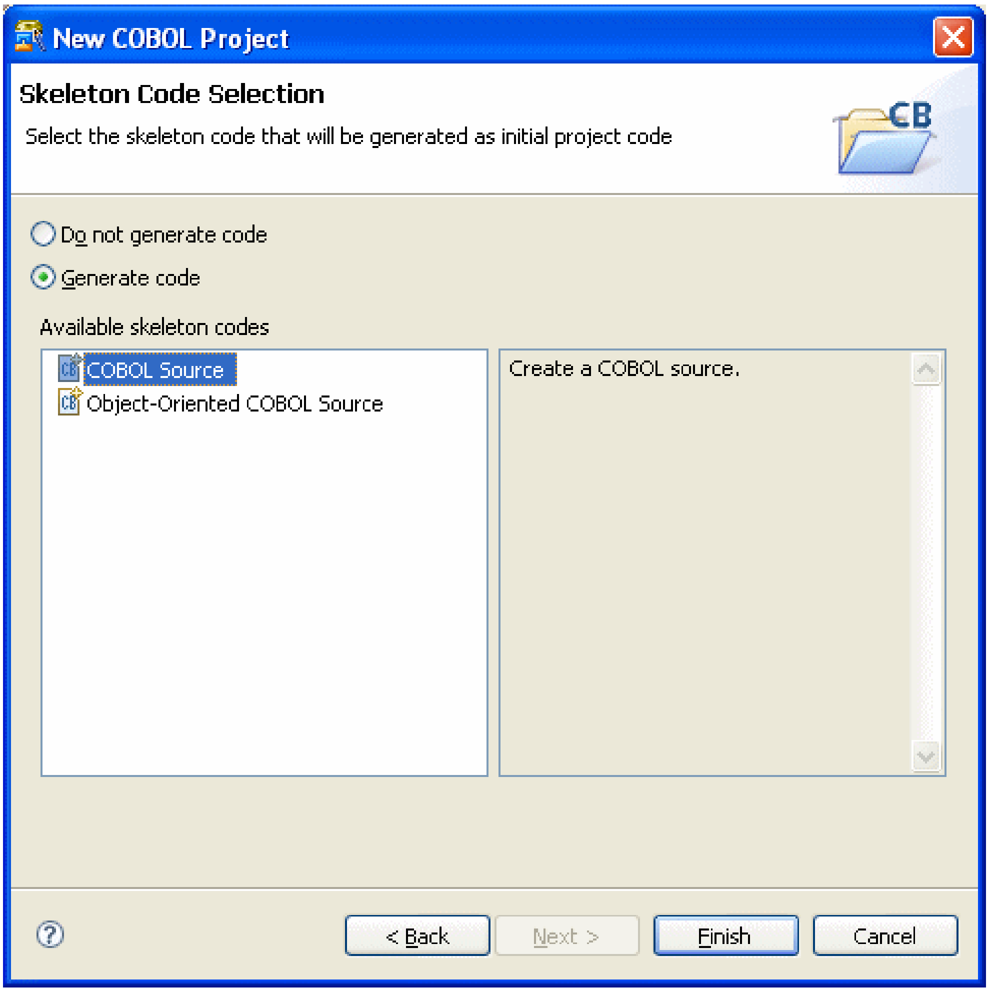
Object type:

● Multi-thread

○ Single thread

☑ Use CDCORBA class

7. You can select the option "Generate code". Make sure that "COBOL Source" is selected as the code generation wizard, and click the **Finish** button.



New COBOL Project

Skeleton Code Selection

Select the skeleton code that will be generated as initial project code

○ Do not generate code

● Generate code

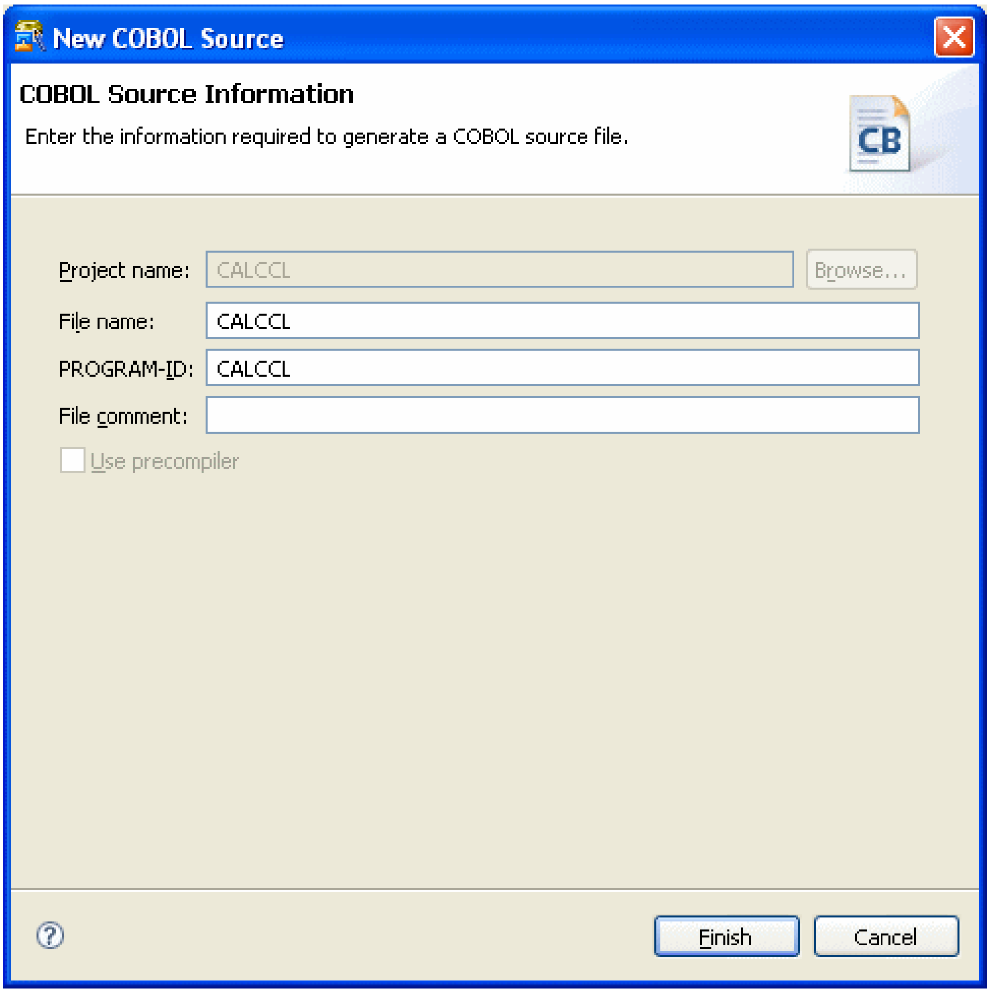
Available skeleton codes

- COBOL Source
- Object-Oriented COBOL Source

Create a COBOL source.

3. Creating a Template with COBOL Source Wizard

Create a program template by using the **New COBOL Source wizard**.


New COBOL Source
COBOL Source Information
Enter the information required to generate a COBOL source file.
CB

Project name: CALCCL

File name: CALCCL

PROGRAM-ID: CALCCL

File comment:

□ Use precompiler

In this tutorial, specify the following COBOL source information.

Setting item	Setting value
Project name	CALCCL
File name	CALCCL
PROGRAM-ID	CALCCL
File comment	(Write any comment.)

Click the **Finish** button. The "CALCCL.cob" and "CDCORBA.cob(*1)" file is created.

*1: When "Use CDCORBA class" is checked on the **Build Environment** page, this file is created.

4. Editing the Program

Edit the COBOL source generated by the wizard.

1. Displaying the **Template view**

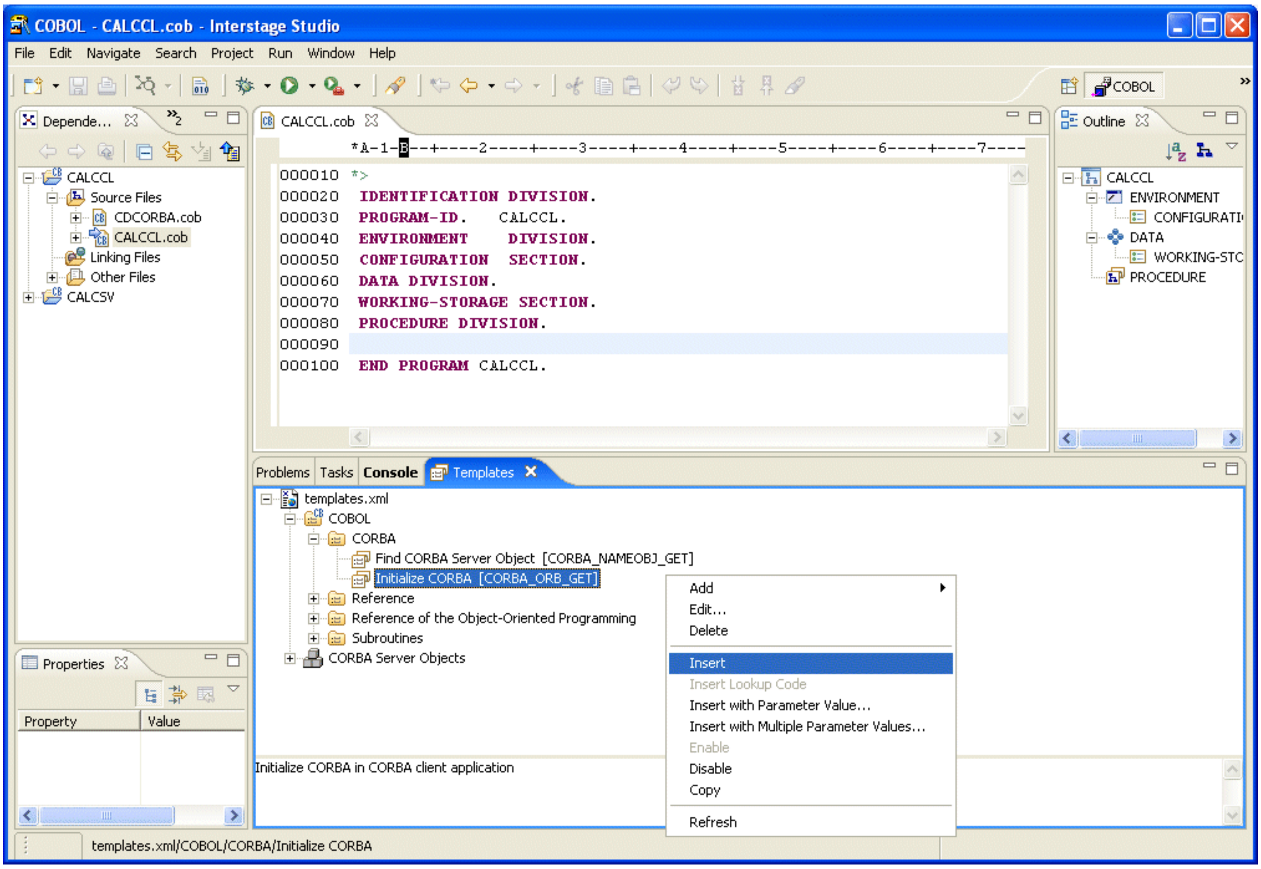
   1) Select **Window** > **Show View** > **Other** from the workbench menu. The **Show View** is displayed.

   2) Select **General** > **Template**, and click the **OK** button.

2. Adding CORBA initialization processing

   In the editor, move the cursor to the location at which this processing is to be added. Select **COBOL** > **CORBA** > **Initialize CORBA** from the **Template view**, then select "Insert" from the context menu to add the processing.

In this tutorial, insert "Initialize CORBA" processing onto line 9 (line number 90) of CALCCL.cob.



```
       *A-1-B--+----2----+----3----+----4----+----5----+----6----+----7----
000010 *>
000020 IDENTIFICATION DIVISION.
000030 PROGRAM-ID.    CALCCL.
000040 ENVIRONMENT    DIVISION.
000050 CONFIGURATION  SECTION.
000060 DATA DIVISION.
000070 WORKING-STORAGE SECTION.
000080 PROCEDURE DIVISION.
000090
000100 END PROGRAM CALCCL.
```

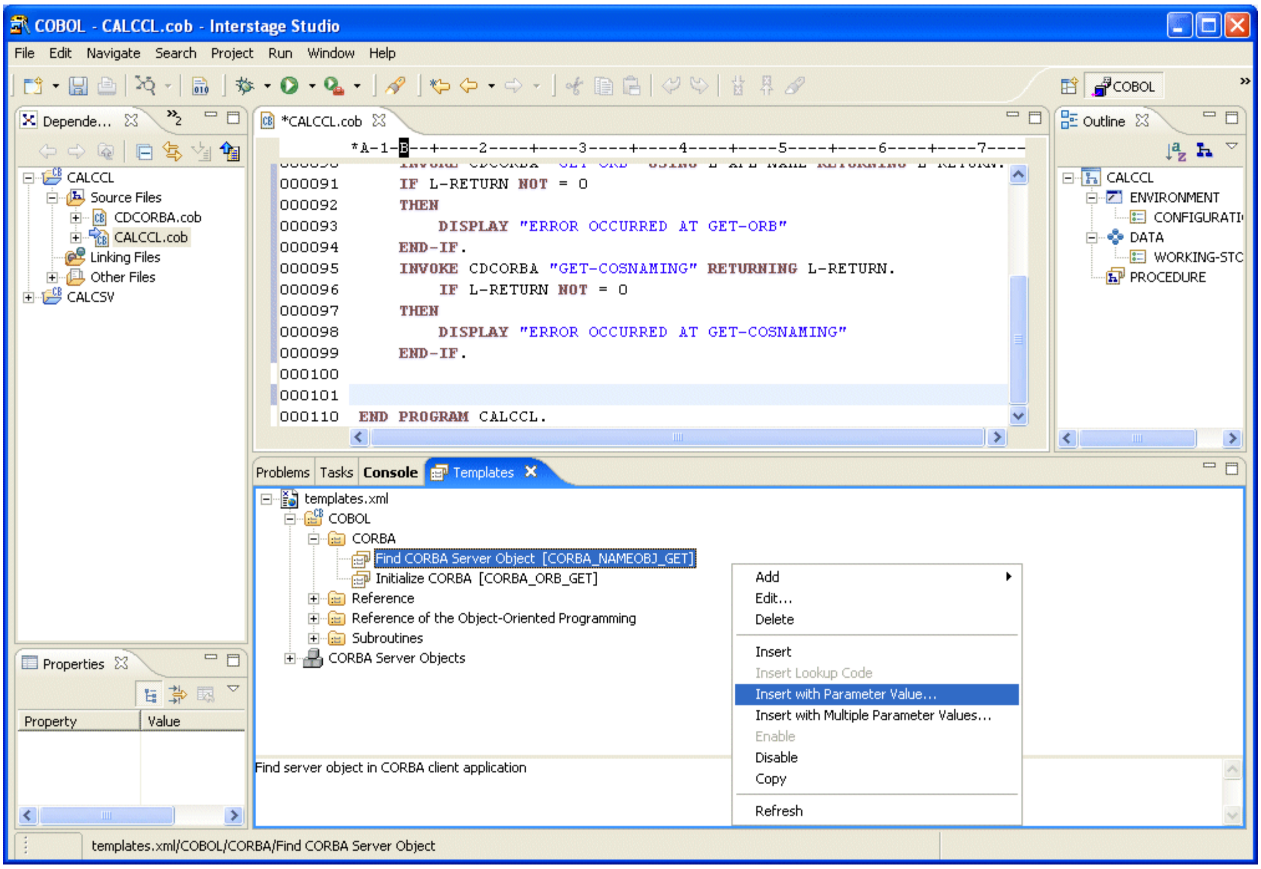
Initialize CORBA in CORBA client application

templates.xml/COBOL/CORBA/Initialize CORBA

3. Adding CORBA server object search processing

1) In the editor, move the cursor to the location at which this processing is to be added. Select **COBOL** > **CORBA** > **Find CORBA Server Object** from the **Template view**, then select "Insert with Parameter Value" from the context menu to add the processing.

In this tutorial, create a line immediately under line 19 (line number 100) of CALCCL.cob, and insert "Find CORBA Server Object" there.

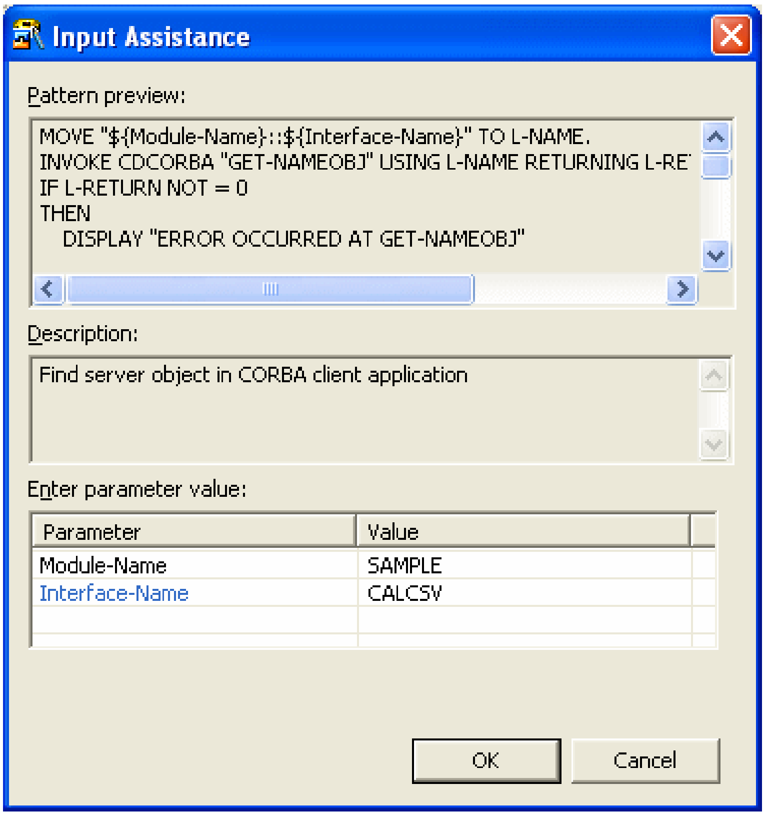


```
      *A-1-B--+----2----+----3----+----4----+----5----+----6----+----7----
000091     IF L-RETURN NOT = 0
000092     THEN
000093         DISPLAY "ERROR OCCURRED AT GET-ORB"
000094     END-IF.
000095     INVOKE CDCORBA "GET-COSNAMING" RETURNING L-RETURN.
000096         IF L-RETURN NOT = 0
000097     THEN
000098         DISPLAY "ERROR OCCURRED AT GET-COSNAMING"
000099     END-IF.
000100
000101
000110 END PROGRAM CALCCL.
```

Find server object in CORBA client application

templates.xml/COBOL/CORBA/Find CORBA Server Object

2) In the **Input Assistance** dialog that is displayed, specify values for the Replace key as shown below, and click the **OK** button to add the processing.



```
MOVE "${Module-Name}::${Interface-Name}" TO L-NAME.
INVOKE CDCORBA "GET-NAMEOBJ" USING L-NAME RETURNING L-RE
IF L-RETURN NOT = 0
THEN
   DISPLAY "ERROR OCCURRED AT GET-NAMEOBJ"
```

Find server object in CORBA client application

Parameter	Value
Module-Name	SAMPLE
Interface-Name	CALCSV

4. Adding method invocation processing

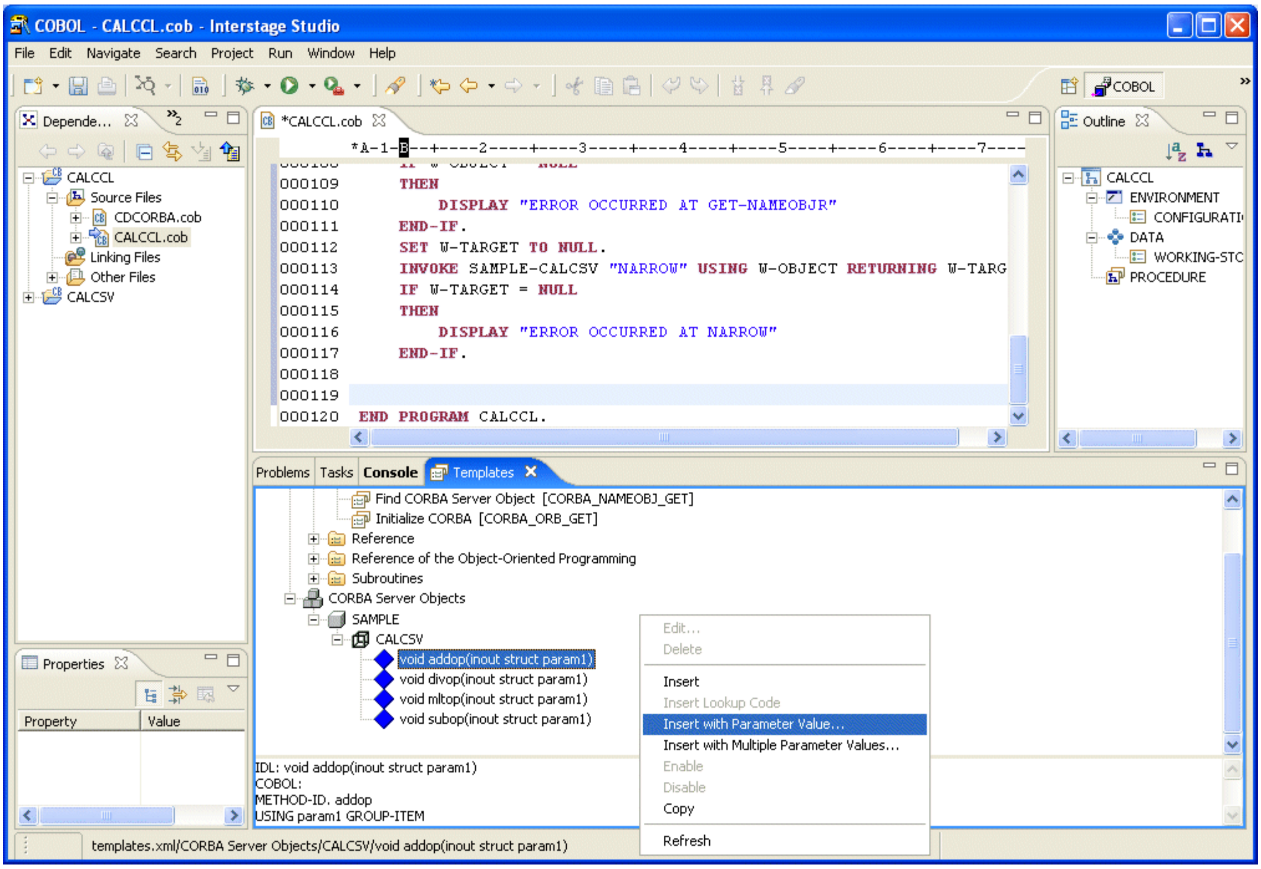
1) In the editor, move the cursor to the location at which this processing is to be added. Select "Module Name" > "Interface Name" > "Method" from "CORBA Server Objects" in the **Template view**, then select "Insert with Parameter Value" from the context menu.

In this tutorial, create a line immediately under line 37 (line number 118) of CALCCL.cob, and insert "addop" method invocation processing there.

Note

To obtain a list of CORBA server objects from the template, Interstage service and the services for management of Interstage(platform services) must be started. If they are not started, restart the workbench.

To start the platform service, use the Interstage Management Service Operation Tool. To start the tool, select **Interstage** > **Studio** > **Interstage Management Service Operation Tool** from the start menu.



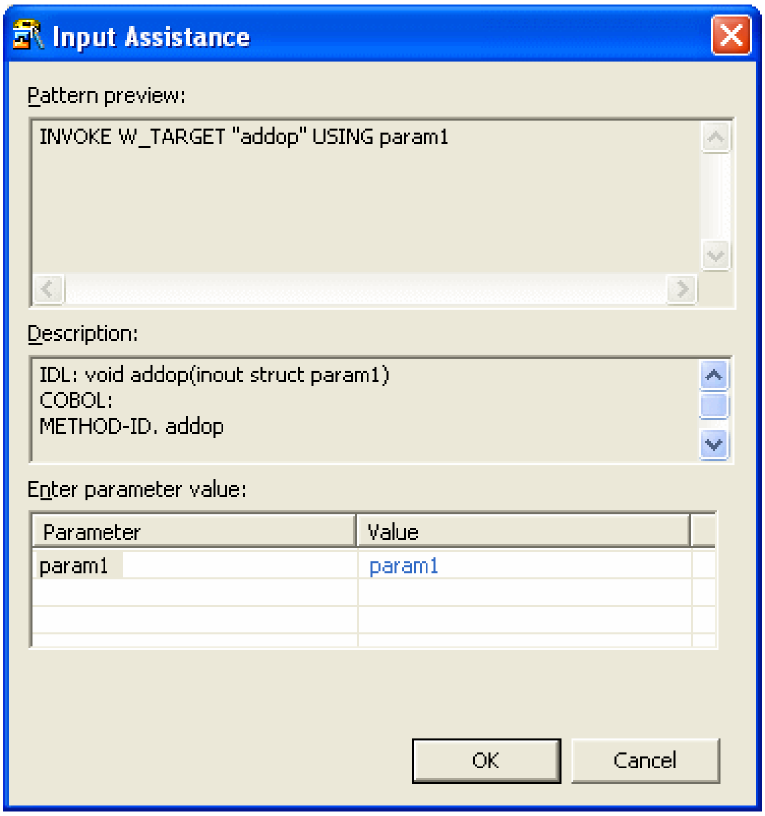
COBOL - CALCCL.cob - Interstage Studio
File Edit Navigate Search Project Run Window Help
COBOL
Depende...
CALCCL
Source Files
CDCORBA.cob
CALCCL.cob
Linking Files
Other Files
CALCSV
*CALCCL.cob
Outline
CALCCL
ENVIRONMENT
CONFIGURATI
DATA
WORKING-STC
PROCEDURE
Properties
Property
Value
Problems
Tasks
Console
Templates
Find CORBA Server Object [CORBA_NAMEOBJ_GET]
Initialize CORBA [CORBA_ORB_GET]
Reference
Reference of the Object-Oriented Programming
Subroutines
CORBA Server Objects
SAMPLE
CALCSV
void addop(inout struct param1)
void divop(inout struct param1)
void mltop(inout struct param1)
void subop(inout struct param1)
Edit...
Delete
Insert
Insert Lookup Code
Insert with Parameter Value...
Insert with Multiple Parameter Values...
Enable
Disable
Copy
Refresh

```
*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7----
000109        THEN
000110            DISPLAY "ERROR OCCURRED AT GET-NAMEOBJR"
000111        END-IF.
000112        SET W-TARGET TO NULL.
000113        INVOKE SAMPLE-CALCSV "NARROW" USING W-OBJECT RETURNING W-TARG
000114        IF W-TARGET = NULL
000115        THEN
000116            DISPLAY "ERROR OCCURRED AT NARROW"
000117        END-IF.
000118
000119
000120    END PROGRAM CALCCL.
```

```
IDL: void addop(inout struct param1)
COBOL:
METHOD-ID. addop
USING param1 GROUP-ITEM
```

templates.xml/CORBA Server Objects/CALCSV/void addop(inout struct param1)

2) Specify a value for the Replace key, and click the **OK** button to add the processing.



Pattern preview:

```
INVOKE W_TARGET "addop" USING param1
```

Description:

```
IDL: void addop(inout struct param1)
COBOL:
METHOD-ID. addop
```

Enter parameter value:

Parameter	Value
param1	param1

Add and/or modify necessary data items and processing codes in order to make the source program as shown below.

Note

In this source program, the method invocation processing that has been added last must be modified before the program can be used. If you copy the source program, delete the added method invocation processing before program execution.

```
000010*
000020 IDENTIFICATION DIVISION.
000030 PROGRAM-ID.   CALCCL.
000040 ENVIRONMENT    DIVISION.
000050 CONFIGURATION  SECTION.
000060   REPOSITORY.
000070* Standard library of ObjectDirector (for repository declaration)
000080     COPY CORBA--REP.
000090* Standard library of naming service (for repository declaration)
000100     COPY CosNaming--REP.
000110* Library output by IDL compiler (for repository declaration)
000120     COPY USCALCSV--REP.
000130* CORBA client development class provided by Interstage Studio
000140     CLASS CDCORBA.
000150
000160  SPECIAL-NAMES.
000170     ARGUMENT-NUMBER IS ARG-NO
000180     ARGUMENT-VALUE  IS ARG-VAL
000190     SYMBOLIC CONSTANT
000200* Standard library of  ObjectDirector (for constant declaration)
000210     COPY CORBA--CONST.
000220* Standard library of naming service (for constant declaration)
000230     COPY COSNAMING--CONST.
000240* Library output by IDL compiler (for constant declaration)
```

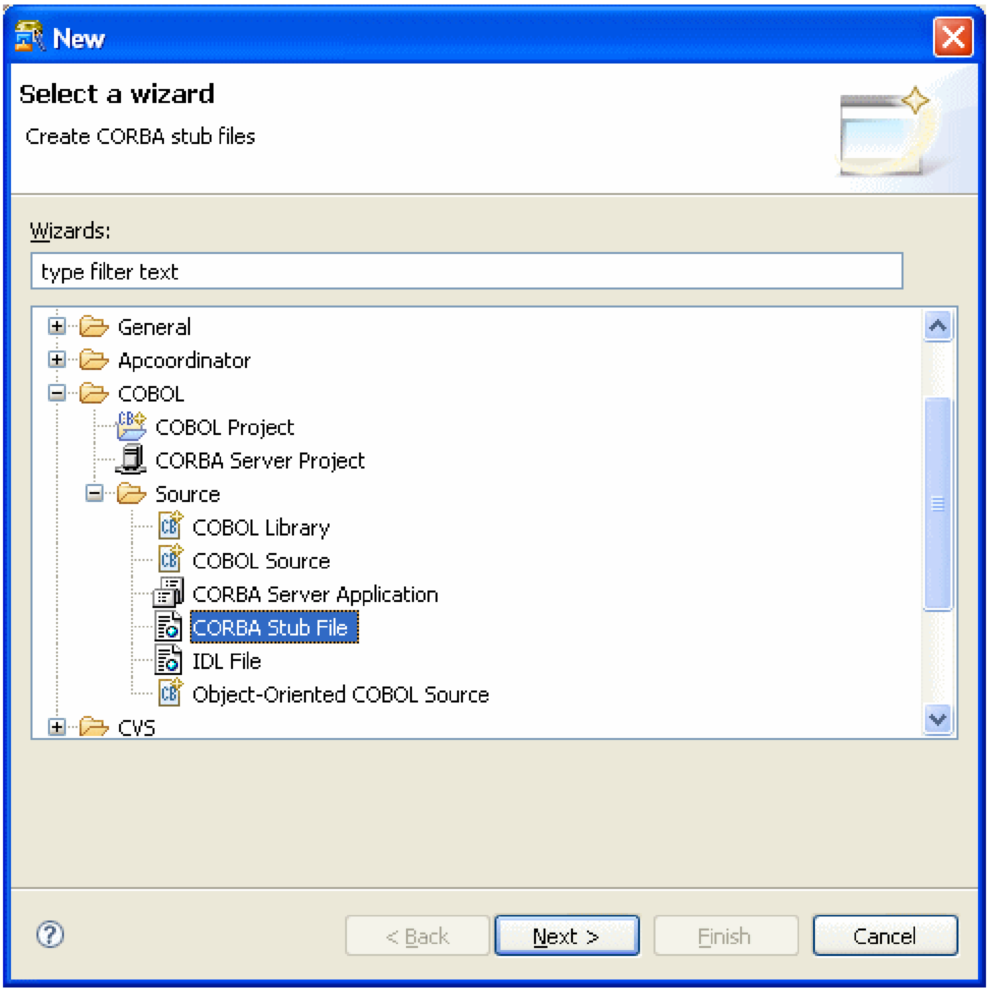
```
000250     COPY USCALCSV--CONST.
000260     .
000270 DATA DIVISION.
000280 WORKING-STORAGE SECTION.
000290 COPY CORBA--COPY.
000300 COPY USCALCSV--COPY.
000310 01 L-APL-NAME PIC X(50) VALUE "CALCCL".
000320 01 W-OBJECT OBJECT REFERENCE CORBA-OBJECT.
000330 01 W-TARGET OBJECT REFERENCE SAMPLE-CALCSV.
000340 01 STRUCT1  TYPE SAMPLE-S1.
000350 01 L-RETURN PIC S9(9) COMP-5.
000360 01 L-NAME PIC X(128) VALUE "SAMPLE::CALCSV".
000370 01 ERR-MSG PIC X(128).
000380 01 CDEXCEPTIONMSG OBJECT REFERENCE CORBA-STRING.
000390 01 CDEXCEPTIONCODE TYPE CORBA-LONG.
000400
000410 PROCEDURE DIVISION.
000420  DECLARATIVES.
000430   ERR SECTION.
000440     USE AFTER EXCEPTION SAMPLE-CDEXCEPTION.
000450     MOVE CDEXCEPTIONCODE OF EXCEPTION-OBJECT AS SAMPLE-CDEXCEPTION TO
000451                                 CDEXCEPTIONCODE.
000460     SET CDEXCEPTIONMSG TO CDEXCEPTIONMSG OF EXCEPTION-OBJECT AS SAMPLE-CDEXCEPTION.
000470     INVOKE CDEXCEPTIONMSG "GET-VALUE" RETURNING ERR-MSG.
000480     DISPLAY ERR-MSG.
000490  END DECLARATIVES.
000500     INVOKE CDCORBA "GET-ORB" USING L-APL-NAME RETURNING L-RETURN.
000510     IF L-RETURN NOT = 0
000520     THEN
000530         DISPLAY "ERROR OCCURRED AT GET-ORB"
000540     END-IF
000550     INVOKE CDCORBA "GET-COSNAMING" RETURNING L-RETURN.
000560         IF L-RETURN NOT = 0
000570     THEN
000580          DISPLAY "ERROR OCCURRED AT GET-COSNAMING"
000590     END-IF
000600
000610     MOVE "SAMPLE::CALCSV" TO L-NAME.
000620     INVOKE CDCORBA "GET-NAMEOBJ" USING L-NAME RETURNING L-RETURN.
000630     IF L-RETURN NOT = 0
000640     THEN
000650        DISPLAY "ERROR OCCURRED AT GET-NAMEOBJ"
000660     END-IF
000670     INVOKE CDCORBA "GET-NAMEOBJR" RETURNING W-OBJECT.
000680     IF W-OBJECT = NULL
000690     THEN
000700        DISPLAY "ERROR OCCURRED AT GET-NAMEOBJR"
000710     END-IF
000720
000730     SET W-TARGET TO NULL.
000740     INVOKE SAMPLE-CALCSV "NARROW" USING W-OBJECT RETURNING W-TARGET.
000750     IF W-TARGET = NULL
000760     THEN
000770        DISPLAY "ERROR OCCURRED AT NARROW"
000780     END-IF
000790
000800*    Invocation of server application methods
000810     DISPLAY "Enter the first argument: " WITH NO ADVANCING.
000820     ACCEPT  item1 OF STRUCT1.
000830     DISPLAY "Enter the second argument: " WITH NO ADVANCING.
000840     ACCEPT  item2 OF STRUCT1.
000850
000860     INVOKE W-TARGET "addop" USING STRUCT1
```

```
000870     DISPLAY "Addition result: " result OF STRUCT1.
000880
000890     INVOKE W-TARGET "subop" USING STRUCT1
000900     DISPLAY "Subtraction result: " result OF STRUCT1.
000910
000920     MOVE 0 TO CDEXCEPTIONCODE.
000930     DISPLAY "Multiplication result: " WITH NO ADVANCING.
000940     INVOKE W-TARGET "mltop" USING STRUCT1
000950     IF CDEXCEPTIONCODE NOT = -1 THEN
000960       DISPLAY result OF STRUCT1
000970     END-IF.
000980
000990     MOVE 0 TO CDEXCEPTIONCODE.
001000     DISPLAY "Division result: " WITH NO ADVANCING.
001010     INVOKE W-TARGET "divop" USING STRUCT1
001020     IF CDEXCEPTIONCODE NOT = -1 THEN
001030       DISPLAY result OF STRUCT1
001040     END-IF.
001050
001060 END PROGRAM CALCCL.
```

5. Adding a Stub

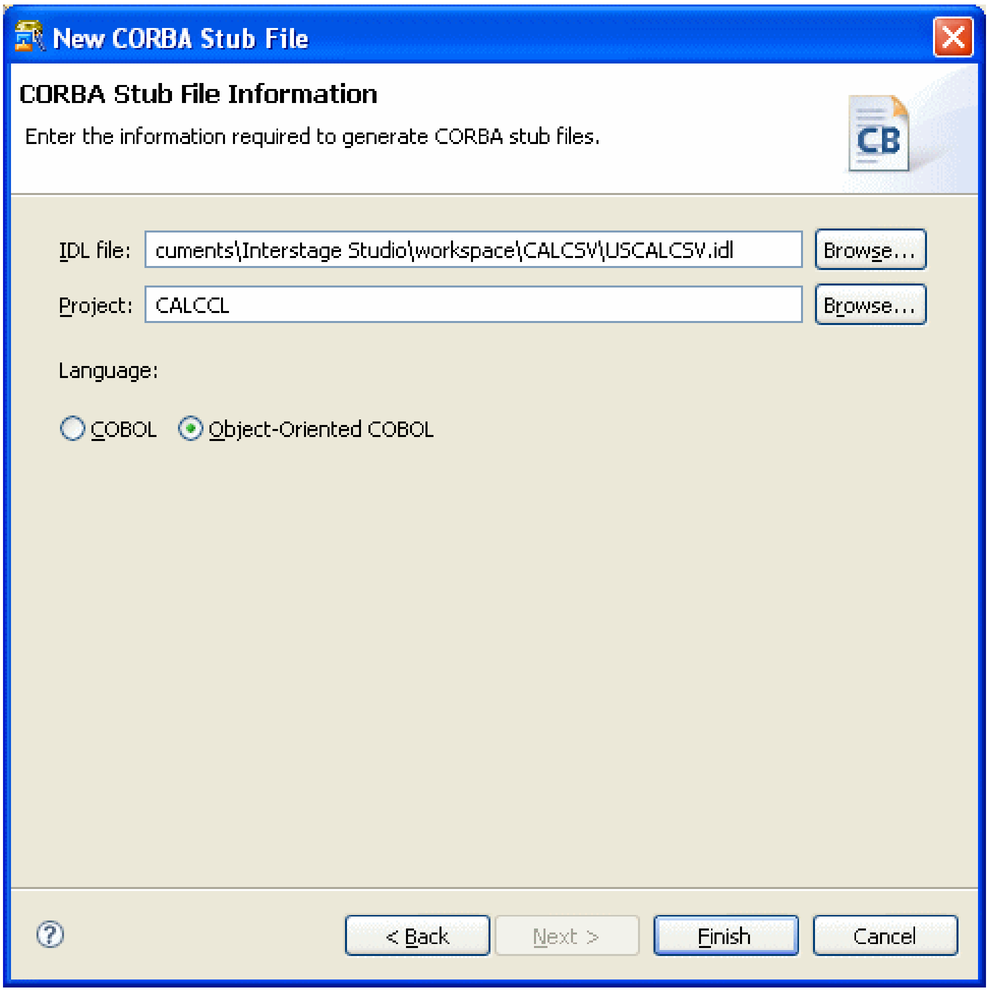
To call the CORBA server application, a stub file and other files required for the CORBA client must be created from the IDL file and added to the project. Use the **New CORBA Stub File wizard** to create a stub file and the other files, and add them to the project.

1. Select **File** > **New** > **Other** from the workbench menu. **New wizard** starts.
2. Select **COBOL** > **Source** > **CORBA Stub File**.



3. The **CORBA stub file wizard** starts.

   Specify the IDL file, the registration target project, and the language used to create the files.



New CORBA Stub File

CORBA Stub File Information

Enter the information required to generate CORBA stub files.

IDL file: cuments\Interstage Studio\workspace\CALCSV\USCALCSV.idl

Project: CALCCL

Language:

○ COBOL ● Object-Oriented COBOL

Setting item	Setting value
IDL File	Specify USCALCSV.idl of the CALCSV project.
Project	CALCCL
Language	Object-oriented COBOL

Click the **Finish** button. The following files are created in the project.

File type	File name
Repository paragraph declaration library	USCALCSV--REP.cbl
Constant declaration library	USCALCSV--CONST.cbl
TYPEDEF type declaration library	USCALCSV--COPY.cbl
Interface file	SAMPLE-CALCSV.cob
Helper class file	SAMPLE-CALCSV--HELPER.cob
Stub file	SAMPLE-CALCSV--STUB.cob
Narrow stub file	SAMPLE-CALCSV_NARROW.cob
Data-type class file	SAMPLE-CDEXCEPTION.cob

File type	File name
Data-type Helper file	SAMPLE-S1--HELPER.cob SAMPLE-CDEXCEPTION--HELPER.cob

Among the files created, COBOL sources other than libraries are registered as the source files in the **Dependency view**.

6. Dependency Analysis

Before a project including an object-oriented COBOL source file is built, the dependencies between COBOL source files must be defined. When the dependencies are correctly defined, COBOL sources are built in the correct order. If the dependencies are not correctly defined, a compile error may occur during building.

When a project is newly created or a COBOL source file is added to "Source Folder", dependencies for the added source file are automatically analyzed and defined. However, when the REPOSITORY paragraph of the COBOL source file has been edited as is shown in this tutorial, a dependency analysis must be executed manually before build.

Select the target project from the workbench **Dependency view**, then select **Edit** > **Analyze Dependency** > **All** from the menu to execute a dependency analysis.

Note

During a dependency analysis, the message "xxx.rep: does not exist" is displayed on the log console. Normally, an object-oriented COBOL source file in a project is compiled before a repository file (*.rep) for the source file is created. For this reason, the message is displayed if a dependency analysis is executed before build. In such cases, the dependency analysis is executed normally, so you can continue to build the project.

7. Building the Project

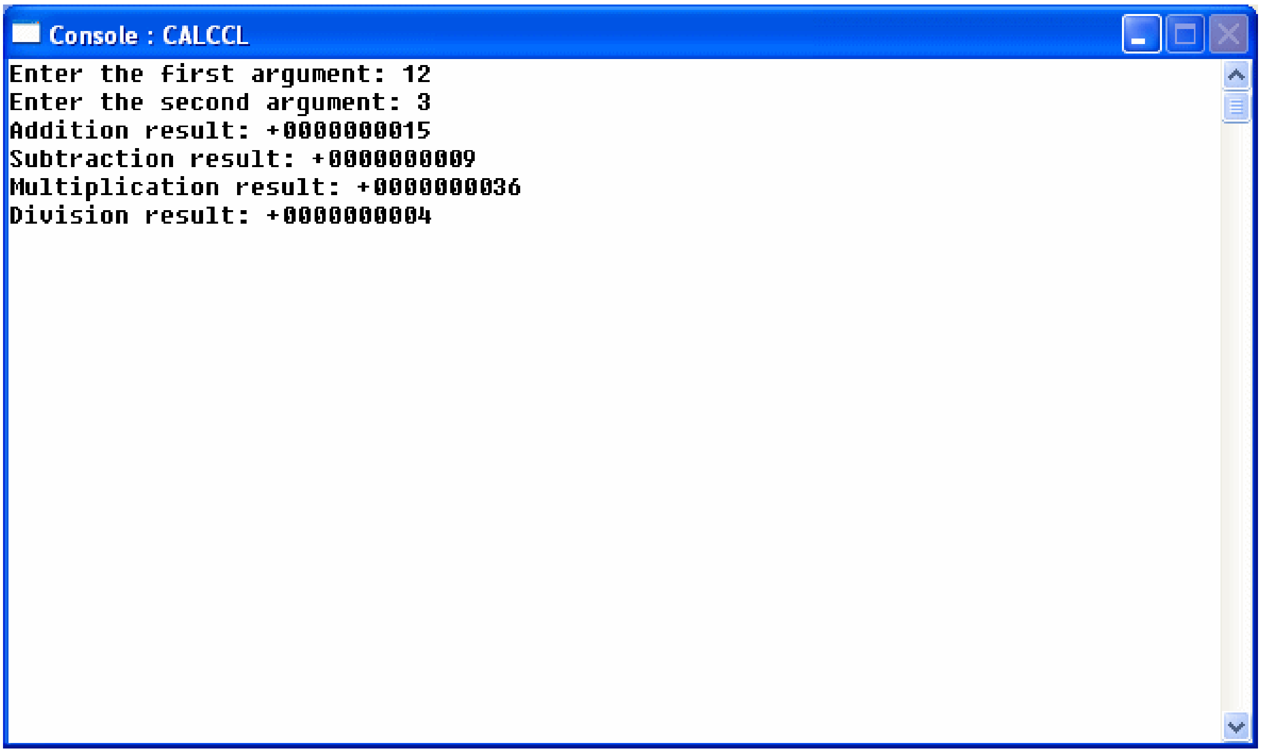
By following the procedure to this point, you have completed the work to build the CORBA client application.

However, if **Project** > **Build Automatically** was not previously selected (the **Build Automatically** menu item will have a check if it was selected), the application has not been built.

Select the CORBA server project in the **Dependency view** or **Structure view**, and select **Project** > **Project Build** from the menu bar. The CORBA server project is built.

8. Verifying Operation

Start the WorkUnit to which the CORBA server application has been deployed, select the project from the workbench view, and then select **Run** > **Run As** > **COBOL Application** from the menu to run the client application.



```
Enter the first argument: 12
Enter the second argument: 3
Addition result: +0000000015
Subtraction result: +0000000009
Multiplication result: +0000000036
Division result: +0000000004
```

11.3 Preparation for Developing CORBA Applications

The following programs are necessary to developing CORBA applications.

- Interstage Studio Workbench (*1)
- COBOL Plug-in (*2)
- Interstage Application Server Function or Client Function(*3)
- NetCOBOL Development Package

*1: Program provided by Interstage Studio Standard-J Edition, which is sold separately.

*2: Point to the "COBOL plug-in for Interstage Studio", which is contained in the NetCOBOL development package.

*3: Program provided by Interstage Application Server Enterprise Edition, which is sold separately.

Checking the environment settings of Interstage Application Server

The IDL file is compiled for CORBA application development. An environment that allows access to the CORBA server must be set up for this purpose.

Environment setup is not required for such development, which is combined with use of the server functions of Interstage Application Server. However, environment setup is required for such development, which is combined with use of the client functions.

When an IDL compiler error occurs during the build of the CORBA application, confirm the operating environment file of the CORBA service. For the content of the file, refer to "Interstage Application Server/Interstage Web Server Tuning guide".

Starting the Interstage basis service

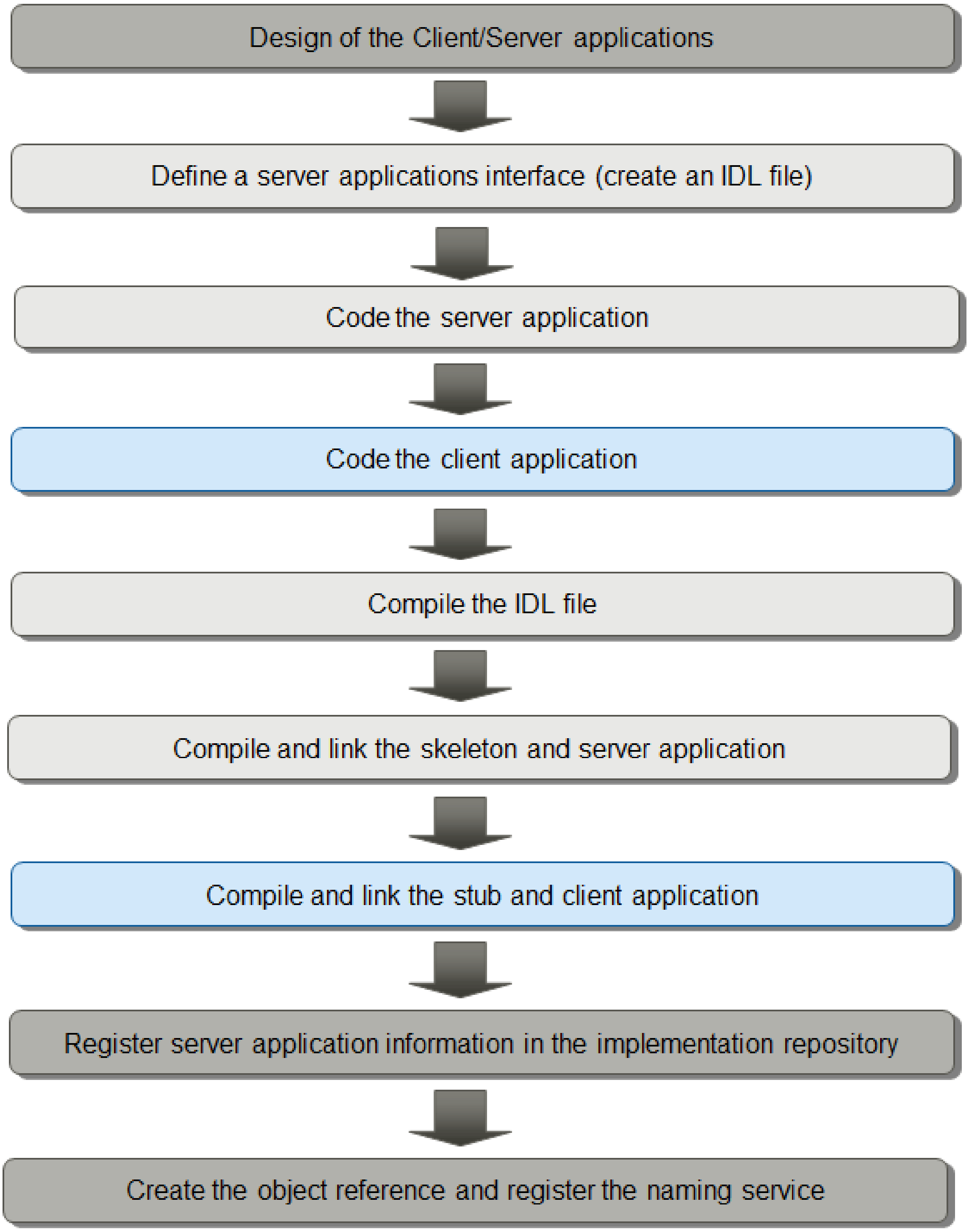
Before creating CORBA applications, it is necessary to start the J2EE execution environment service. Because the J2EE execution environment service is not started by default, use the Interstage basis service operation tool to start the service.

1. Select **Interstage** > **Studio** > **Interstage basis service operation tool** from the start menu. The Interstage basis service operation tool is started.
2. Select the **Using J2EE execution environment** checkbox of "Debugging environment used in localhost".

3. When the **Make only required service become starting state** button is available, click the button and start the service. When this button is not available, this operation is unnecessary.
4. Click the **Details** button. The Detail display area is displayed at the bottom of the screen.
5. Make sure that the < Necessary Service> in the detail display area are all starting, and then click the **Close** button. The Interstage basis service operation tool ends.

11.4 Flow of CORBA application development

The following figure shows the general CORBA application development sequence


Design of the Client/Server applications
Define a server applications interface (create an IDL file)
Code the server application
Code the client application
Compile the IDL file
Compile and link the skeleton and server application
Compile and link the stub and client application
Register server application information in the implementation repository
Create the object reference and register the naming service

▨ Common

□ Server Application Development

🟦 Client Application Development

1. Design the Client/Server Application.

   For the design of the CORBA application, refer to the "Interstage Application Server Distributed Application Development Guide (CORBA Service Edition)".

2. Define a server application interface (create an IDL file).

   Create a server application template and a template of the IDL file (interface definition file) required for creating a CORBA server application. Use the **CORBA server application generation wizard** offered in the COBOL plug-in.

3. Code the server application.

   Write implementation code in the template source of the business method class generated by the **CORBA server application wizard**. The main program and server application registration program can be used without modification.

4. Code the Client application.

   Create a COBOL project that will be a CORBA client application using the **COBOL Project wizard**.

   The processing to access the CORBA server can easily be added to the COBOL source file by using the template.

5. Compile the IDL file.

   The IDL file is compiled during the build of the CORBA server project and generates stub and skeleton files.

6. Compile and link the skeleton and server application.

   The skeleton and server application are compiled and linked at the build of the CORBA server project.

7. Compile and link the stub and client application.

   The stub and client application are compiled and linked at the build of the CORBA client project.

8. Register the server application information in the implementation repository.

   Registration of the implementation repository is essential. It stores information necessary for server application execution. Perform registration by using the Interstage Management Console or the CORBA service command on the server.

9. Create the object reference and register the naming service.

   A naming service name must be registered for each CORBA server application created with object-oriented COBOL. You can register it by using either of two methods. One method is to specify settings from the Interstage Management Console so that the name is automatically registered when a WorkUnit starts. The other method is to execute the CORBA service command for registration.

11.5 Development of CORBA Server Application

This section explains how to develop the CORBA server application.

11.5.1 Development procedure

11.5.1.1 Creation of CORBA server project

Note

Generation rules for CORBA server application targets

Targets of the main program (.EXE) and the library of business logic (.DLL) are created at the time of building on the CORBA server application.

The main program is created from the COBOL source that is set as a main program in the **Dependence view**, and other COBOL sources are all linked with library of business logic.

When creating UNICODE object

The repository file and library file used for UNICODE are provided in the Interstage Application Server. For details about developing a UNICODE application, refer to "NetCOBOL Users' Guide" and "Interstage Application Server Application Creation Guide (CORBA service)"

By specifying "Using Unicode" in the **CORBA server project generation wizard**, a build can be executed using the file used for UNICODE.

Use the **CORBA server project generation wizard** to create the CORBA server project for CORBA server application.

11.5.1.1.1 CORBA Server Project Wizard

Create a server application template and a template of the IDL file (interface definition file) required for creating a CORBA server project.

Note

Of the types of CORBA server applications, only applications with CORBA server as the target can be created.
The applications are actually created as exe and dll files.

Creating a project

Create a CORBA server project by following the procedure below.

1. Starting the Interstage Studio workbench.
2. Select **File** > **New** > **CORBA Server Project** from the menu bar. The **New CORBA Server Project wizard** is started.
3. Specify information in the **New CORBA Server Project wizard** as follows:
   - On the **CORBA Server Project** page, specify Basic Project Information.
   - On the **CORBA Server Project** page, define a target.
   - If the target definitions specify a project that uses a precompiler, then specify precompiler linkage information.
   - On the **Build Environment** page, define the build environment for the CORBA server.
   - On the **Select** page, select "Generate code".
4. Click the **Finish** button.

Table 11.1 Defining a Target

Option		Description
Target type		Specifies the target type of the COBOL application to be created.
	CORBA server	"CORBA server" is selected.

For the other information of target, refer to "Table 4.3 Defining the target".

Table 11.2 Build Environment

Option	Description
Using Unicode	Selected (checked) when Unicode is used.

11.5.1.2 Creating and Editing the template (IDL file and COBOL source file)

Create templates of an IDL file and a COBOL source file using the **CORBA server application wizard**.

11.5.1.2.1 Files with created templates

Template file		Description	Creating file name
IDL file(*1)		Interface definition language file	US*interface-name*.idl
COBOL source file	Main program source	CORBA server initial and termination processing	*interface-name*.cob
	Business method program source	Business method definitions	*module-name-interface*-IMPL.cob
	Server application registration program source	Server application registration	*module-name-interface*--INIT.cob
	Skeleton File generated with IDL compiler		*module-name-COBO-source-file*

*1: Interface Definition Language (IDL):

Format of the IDL file to be created

One module can be defined for one project. Defining a class for business methods within the module generates an IDL file interface declaration. For details on the IDL file, refer to the "Interstage Application Server Distributed Application Development Guide (CORBA Service Edition)". This section describes the format of the IDL file to be created and the declarations generated by the IDL file.

```
// Module declaration
module M1 {
    // Interface declaration
    interface intf1 {
         // Constant declaration
         const long a = 1 ;
         // Type declaration
         typedef sequence<long 10> b ;
         // Structure declaration
         struct c {
             short item1 ;
             long item2 ;
             long long item3 ;
         } ;
        // Exception declaration
        exception CDException {
            string CDExceptionMsg ;
            long  CDExceptionCode ;
        } ;
        // Operator declaration
        long op1(in short param1, out long param2, inout long param3)
        raises ( CDException );
    };
};
```

Table 11.3 Declarations generated by the IDL file

definition	Explanation
Module declaration	Declaration for a server application. One module is defined for one application.
Interface declaration	Declaration for a class. The class is for business method definition.
Constant declaration	Declaration of a constant used in a business method
Type declaration	Declaration of a type used in a business method. This declaration is used to define iteration items.

definition	Explanation
Structure declaration	Declaration of a structure. For object-oriented COBOL, this corresponds to a definition of group items. The **CORBA server application wizar**d does not allow a structure to be defined as a structure element type.

Exception declaration	Declaration of the contents of a record that is posted if an exception is thrown. The processing of actual exception notification must be coded within the program.
	The following element is automatically generated as an option. The contents of the element are fixed. Therefore, if modification is required, modify the IDL file created by the **CORBA server application generation wizard**.
	`exception CDException {`
	`string CDExceptionMsg ;`
	`long CDExceptionCode ;`
	`} ;`

Operator declaration	Generates one operator per business method declaration. If an exception declaration is selected as an option, a definition is generated for returning the declared record in the exception declaration in the event that an exception is thrown.

11.5.1.2.2 Edit of Files

After the model has been created, editing of the IDL file and the COBOL source file can be performed freely. However, it is necessary to synchronize the contents of the IDL file with the COBOL source file.

For example, it is necessary to modify both the IDL file and the COBOL source file when changing the predefined method definition.

IDL file

When changing the contents of the interface, modify it to be equivalent to the business method program.

Main program

In order to maintain the selected standard application format, modify only when it is necessary.

Business method program

Execute the procedures of each method.

For details about the adding process, refer to "Interstage Application Server Application Creation Guide (CORBA service)".

Server application registration program

The server application performs the registration in order to use it as a CORBA server. Use it without modification.

File used for skeleton that is created by IDL compiler

Use it without modification.

11.5.1.2.3 CORBA Server Application generation wizard

The **CORBA Server Application generation wizard** generates an interface definition file (IDL file) and a sample CORBA server application.

Start the **CORBA Server Application Skeleton Code wizard** by following the procedure below.

1. Select **File** > **New** > **Other** from the menu bar. The **New wizard** starts.
2. Select **COBOL** > **Source** > **CORBA Server Application** in the **New wizard**, and click the **Next** button. The **CORBA Server Application generation wizard** starts.

   Specify information in "New CORBA Server Application Skeleton Code" as follows:

   - On the **Module Declaration** page, specify information including a module name and exception declaration.
   - On the **Constant Declaration** page, specify information on the literals required for an interface declaration.
   - On the **Type Declaration** page, specify the type information required for an interface declaration.
   - On the **Structure Declaration** page, specify information on the structure used for an interface declaration.

- On the **Method Declaration** page, specify information on the method used for an interface declaration.

Table 11.4 Module Declaration

Option	Description
Project name	Specifies the project in which the generated CORBA server application source is to be stored. A project name must be specified.
Module name	Specifies the module name for the generated CORBA server application source. A module name must be specified.
Class name	Displays the class name for the generated CORBA server application. The class name is the same as the project name. A class name is generated as an interface name in the IDL file after the wizard is completed.
Generate exceptions	Specifies whether to make the default exception declaration in the generated CORBA server application. If this option is specified, the following declaration is generated in the IDL file: exception CDException { string CDExceptionMsg ; long CDExceptionCode ; } ;
Generate default code	Specifies whether to generate the default process for the generated CORBA server application. If this option is specified, the default process of CORBA server applications is generated in the program source.
Generate comments	Specifies whether to generate a comment in the generated CORBA server application. If this option is specified, a comment is generated in the IDL file.
Use precompiler	When the pre-compiler link information is set to the project of "Project name", this item is enabled. When checked, the extension of the COBOL source file becomes an extension set to the precompiler link information. For the precompiler link information, see "Table 4.4 Precompiler link information ".

Note

For IDL compilation during source generation, the Interstage service must be started in advance. If the wizard displays the message "The environment which compiles IDL is not prepare." on the **Module Declaration** page during source generation, click "Cancel" to quit the wizard. Start the Interstage service before restarting the source generation wizard.

Table 11.5 Constant Declaration

Option	Description
Type	Used to select the type of literal to be declared. For details on the types that can be defined and COBOL and IDL mapping, refer to "Table 11.6 Literal types".
Constant Name	Specifies the name of the literal being defined.
Initial Value	Specifies the default value of a literal. The initial value is generated in the IDL file. Specify Initial values of literals in the IDL format.
Add	Adds a new constant declaration.
Delete	Deletes the selected constant declaration.

Table 11.6 Literal types

Type that can be defined	COBOL	IDL
2-byte integer	PIC S9(4) COMP-5	short

Type that can be defined	COBOL	IDL
2-byte integer (unsigned)	PIC 9(4) COMP-5	unsigned short
4-byte integer	PIC S9(9) COMP-5	long
4-byte integer (unsigned)	PIC 9(9) COMP-5	unsigned long
Single precision floating-point number	COMP-1	float
Double precision floating-point number	COMP-2	double
Alphanumeric character	PIC X(1)	char
Boolean	PIC 1(1)	boolean
Alphanumeric character string	PIC X(n)	string

Table 11.7 Initial values of literals

Character literal	Character enclosed by single quotation marks (e.g., 'A')
String literal	Character string enclosed by double quotation marks (e.g., "ABC")
4-byte integer	Numeric value (e.g., 1)
Logical value	TRUE or FALSE

Table 11.8 Type Declaration

Option	Description
Type	Used to select the type of variable name to be declared. For details on the types that can be defined and COBOL and IDL mapping, refer to Table 11.9 variable-name types.
Variable Name	Specifies the name of the variable being defined.
No. of Digits/Characters	Specifies the total number of columns when the type is alphanumeric character string, national character string, or packed-decimal number.
Scale	Specifies the number of decimal places when the type is packed-decimal number.
No. of Repitions	Specifies the repetition count of a one-dimensional element.
Add	Adds a new type declaration.
Delete	Deletes the selected type declaration.

Table 11.9 variable-name types

Type that can be defined	COBOL	IDL
2-byte integer	PIC S9(4) COMP-5	short
2-byte integer (unsigned)	PIC 9(4) COMP-5	unsigned short
4-byte integer	PIC S9(9) COMP-5	long
4-byte integer (unsigned)	PIC 9(9) COMP-5	unsigned long
8-byte integer	PIC S9(18) COMP-5	long long
Packed-decimal number	PIC xx(n) PACKED-DECIMAL (*)	fixed
Single precision floating-point number	COMP-1	float
Double precision floating-point number	COMP-2	double
Alphanumeric character	PIC X(1)	char

Type that can be defined	COBOL	IDL
National character	PIC N(1)	wchar
Boolean	PIC 1(1)	boolean
Alphanumeric character string	PIC X(n)	string
National character string	PIC N(n)	wstring

*1: The COBOL type of a packed-decimal number depends on the combination of the total number of columns and the number of decimal places. For details, refer to "COBOL Programming Guide" in the "Interstage Application Server Distributed Application Development Guide (CORBA Service Edition)".

Table 11.10 Structure Declaration

Option	Description
Structures	Lists defined structures. The structure selected from this list is subject to "Edit" or "Delete".
Add	Adds a new structure. Clicking **Add** displays the "Table 11.11 Structure Definition" screen.
Edit	Modifies definition information on the selected structure. Clicking "Edit" displays the "Table 11.11 Structure Definition" screen.
Delete	Deletes the selected structure.

Table 11.11 Structure Definition

Option	Description
Structure name	Specifies the name of the structure being defined.
Variable Name	Specifies the name of the variable being defined.
Type	Used to select the type of variable name to be declared. For details on the types that can be defined and COBOL and IDL mapping, refer to "Table 11.12 Variable-name type".
No. of Digits/ Characters	Specifies the total number of columns when the type is alphanumeric character string, national character string, or packed-decimal number.
Scale	Specifies the number of decimal places when the type is packed-decimal number.
Sequence Size	Specifies the repetition count of a one-dimensional element.
Add	Adds a new element to the structure.
Delete	Deletes the specified structure element.

Table 11.12 Variable-name type

Type that can be defined	COBOL	IDL
2-byte integer	PIC S9(4) COMP-5	short
2-byte integer (unsigned)	PIC 9(4) COMP-5	unsigned short
4-byte integer	PIC S9(9) COMP-5	long
4-byte integer (unsigned)	PIC 9(9) COMP-5	unsigned long
8-byte integer	PIC S9(18) COMP-5	long long
Packed-decimal number	PIC xx(n) PACKED-DECIMAL (*1)	fixed
Single precision floating-point number	COMP-1	float
Double precision floating-point number	COMP-2	double

Type that can be defined	COBOL	IDL
2-byte integer	PIC S9(4) COMP-5	short
Alphanumeric character	PIC X(1)	char
National character	PIC N(1)	wchar
Boolean	PIC 1(1)	boolean
Alphanumeric character string	PIC X(n)	string
National character string	PIC N(n)	wstring
Any type (*2)	Any type	Any type

*1: The COBOL type of a packed-decimal number depends on the combination of the total number of columns and the number of decimal places. For details, refer to "COBOL Programming Guide" in the "Interstage Application Server Distributed Application Development Guide (CORBA Service Edition)".

*2: Enter the type specified in a type definition (type of repetition item). "Any type" must be one that is defined in a type definition.

Table 11.13 Method Declaration

Option	Description
User methods	Lists defined user methods. The method selected from this list is subject to "Edit" or "Delete".
Details	Displays detailed information on the user method selected from **User Method Definition**.
Add	Adds a new user method (business method). Clicking **Add** displays the "Table 11.14 User Method Definition" screen.
Edit	Modifies definition information on the method selected from **User Method Definition**. Clicking "Edit" displays the "Table 11.14 User Method Definition" screen.
Delete	Deletes the user method (business method) selected from **User Method Definition**.

Table 11.14 User Method Definition

Option	Description
Method name	Specifies the name of the user method being defined.
Return type	Used to select the type of method return value. For details on the types that can be defined and COBOL and IDL mapping, refer to Table 11.15 Return-value types.
No. of Digits/ Characters	Specifies the total number of columns for the return value. Specifies the total number of columns when the type of the return value is alphanumeric character string, national character string, or packed-decimal number.
Scale	Specifies the number of decimal places for the type of the return value. Specify the number of decimal places when the type of the return value is packed-decimal number.
Throws an exception	Specifies whether to generate a process that reports multiplication and division exceptions in the generated CORBA server application. If this option is specified, a process that reports multiplication and division exceptions is generated in the program source.
Parameters	Can be used to add and delete parameters. The variable and type of each parameter can be edited. A parameter type can be selected from the list of types or specified directly. The character string specified for a variable name cannot consist of a mixture of en-size and em-size characters. No em-size character can be used to specify the type.
Variable Name	Specifies the name of the variable being defined.
Type	Used to select the type of variable name to be declared. For details on the types that can be defined and COBOL and IDL mapping, refer to Table 11.16 Variable-name types.

Option	Description
No. of Digits/ Characters	Specifies the total number of columns when the type is alphanumeric character string, national character string, or packed-decimal number.
Scale	Specifies the number of decimal places when the type is packed-decimal number.
Parameter Type	Used to select the parameter type. If in is selected, an input parameter is generated in the program source. If out is selected, an output parameter is generated in the program source. If inout is selected, an input-output parameter is generated in the program source.
Add	Adds a new parameter to the user method.
Delete	Deletes the selected user method parameter.

Table 11.15 Return-value types

Type that can be defined	COBOL	IDL
None	-	oneway void
2-byte integer	PIC S9(4) COMP-5	short
2-byte integer (unsigned)	PIC 9(4) COMP-5	unsigned short
4-byte integer	PIC S9(9) COMP-5	long
4-byte integer (unsigned)	PIC 9(9) COMP-5	unsigned long
8-byte integer	PIC S9(18) COMP-5	long long
Packed-decimal number	PIC xx(n) PACKED-DECIMAL)*1)	fixed
Single precision floating-point number	COMP-1	float
Double precision floating-point number	COMP-2	double
Alphanumeric character	PIC X(1)	char
National character	PIC N(1)	wchar
Boolean	PIC 1(1)	boolean
Alphanumeric character string	PIC X(n)	string
National character string	PIC N(n)	wstring
Any type (*2)	Entered type	Entered type

*1: The COBOL type of a packed-decimal number depends on the combination of the total number of columns and the number of decimal places. For details, refer to "COBOL Programming Guide" in the "Interstage Application Server Distributed Application Development Guide (CORBA Service Edition)".

*2: Enter the type specified in a type definition or structure definition.

Table 11.16 Variable-name types

Type that can be defined	COBOL	IDL
2-byte integer	PIC S9(4) COMP-5	short
2-byte integer (unsigned)	PIC 9(4) COMP-5	unsigned short
4-byte integer	PIC S9(9) COMP-5	long
4-byte integer (unsigned)	PIC 9(9) COMP-5	unsigned long
8-byte integer	PIC S9(18) COMP-5	long long
Packed-decimal number	PIC xx(n) PACKED-DECIMAL (*1)	fixed

Type that can be defined	COBOL	IDL
Single precision floating-point number	COMP-1	float
Double precision floating-point number	COMP-2	double
Alphanumeric character	PIC X(1)	char
National character	PIC N(1)	wchar
Boolean	PIC 1(1)	boolean
Alphanumeric character string	PIC X(n)	string
National character string	PIC N(n)	wstring
Any type (*2)	Entered type	Entered type

*1: The COBOL type of a packed-decimal number depends on the combination of the total number of columns and the number of decimal places. For details, refer to "COBOL Programming Guide" in the "Interstage Application Server Distributed Application Development Guide (CORBA Service Edition)".

*2: Enter the type specified in a type definition or structure definition.

The **CORBA Server Application wizard** creates the template using the BOA interface. For the details of BOA interface, refer to "interstage Application Server application guide (CORBA Service)".

11.5.1.2.4 IDL File wizard

A client interface (IDL) file that describes a structure can be created based on CORBA/IDL object-oriented COBOL mapping.

Start the **IDL File wizard** by following the procedure below.

1. Select **File** > **New** > **Other** from the menu bar. The **New wizard** is started.
2. Select **COBOL** > **Source** > **IDL File** in the **New wizard.** The **IDL File wizard** is started.
3. Specify basic information for the IDL file on the **IDL File Information** page.

Table 11.17 Basic Information for an IDL File

Option	Description
Project name	Specifies the project in which the IDL file is generated.
File name	Specifies a file name.

4. Click the **Finish** button. The IDL file is created.

11.5.1.3 Build

Build includes the following types of processing executed in the order listed:

- Generating stub and skeleton files through IDL file compilation
- Compiling stub and skeleton files
- Compiling the program source
- Linking

Option set automatically

When a CORBA server project is created, the options listed below are automatically set as build options.

Compiler option

- Unicode is not used

```
LIB(Interstage Application Server installation folder\odwin\include\oocob)
REPIN(Interstage Application Server installation folder\odwin\rep
THREAD(MULTI)
```

- Unicode is used

```
LIB(Interstage Application Server installation folder\odwin\include\oocob)
REPIN(Interstage Application Server installation folder\odwin\rep\Unicode)
THREAD(MULTI)
RCS(UTF16,LE)
```

Linker option

The following libraries are linked:

- Unicode is not used

```
Interstage Application Server installation folder\odwin\lib\odoocobsv.lib
Interstage Application Server installation folder\odwin\lib\odcnsoocob.lib
```

- Unicode is used

```
Interstage Application Server installation folder\odwin\lib\odoocobsvuc.lib
Interstage Application Server installation folder\odwin\lib\odcnsoocobuc.lib
```

To set other options, select **Build** from **Properties** of project, and specify the options.

The **Build** page is displayed as follows:

1. Select the CORBA server project in **Dependency view** or **Structure view**.
2. Select **File** > **Property** from the menu bar or select the **Property** from the context menu. The **Properties** dialog box appears.
3. Select **Build** from the left pane. The **Build** page appears.

Project Build

Select "Save" from the context menu of the edit window in the source program and the CORBA server application is built automatically. However, if **Project** > **Build Automatically** was not previously selected (the **Build Automatically** menu item will have a check if it was selected), the application is not built. In this case, select the project in the **Dependency view** or the **Structure view**, and select **Project** > **Project Build** to build the project.

The IDL compiler compiles the IDL file during building. The skeleton and other files required for server applications are generated and added as compilation targets.

Configuring Build Tool

When the CORBA server project is developed, it registers the following in the build tool.

- IDL compiler
- COBOL compiler
- linker

In the steps below, the **Build Tool** page is displayed and sets the build tool configuration.

1. Select the CORBA server project in the **Dependency view** or the **Structure view**.
2. Select the **Properties** option from the context menu or select **File** > **Properties** from the menu bar. The **Properties** dialog box appears.

3. Select "Build Tools" from the left pane. A list of build tools associated with the application is displayed. Build executes the build tools in the displayed sequence.

Option	Description
Add from Application	Add build tool from the application type. Click on "Add from Application" to include the build tools associated with other application types. The **Add build tools From Application** dialog box with a list of applications appears. The list will show only those applications that have at least one build tool not existing in the list and having the same project type. From the list of applications, select the application from which the build tools should be added.
Add Build Tool	Add a build tool to the project. Click on **Add Build Tool** to include a build tool into the project. The **Add Build Tool** dialog box with a list of all the build tools appears. From this list, select the build tool to be added.
Remove	Removes the selected build tool. Note: Cannot remove the Java compiler in the build tool list.
Up	Changes the execution sequence of the selected build tool. The selected build tool is executed before the execution of the previous build tool.
Down	Changes the execution sequence of the selected build tool. The selected build tool is executed after the execution of next build tool.
Restore Defaults	Restores the build tool configuration to that at the project creation time.
Apply	Changed contents are applied.

Note

The build tool to be added should have the following prerequisites

- The build tool related to the specified build tool should be available.
- Corresponding to the specified build tool, the specified execution sequence should be correct.

When you want to add a build tool or change the executing order of the build tool, be careful about the execution sequence and then make the settings.

- The following build tool is executed before the COBOL compiler.
  - IDL Compiler
  - Precompiler
- The following build tools are executed after the COBOL compiler.
  - Resource Compiler
  - Linker

Point

To make detailed settings for added build tools such as the IDL compiler and resource compiler, click "Apply".

IDL Compiler

You can use the IDL compiler to generate source codes from the IDL definition file. The generated source codes are client stubs and object skeletons. The client stubs are used to develop client program. The object skeletons are used to implement CORBA objects.

Select **Build Tool** > **IDL Compiler** from the left pane of the property dialog box. The **IDL Compiler** page displays.

Option	Description
Include file folders	Specify folder name that is used for searching files specified in "#include" statement in IDL file.
Other options	Specify any additional compiler options. When specifying multiple options, delimit them with a space.
Register to interface repository	Specify this option when you want to register information to the interface repository.

11.5.1.4 Debugging

The program debugging is performed after creating client applications.

On Interstage Application Server, which is installed in the same terminal, use CORBA work unit starting-organization and debug the server application. The debugging of the CORBA server application is the same as attach debugging.

See

For details about debug perspective, refer to "7.2 Debug Perspective".

11.5.1.4.1 Setting the build mode

To debug the program, select **Debug** in "Build mode".

1. Select the CORBA server project in the **Dependency view** or the **Structure view**, and select **File** >**Property** from the menu bar or select the **Property** from the context menu. The **Property** dialog box displays.
2. Select **Target** from the left pane. The **Target** page appears.
3. Specify **Debug** in "Build mode".

If the build mode is changed, the project must be rebuilt.

11.5.1.4.2 Start debugging

1. Select a CORBA server project from the **Dependency view** or the **Structure view**.
2. Select **Run** > **Debug** from the menu bar, or click ▾ of 🐞▾ on the toolbar. The **Debug** dialog box is displayed.
3. Select and double-click **CORBA WorkUnit** in the left pane.
4. The launch configuration setting page is displayed in the right pane.
5. The default launch configuration name is displayed at "Name". You can change it to any name you want.
6. Specify settings on the **Main** tab as follows:
    a. Enter a CORBA WorkUnit name in "CORBA WorkUnit name".
    Alternatively, you can select one from the list of CORBA WorkUnits created by Interstage Application Server.
    b. To automatically deploy resources to Interstage Application Server, check "Deploy before launching".
    c. Enter the CORBA server project name in "Project to be deployed", or click "Browse" and select the CORBA server project.
7. Click **Debug** to start debugging.
8. To quit debugging, select "Terminate" in the **Debug view** to stop the CORBA WorkUnit.

Select a CORBA server project from the **Dependency view** or the **Structure view**, and select **Run** > **Debug** > "CORBA WorkUnit" from the menu bar. This starts debugging under the default settings.

Note

- If services that are required for management of Interstage (Interstage management services) have not started, the list of CORBA WorkUnits in "CORBA WorkUnit name" will not be displayed correctly. In cases such as this, do not create the CORBA WorkUnit launch configuration until after the Management Services have started.
  For starting the Interstage management service, refer to "Checking the environment settings of Interstage Application Server"

- If services that are required for connection to the local Interstage Application Server are stopped by any debugging that uses the CORBA WorkUnit launch configuration, start those services. In an OS that has a User Account Control (UAC) function, when the service starts, a dialog box will appear telling you that you need administrator permissions. In such a case, change your permissions according to the information displayed.

Information

Settings in the .deployment file

CORBA WorkUnits can automatically deploy resources before debugging. They use the information provided in .deployment files to implement resource deployment. The .deployment file information items are as follows.

Tag	Description
implementation-repository-id	Specifies an implementation repository ID.
interface-repository-id	Specifies an interface repository ID.
binding-name	Specifies a naming service registration name.
workunit-name	Specifies a CORBA WorkUnit name.

A .deployment file is created based on the module name and interface name specified in the wizard when the **CORBA Server Application wizard** is executed.

11.5.1.4.3 Creating the CORBA work Unit

The methods for creating a CORBA work unit (work unit of CORBA application) are as follows.

- using Interstage management console

- using the work unit generation command (isaddwudef) according to the CORBA work unit definition file

For details about the CORBA work unit creation method from the Interstage management console, refer to "Interstage Application Server Operation Guide (basic)". This section explains the methods of creating a CORBA work unit (MyCORBADebug) using the work unit generation command according to the CORBA work unit definition file for local debugger and command.

Execution of work unit generation command (isaddwudef)

To use the CORBA work unit definition file as an argument, execute the following commands in a command prompt.

```
isaddwudef NetCOBOL install folder\Samples\CORBA\MyCORBADebug.wu
```

When using spaces in the CORBA work unit definition file name, enclose the file name in double quote characters. For details about the isaddwudef command, refer to "Interstage Application Server Reference Manual (commands)"

Note

- Information such as the work folder is set in the "Path", "Current Directory" property of the CORBA work unit definition file. Generally, although the value is set from the information from the Interstage installed destination folder, correct the setting when errors exist in the path

- Settings for environment variables PATH and CLASSPATH are necessary on creation of the CORBA work unit. When errors happen that involve these environment variables, set environment variables PATH and CLASSPATH, and restart your computer.

11.5.1.5 Executing

Use the following procedure to execute the CORBA server application .

1. Copy the execution resource (execution file and dynamic link library) into the execution environment.
2. Create the application operation current directory for the start of work unit.
3. Deploy the CORBA server application.
4. Start the work unit.
5. Execute the client application.

For detailed procedures, refer to "5. Running the Program" in the tutorial.

11.5.2 Notes

This section provides notes on developing CORBA server applications.

11.5.2.1 Usable Data Types

The table below lists CORBA data types and their corresponding data types in other languages.
For details on data types that can be defined, refer to the "Interstage Studio User's Guide".

Table 11.18 COBOL (object-oriented COBOL) mapping

CORBA type	COBOL (object-oriented COBOL) mapping type	COBOL native type
long	CORBA-long	PIC S9(9) COMP-5
unsigned long	CORBA-unsigned-long	PIC 9(9) COMP-5
short	CORBA-short	PIC S9(4) COMP-5
unsigned short	CORBA-unsigned-short	PIC 9(4) COMP-5
long long	CORBA-long-long	PIC S9(18) COMP-5
unsigned long long	CORBA-unsigned-long-long	PIC 9(18) COMP-5
float	CORBA-float	COMP-1
double	CORBA-double	COMP-2
char	CORBA-char	PIC X
wchar	CORBA-wchar	PIC N
octet	CORBA-octet	PIC X
boolean	CORBA-boolean	PIC 1(1)
fixed <m+n,n>	Cannot be used	PIC S9 (m+n,n) PACKED-DECIMAL
string (fixed length)	PIC X(n)	PIC X(n)
string (variable length)	Cannot be used	Cannot be used
wstring (fixed length)	PIC N(n)	PIC N(n)
wstring (variable length)	Cannot be used	Cannot be used
enum	CORBA-enum	PIC 9(10) COMP-5
any	Cannot be used	Cannot be used
Structure (fixed length)	Group item	Group item
Structure (variable length)	Class	Class
Union	Class	Class
sequence type (fixed length)	Class	Class

CORBA type	COBOL (object-oriented COBOL) mapping type	COBOL native type
sequence type (variable length)	Class	Class
array type	Cannot be used	Cannot be used

The **CORBA server application generation wizard** does not support the data types listed below. To use these data types, create templates and modify the relevant IDL file and program source according to the "Interstage Application Server Distributed Application Development Guide (CORBA Service Edition)".

- octet
- string (variable length)
- wstring (variable length)
- enum
- Structure (variable length)
- Union
- sequence type (variable length)

11.5.2.2 Operation of Iteration Items

Iteration items are mapped to the sequence type as described below.

Table 11.19 Iteration item mapping

Number of dimensions	Object-oriented COBOL
1	sequence type (fixed length)
2 to 5	Cannot be used

Mapping a iteration definition to the IDL file

Mapping of an iteration definition to the IDL file is explained below.

Mapping one-dimensional items to the sequence type

One-dimensional items are mapped to the sequence type.

For object-oriented COBOL

The following example shows mapping with a 4-byte integer specified for the data type, "a" for the type name, and 10 for the iteration count.

```
    typedef sequence<long, 10> a ;
```

Operation of iteration items

Operation of iteration items is explained below.

One dimension (object-oriented COBOL)

Operation of iteration items is shown below. When the following IDL file is compiled, the sequence type is mapped to a sequence class.

IDL file

```
module ODsample {
      typedef sequence<long,10> sampleseq;
      interface seqtest {
               sampleseq op1(in sampleseq param1,
                        out sampleseq param2,
                        inout sampleseq param3);
      };
};
```

The factory method, object methods, and properties of the sequence class are listed below.

Table 11.20 Sequence class

Category	Method/property name	Function
Factory method	NEW-WITH-LENGTH	Creates a sequence class of the specified length
Object method	GET-VALUE	Obtains the value of the element of the specified number
	SET-VALUE	Sets a value for the element of the specified number
	CLONE	Makes a copy of a sequence class
Property	SEQ-MAXIMUM	Gets the value of the maximum sequence length
	SEQ-LENGTH	Gets or sets the sequence length value

Example of using iteration items

```
 METHOD-ID. OP1 AS "OP1" OVERRIDE.
* <IDL-INFO-START>
* sampleseq op1(in sampleseq param1,out sampleseq param2,inout sampleseq param3)
* <IDL-INFO-END>
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 SEQ-VALUE TYPE  CORBA-LONG.
 01 I         TYPE  CORBA-UNSIGNED-LONG.

 LINKAGE SECTION.
 01 RETURN-VALUE  TYPE ODSAMPLE-SAMPLESEQ.
 01 PARAM1  TYPE ODSAMPLE-SAMPLESEQ.
 01 PARAM2  TYPE ODSAMPLE-SAMPLESEQ.
 01 PARAM3  TYPE ODSAMPLE-SAMPLESEQ.
 PROCEDURE DIVISION
         USING
                   PARAM1
                   PARAM2
                   PARAM3
         RETURNING RETURN-VALUE
                   .

*  IN PARAMETER
     PERFORM VARYING I FROM 1 BY 1 UNTIL I > 10
       INVOKE PARAM1 "GET-VALUE" USING I RETURNING SEQ-VALUE
     END-PERFORM.

*  OUT PARAMETER
     INVOKE SEQUENCE-LONG-10 "NEW" RETURNING PARAM2.
     MOVE 10 TO SEQ-LENGTH OF PARAM2.
     PERFORM VARYING I FROM 1 BY 1 UNTIL I > 10
       COMPUTE SEQ-VALUE = I * 100
       INVOKE PARAM2 "SET-VALUE" USING I SEQ-VALUE
     END-PERFORM.

*  INOUT PARAMETER
     PERFORM VARYING I FROM 1 BY 1 UNTIL I > 10
       INVOKE PARAM3 "GET-VALUE" USING I RETURNING SEQ-VALUE
       COMPUTE SEQ-VALUE = SEQ-VALUE * 100
       INVOKE PARAM3 "SET-VALUE" USING I SEQ-VALUE
     END-PERFORM.

*  RESULT
     INVOKE SEQUENCE-LONG-10 "NEW" RETURNING RETURN-VALUE.
     MOVE 10 TO SEQ-LENGTH OF RETURN-VALUE.
     PERFORM VARYING I FROM 1 BY 1 UNTIL I > 10
       COMPUTE SEQ-VALUE = I * 10000
       INVOKE RETURN-VALUE "SET-VALUE" USING I SEQ-VALUE
     END-PERFORM.

 END METHOD OP1.
```

Point

As for the sequence type, in cases where the mode is "in" or "inout", a data area already exists on the calling side and need not be allocated. However, in cases of "out" mode or the return value, the server application must allocate a data area.

11.5.2.3 Operating Structures

Structures are mapped to group items as described below.

Table 11.21 Structure mapping

Type	Object-oriented COBOL
Structure (fixed length)	Group item
Structure (variable length)	Cannot be used

An example of using structures when the following IDL file is compiled, is shown below.

IDL file

```
module ODsample {
      struct samplestruct {
               long item1;
               long item2;
      };
      interface structtest {
               samplestruct op1(in samplestruct param1,
                         out samplestruct param2,
                         inout samplestruct param3);
      };
};
```

Example of structure items

```
 METHOD-ID. OP1 AS "OP1" OVERRIDE.
* <IDL-INFO-START>
* samplestruct op1(in samplestruct param1,out samplestruct param2,inout samplestruct param3)
* <IDL-INFO-END>
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 LINKAGE SECTION.
 01 RETURN-VALUE TYPE ODSAMPLE-SAMPLESTRUCT.
 01 PARAM1 TYPE ODSAMPLE-SAMPLESTRUCT.
 01 PARAM2 TYPE ODSAMPLE-SAMPLESTRUCT.
 01 PARAM3 TYPE ODSAMPLE-SAMPLESTRUCT.
 PROCEDURE DIVISION
         USING
                   PARAM1
                   PARAM2
                   PARAM3
         RETURNING RETURN-VALUE
                   .

*   IN PARAMETER
*   OUT PARAMETER
*  INOUT PARAMETER
     MOVE ITEM1 OF PARAM1 TO ITEM1 OF PARAM2.
     MOVE ITEM2 OF PARAM3 TO ITEM2 OF PARAM2.

     MOVE 2 TO ITEM1 OF PARAM3.
     MOVE 3 TO ITEM2 OF PARAM3.

*  RESULT
     MOVE 4 TO ITEM1 OF RETURN-VALUE.
     MOVE 5 TO ITEM2 OF RETURN-VALUE.

 END METHOD OP1.
```

11.5.2.4 Inheritance

The IDL generator can generate neither inheritance code nor #include statements.
To inherit a module or interface, edit the IDL file, using an editor to add coding of the inherited part.

11.5.2.5 Multi-instance System

For a program to be operated on multi-instance system, it is mandatory that the ORB initialization part of the program is modified.
For details about the multi-instance system and ORB initialization method, refer to the "Interstage Application Server Distributed Application Development Guide (CORBA Service Edition)" and "Interstage Application Server Reference Manual (Command Edition)".

11.5.2.6 Database Access

Following are two methods for creating the CORBA server application that performs database access .

- Create the class library for accessing the database in advance, and create the CORBA server application that uses this class. Class library can be used as a component in other applications.

- Describe the instructions that access the database directly in the business method of the CORBA server application.

For details and notes about the database access method (ESQL/COBOL etc.) provided by NetCOBOL, refer to "NetCOBOL Users' Guide" and "NetCOBOL Software Instructions".

11.6 Development of CORBA Client Application

This section explains how to develop CORBA client applications.

11.6.1 Development procedure

11.6.1.1 Creating the CORBA client project

This section explains how to develop CORBA client applications.

11.6.1.1.1 COBOL Project Wizard

The **COBOL Project wizard** can be used to create CORBA client applications written in COBOL.
Use the **COBOL Project wizard** to define a target and set up a build environment.

Create a COBOL project by following the procedure below.

1. Start the Interstage Studio workbench.

2. Select **File** > **New** > **COBOL Project** from the menu bar. The **New Project wizard** is started.

3. Specify information in the **New Project wizard** as follows:

    - On the **COBOL Project** page, specify Basic Project Information.

    - On the **COBOL Project** page, define a target.

- If the target definitions specify a project that uses a precompiler, then specify precompiler linkage information.
- On the **Build Environment** page, define the build environment for CORBA clients.
- On the **Skeleton Code Selection** page, select "Generate code". In "Available skeleton codes", select the source type to be created.
  - COBOL Source
  - Object-Oriented COBOL Source

For details on the **COBOL source generation wizard** and **Object-oriented COBOL source generation wizard**, refer to "4.2.4 COBOL source generation wizard" and "4.2.5 Object-oriented COBOL source generation wizard."

Table 11.22 Basic Project Information

Option	Description
Project name	Specify name of the project.(e.g. "CALCLL")
Project contents	Specify the save destination for project resources.
Create new project in workspace	Saves the project resources to the workspace folder.
Create project from existing source	Saves the project resources to a location outside the workspace folder. You can select the resource storage folder by clicking "Browse".

Table 11.23 Defining a Target

Option	Description
Target type	Specifies the target type of the COBOL application to be created. - To create an executable file (exe), select "Executable". - To create a dynamic link library (dll), select "Dynamic-link library".
Using the runtime initialization file(COBOL85.CBR) under DLL	If the target type is dynamic link library, then specify whether an initialization file for DLL-specific execution is to be used. If this option is selected, then the dynamic link library uses an initialization file for DLL-specific execution.
Target name	Specifies the file name for the target file (exe/dll) that is created after linkage. (e.g. CALCCL)
Application Format	Specifies the type of console (COBOL console/system console) used by the application to create an executable file (exe).
Use precompiler	Select this item if you are generating a project that uses a precompiler.
Text file encoding(*1)	Select the text file encoding to use when creating a new file for the project.

*1: To change the "Text file encoding" after the project is created, follow the procedure below:

1. Select the project. In the **Dependency view** or the **Structure view**, and select **Property** from the context menu. The **Property** dialog box displays.
2. Select "Resource" in the left pane. The **Resource** page displays.
3. Select the encoding to be changed in "Text file encoding" of the **Resource** page, and click the **OK** button.

Note

- Changing the encoding on the **Info** page at "Text file encoding" does not change the encoding of existing files in the project.
- NetCOBOL V10.0.0 or later must be installed in order to build COBOL source files using the text file encoding "UTF-8".

Table 11.24 Precompiler linkage information

Option	Description
Precompiler command	Specify the name of the command that runs the precompiler.
Precompiler parameters	Specify parameters of the precompiler command.
Precompiler source extension	Specify the extension of the precompiler input source file. Note that the following extensions cannot be specified: cobol cob cbl lcai
Precompiler output source extension	Select the extension of the precompiler output source file.
Use original source line numbers for error messages	If this item is checked, the COBOL compiler error messages are displayed using the line number in the original source, rather than the line number of the preprocessed source. (The INSDBINF command is invoked (*1).) The default setting is unchecked.
INSDBINF parameters	Specify parameters of the INSDBINF command that reflects line correction information on an input source for the precompiler to the COBOL source file created by the precompilation. This option does not need to be specified, since the input and output source file names are determined from the name of the precompiler input source file.

*1: For details on the INSDBINF command, refer to "6.2.2 INSDBINF command".

For details on the precompiler linkage information, refer to "6.2.3.2 Setting and changing precompiler link information".

Table 11.25 Build Environment

Option		Description
Set build environment for CORBA client		Specifies the build environment when this project is used as a CORBA client.
	Language(*1)	Specifies the CORBA client description language. Select "Object-Oriented COBOL" or **COBOL**.
	Object type(*1)	Specifies the object thread format. Select "Multi-thread" or "Single thread".
Use CDCORBA class(*1)		Specifies the CORBA client creation support class (CDCORBA) provided by Interstage Studio when it is used.

*1: If "Set build environment for CORBA client" is checked, each optional button and the checkbox become effective.

Table 11.26 Skeleton Code Selection

Option	Description
Do not generate code	When this option is selected, the project is created without the generation of any initial source code.
Generate code	Creates a project through use of a wizard that generates sample source code. Select the skeleton code type from the "Available skeleton codes" list.
Available skeleton codes	If "Generate Code" is selected, "Available skeleton codes" list is enabled to select the initial skeleton code generation.

11.6.1.2 Creating and Editing the COBOL source file

The COBOL source file is created using the COBOL source wizard. After the creation, processing is added.

- Initialization of CORBA(ORB, Naming Service)
- Search the CORBA server object
- Invoke the CORBA server application method

This section shows and explains a sample program that receives two parameters from a console, transfers them to a server program, and displays the result on the console.

IDL file of the CORBA server application

The server application receives two parameters and returns the result of their addition as a return value. The registered naming service name is assumed to be "Sample::POAintf".

```
// Module declaration
module SAMPLE {
   // User interface declaration
   interface CALCULATE_ADD {
           long CALCULATE(in long param1,in long param2);
   };
};
```

11.6.1.2.1 Sample coding of object-oriented COBOL program (static invocation)

The program shown below is used as an example to explain the points on creating object-oriented COBOL programs using static invocation interface.

Points on creating the programs

Code the CONFIGURATION SECTION as follows:

- REPOSITORY
  - COPY CORBA--REP (provided by the CORBA service)
  - COPY CosNaming--REP (provided by the CORBA service)
  - COPY IDL-file-name--REP (generated from the IDL file)
  - CLASS CDCORBA (provided with this product)
- SYMBOLIC CONSTANT
  - COPY CORBA--CONST (provided by the CORBA service)
  - COPY COSNAMING--CONST.
  - COPY project-name--CONST (generated from the IDL file)

Define the variables used for ORB initialization in the WORKING-STORAGE SECTION.

Declare the following variables:

- W-OBJECT (CORBA-OBJECT type object variable that stores the results of object acquisition by the server application)
- W-TARGET (server object type object variable generated from the IDL file): Generated as "*module name-interface name*"

Code the following types of processing, which can be entered from templates:

- ORB initialization
- Naming service initialization
- Server object acquisition
- Target object acquisition

- Method invocation by server application(*1)

*1: When you input the method invocation using the template, use the "CORBA Server Objects".

Main program: CALCULATE_ADD_CLIENT.cob

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. MAIN.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.

   REPOSITORY.
* Standard library for ObjectDirector (for repository declaration)
     COPY CORBA--REP.
* Standard library for naming service (for repository declaration)
     COPY CosNaming--REP.
* Library output by IDL compiler (for repository declaration)
     COPY USCALCULATE_ADD--REP.
* CORBA client development class provided by Interstage Studio
     CLASS CDCORBA.

  SPECIAL-NAMES.
     ARGUMENT-NUMBER IS argument-number
     ARGUMENT-VALUE  IS argument-content
     SYMBOLIC CONSTANT
* Standard library for ObjectDirector (for constant declaration)
     COPY CORBA--CONST.
* Standard library for naming service (for constant declaration)
     COPY COSNAMING--CONST.
* Library output by IDL compiler (for constant declaration)
     COPY USCALCULATE_ADD--CONST.
     .

 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  L-APL-NAME PIC X(50) VALUE "CALCULATE_ADD_CLIENT".
 01 W-OBJECT OBJECT REFERENCE CORBA-OBJECT.
 01 W-TARGET  OBJECT REFERENCE SAMPLE-CALCULATE_ADD.

 01 W-PARAM1 PIC S9(9) COMP-5.
 01 W-PARAM2 PIC S9(9) COMP-5.
 01 W-RETURN PIC S9(9) COMP-5.
 01  L-RETURN PIC S9(9) COMP-5.
 01  L-NAME PIC X(128) VALUE "SAMPLE::CALCULATE_ADD".

 PROCEDURE DIVISION.
*    ORB initialization
     INVOKE CDCORBA "GET-ORB" USING L-APL-NAME RETURNING L-RETURN.
     IF L-RETURN NOT = 0
     THEN
        DISPLAY "ERROR OCCURRED AT GET-ORB"
     END-IF
*    Naming service initialization
     INVOKE CDCORBA "GET-COSNAMING" RETURNING L-RETURN.
     IF L-RETURN NOT = 0
     THEN
        DISPLAY "ERROR OCCURRED AT GET-COSNAMING"
     END-IF

     INVOKE CDCORBA "GET-NAMEOBJ" USING L-NAME RETURNING L-RETURN.
     IF L-RETURN NOT = 0
     THEN
        DISPLAY "ERROR OCCURRED AT GET-NAMEOBJ"
     END-IF
```

```
*    Server object acquisition
     SET W-OBJECT TO NULL.
     INVOKE CDCORBA "GET-NAMEOBJR" RETURNING W-OBJECT.
     IF W-OBJECT = NULL
     THEN
        DISPLAY "ERROR OCCURRED AT GET-NAMEOBJR"
     END-IF

*    Target object acquisition
     SET W-TARGET TO NULL.
     INVOKE SAMPLE-CALCULATE_ADD "NARROW" USING W-OBJECT RETURNING W-TARGET.
     IF W-TARGET = NULL
     THEN
        DISPLAY "ERROR OCCURRED AT NARROW"
     END-IF

*    Server application method invocation
     DISPLAY "Enter the first argument:" WITH NO ADVANCING.
     ACCEPT  W-PARAM1.
     DISPLAY "Enter the second argument:" WITH NO ADVANCING.
     ACCEPT  W-PARAM2.

     INVOKE W-TARGET "CALCULATE" USING W-PARAM1 W-PARAM2 RETURNING W-RETURN.

     DISPLAY "result of addition:" W-RETURN.

*    End of execution
     EXIT PROGRAM.
 END PROGRAM MAIN.
```

11.6.1.2.2 Sample coding of COBOL program (static invocation)

A coding sample of a COBOL program using the static invocation interface is shown below. The flow of processing is same as for OOCOBOL although the detailed coding of each type of processing is different. For details on ENVIRONMENT DIVISION, DATA DIVISION, and each type of processing, refer to the "Interstage Application Server Distributed Application Development (CORBA Service Edition)".

Program coding sample(Main program: CALCULATE_ADD_COBCLIENT.cob)

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. MAIN.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
   SPECIAL-NAMES.
     ARGUMENT-NUMBER IS ARG-C
     ARGUMENT-VALUE  IS ARG-V
     SYMBOLIC CONSTANT
     COPY SYMBOL-CONST IN CORBA.
     .

 DATA DIVISION.
 WORKING-STORAGE SECTION.
 COPY CONST IN CORBA.
* ORB SETTING PARAMETER
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY CURRENT-ARG-C.
 01 CURRENT-ARG-V.
    02 FILLER OCCURS 6.
       03 CURRENT-ARG-V-VALUE USAGE POINTER.
 01 APLI-NAME PIC X(30) VALUE "CALCULATE_ADD_CLIENT".
 01 TMP-STRING-BUF PIC X(20).
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY ARG-COUNT.
```

```
* Character string length
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY STRING-LENGTH.

* Work pointer
 01 TEMP-BUF POINTER.

* CORBA-ENVIRONMENT
 01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.

* CORBA-ORB
 01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.

* CORBA-BOA
 01 COPY BOA IN CORBA REPLACING CORBA-BOA BY BOA.

* CORBA-OBJECT (for each type of task)
 01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.

* Exception ID used if an exception is thrown
 01 MESS PIC X(30).

* Naming Service repository
 01 COPY COSNAMING-NAMINGCONTEXT IN CORBA REPLACING
         COSNAMING-NAMINGCONTEXT BY COS-NAMING.

* Naming service name
 01 STR-BUF PIC X(30).

* For other naming context operations
 01 COPY COSNAMING-NAME IN CORBA REPLACING COSNAMING-NAME BY NAME.
 01 NAME-A USAGE POINTER.
 01 COPY COSNAMING-NAMECOMPONENT IN CORBA REPLACING
         COSNAMING-NAMECOMPONENT BY NAME-COMPONENT.
 01 NAME-COMPONENT-A USAGE POINTER.
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.

* Method return value
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY RET.
* Method parameters
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARAM1.
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARAM2.

 PROCEDURE DIVISION.
* Set server application start parameters.
* Set the client application name at the beginning of the start parameters.
     ACCEPT CURRENT-ARG-C FROM ARG-C.
     COMPUTE CURRENT-ARG-C = CURRENT-ARG-C + 1.
     PERFORM VARYING ARG-COUNT FROM 1 BY 1 UNTIL ARG-COUNT > CURRENT-ARG-C
        IF ARG-COUNT = 1
           MOVE APLI-NAME TO TMP-STRING-BUF
        ELSE
           ACCEPT TMP-STRING-BUF FROM ARG-V
        END-IF
        MOVE FUNCTION LENG (TMP-STRING-BUF) TO STRING-LENGTH
        CALL "CORBA-STRING-SET" USING
              CURRENT-ARG-V-VALUE (ARG-COUNT)
              STRING-LENGTH
              TMP-STRING-BUF
     END-PERFORM.
     SET CURRENT-ARG-V-VALUE (ARG-COUNT) TO NULL.

*
*    ORB initialization
*    (CORBA-ORB-INIT,CORBA-ORB-BOA-INIT)
*
     MOVE       12 TO STRING-LENGTH.
     CALL       "CORBA-STRING-SET" USING
           TEMP-BUF
           STRING-LENGTH
           FJ-OM-ORB-ID.
     CALL "CORBA-ORB-INIT" USING
           CURRENT-ARG-C
           CURRENT-ARG-V
           TEMP-BUF
           ENV
           ORB.
     CALL "CORBA-FREE" USING TEMP-BUF.
     PERFORM ENV-CHECK.

     MOVE 15 TO STRING-LENGTH.
     CALL       "CORBA-STRING-SET" USING
           TEMP-BUF
           STRING-LENGTH
           CORBA-BOA-OA-ID.
     CALL "CORBA-ORB-BOA-INIT" USING
           ORB
           CURRENT-ARG-C
           CURRENT-ARG-V
           TEMP-BUF
           ENV
           BOA.
     CALL "CORBA-FREE" USING TEMP-BUF.
     PERFORM ENV-CHECK.

*
*    Acquisition of Naming Service repository

     MOVE FUNCTION LENG ( CORBA-ORB-OBJECTID-NAMESERVICE ) TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
           TEMP-BUF
           STRING-LENGTH
           CORBA-ORB-OBJECTID-NAMESERVICE.
     CALL "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" USING
           ORB
           TEMP-BUF
           ENV
           COS-NAMING.
     CALL "CORBA-FREE" USING TEMP-BUF.
     PERFORM ENV-CHECK.

*
*    Search for the server application.
*
     MOVE FUNCTION LENG (STR-BUF) TO STRING-LENGTH.
     MOVE "SAMPLE::CALCULATE_ADD" TO STR-BUF.
     CALL "CORBA-STRING-SET" USING
           IDL-ID OF NAME-COMPONENT
           STRING-LENGTH
           STR-BUF.
     MOVE " " TO STR-BUF.
     CALL "CORBA-STRING-SET" USING
           KIND OF NAME-COMPONENT
           STRING-LENGTH
           STR-BUF.
     MOVE 1 TO SEQ-LENGTH OF NAME.
     MOVE 1 TO SEQ-MAXIMUM OF NAME.
     MOVE 1 TO NUM.
     CALL "CORBA-SEQUENCE-COSNAMING-NAMECOMPONENT-ALLOCBUF" USING
           SEQ-MAXIMUM OF NAME
           SEQ-BUFFER OF NAME.
     MOVE FUNCTION ADDR ( NAME ) TO NAME-A.
     MOVE FUNCTION ADDR ( NAME-COMPONENT ) TO NAME-COMPONENT-A.
     CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
           NAME-A
           NUM
           NAME-COMPONENT-A.
     CALL "COSNAMING-NAMINGCONTEXT-RESOLVE" USING
              COS-NAMING
              NAME
              ENV
              OBJ.
     MOVE "COSNAMING-NAMINGCONTEXT-RESOLVE" TO MESS.
     PERFORM ENV-CHECK.

*
*    Server object invocation
*    Sample name - interface name - operator name
*

*    Addition
     DISPLAY "Enter the first argument:" WITH NO ADVANCING.
     ACCEPT  PARAM1.
     DISPLAY "Enter the second argument:" WITH NO ADVANCING.
     ACCEPT  PARAM2.

     CALL "SAMPLE-CALCULATE-ADD-CALCULATE" USING
          OBJ
          PARAM1
          PARAM2
          ENV
          RET.
     PERFORM ENV-CHECK.

     DISPLAY "result of addition:" RET.
*
*    Post-processing
*

     CALL "CORBA-OBJECT-RELEASE" USING OBJ ENV.
     CALL "CORBA-OBJECT-RELEASE" USING COS-NAMING ENV.
     PERFORM ENV-CHECK.
     CALL "CORBA-FREE" USING SEQ-BUFFER OF NAME.
*
* Refer to CORBA-ENVIRONMENT information to check whether an exception was thrown.
*
 ENV-CHECK SECTION.
     EVALUATE TRUE
         WHEN CORBA-NO-EXCEPTION OF MAJOR OF ENV
              CONTINUE
         WHEN CORBA-USER-EXCEPTION OF MAJOR OF ENV
              DISPLAY "USER-EXCEPTION   : "
              MOVE FUNCTION LENG (MESS) TO STRING-LENGTH
              CALL "CORBA-STRING-GET" USING
                                  IDL-ID OF ENV
                                  STRING-LENGTH
                                  MESS
              DISPLAY "ID : " MESS
              EXIT PROGRAM
         WHEN CORBA-SYSTEM-EXCEPTION OF MAJOR OF ENV
```

```
             DISPLAY "SYSTEM-EXCEPTION : "
             MOVE FUNCTION LENG (MESS) TO STRING-LENGTH
             CALL "CORBA-STRING-GET" USING
                                 IDL-ID OF ENV
                                 STRING-LENGTH
                                 MESS
             DISPLAY "ID : " MESS
             EXIT PROGRAM
    END-EVALUATE.
ENV-CHECK-END.
    EXIT.

END PROGRAM MAIN.
```

11.6.1.2.3 CORBA Server Objects

When you use the "CORBA Server Object" in the **Template view**, you can add the method invocation.

When using the editor to add the method invocation in the source file, move the cursor to the location at which this processing is to be added and display the **Template view**. Select module name > interface name > method in "CORBA Server Objects" and select **input argument** from the context menu.

The CORBA server objects stored in the Interstage interface repository are displayed as a list.

- Templates or categories cannot be added, edited, deleted in CORBA server objects.
- The tree structure of the CORBA server object root categories is as follows:
  - Modules
    Modules are shown under the CORBA server object folder.
  - Interfaces
    Interfaces defined for a module is shown under the folder of the module.
  - Operators
    Operators defined for an interface is shown under the folder of the interface.

Note

Individual elements are displayed for CORBA server objects only in an environment where reference to the interface repository is possible. If no elements are displayed for CORBA server objects, check the environment. If the Interstage server is installed, check whether the service required to refer to the interface repository is activated. If the Interstage client is installed, check whether the appropriate server name is set in "Interstage-installation-folder\ODWIN\etc\INITHOST" and whether the server is activated.

Context menu specific to CORBA server objects

Element type	Menu	Description
Interface	Insertion of object	Inserts processing for a server object search in the selected interface at the cursor position in the active COBOL editor.
Operator	Insert	Inserts the operator call at the cursor position in the active COBOL editor. Arguments are inserted as data items named param1 and param2, and the return value is inserted as the data item named "return value".
	Insert with Parameter Value	Displays a dialog box for entering operator arguments and return value data item names, and inserts the operator call at the cursor position in the active COBOL editor. The data item names can be specified in the dialog box.
	Insert with Multiple Parameter Values	Inserts as many sets of the operator call as specified at the cursor position in the active COBOL editor. A dialog box for entering arguments appears, and data item names can be entered in it.
	Copy	Copies the operator call to the clipboard.

Note

- "Refresh" can update only the elements under the selected node.
- The operation of saving an object to the interface repository is not automatically reflected in CORBA server objects. To do so, execute "Refresh".

11.6.1.3 Preparation of the Stub file

To generate a CORBA client application, create stub files and other files required for CORBA clients using the IDL file generated together with the CORBA server application, and register the files in the relevant project.

11.6.1.3.1 CORBA Stub File wizard

Start the **CORBA Stub File wizard** by following the procedure below.

1. Select **File** > **New** > **Other** from the menu bar. The **New wizard** is started.
2. Select **COBOL** > **Source** > **CORBA Stub File** in the **New wizard**, and click the **Next** button. The **CORBA Stub File wizard** is started.
3. Specify basic information for the CORBA stub file on the **CORBA Stub File Information** page.

Table 11.27 Generating a CORBA Stub File

Option	Description
IDL file	Specifies the absolute path of the IDL file generated together with the CORBA server application.
Project	Specifies the project in which the generated stub file is to be registered.
Language	Specifies the generation language. Select "COBOL" or "Object-Oriented COBOL".

4. Click the **Finish** button. The stub file is created.

11.6.1.4 Building

Use the **COBOL project wizard**. When you check "Set the build environment of CORBA client project" in the **Build Environment** page, the following options are automatically set for the build option.

- Build of object-oriented COBOL program
  - Compile options

```
LIB(Interstage Application Server install folder\odwin\include\oocob)
REPIN(Interstage Application Server install folder\odwin\rep)
THREAD(MULTI)or THREAD(SINGLE)
```

  - Linker options

    Link the following library.

```
Interstage Application Server install folder\odwin\lib\odoocob.lib
Interstage Application Server install folder\odwin\lib\odcnoocob.lib
```

- Build of COBOL program
  - Compile options

```
THREAD(MULTI) or THREAD(SINGLE)
```

- Library name

  ```
  CORBA=Interstage Application Server install folder\odwin\include\cobol
  ```

- Linker options

  Link the following library.

  ```
  Interstage Application Server install folder\odwin\lib\odcobcbl.lib -- For the single thread
  Interstage Application Server install folder\odwin\lib\odcobcblmt.lib  -- For the multi thread
  ```

When setting options other than the above, set them in the **Build** page of the **Property** dialog box. The **Build** page is displayed using the following procedures.

1. Select CORBA server project from the **Dependence view** or the **Organization view**.
2. Select **File** > **Property** from the menu bar or select **Property** from the context menu. The **Property** dialog box is displayed.
3. Select **Build** in left pane. The **Build** page is displayed.

Build of Project

When automatic build is set, the build is executed after saving the COBOL source program. When automatic build is not set, select object-project from the **Dependence view** or the **Structure view**, select **Project** > "Build of Project" from menu bar to execute the build.

11.6.1.5 Debugging

See

For details about debug perspective, refer to "Debug Perspective".

11.6.1.5.1 Setting of build mode

When debugging the program, specify "Build Mode" to **Debug**. For "Build Mode" procedures, refer to "11.5.1.4.1 Setting the build mode".

11.6.1.5.2 Start-organization When Debug Starts

Use the COBOL application start-organization to debug a CORBA client application.

- Local debug of COBOL application

  To perform a local debug of a COBOL application, use the COBOL application start-organization. For details about creating a COBOL application start-organization and starting the debugger, refer to "7.1.3 Starting debugging".

- Remote debug of COBOL application

  To perform a remote debug of a COBOL application that is built on the server side, use the remote COBOL application start-organization. For details about creating a remote COBOL application start-organization and starting the debugger, refer to the following.

  - Normal Debugging: "9.6.1.2 Starting the remote debugger"
  - Attach Debugging:"9.6.2.1 Starting the remote debugger"

11.6.1.6 Executing

After starting the work unit that has deployed the CORBA server application, select the object-project from the **Dependence view** or the **Structure view**, select **Executing** > **Executing(S)** > **COBOL Application** from menu bar, and execute the client application.

11.6.2 Notes

For CORBA data type, refer to "11.5.2.1 Usable Data Types".

11.7 Remote Development

Develop the CORBA application that operates with the Solaris server.

For the procedure for remote development of COBOL project, refer to "Chapter 9 Remote Development Function".

It is a part as follows different from COBOL project.

11.7.1 Creating a Makefile

A makefile is required for remote build on a server. You can use the makefile generation function to generate a makefile required for build on a server.

Create one makefile for each COBOL project, and two makefiles, one for executable files and one for libraries, for each CORBA server project. However, for a CORBA server project that does not have a COBOL source file specifying the main program, create only a makefile for libraries.

	Server file name	Local PC file name
For an executable file	Makefile	Makefile
For a shared library or dynamic link library	lib/Makefile(*)	Makefile_lib

*: Relative path from server directory

Target name

Displays the name of the executable file or dynamic link library (shared library) that becomes the makefile target.

For a CORBA server project, an executable file name and library name are displayed. However, if the project does not have a COBOL source file specifying the main program, only a library name is displayed.

Server system	Target type	Target name
Solaris system	Executable file	Target_name
	Shared library	libTarget_name.so

Compile option

Displays the compile options used in the makefile during COBOL source compilation.

The target for a CORBA server project must not have the dynamic program structure. For this reason, do not specify the DLOAD compile option.

Link option

Displays the link options used in the makefile during COBOL source linking.

You can change link options specific to the OS on the server by selecting the **Linker Options2** tab in the **Option setting** dialog box.

For details on the link options on the server, refer to the "NetCOBOL User's Guide" on the server.

Option		Description
Linkage mode		Specifies the linkage mode. For a CORBA server project, the linkage mode is dynamic linkage.
	Dynamic linkage"-dy"	Creates COBOL applications in dynamic linkage mode.
	Static linkage"-dn"	Creates COBOL applications in static linkage mode.
Use screen form descriptor		For a CORBA server project, this option is disabled.
Uses screen handling module		For a CORBA server project, this option is disabled.

Option	Description
Use C-ISAM	Specifies that a program using C-ISAM be linked. This option is not selected by default.
Program invoked from C	For a CORBA server project, this option is disabled.
Use C Runtime Library	For a CORBA server project, this option is disabled.
Output debugging information	For a CORBA server project, this option is disabled.
Link Option"-Wl"	Specifies the link option used by the ld command.

11.7.2 Remote Debugging

To implement remote debugging of a CORBA application, use attachment debugging. Run the CORBA application on the server while the debugger on the local PC is in the wait state. The procedure for starting the debugger is the same as that for attachment debugging.

The following settings are necessary. .

- When the make file generates, you set the THREAD(SINGLE) compiler option.
- Before starting the CORBA work unit, register various repositories. For details, refer to "11.8.1 Operation of CORBA Server Application". The registration of the interface repository executes the following commands.

  ```
  IDLc -R -create IDL-filename
  ```

  The IDL File is stored in the directory specified by a remote development of the CORBA server project.

  When the following errors are output, change from "-create" to "-update", and reexecute it.

  ```
  UX:OD: ERROR: od51107:IDLparser: Redeclaration of module name as an identifier. FILE=IDL-filename
  LINE=2
  UX:OD: error: od51229:IDLc: Command error.STATUS=4
  ```

  For other errors and details of the error, refer to "Interstage Application Server/Interstage Web Server reference manual (Command)"
- When the CORBA work unit is started, set the following to the library path;

  Interstage Application Server install folder/lib/nt (default is "/opt/FSUNod/lib/nt")

Point

During remote debugging, when the CORBA work unit is started, set the CBR_ATTACH_TOOL environment variable.

When you operate the CORBA server application, change the THREAD(MULTI) compiler option as needed and rebuild.

If you rebuild, set the following to the library path;

```
Interstage Application Server install folder/lib/  (default is "/opt/FSUNod/lib/")
```

11.8 Operation

11.8.1 Operation of CORBA Server Application

The operations described in the following section are required in actual operation of the developed CORBA server applications.

The section explains the setup of the operating environment and registration of the server applications.

Registering a CORBA Server Application

Before you can execute a CORBA server application, it is necessary to register it in several types of repositories. For the registration procedure, refer to the "Interstage Application Server Distributed Application Development Guide (CORBA Service Edition)" and "5. Running the Program"

Interface repository

The interface repository is required if:

- The client application uses the OLE-GATEWAY method.
- The CORBA server object listing function is used during workbench-based CORBA client application development.

For registration in the interface repository, select the IDL file to be registered from the **Dependency view**, then select "Register to Interface repository" from the popup menu.

If "Register to Interface repository" of IDLc is specified as a build option, registration is performed automatically during the build.

Note

If the interface repository already has the interface to be registered, the existing interface is overwritten.

Implementation repository

Registration of the implementation repository is essential. It stores information necessary for server application execution. Perform registration by using the Interstage Management Console or the CORBA service command on the server.

Naming service name

A naming service name must be registered for each CORBA server application created with object-oriented COBOL. You can register it by using either of two methods. One method is to specify settings from the Interstage Management Console so that the name is automatically registered when a WorkUnit starts, and the other is to execute the CORBA service command for registration.

11.8.2 Operation of CORBA Client Application

Check the following environment settings before running the CORBA client application.

11.8.2.1 Common Items

This section explains the common items that are independent of the implementation method. For details, refer to the "Interstage Application Server Distributed Application Development Guide (CORBA Service Edition)". For notes on running applications, refer to the relevant software release guides.

- If the server application must be started in advance, when the server application type is PERSISTENT, ask the server administrator to start the server application.
- Though it's not directly related with the client application but, to run the server application it is necessary to register the implementation repository information beforehand on the server.
- Before the CORBA client application can refer to a server object, server object information must be stored in the naming service repository.
- If the Interstage Application Server client function is installed, specify the naming server used for it in the "inithost" file provided by the CORBA service.
    - Naming server specification example 1:
      If Interstage Application Server is installed in "c:\interstage\APS" and the naming service of the "SERV01" server is to be used, the following must be defined in "c:\interstage\APS\odwin\etc\inithost".

```
SERV01    8002
  :         :
  :         :
```

- Naming server specification example 2:
  When connecting the client system to a server in a multi-instance system, the system to be connected is determined according to the port number.

```
SERV01    8002    Access to instance 1 of server SERV01
SERV01    8003     Access to instance 2 of server SERV01
```

11.8.2.2 Environment Settings for Object-Oriented COBOL Static Invocation Interface

The following environment settings are required for the object-oriented COBOL static invocation interface:

- The folder containing the runtime system for this product and the folder containing the object-oriented COBOL runtime system must be set in the "PATH" environment variable.

11.8.2.3 Environment Settings for COBOL Static Invocation Interface

The following environment setting is required for the COBOL static invocation interface:

- The installation folder of the COBOL runtime system must be set in the "PATH" environment variable.

11.9 IDL Editor

The IDL editor is used to edit files written in the Interface Definition Language (IDL). The IDL editor has a variety of useful functions for IDL coding.

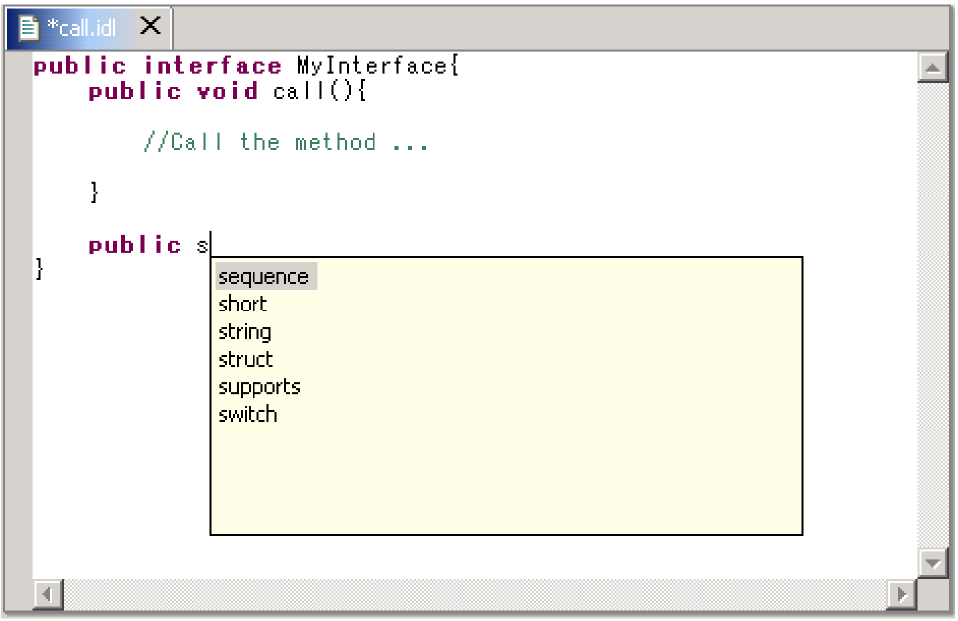
11.9.1 Input Candidate List (Content Assist)

Content Assist (also referred to as Code Assist) can be used for coding. Place the cursor at the appropriate position on a line, and perform one of the operations listed below. A list of available input candidates is then displayed in a box in the window.Select **Edit** > **Content Assist** from the menu bar.

- Select **Content Assist** from the editor context menu.

- Press the **Ctrl+Space** keys.

The following figure is a display example of Content Assist of the IDL editor.



```
public interface MyInterface{
    public void call(){

        //Call the method ...

    }

    public s
}
```

When suitable input candidates are found, the IDL editor displays a list of the available input candidates in a box in the window. A keyword can be entered to further narrow down the candidates on the list.

For example, type "s" and press the **Ctrl+Space** key. The box in the window will display only the candidates that begin with "s".

Note

Content Assist cannot be used while the cursor is:

- In a single-line comment
- In a multiple-line comment
- Within a string or string literal

11.9.2 Setting Highlight Colors

Keywords can be highlighted using colors and font styles (boldface type).

Highlight colors can be set for the following items.

Item	Explanation
line comment	Lines beginning with // are handled as single-line comments.
Multiple-line comment	Text enclosed by /* and */ is handled as a multiple-line comment.
Reserved word	Uppercase and lowercase letters are distinguished in reserved words. For reserved words, refer to "Interstage Application Server Application Development Guide (CORBA Service Edition)".
String literal	A character string enclosed between ' ' or " " is handled as a string literal.

Displaying the highlight color settings

The highlight colors settings for the above items can be displayed as follows:

1. Select **Window** > **Preferences** from the menu bar. The **Preferences** dialog box is displayed.
2. Select **IDL** > **Editor** in the left pane. The **Editor** page is displayed.
3. Click the **Colors** tab. The "Items", "Color", "Bold", and "Preview" areas are displayed.
4. When an item is selected, styles are displayed for the set "Color" and "Bold". The current settings are displayed in "Preview".

Changing the highlight color settings

The color settings for the above items can be changed as follows:

1. To change the display color of an item, select the item from "Items".
2. Click "Color". The **Color** dialog box is displayed.
3. Click the target color in the "Basic colors" pallet. Alternatively, click "Define Custom Colors" to create the desired color.
4. To apply the selected color and close the dialog box, click **OK**. The selected color is reflected in the "Preview" area. To close the dialog box without applying the selected color, click "Cancel".
5. To display the text in boldface type, select "Bold".

Note

- By default, reserved words are displayed in boldface type, and all other items are displayed in the normal font style.
- The settings remain in effect even after the next time the editor is opened.

11.9.3 Setting Fonts

The font used in the IDL editor can be changed using the **Preferences** dialog box. By default, the font that is set for workbench is used.

Changing the font setting

1. Select **Window** > **Preferences** from the menu bar. The **Preferences** dialog box is displayed.
2. Select **IDL** > **Edito**r in the left pane. The **Editor** page is displayed.
3. Click the **General** tab.
4. Click "Change". The **Font** dialog box is displayed.
5. Change "Font", "Font style", and "Size" as necessary.

11.9.4 Displaying line numbers

The editor can be set up to show or hide line numbers.

Showing or hiding line numbers

1. Select **Window** > **Preferences** from the menu bar. The **Preferences** dialog box is displayed.
2. Select **General** > **Editors** > **Text Editors** in the left pane. The **Text Editors** page is displayed.
3. To show line numbers, select "Show line numbers".

11.9.5 Highlighting the Current Line

The editor can be set up to highlight the current line.

Highlighting the current line

1. Select **Window** > **Preferences** from the menu bar. The **Preferences** page is displayed.
2. Select **General** > **Editors** > **Text Editors** in the left pane. The **Text Editors** page is displayed.
3. To highlight the current line, select "Highlight current line".

11.9.6 Displaying the Cursor Position

The line and column numbers indicating the cursor position in the active file are displayed on the status bar.

11.9.7 "Edit" Menu Commands

This section explains the "Edit" menu commands available when the IDL editor is active.

Name	Function	Keyboard Shortcut
Undo	Cancels the preceding edit operations (up to 25 operations). Select "Undo" from the context menu, or click on the toolbar.	Ctrl + Z
Redo	Redoes the edit operations that were undone (up to 25 operations). Select "Redo" from the context menu or click on the toolbar.	Ctrl + Y
Cut	Cuts out the selected range of data and saves it on the clipboard. Select "Cut" from the context menu, or click on the toolbar.	Ctrl + X
Copy	Copies the selected range of data and saves it on the clipboard. Select "Copy" from the context menu, or click on the toolbar.	Ctrl + C
Paste	Pastes the contents of the clipboard. Select "Paste" from the context menu, or click on the toolbar.	Ctrl + V
Delete	Deletes the selected range of data.	Delete

Name	Function	Keyboard Shortcut
Select All	Selects all contents of the edited file. Selecting "Select All" from the context menu performs the same operation.	Ctrl + A
Forward	Searches forward for a match of the character string specified in the **Find/ Replace** dialog box.	Ctrl + K
Backward	Searches backward for a match of the character string specified in the **Find/ Replace** dialog box.	Ctrl + Shift + K
Incremental Find Next	Begins an incremental search for the next match.	Ctrl + J
Find/Replace	Displays the **Find/Replace** dialog box so that the specified character string can be found and/or replaced. Clicking 🔦 on the toolbar performs the same operation.	Ctrl + F
Add Bookmark	Adds a bookmark to the currently selected text.	-
Add Task	Adds a user-defined task to the currently selected text.	-
Content Assist	Enables Content Assist.	Ctrl + Space
Undo	Cancels the preceding edit operations (up to 25 operations). Select "Undo" from the context menu, or click ↶ on the toolbar.	Ctrl + Z
Redo	Redoes the edit operations that were undone (up to 25 operations). Select "Redo" from the context menu or click ↷ on the toolbar.	Ctrl + Y
Cut	Cuts out the selected range of data and saves it on the clipboard. Select "Cut" from the context menu, or click ✂ on the toolbar.	Ctrl + X

11.10 COBOL Service class (CDCORBA class)

The COBOL service class is provided by the COBOL plug-in.

The CDCORBA class is used to develop CORBA client applications.
This section explains each method of the CDCORBA class.

11.10.1 Methods

Method name	Function
CREATE	Initializes member variables
GET-ORB	Initializes ORB and obtains an ORB object reference
GET-COSNAMING	Obtains an object reference of the naming service
GET-NAMEOBJ	Obtains an object reference of the server application
GET-ERROR-MSG	Returns the error message of the error occurred in the previous method call
GET-ORBR	Returns the obtained ORB object reference
SET-ORBR	Sets an ORB object reference
GET-COSNAMINGR	Returns the obtained object reference of the naming service
SET-COSNAMINGR	Sets an object reference of the naming service
GET-NAMEOBJR	Returns the obtained object reference of the server application
SET-NAMEOBJR	Sets an object reference of the server application

11.10.2 Method Details

This section provides details on methods.

11.10.2.1 CREATE method

Class

CDCORBA

Description

The CREATE method executes essential processing and then initializes the member variables of the CDCORBA class.
For details on the CREATE method, refer to the "NetCOBOL Language Reference".

Syntax

```
INVOKE CDCORBA "CREATE" RETURNING L-RETURN.
```

Parameter

None

Return value

```
L-RETURN "attribute: OBJECT REFERENCE SELF"
   The object reference of CDCORBA itself is returned.
```

See also

CREATE method of the FJBASE class

11.10.2.2 GET-ORB method

Class

CDCORBA

Description

This method initializes ORB and obtains an ORB object reference.
Use the GET-ORBR method to refer to the obtained object reference.

Syntax

```
INVOKE CDCORBA "GET-ORB" USING L-PARAM RETURNING L-RETURN.
```

Parameter

```
L-PARAM "attribute: PIC X(50)"
   The name of client application
```

Return value

```
L-RETURN "attribute: PIC S9(9) COMP-5"
   The result of function processing is returned.
```

0: Normal end
Negative number: Abnormal end (continuous operation disabled)
Positive number: ORB object reference already obtained

See also

GET-ORBR method

11.10.2.3 GET-COSNAMING method

Class

CDCORBA

Description

This method obtains an object reference of the naming service.
Use the GET-COSNAMINGR method to refer to the obtained object reference.

Syntax

```
INVOKE CDCORBA "GET-COSNAMING" RETURNING L-RETURN.
```

Parameter

None

Return value

```
L-RETURN "attribute: S9(9) COMP-5 "
```

The result of function processing is returned.

0: Normal end

Negative number: Abnormal end

See also

GET-COSNAMINGR method

11.10.2.4 GET-NAMEOBJ method

Class

CDCORBA

Description

This method obtains a server object reference with the naming service name specified in the parameter.
Use the GET-NAMEOBJR method to refer to the obtained object reference.

Syntax

```
INVOKE CDCORBA "GET-NAMEOBJ" USING L-PARAM RETURNING L-RETURN.
```

Parameter

```
L-PARAM "attribute: X(128) "
```

Naming service name of the server application to be obtained

For details on naming service names, refer to the "Interstage Application Server Distributed Application Development Guide (CORBA Service Edition)".

Return value

```
L-RETURN "attribute: S9(9) COMP-5 "
```

The result of function processing is returned.
0: Normal end
Negative number: Abnormal end

See also

GET-NAMEOBJR method

11.10.2.5 GET-ERROR-MSG method

Class

CDCORBA

Description

This method returns error information of the error occurred in the previous method call.
The method returns EXCEPTION-ID that is returned to CORBA-Exception. For details, refer to the "Interstage Application Server Distributed Application Development Guide (CORBA Service Edition)".

Syntax

```
INVOKE CDCORBA "GET-ERROR-MSG" RETURNING L-RETURN.
```

Parameter

None

Return value

```
L-RETURN "attribute: X(128) "
```

Information returned from CORBA-Exception

See also

CORBA-Exception (Refer to the "Interstage Application Server Distributed Application Development Guide (CORBA Service Edition).)"

11.10.2.6 GET-ORBR method

Class

CDCORBA

Description

This method returns an obtained ORB object reference.

Syntax

```
INVOKE CDCORBA "GET-ORBR" RETURNING L-RETURN.
```

Parameter

None

Return value

```
L-RETURN "attribute: OBJECT REFERENCE CORBA-ORB "
```

An ORB object reference obtained by the GET-ORB method is returned.

See also

GET-ORB method
CORBA-ORB (Refer to the "Interstage Application Server Distributed Application Development Guide (CORBA Service Edition)".)

11.10.2.7 SET-ORBR method

Class

CDCORBA

Description

This method sets an ORB object reference.

Syntax

```
INVOKE CDCORBA "SET-ORBR" USING L-PARAM.
```

Parameter

```
L-PARAM "attribute: OBJECT REFERENCE CORBA-ORB "
```

Object reference of ORB

Return value

None

See also

GET-ORB method
GET-ORBR method
CORBA-ORB (Refer to the "Interstage Application Server Distributed Application Development Guide (CORBA Service Edition)".)

11.10.2.8 GET-COSNAMINGR method

Class

CDCORBA

Description

This method returns an obtained object reference of the naming service.

Syntax

```
INVOKE CDCORBA "GET-COSNAMINGR" RETURNING L-RETURN.
```

Parameter

None

Return value

```
L-RETURN "attribute: OBJECT REFERENCE COSNAMING-NAMINGCONTEXT "
```

Obtained object reference of the naming service

See also

GET-COSNAMING method
COSNAMING-NAMINGCONTEXT (Refer to the "Interstage Application Server Distributed Application Development Guide (CORBA Service Edition)".)

11.10.2.9 SET-COSNAMINGR method

Class

CDCORBA

Description

This method sets an object reference of the naming service.

Syntax

```
INVOKE CDCORBA "SET-COSNAMINGR" USING L-PARAM.
```

Parameter

```
L-PARAM "attribute: OBJECT REFERENCE COSNAMING-NAMINGCONTEXT "
```

Object reference of the naming service

Return value

None

See also

GET-COSNAMING method
GET-COSNAMINGR method
COSNAMING-NAMINGCONTEXT (Refer to the "Interstage Application Server Distributed Application Development Guide (CORBA Service Edition)".)

11.10.2.10 GET-NAMEOBJR method

Class

CDCORBA

Description

This method returns an object reference of the server application.

Syntax

```
INVOKE CDCORBA "GET-NAMEOBJR" RETURNING L-RETURN.
```

Parameter

None

Return value

```
L-RETURN "attribute: OBJECT REFERENCE CORBA-OBJECT "
```

Object reference of the server application

See also

GET-NAMEOBJ method
CORBA-OBJECT (Refer to the "Interstage Application Server Distributed Application Development Guide (CORBA Service Edition)".)

11.10.2.11 SET-NAMEOBJR method

Class

CDCORBA

Description

This method sets an object reference of the server application.

Syntax

```
INVOKE CDCORBA "SET-NAMEOBJR" USING L-PARAM
```

Parameter

```
L-PARAM "attribute: OBJECT REFERENCE CORBA-OBJECT "
```

Object reference of the server application

Return value

None

See also

GET-NAMEOBJ method
GET-NAMEOBJR method
CORBA-OBJECT (Refer to the "Interstage Application Server Distributed Application Development Guide (CORBA Service Edition)".)

11.11 Resource Migration from a Previous Version

This section explains the migration of resources pertaining to a COBOL plug-in from a previous version. For migration of another resource, refer to "Appendix D Resource Migration from a Previous Version" of the "Interstage Studio user's guide".

11.11.1 About migration from the component designer

Migrating the CORBA server project written in COBOL language to the workbench.

A CORBA server project written in the COBOL language can be migrated to the workbench with the following procedures.

1. Create an empty CORBA server project.

   Select **File** > **New** > **Project** from menu bar, and then select **COBOL** > **CORBA Server Project** on the **New project** dialog box.

   Project name and the target name are set to the same name as the project name to be migrated. On the **Skeleton Code Selection** page, select "Do not generate code" and click the **Finish** button.

2. Set the compiler options and the linker options for the created project.

   When the compiler options, library name, linker options, and the IDL compiler options have been set in the original project to be migrated, select the same settings on the **Property** dialog of the project.

3. Copy the source files from the original project to be migrated.

   In the Component Designer, open the original project to be migrated and confirm the source files (COBOL source file and IDL file) are displayed in the project display region. Then copy these source files from the folder of the original project to be migrated into the folder of the created project that is the migration destination. After confirming the source file, the Component designer can be closed.

   - the original project to be migrated:<Installation folder of old product>/AP/Projects/<project name>
   - destination of migration:<User's Document folder>/ Interstage Studio/<product version>/ workspace/<project name>

   Note

   - In the folder of the original project to be migrated, those files that are not displayed in the project display region need not be copied.
   - When the workspace folder of the workbench has been changed from the default position, the (project name) folder under the workspace folder becomes the destination of migration.

4. Add the copied files to the project.

   Right click the project from the **Dependency view**, and select "Refresh" from the context menu. The copied files will be added to the "Other Files" folder.

5. Move the copied files to the "Source File" folder.

   Select the copied files from "Other File" folder, and execute "Add to Source File" from the context menu. These files will be moved from the "Other file" folder to the "Source File" folder.

6. Set the main program.

   Open the "Source file" folder, select "(project name).cob" under it, and then execute the "Main program" from the context menu.

7. Compile the IDL file.

   Select the existing IDL file in the "Source File" folder, and execute "Compile File" from the context menu. Before compiling IDL files, it is necessary to start Interstage Application Server and the service (basic service) required for the management of Interstage.

   To start the basic service, use the Interstage basic service operation tool. To start the tool, select "Interstage"> "Studio" > "Interstage Base Service Operation Tool" from the start menu.

   To start Interstage Application Server, use the Interstage management console.

8. Move the skeleton file to the "Source File" folder.

   If the IDL file has been compiled, COBOL source files and COBOL library files that act as stub and skeleton are generated in the "Other File" folder.

   Move these files that act as the skeleton to the "Source File" folder. Move the following files to the "Source File" folder using the same procedures above.

   - Interface file: (module name)-(interface name).cob
   - Helper class file: (module name)-(interface name)--HELPER.cob

- Narrow skeleton file: (module name) - (interface name) _ NARROW.cob
- Tie class file: (module name)-(interface name)--TIE.cob
- Implementation registration file: (module name)-(interface name)--NEW.cob
- Data type class file: (data type name).cob
- Data type Helper file: (data type name)--HELPER.cob

Note

Do not move the library file (*.cbl) and stub file (*--stub.cob,*--NarrowStub.cob) to the "Source Files" folder.

Troubleshooting CORBA Application

Problems related to CORBA Application

1. The IDL file is not compiled.

Action

To compile the IDL file, the service (management service) required for the management of the Interstage Application Server and Interstage must be started in advance. If it has not started, start it first. To start the management service, the Interstage Management Service Operation Tool is used. To start the tool, select "Interstage" > "Studio" > "Interstage Management Service Operation Tool" from the start menu. To start the Interstage Application Server, the Interstage Management Console must be used.

2. The IDL compiler causes an error during build.

Action

Although environment setup is not required when the server function of Interstage Application Server is used in combination with development work, it is required when the client package is used in combination with development work.
To use the client package in combination with development work, the server to be accessed must be specified in the "Interstage-Application-Server-installation-folder"\odwin\etc\inithost file.

3. The displayed list of CORBA server objects is not updated even after the connected host is changed.

Action

If the host of the Interface Repository service is changed while workbench is running, the change is not reflected in the list of CORBA server objects as long as workbench remains active. If the host of the Interface Repository service has been changed, restart workbench.

4. A CORBA server project cannot be built when the main program is changed from that in the default file.

Action

When a CORBA server project is being built, the main program name must match the project name. Do not change the main program from that in the default file.

5. CORBA server project building when Interstage Application Server is not active may not be successful.

The following error message is displayed in the **Console view**:

```
OD: Error: od51401:IDLparser: CORBA_Container_lookup function caused IDL: CORBA/StExcep/
COMM_FAILURE: 1.0 in exception information.  Minor code is 0x464a010a
OD: Error: od51221:IDLc: IDLparser command error.  Error code=4
```

Action

The IDL compiler is executed if the IDL file has been updated. The Build fails if attempted while Interstage Application Server is in the forced stop state because IDL compiler execution fails. In this event, use the Interstage Management Console to start Interstage Application Server, and re-execute build.

6. A warning message indicating failure in deletion from the interface repository is output during the initial building of a CORBA server project.

The following error message is displayed in the Console view:

```
OD: Warning: od51006:IDLinst: Any object::SAMPLE to be deleted was not found.
```

Action

If "Register to Interface Repository" is selected in the IDL compiler setup window, the IDL compiler first deletes existing information stored in the interface repository before saving new information. During the initial building of a project, no information is stored in the interface repository, so a warning message indicating a failure to delete stored information is displayed. The project is built successfully regardless of the warning message.

Appendix A Compile Options

A.1 Compile option details

A dialog box for specifying detailed information is displayed according to the options selected from the *Compiler options* list in the **Add Compiler Options** dialog box.

Options not specified by the **Compiler Options** dialog box follows the NetCOBOL compiler option default. For details about NetCOBOL compile options, refer to "NetCOBOL User's Guide".

Note

- When specifying the project from the Selection from project dialog in selecting compiler options, the path separator backslash is displayed as a forward slash. The forward slash is treated as backslash.

- The compiler option that cannot be selected from the **Add Compiler Option** dialog box can be specified by using the **Other compiler options** on the **Compiler Options** page.

A.1.1 Compiler option list

Compilation resources options

A.1.17 FORMLIB compile option

A.1.20 LIB compile option

A.1.36 REP compile option

A.1.37 REPIN compile option

Compilation list options

A.1.8 COPY compile option

A.1.21 LINECOUNT compile option

A.1.22 LINESIZE compile option

A.1.23 LIST compile option

A.1.24 MAP compile option

A.1.25 MESSAGE compile option

A.1.30 NUMBER compile option

A.1.33 PRINT compile option

A.1.43 SOURCE compile option

A.1.54 XREF compile option

Compilation messages options

A.1.7 CONF compile option

A.1.15 FLAG compile option

A.1.16 FLAGSW compile option

COBOL program interpretation options

A.1.2 ALPHAL compile option

A.1.5 BINARY compile option

A.1.10 CURRENCY compile option

A.1.12 DUPCHAR compile option

A.1.18 INITVALUE compile option

A.1.19 LANGLVL compile option

A.1.27 NCW compile option

A.1.28 NSP compile option

A.1.29 NSPCOMP compile option

A.1.34 QUOTE/APOST compile option

A.1.38 RSV compile option

A.1.40 SDS compile option

A.1.41 SHREXT compile option

A.1.44 SQLGRP compile option

A.1.45 SRF compile option

A.1.48 STD1 compile option

A.1.49 TAB compile option

A.1.55 ZWB compile option

Source program analysis options

A.1.39 SAI compile option

Object program creation options

A.1.3 ARITHMETIC compile option

A.1.4 ASCOMP5 compile option

A.1.13 ENCODE compile option

A.1.11 DLOAD compile option

A.1.26 MODE compile option

A.1.31 OBJECT compile option

A.1.32 OPTIMIZE compile option

A.1.35 RCS compile option

A.1.51 THREAD compile option

Execution time processing options

A.1.14 EQUALS compile option

A.1.53 TRUNC compile option

Execution time resources options

A.1.42 SMSIZE compile option

A.1.46 SSIN compile option

A.1.47 SSOUT compile option

Debugging functions at execution time options

A.1.6 CHECK compile option

A.1.9 COUNT compile option

A.1.50 TEST compile option

A.1.52 TRACE compile option

A.1.2 ALPHAL compile option

This option specifies whether to treat lowercase letters in source programs the same as uppercase letters (ALPHAL) or not (NOALPHAL).

Item	Description
Lower-case character usage	Specifies how to treat lowercase letters in source programs. ALPHAL is the default value.
ALPHAL	Treats lowercase letters the same as uppercase letters (not case-sensitive).
NOALPHAL	Distinguishes between lowercase letters and uppercase letters (case-sensitive).
Scope of words taken as upper case	This item can be selected only if ALPHAL is selected for Lower-case character usage. ALL is the default value.
ALL	Treats the lowercase letters that appear in program-name literals and literals in the CALL, CANCEL, ENTRY, and INVOKE statements the same as uppercase letters.
WORD	Treats literals as they are described.

A.1.3 ARITHMETIC compile option

This option specifies whether to use 18-digit compatible arithmetic mode or 31-digit extended arithmetic mode.

Item	Description
Calculation mode	Specifies which calculation mode to use. The default is 18 digits compatible arithmetic mode.
18 digits compatible arithmetic mode	Uses the 18-digit compatible arithmetic mode.
31 digits extended arithmetic mode	Uses the 31-digit extended arithmetic mode.
Show I-level diagnostic message	Checks for the following: - When compiling using 18 digits compatible arithmetic mode, the following message is output: JMN3024I-W The intermediate result cannot contain more than 30 digits. The intermediate result is assumed to have 30 digits. - Checks for arithmetic expressions with an intermediate result (fixed-point or floating-point) that is not in18 digits compatible arithmetic mode.

Note

- When 31 digits extended arithmetic mode is specified, compiler option BINARY can specify only BINARY(WORD,MLBON).
- Specify 18 digits compatible arithmetic mode for compatibility with V10.2.0 or earlier, or with other systems.
- Refer to "Operation mode" in Chapter 1 of the "NetCOBOL Language Reference" for details.

A.1.4 ASCOMP5 compile option

This option specifies how to interpret binary items.

Item		Description
Specification of interpretation binary data		Specifies how to treat binary items. The default is NONE.
	NONE	Treats binary items as they are declared.
	ALL	Treats binary items declared as USAGE BINARY, USAGE COMP, or USAGE COMPUTATIONAL as USAGE COMP-5 items.
	BINARY	Treats binary items declared as USAGE BINARY as USAGE COMP-5 items.
	COMP	Treats binary items declared as USAGE COMP or USAGE COMPUTATIONAL as USAGE COMP-5 items.

Note

- Using this option may affect the internal format of data.
- Note that care is required if you use this option with code that is aware of internal binary numeric formats because the ALL, BINARY, and COMP options will cause the internal formats to change.
- NONE must be specified when using the CBL routine.

A.1.5 BINARY compile option

This option specifies whether to assign an elementary item of binary data to an area of a length in units of words (2, 4, or 8) (BINARY(WORD)) or bytes (1 to 8) (BINARY(BYTE)), which is calculated based on the number of digits. This option can also specify how to treat the high-order-end bit of an unsigned binary item.

Item		Description
Binary data item usage		Specifies how to treat binary items. The default is WORD,MLBON.
	WORD,MLBON	The area length is in units of words, and the high-order-end bit is treated as a sign symbol.
	WORD,MLBOFF	The area length is in units of words, and the high-order-end bit is treated as a numerical value.
	BYTE	The area length is in units of bytes, and the high-order-end bit is treated as a sign symbol.

Note

- BINARY(BYTE) cannot be used in a class definition.
- If BINARY(BYTE) is specified, the high order end bit is treated as a numeric value.
- BINARY(WORD, MLBON) cannot be excluded when specifying ARITHMETIC(31).

Information

The following table shows the relationship between the number of declared digits and the area length.

Number of PIC Digits		Assigned Area Length	
Signed	unsigned	BINARY(BYTE)	BINARY(WORD)
1 - 2	1 - 2	1	2
3 - 4	3 - 4	2	2
5 - 6	5 - 7	3	4
7 - 9	8 - 9	4	4
10 - 11	10 - 12	5	8
12 - 14	13 - 14	6	8
15 - 16	15 - 16	7	8
17 - 18	17 - 18	8	8
19 - 28 (*)	19 - 28	-	12
29 - 31 (*)	29 - 31	-	16

*: For 31-digit extension operation mode.

A.1.6 CHECK compile option

This option specifies whether to use the CHECK function (CHECK) or not (NOCHECK). An integer ranging from 0 to 999999 is specified for n as the number of messages to be displayed. If the option is omitted, 1 is assumed to be specified.

Item		Description
Check function		Specifies whether the CHECK function can be used. The default is NOCHECK.
	CHECK	Uses the CHECK function.
	NOCHECK	Does not use the CHECK function.
Number of messages to be displayed		Specifies the number of messages to be displayed.
Check items		
	ALL	Checks all of NUMERIC, BOUND, ICONF, LINKAGE, and PRM.
	BOUND	Checks whether subscripts, indexes, and reference modifications are in their correct ranges.
	ICONF	Checks whether the INVOKE statement parameters are compatible with the tentative parameters of the method to be called.
	NUMERIC	Checks for a data exception. If a numeric item contains a value that does not match its attribute, or if the divisor of division operation is 0, a data exception occurs.
	PRM	Checks the following during compilation related to data items described in the USING or RETURNING phrases of the CALL statement (except call identifiers) invoking internal programs, and the USING or RETURNING phrase of the internal programs: - Whether the number of parameters in the USING phrases match - Consistency in the parameters of individual RETURNING phrases - Whether the lengths of data items that do not reference objects match. This check of the lengths is performed only if the length is determined during compilation.

Item	Description
	- Whether the class names specified for the USAGE OBJECT REFERENCE clause match, the FACTORY phrase matches, and the ONLY phrase matches if the data items reference objects A check of the following is performed during execution related to data items described in both the USING or RETURNING phrase of the CALL statement invoking external programs, and the USING or RETURNING phrase of the external programs: - Whether the number of parameters in the USING phrases match, and whether the data items lengths match. ***Note***: If the USING phrases have four or more parameter number mismatch errors, an error may not be detected. - Whether the lengths of parameters in the USING phrases match. If the RETURNING phrase is not specified, data items with a length of four bytes are assumed specified because PROGRAM-STATUS is implicitly handed over. Also, if the length of a parameter is determined during execution, this check is performed during compilation, using the maximum value of the length described.

Note

- While the CHECK function is enabled, the program continues running until the n-th message is output. However, the program may not run as expected due to destruction of an area or other issue. If 0 is specified for n, the program continues running regardless of the number of messages that have been displayed.

- If CHECK is specified, processing for the checks described above is embedded in an object program, decreasing execution performance. After debugging, recompile the program with NOCHECK specified.

- A check for a zero divisor for ON SIZE ERROR is performed on an arithmetic statement with the ON SIZE ERROR or NOT ON SIZE ERROR phrase. However, this divisor check is not performed on CHECK (NUMERIC).

- A check for a data exception on CHECK (NUMERIC) is performed only if zoned decimal items or packed decimal items are referenced, or alphanumeric data items or group items are moved to zoned decimal items or packed decimal items. However, the following are not checked:

- Table element for which ALL is specified as a subscript

  - Key items in the SEARCH ALL statement, except in cases where the subscripts for the key items are one-dimensional and a single WHEN condition is specified

  - Key items for the SORT/MERGE statement

  - Host variables used in SQL statements

  - BY REFERENCE parameter of the CALL statement, the INVOKE statement, and an in-line invocation

  - Arguments of the following intrinsic functions:

    - FUNCTION ADDR

    - FUNCTION LENG

    - FUNCTION LENGTH

  - Moving of alphanumeric data items or group data items to object properties of zoned decimal items or packed decimal items.

A.1.7 CONF compile option

This option specifies whether to output diagnostic messages when items are detected for which there are incompatibilities between new and old COBOL standards (CONF) or not (NOCONF). If CONF is specified, I-level diagnostic messages will be output when items with such incompatibilities are detected. CONF is useful for converting programs that were created in conformance to old standards into programs conforming to the '85 ANSI COBOL standard.

Item	Description
Posting of incompatibility with the COBOL standards	Specifies whether to output messages on items for which there are incompatibilities between new and old COBOL standards. The default is NOCONF.
CONF	Outputs messages on items for which there are incompatibilities between new and old COBOL standards.
NOCONF	Does not generate messages when incompatibilities between new and old COBOL standards are detected.
indicating post items	This item is enabled only if CONF is selected for Posting of incompatibility with the COBOL standards. The default value is 68.
68	Outputs messages when items are detected for which there are incompatibilities between '68 ANSI COBOL and '85 ANSI COBOL. The specification is valid only if LANGLVL (85) is specified as a compile option.
74	Outputs messages when items are detected for which there are incompatibilities between '74 ANSI COBOL and '85 ANSI COBOL. The specification is valid only if LANGLVL (85) is specified as a compile option.
OBS	Outputs messages when obsolete elements in language specifications and functions are detected.

Note

The compiler options CONF(68) and CONF(74) are effective only if the compiler option LANGLVL(85) is specified

A.1.8 COPY compile option

This option specifies whether to display library text embedded by the COPY statement in a source program list (COPY) or not (NOCOPY). The COPY statement is valid only if the SOURCE compile option is specified.

Item	Description
Library text output option	Specifies whether to display library text. The default is NOCOPY.
COPY	Displays library text.
NOCOPY	Does not display library text.

Note

COPY is only effective when the compiler option SOURCE is specified.

A.1.9 COUNT compile option

This option specifies whether to use the COUNT function (COUNT) or not (NOCOUNT).

Item	Description
Determines whether to use COUNT function	Specifies whether the COUNT function can be used. The default is NOCOUNT.
COUNT	Uses the COUNT function.
NOCOUNT	Does not use the COUNT function.

Note

- If COUNT is specified, processing to output COUNT information is embedded in an object program, decreasing execution performance. After debugging, recompile the program with NOCOUNT specified.
- COUNT cannot be specified together with the TRACE compile option. If both of these options are specified together, the option specified last is used.

A.1.10 CURRENCY compile option

This option specifies whether to use $ (CURRENCY($)) or to use another symbol (CURRENCY(currency-symbol)) as the character for the currency symbol. If CURRENCY(currency-symbol) is specified, refer to the CURRENCY SIGN clause explained in the "COBOL Language Reference" for the currency symbol that can be used.

Item	Description
Specification of currency sign	Specifies what to use as the currency symbol. The default is $.
$	Uses $ as the currency symbol.
Specify currency symbol character	Uses another symbol.
Currency symbol character	The currency symbol character is specified.

A.1.11 DLOAD compile option

This option specifies whether to adopt the dynamic program structure for the program structure (DLOAD) or not (NODLOAD).

Item	Description
Program structure	Specifies the program structure. The default is NODLOAD.
DLOAD	Uses the dynamic program structure.
NODLOAD	Does not use the dynamic program structure.

A.1.12 DUPCHAR compile option

Specify whether the JIS non-Kanji minus sign is "MINUS SIGN (STD)" or "FULLWIDTH HYPHEN-MINUS (EXT)" when source programs and library texts are created in UTF-8.

Item	Description
Duplicate characters	Specifies how duplicate characters are treated. The default is EXT.
STD	MINUS SIGN
EXT	FULLWIDTH HYPHEN-MINUS

	UTF-8	UTF-16
MINUS SIGN	X"E28892"	X"2212"

FULLWIDTH HYPHEN-MINUS	X"EFBC8D"	X"FF0D"

A.1.13 ENCODE compile option

Specify the encoding of the alphanumeric data item and the national data item.

Item		Description
Encoding of an alphanumeric character item		The default is SJIS.
	SJIS - Shift-JIS	The encoding of an alphanumeric character item is specified as SJIS.
	UTF-8	The encoding of an alphanumeric character item is specified as UTF8.
Encoding of a national item		The default is SJIS.
	SJIS - Shift-JIS	The encoding of a national item is specified as SJIS.
	UTF-16	The encoding of a national item is specified as UTF16.
	UTF-32	The encoding of a national item is specified as UTF32.
Endian of a national item		When the national item is UTF16 or UTF32, then specify little endian or big endian.
	LE - Little Endian	The national item is little endian.
	BE - Big Endian	The national item is big endian.

When compilation option ENCODE is specified, then the runtime code will be Unicode. However, when compilation option RCS is specified, then it is given priority.

Point

When compile option ENCODE (UTF8, UTF16) or ENCODE (UTF8, UTF32) is specified explicitly or implicitly, then spaces related to a national item become national spaces (alphabetic character spaces). This handling can be done by specifying the compile option NSP.

A.1.14 EQUALS compile option

For multiple records with the same key in the SORT statement, this option specifies whether to guarantee that the SORT output records are in the same order as the SORT input records at execution time (EQUALS), or not (NOEQUALS).

Item		Description
Processing mode of same key data in SORT statement		Specifies how to process data with the same key in the SORT statement. The default is NOEQUALS.
	EQUALS	Guarantees that the SORT output records are in the same order as the SORT input records at execution time.
	NOEQUALS	Does not guarantee that the SORT output records are in the same order as the SORT input records at execution time. Note: This option increases the SORT processing speed.

Note

If EQUALS is specified, special processing is executed to guarantee the order of input during the sorting operation, and the execution speed decreases.

A.1.15 FLAG compile option

This option specifies the diagnostic messages to be displayed.

Item		Description
Level of diagnostic message		Specifies a level of diagnostic messages. The default is I.
	I	Displays all diagnostic messages.
	W	Displays diagnostic messages of only W-level or higher.
	E	Displays diagnostic messages of only E-level or higher.

Note

The diagnostic message specified in the compiler option CONF is displayed regardless of the FLAG specification.

A.1.16 FLAGSW compile option

This option specifies whether to display messages that identify types of COBOL language elements (FLAGSW) or not (NOFLAGSW).

Item		Description
Display messages COBOL syntax language construction		Specifies whether to display messages that identify language elements. The default is NOFLAGSW.
	FLAGSW	Displays messages that identify COBOL language elements.
	NOFLAGSW	Does not display messages that identify COBOL language elements.
Language constructions		Select the language elements to be identified.
	STDM	Identifies all non-low-level '85 ANSI COBOL standard language elements.
	STDI	Identifies all non-mid-level '85 ANSI COBOL standard language elements.
	STDH	Identifies all non-high-level '85 ANSI COBOL standard language elements.
	SIA	Identifies language elements that are outside the range of the Fujitsu System Integration Architecture (SIA).
	RPW	Identifies language elements of the report writer function of the '85 ANSI COBOL standard. This item can be selected only if STDM, STDI, or STDH is selected.

Point

FLAGSW (SIA) is useful when creating a program that will run in another system.

Note

FLAGSW sub operands {STDM | STDI | STDH} and RPW cannot be omitted simultaneously.

A.1.17 FORMLIB compile option

This option specifies the folder of the screen form descriptor files from which record definitions are imported by the COPY statement with the IN/OF XMDLIB phrase. If the screen form descriptor files are stored in multiple folders, specify these folders using semicolon delimiters. If multiple folders are specified, the folders are searched in the order specified.

Item	Description
Screen form definition file folder	Specifies a folder of screen form descriptor files. Multiple folders can be specified using semicolons as delimiters. Click **Browse** to display the **Folder type selection** dialog box. Specify the absolute path of the folder. For some items selected in this way, the Folder selection dialog box appears, and you can select the folder to be specified.

A.1.18 INITVALUE compile option

This option specifies whether to initialize items without the VALUE clause in working-storage section data to their specified values (INITVALUE) or not (NOINITVALUE).

Item		Description
Initialize value of data item in WORKING-STORAGE		Specifies whether to initialize items without the VALUE clause in working-storage section data to their specified values. The default is NOINITVALUE.
	INITVALUE	Initializes the items.
	NOINITVALUE	Does not initialize the items.
Value		Specifies a two-digit hexadecimal number if INITVALUE is selected. If INITVALUE is selected, an entry must be made for this item.

A.1.19 LANGLVL compile option

This option specifies the base standard for interpreting source program items for which there are differences between new and old COBOL standards.

Item		Description
Specification of the ANSI COBOL standards		Specifies the base standard for interpreting a source program. The default is 85.
	85	Interpretation is based on the '85 ANSI COBOL standard.
	74	Interpretation is based on the '74 ANSI COBOL standard.
	68	Interpretation is based on the '68 ANSI COBOL standard.

A.1.20 LIB compile option

This option specifies a folder of library files used for a library function (COPY statement). If the library files are stored in multiple folders, use semicolons to delimit the folders. If multiple folders are specified, the folders are searched in the order specified.

Item	Description
Folder where the library texts exist	Specifies a folder of library files. Multiple folders can be specified by using semicolon delimiters. Click **Browse** to display the **Folder type selection** dialog box. Specify the absolute path of the folder. For some items selected in this way, the Folder selection dialog box appears, and you can select the folder to be specified.

A.1.21 LINECOUNT compile option

This option specifies the number of lines contained on each page of a compilation list. If the specified number is in a range of 0 to 12, the compilation list is not organized into pages, and data is output continuously across pages.

Item	Description
Number of lines per page for the compiler listings	Specifies the number of lines contained on each page of a compilation list. The default is 60. Specify an integer consisting of three or fewer digits. If this item is not specified, 60 is assumed.

A.1.22 LINESIZE compile option

This option specifies the maximum number of characters per line in a compilation list (i.e., A/N characters displayed in the list).

Item	Description
Number of characters per line for the compiler listings	Specifies the maximum number of characters per line in a compilation list. The default is 136. The specified value can be 80 or a three-digit integer ranging from 120 to 136. If this item is not specified 136 is assumed.

Note

- Source program lists, option information lists, diagnostic message lists, and compilation unit statistical information lists are output in a format with a fixed number of characters per line (120), regardless of the maximum number of characters specified for the LINESIZE compile option.
- The maximum valid value for the number of characters is 136. If a value exceeding 136 is specified for the LINESIZE compile option, 136 is assumed.

A.1.23 LIST compile option

This option specifies whether to output an object program list (LIST) or not (NOLIST). If enabled, an object program is output to the folder specified by the PRINT compile option.

Item	Description
Print program list or not	Specifies whether to output an object program list. The default is NOLIST.
LIST	Outputs an object program list.
NOLIST	Does not output an object program list.

Note

If the PRINT compile option is not specified, no object program list is output regardless of the specification of this option.

A.1.24 MAP compile option

This option specifies whether to output a data map list, program control information list, and section size list (MAP) or not (NOMAP). If enabled, a data map list, program control information list, and section size list are output to the file specified by the PRINT compile option.

Item	Description
Output data map list	Specifies whether to output a data map list, program control information list, and section size list. The default is NOMAP.

Item	Description
MAP	Outputs a data map list, program control information list, and section size list.
NOMAP	Does not output a data map list, program control information list, and section size list.

Note

To output a data map list, program control information list, and section size list, the PRINT compile option must be specified in addition to this option.

A.1.25 MESSAGE compile option

This option specifies whether to output an option information list and compilation unit statistical information list (MESSAGE) or not (NOMESSAGE).

Item	Description
Output of compiler options and statistical information	Specifies whether to output an option information list and a compilation unit statistical information list. The default is NOMESSAGE.
MESSAGE	Outputs an option information list and a compilation unit statistical information list.
NOMESSAGE	Does not output an option information list or a compilation unit statistical information list.

A.1.26 MODE compile option

This option specifies whether use right justification (MODE(STD)) or left justification (MODE(CCVS)) for copying numeric data during execution of the ACCEPT statement. This option specifies right or left justification when moving numeric data to an item on the receiving side, with a numeric receiving-side item in the coding format of "ACCEPT *identifier* FROM *mnemonic-name*".

Item	Description
Specification of ACCEPT statement's action	Specifies the operation of the ACCEPT statement. The default is STD.
STD	Copies numeric data with right justification.
CCVS	Copies numeric data with left justification.

Note

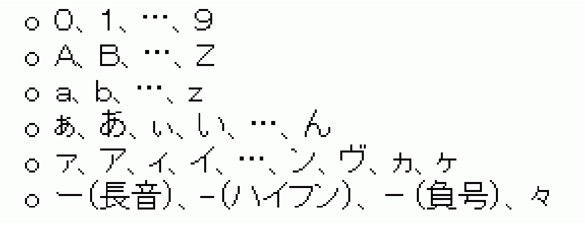
If MODE(CCVS) is specified, only an external decimal data item can be specified as a receiving item in the ACCEPT statement.

A.1.27 NCW compile option

This option specifies whether the national language character set that can be specified in user-defined words is the common one in the system (NCW(STD)) or the national character set of the computer (NCW(SYS)).

If STD is specified, the following national language character sets can be used for national language user-defined words:

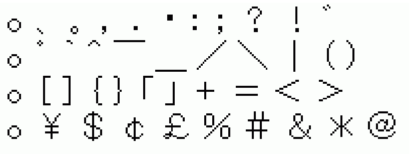
- JIS level-1 set
- JIS level-2 set
- JIS non-kanji set



○ 0、1、…、9
○ A、B、…、Z
○ a、b、…、z
○ ぁ、あ、ぃ、い、…、ん
○ ァ、ア、ィ、イ、…、ン、ヴ、ヵ、ヶ
○ ー(長音)、-(ハイフン)、－(負号)、々

If SYS is specified, the following national language character sets can be used for national language user-defined words:

- Character set with STD specified
- Extended characters
- Extended non-kanji set
- User-defined characters
- JIS non-kanji set



○ 、。，．・：；？！゛
○ ゜´＾￣＿／＼｜（）
○ ［］｛｝「」＋＝＜＞
○ ￥＄￠￡％＃＆＊＠

Item		Description
Character set of Japanese user-defined word		Specifies a character set for national language user-defined words. The default is STD.
	STD	Specifies the national language character set that is the common one in the system.
	SYS	Specifies the national language character set of the computer.

A.1.28 NSP compile option

Specify whether spaces related to a national data item are handled as national spaces (NSP) or as alphabetic character spaces (NONSP). In this option, encode is enabled for handling the trailing spaces related to a national data item of Unicode and a figurative constant SPACE. It is irrelevant when the encoding of national data items is other than Unicode.

Item		Description
Handling of spaces related to national data item		Specify whether spaces related to a national data item are handled. The default is NONSP.
	NSP	The national data item are handled as national spaces.
	NONSP	The national data item are handled as alphabetic character spaces.

A.1.29 NSPCOMP compile option

Use this option to specify how to treat a national space in a comparison. Specify NSPCOMP(NSP) to treat the national space as a national space; specify NSPCOMP(ASP) to treat the national space as an ANK space.

When NSPCOMP(ASP) is specified, an encode national space decided by compiler options is treated the same as the ANK blank in two bytes for UTF16, and treated to the same as the ANK blank in four bytes for UTF32.

The NSPCOMP(ASP) option is valid for the following comparisons:

- National character comparison that has a national item as an operand
- Nonnumeric comparison that has a group item as an operand

The NSPCOMP(ASP) option is invalid for the following comparisons:

- Comparison between group items that do not include national items
- Comparison between group items that include items that are not explicitly or implicitly used for display
- Comparison between group items including a national item with a different encode method
- Comparison of a national character that has an encode method different from the encode decided by the compiler options

Item		Description
Japanese-language space comparison specification		Use this option to specify how to treat a national space in a comparison. The default is NSP.
	NSP	Treats national spaces as national spaces.
	ASP	Treats national spaces as ANK spaces.

Note

- In a non-numeric comparison or national character comparison in processing INSPECT, STRING, or UNSTRING statements, or in a key operation for an index file, national spaces are not treated as equivalent to ANK spaces regardless of the specification of the NSPCOMP(ASP) option.
- If NSPCOMP(ASP) is specified, each ANK space is treated as a national character under the JAPANESE class condition.

A.1.30 NUMBER compile option

Information that identifies each line in a source program is contained in different lists that are output at compilation time and at execution time. This option specifies whether to use values in the sequence number area of a source program (NUMBER), or values generated by the compiler (NONUMBER) for such line number information.

Item		Description
Treatment of sequential number area of source program		Specifies how to treat the sequence number area of a source program. The default is NONUMBER.
	NUMBER	If a data item on any line in the sequence number area contains a character that is not a digit, or if sequence numbers are not assigned in ascending order, the line number of this line is changed to the number of the previous line (a correct sequence number) plus 1. If sequence numbers are assigned in descending order, a unique compensated number is added in the same format as that of a COPY qualification-value.
	NONUMBER	Line numbers are assigned in ascending order beginning from 1, in increments of 1.

Note

- If the same sequence number is repeated on two contiguous lines and NUMBER is specified, it is not regarded as an error.
- If NUMBER is specified, Go To on the context menu of the **Problems view** cannot be used.
- If the sequence number is in descending order, the source analysis information file output with compiler option SAI specified can only be used by a project browser. See "A.1.39 SAI compile option".
- When NUMBER is specified, you cannot debug it. Specify NONUMBER when debugging it.
- NUMBER cannot be specified for the pre-compiler. Specify NONUMBER.

A.1.31 OBJECT compile option

This option specifies whether to output an object program (OBJECT) or not (NOOBJECT). When an object program is output, its file is usually created in the same folder as that of the source program. To change the folder, specify a storage folder.

Item	Description
Output of object program	Specifies whether to output an object program. The default is OBJECT.
OBJECT	Outputs an object program.
NOOBJECT	Does not output an object program.
Folder for object program	Specifies the storage folder other than the source program folder for output of an object program. Click **Browse** to display the **Folder type selection** dialog box. Specify the absolute path of the folder. For some items selected this way, the **Folder selection** dialog box appears, and you can select the folder to be specified.

Note

In NetCOBOL Studio, when the object program folder and the source program folder are different, the target file cannot be created. When you create the target file, specify the same folder for the source program folder and for the object program folder.

A.1.32 OPTIMIZE compile option

This option specifies whether to create a globally optimized object program (OPTIMIZE) or not (NOOPTIMIZE).

Item	Description
Global optimization	Specifies whether a globally optimized object program is to be created. The default is: NOOPTIMIZE.
OPTIMIZE	Creates a globally optimized object program.
NOOPTIMIZE	Does not create a globally optimized object program.

Note

If this option is specified at the same time as the TEST compile option, the created debug information file can be used by the diagnostic function but not by the interactive debugger.

A.1.33 PRINT compile option

This option is used to specify the output of a compilation list. When a compilation list is output, its file is usually created in the same folder as the source program. To change the folder, specify a storage folder.

Item	Description
Compiler listing folder	Specifies the name of the compilation list output folder. Click **Browse** to display the **Folder type selection** dialog box. Specify the absolute path of the folder. For some items selected this way, the Folder selection dialog box appears, and you can select the folder to be specified.

A.1.34 QUOTE/APOST compile option

This option specifies whether to use a quotation mark (") (QUOTE) or an apostrophe (') (APOST) as the figurative constants QUOTE and QUOTES.

Item	Description
Value of figurative constants QUOTE and QUOTES	Specifies how to treat the figurative constants QUOTE and QUOTES. The default is QUOTE.
QUOTE	Uses the quotation mark (").
APOST	Uses the apostrophe (').

Note

Either the quotation mark or the apostrophe can be used in a source program to indicate a quotation mark, regardless of the specification of this option. However, the same character - quotation mark or apostrophe - must be used on the left and right sides of items in quotations. Examples of correct usage: "ABC", ' ABC'. Examples of incorrect usage: "ABC', 'ABC".

A.1.35 RCS compile option

This option specifies whether to use ASCII(RCS(ASCII)) or Shift-JIS (RCS (SJIS)) or Unicode (RCS(UTF16) or RCS(UCS2)) as the code set during execution. When ANSI code page (RCS(ACP)) is specified, it is specified in the Other compiler options text box.

Item	Description
Runtime code set	Specifies whether to use ASCII or Unicode during execution. The default value is ASCII.
ASCII	Uses ASCII as the code set during execution.
SJIS	Uses Shift-JIS as the code set during execution.
UTF16	Uses Unicode as the code set during execution.
UCS2	Uses Unicode as the code set during execution.
Endian in Unicode environment	Specifies which endianness is used when runtime code set is Unicode. The default value is LE.
LE	Unicode Little Endian is specified for the runtime code set.
BE	Unicode Big Endian is specified for the runtime code set.

Point

UTF16 and UCS2 are equivalent, but UTF16 is recommended.

Note

- For V11 and later specifying the ENCODE compile option is recommended.
- When compile option RCS is specified, it is assumed that the ENCODE compile option is specified in the following way.
  - When RCS(ACP) or RCS(ASCII) is specified explicitly, the ENCODE compile option cannot be specified.
  - When RCS(SJIS) is specified, it is as assumed that ENCODE(SJIS,SJIS) is specified.
  - When RCS(UTF16) or RCS(UCS2) is specified, it is as assumed that ENCODE(UTF8,UTF16) is specified.
- If the system locale is Japanese, the RCS(ACP) or RCS(ASCII) or RCS(SJIS) settings produce the same result.

  However, if you explicitly specify the RCS (ACP) or RCS(ASCII), the encoding of an alphanumeric character item works as ASCII. Behavior of National item is not guaranteed. In addition, it does not allow you to specify the ENCODING clause.

- For the operation when the system locale is not Japanese, refer to the "The system locale in Windows system" of the "NetCOBOL User's Guide."

A.1.36 REP compile option

Repository files are usually created in the same folder as that of the source program. Use this option to change the storage folder.

The specified folder can be the repository file input folder.

Item	Description
Repository file input and output folder	Specifies the repository file input-output folder. Click **Browse** to display the **Folder type selection** dialog box. Specify the absolute path of the folder. For some items selected in this way, the Folder selection dialog box appears, and you can select the folder to be specified.

A.1.37 REPIN compile option

This option specifies the repository file input folder. If repository files are stored in multiple folders, use semicolons to delimit the folders. If multiple folders are specified, the folders are searched in the order specified.

Item	Description
Repository file input folders	Specifies the repository file input folder. Multiple folders can be specified using semicolons as delimiters. Click **Browse** to display the **Folder type selection** dialog box. Specify the absolute path of the folder. For some items selected in this way, the Folder selection dialog box appears, and you can select the folder to be specified.

A.1.38 RSV compile option

This option specifies types of reserved words.

Item	Description
Kinds of reserved word	Specifies types of reserved words. The default is ALL.
ALL	For this product (Fujitsu NetCOBOL latest version)
V111	For OSIV COBOL85 V11L11
V112	For OSIV COBOL85 V11L20
V122	For OSIV COBOL85 V12L20
V125	For COBOL85 V12L50
V30	For COBOL85 V30
V40	For COBOL97 V40
V61	For COBOL97 V61
V70	For NetCOBOL V7.0
V81	For NetCOBOL V8.0
V90	For NetCOBOL V9.0
V1020	For NetCOBOL V10.0, V10.1, and V10.2
V1040	For NetCOBOL V10.4
V1100	For NetCOBOL V11.0
VSR2	For VS COBOLII REL2.0
VSR3	For VS COBOLII REL3.0

A.1.39 SAI compile option

This option specifies whether to output a source analysis information file (SAI) or not (NOSAI). When a source analysis information file is output, its file is usually created in the same folder as that of the source program. Use this option to change the storage folder.

Item	Description
Source analysis information file	Specifies whether to output a source analysis information file. The default is NOSAI.
SAI	Outputs a source analysis information file.
NOSAI	Does not output a source analysis information file.
Output location folder	Specifies the source analysis information file output folder. This item can be selected only if SAI is selected for Source analysis information file. Click **Browse** to display the **Folder type selection** dialog box. Specify the absolute path of the folder. For some items selected in this way, the Folder selection dialog box appears, and you can select the folder to be specified.

A.1.40 SDS compile option

When moving a signed packed-decimal item to another signed packed-decimal item, this option specifies whether to move the sign of a sending-side item as is (SDS) or move a formatted sign (NOSDS). There are two types of negative signs: X'B' and X'D'. The others are treated as positive signs. Here, formatting a sign means to convert the sign of the sending-side item from either the positive sign to X'C' or the negative sign to X'D'.

Item	Description
Arrangement of sign of decimal item	Specifies whether to format the sign of a signed decimal item. The default is SDS.
SDS	Moves the sign as is.
NOSDS	Formats the sign before moving it.

When NOSDS is specified, if it is a value for the sign to mean negative, it changes in minus symbol and if it is a value for the sign to mean positive, it changes to plus symbol. When minus symbol attaches to value 0, it changes to plus symbol.

Example

Program into which operation changes by option

```
    01  A PIC S9V9  VALUE -0.1
    01  B PIC S9.
        :
    MOVE A TO B.
```

When SDS is specified, the decimal point of -0.1 is rounded down and -0 is stored in B.

When NOSDS is specified, the decimal point of -0.1 is rounded down, it becomes -0, convert of the sign is done, and +0 is stored in B.

A.1.41 SHREXT compile option

In cases where the object format is multithread (THREAD(MULTI) phrase), this option specifies whether to share data and files with the external attribute (EXTERNAL phrase) among multiple threads (SHREXT) or not (NOSHREXT).

Item	Description
External data and files usage	Select whether to share data and files with EXTERNAL phrase among multiple threads or not. The default is NOSHREXT.
SHREXT	Share data and files with the EXTERNAL phrase among multiple threads.

Item		Description
	NOSHREXT	Do not share data and files with the EXTERNAL phrase among multiple threads.

Note

If the object format is single thread (THREAD(SINGLE)), SHREXT is displayed for the fixed compilation option, but compilation is performed under the condition of NOSHREXT.

A.1.42 SMSIZE compile option

This option specifies the amount of memory used by PowerBSORT, in kilobytes.

Item	Description
Specified capacity of memory used by PowerBSORT	Specifies the amount of memory used by PowerBSORT, in kilobytes. The default is 0 KByte.

Note

- If the PowerBSORT product is installed, this option is valid. If PowerBSORT is not installed, this option is ignored.
- Specify this option to limit the memory space used by PowerBSORT, which is invoked by the SORT and MERGE statements. For details on whether a specified value is valid, see the online manual for PowerBSORT.
- This option is equivalent to the values specified for the execution-time option smsize and the special register SORT-CORE-SIZE. If they are specified concurrently, the special register SORT-CORE-SIZE has highest priority, the execution-time option smsize has second priority, and the SMSIZE () compile option has third priority.

Example

- Special register

```
    MOVE 102400 TO SORT-CORE-SIZE
```

(102400 = 100 kilobytes)

- Compile option

```
  SMSIZE (500 KB)
```

- Execution-time option

```
  smsize 300 KB
```

In the above example, the value of 100 KB for the special register SORT-CORE-SIZE, which has top priority, is used.

A.1.43 SOURCE compile option

This option specifies whether to output a source program list (SOURCE) or not (NOSOURCE). If output, a source program list is output to the folder specified by the PRINT compile option.

Item	Description
Output of the source program list	Specifies whether to output a source program list. The default is NOSOURCE.

Item	Description
SOURCE	Outputs a source program list.
NOSOURCE	Does not output a source program list.

Note

If the PRINT compile option is not specified, no source program list is output, regardless of the specification of this option.

A.1.44 SQLGRP compile option

This option specifies whether to expand the definition method of SQL host variables (SQLGRP) or not (NOSQLGRP). When NOSQLGRP is specified, a REDEFINE clause cannot be specified to the host variable.

Item	Description
SQL host variable definition expansion	Specifies whether to expand the definition method of SQL host variables. The default is SQLGRP.
SQLGRP	Expands the definition method.
NOSQLGRP	Does not expand the definition method.

Note

When SQLGRP is specified, you cannot determine whether the data item has a correct format as a host variable just with the data description entry. This means that the syntax check for the host variable definitions during compilation differs as follows:

- The host variables not referenced in an embedded SQL statement are not subject to syntax check. Here, only the host variable names for the group items having global attribute are checked.
- More strict syntax check is made to variable-length character-type host variables. Group items containing items of level number 49 are handled as variable-length character-type host variables.

Specify NOSQLGRP for the conventional syntax check without using the host variables which are defined as group items.

A.1.45 SRF compile option

This option specifies whether to use the fixed format, variable format or free format for the reference format types of COBOL source programs and library files.

Item	Description
Source reference format	Specify the reference formats of a COBOL source program and library. "SRF and TAB compile option setting to be consistent with the applicable editor setting". If the checkbox is selected, SRF setting is to be consistent with the applicable editor setting. The value of SRF is initialized according to the applicable editor setting.
Source program	To specify the reference format for a COBOL source program, select Fixed, Variable or Free for Source program.
Library text	To specify the reference format for a library, select Fixed, Variable or Free for Library text.

A.1.46 SSIN compile option

This option specifies the data input source for the ACCEPT statement of the ACCEPT/DISPLAY function.

Item		Description
ACCEPT statement data input		Specifies the data input source for the ACCEPT statement. The default is SSIN(SYSIN).
	SSIN	Uses a file as the data input source.
	SSIN(SYSIN)	Uses the console window as the data input source.
Environment variable to specify an input file		Specifies the name of the environment variable that sets the input source file.

Note

The environment variable information name must consist of eight or fewer upper-case letters and/or numeric characters, beginning with an upper-case letter (A to Z). The environment variable information name must not match one used by another file (file identifier name).

A.1.47 SSOUT compile option

This option specifies the data output destination for the DISPLAY statement of the ACCEPT/DISPLAY function.

Item		Description
DISPLAY statement data output		Specifies the data output destination for the DISPLAY statement. The default is SSOUT(SYSOUT).
	SSOUT	Uses a file as the data output destination.
	SSOUT(SYSOUT)	Uses the console window as the data output destination.
Environment variable to specify an output file		Specifies the name of the environment variable that sets the output destination file.

Note

The environment variable information name must consist of eight or fewer upper-case letters and/or numeric characters, beginning with an upper-case letter (A to Z). The environment variable information name must not match one used by another file (file identifier name).

A.1.48 STD1 compile option

This option specifies that alphanumeric codes (standard one-byte character codes) in the EBCDIC phrase of the ALPHABET clause be treated as ASCII codes (ASCII), JIS8 unit codes (JIS1), or JIS7 unit Roman letter codes (JIS2).

Item		Description
Treatment code of alphanumeric character EBCDIC in ALPHABET clause		Specifies how to treat alphanumeric codes in the EBCDIC phrase of the ALPHABET clause. The default is JIS2(JIS7 code).
	ASCII(ASCII)	Uses EBCDIC (ASCII) for the character code convention set.
	JIS1(JIS8 code)	Uses EBCDIC (kana) for the character code convention set.
	JIS2(JIS7 code)	Uses EBCDIC (lower-case alphabetic characters) for the character code convention set.

A.1.49 TAB compile option

This option specifies whether to treat the tab character as having a width of four columns (TAB(4)) or eight columns (TAB(8)).

Item		Description
Set tab positions		Specifies whether to treat the tab character as having a width of four or eight columns. The value set for editors is the default.
	8	Designates a width of 8 columns to the tab character.
	4	Designates a width of 4 columns to the tab character.

A.1.50 TEST compile option

This option specifies whether to enable the NetCOBOL Studio debugging function and the COBOL Error Report at the execution time (TEST) or not (NOTEST). If TEST is specified, a debugging information file for use by the NetCOBOL Studio debugging function and the COBOL Error Report is created. The file is usually stored in the same folder as that of the source program. You can use this option to change the storage folder.

Item		Description
Using the debugger		Specifies whether to use the debugging function of NetCOBOL Studio. The default is NOTEST. If debugging is the target, NOTEST is cleared and EST is selected.
	TEST	Uses the debugging function of NetCOBOL Studio.
	NOTEST	Does not use the debugging function of NetCOBOL Studio.
Folder for debugging information		Specifies the debugging information file storage folder.

Note

- If this option is specified with OPTIMIZE, the debugging information file can be used by the COBOL Error Report, and cannot be used by the debugging function of NetCOBOL Studio. Do not specify OPTIMIZE when you use the debugging function of NetCOBOL Studio.
- When debugging, store the debugging information file in the same folder as the target file.

A.1.51 THREAD compile option

This option specifies whether the object format is multithread (THREAD(MULTI)) or single thread (THREAD(SINGLE)).

Item		Description
Type of thread mode		Specifies an attribute of an object file as multi-thread or single thread. The default is SINGLE.
	MULTI	Specifies an attribute of an object file as multi-thread.
	SINGLE	Specifies an attribute of an object file as single thread

A.1.52 TRACE compile option

This option specifies whether to use the TRACE function (TRACE) or not (NOTRACE).

Item		Description
Tracing procedure names		Specifies whether to use the TRACE function. The default is NOTRACE.
	TRACE	Uses the TRACE function.
	NOTRACE	Does not use the TRACE function.

Item	Description
Number of variables showing trace information	Specifies an integer ranging from 1 to 999999 as the number of trace information items to be output. The default is 200.

Note

- If TRACE is specified, processing to display trace information is embedded in an object program, decreasing execution performance. After debugging, recompile the program with NOTRACE specified.
- TRACE cannot be specified together with the COUNT compile option. If both of these options are specified together, the option specified last is valid.

A.1.53 TRUNC compile option

This option specifies whether to truncate upper digits in numeric characters when converting into binary (TRUNC), or to give priority to execution speed regarding truncation (NOTRUNC).

Item		Description
Truncation of high-order digits when converting into binary items		Specifies whether to truncate the upper digits of numeric characters when converting to binary. The default is NOTRUNC.
	TRUNC	Truncates the number of upper digits according to the PICTURE clause of the receiving-side item, and saves the resultant value in the receiving-side item. If this item is specified with the OPTIMIZE compile option, optimization truncates the number of upper digits of data items that are from zoned decimal items or packed decimal items. The number of upper digits is truncated as described above only if the number of digits in the integer part of the sending-side item is greater than that of the receiving-side item.
	NOTRUNC	Gives priority to the object program execution speed. If the execution speed would be higher without truncation of the upper digits, the digits will not be truncated.

Point

In the PICTURE clause:

- Moving S999V9 (three-digit integer part) to S99V99 (two-digit integer part): The digits are truncated.
- Moving S9V999 (one-digit integer part) to S99V99 (two-digit integer part): The digits are not truncated.

Note

- If NOTRUNC is specified and the number of digits in the integer part of the sending-side item is greater than that of the receiving-side item, the results are not defined.
- To specify NOTRUNC, the applicable program must be designed such that no value exceeding the digit number in the PICTURE clause is stored in the receiving-side item, even if the truncation of digits is not applied.
- The criteria for whether to truncate the digits in cases where NOTRUNC is specified are different for different compilers. The compatibility with another system is not guaranteed for a program that depends on the specification of NOTRUNC to suppress truncation.

A.1.54 XREF compile option

This option specifies whether to output a cross reference list (XREF) or not (NOXREF). A cross reference list displays information such as user-defined words and special registers in ascending order of the collating sequence. If XREF is specified, the cross reference list is output to the folder specified by the PRINT compile option.

Item	Description
Output of the cross-reference list	Specifies whether to output a cross reference list. The default is NOXREF.
XREF	Outputs a cross reference list.
NOXREF	Does not output a cross reference list.

Note

- If the PRINT compile option is not specified, no cross reference list is output regardless of the specification of this option.
- If the maximum severity code is S level or higher as a result of compilation with the XREF compile option specified, output of a cross reference list is suppressed.

A.1.55 ZWB compile option

In a comparison between a signed zoned decimal item and an alphanumeric character field, this option specifies whether to ignore the sign part of the zoned decimal item (ZWB) or not (NOZWB). Here, an alphanumeric character means an alphanumeric data item, alphabetic item, alphanumeric edited data item, numeric edited data item, nonnumeric literal, or figurative constant other than ZERO.

Item	Description
Comparison of signed external decimal item and alphanumeric item	Specifies a method for comparing a signed zoned decimal item and an alphanumeric character field. The default is ZWB.
ZWB	Compares the items ignoring the signed part.
NOZWB	Compares the items including the signed part.

Example

```
77  ED  PIC  S9(3)  VALUE  +123.
77  AN  PIC  X(3)  VALUE  "123".
```

Under these conditions, the logical value of the conditional expression ED = AN is as follows:

- If ZWB is specified ... True
- If NOZWB is specified ... False

Appendix B Troubleshooting Guide

B.1 Build Problems

B.1.1 After a resource is updated with an external editor, build is not executed when Build Project or Build All is selected.

Issue: Resource changes have not been reflected in the workspace concerned.

Solution: Take any of the following actions:

- Select **Window** > **Preferences** > **General** > **Workspace** from the menu bar, and then select "Refresh automatically". If this option is selected, resources in the workspace are automatically synchronized with the corresponding resources in the file system.
- When an external editor has been used to update a resource, before executing a build, select the updated resource or the folder containing it in the **Dependency** or **Structure view**, and then select **File** > **Refresh** from the menu bar.

B.1.2 A message "repository file (*.rep) was not found" is displayed in a dependency analysis.

Issue: The **Console view** displays the following error message in a dependency analysis:

full-path-name-for-a-cobol-source-file: line-number: repository-file-name: does not exist.

Solution: Check for the following:

- A repository file of COBOL source files in the project. Since a project contains no repository file immediately after the project is created or cleaned, this error message is displayed. Repository files are created when a build for the project is executed, so no special response is required for this message.
- A repository file outside the project. Confirm that coding in the COBOL source files is correct and that all the necessary repository files have been provided.

B.1.3 A static library (lib) cannot be made

Issue: Static library (lib) cannot be made with NetCOBOL Studio though executable file (exe) or dynamic link library (dll) can be made.

Solution: Use the LIB command to make static library (lib). Refer to the technical intelligence of Microsoft Corporation for detail of the LIB command.

B.2 Debugger problems

B.2.1 Menu items in the Run menu cannot be used during debugging.

Issue: The following menu items in the Run menu are grayed out and their functions are disabled during debugging of a COBOL application:

- Run to Line
- Use Step Filters
- Watch
- Inspect
- Display
- Execute
- Step Into Selection
- Toggle Line Breakpoint

- Toggle Method Breakpoint
- Toggle Watchpoint

Solution: These menu items are used for debugging a Java application, and cannot be used for a COBOL application.

- To add or delete a breakpoint, right-click on the vertical ruler of the COBOL editor, and select **Add Breakpoint** or **Remove Breakpoint** from the context menu.
- To execute a program up to the specified line, select "Run to Line" from the context menu of the COBOL editor.

B.2.2 Values of data items outside the scope are displayed during debugging.

Issue: Values of data items outside the scope are also displayed, but they are not correct values.

Solution: Do not use the values displayed for data items outside the scope.

B.2.3 An error dialog box containing the message "Target executable not found. Could not proceed" appears.

Issue: If a COBOL application is executed or debugged in a workspace created by copying or moving a workspace folder containing the COBOL project, an error dialog box containing the message "Target executable not found. Could not proceed." appears.

Solution: Although the workspace folder containing the COBOL project was copied or moved, the path name used before the copy or move remains defined in Executable file on the **Main** tab of Debug or Run for the COBOL application. This discrepancy causes this message. In this case, click the **Browse** button of Project name on the **Main** tab of Debug or Run, and select the project name again in the **Project Selection** dialog box. The path name is changed to the correct one.

B.2.4 Program execution cannot be restarted after it stops at a breakpoint.

In the **Debug view**, if a selection is made outside the tree element displaying the path of the program being debugged, the program remains stopped even if "Resume" or "Step Into" is selected to restart program execution.

Solution: In the **Debug view**, make a selection from the tree element displaying the path of the program, and then select "Resume" or "Step Into" to restart program execution.

B.3 Execution Problems

B.3.1 A runtime error occurs when an executable file is double-clicked.

Issue: When you double-click an executable file (.exe), a runtime initialization failure occurs.

Solution: See Chapter 8 Execution Function", and verify that the correct folder is specified in the Working folder field.

B.4 Remote Development Problems

B.4.1 The stty command causes an error in link with a server.

Issue: The sty command causes an error and fails when linking to a Solaris or Linux server that uses the ftpd/rexec service.

Solution: Comment out the stty command coded in the .bashrc file (if bash is used) or .cshrc file (if csh is used).

Before modification:

```
stty erase ^H
```

After modification:

```
#stty erase ^H
```

B.4.2 NetCOBOL Studio freezes when linking with a server.

Issue: The server does not respond when linking to a Solaris or Linux server that uses the ftpd/rexec service and NetCOBOL Studio freezes.

Solution: This problem occurs when the .bashrc file (if bash is used) or .cshrc file (if csh is used) has been coded with a script that requests user input and involves waiting for a response. Delete the script that causes waiting for a response, such as one that requests user input, from the file.

B.4.3 The error message "Error occurred while sending command. Remote computer reset connection" is displayed.

Issue: The "Error occurred while sending command. Remote computer reset connection." error occurs when linking to a Solaris or Linux server that uses the ftpd/rexec service, and connection to the server is not possible.

Solution: The domain name system (DNS) controls domain names and IP addresses. With some DNS settings, the rexec daemon fails in a reverse search for the host name of a personal computer connected to the server. Edit the /etc/hosts file on the server to add the following information about the client personal computer:

```
IP address  host name
```

B.4.4 The line feed code is not converted when a text file is transferred.

Issue: If vsftpd is used for file transfer (FTP) in Linux server remote development, when a text file is transferred the line feed codes remain as Windows line feed codes (0x0d0a) rather than changing to Unix line feed codes (0x0a).

Solution: To enable conversion of the line feed code, delete # at the beginning of the following lines, which are commented out, in the /etc/vsftpd/vsftpd.conf file:

Before modification

```
#ascii_upload_enable=YES
#ascii_download_enable=YES
```

After modification

```
ascii_upload_enable=YES
ascii_download_enable=YES
```

B.4.5 The error message "Parameter of server command is invalid" is displayed when using remote development functions.

Issue: The line feed code is not converted according to vsftpd settings.

Solution: To enable conversion of the line feed code, delete # at the beginning of the following lines, which are commented out, in the /etc/vsftpd/vsftpd.conf file:

Before modification

```
#ascii_upload_enable=YES
#ascii_download_enable=YES
```

After modification

```
ascii_upload_enable=YES
ascii_download_enable=YES
```

B.4.6 Local project build runs when starting remote debug.

Issue: The **Build (if required) before launching** checkbox is checked.

Solution: Following the steps below to uncheck the checkbox.

1. Select **Window** > **Preferences** from the menu bar. The **Preferences** dialog box is displayed.
2. Select **Run/Debug** > **Launching** from the left pane. The **Launching** page is displayed.
3. Uncheck the **Build (if required) before launching** checkbox in the General Options group box.

B.4.7 Nothing appears in the Problems view even though the remote build failed.

For remote builds, the **Problems view** displays only compile errors. To check other errors, select "COBOL Remote" from the toolbar icon ("Open Console") in the **Console view**.

B.5 NetCOBOL Studio General Problems

B.5.1 An error dialog box stating "for details refer to log" appears.

Issue: An error occurs directing you to refer to the log for more information.

Solution: The log can be referenced by selecting **Help** > **About NetCOBOL Studio(x64)** from the menu bar, selecting "Configuration Details" in the dialog box that appears, and then selecting "View Error Log" in the **Configuration Details** dialog box.

To display the Error Log view and reference the log:

1. Select **Window > Show View** > **Other** from the menu bar.
2. Select **General** > **Error Log** in the tree in the **Show View** dialog box that appears.
3. Click the **OK** button to display the Error Log view.

Information such as when each workspace was created, when NetCOBOL Studio was started, and what errors have occurred in NetCOBOL Studio are recorded in the log. Some log information will indicates normal NetCOBOL Studio operation. Problems are recorded at the time they occur in NetCOBOL Studio. If no NetCOBOL Studio operation problems are indicated in the error log, the error dialog box display may be caused by an erroneous operation, such as trying to do a build for remote development with incorrect server information. If a NetCOBOL Studio operation problem does appear in the error log, collect the log information listed below and contact a Fujitsu certified service engineer.

The log files are created in the following folder:

```
<work-space-folder>\.metadata\
```

The log files are named as follows:

```
.bak_0.log
.bak_1.log
:
:
.bak_9.log
```

Existing log files contain information sorted in descending time stamp order, with the most recent item at the top of the list.

B.5.2 Entire message containing "..." cannot be displayed.

Issue: If a message to be displayed in a dialog box is longer than the message display field, the middle of the message may be represented by "...".

Solution: The entire message can be displayed using one of the following methods:

- Move the mouse cursor onto the displayed message whose middle part is omitted. A tooltip appears displaying the entire message.
- If the dialog box can be resized, expand the width of the dialog box so that the entire message can be displayed.

B.5.3 Files, folders, and projects cannot be deleted.

Issue: If a file is generated with a path that exceeds the maximum value for file system path lengths, deletion fails for that file and for the folder and project that include that file.

Solution: Use the method below to delete the file and folder

```
java -classpath (NetCOBOL install folder) \eclipse\f5drprfc.jar RemoveFolderContents folder name
```

This command requests confirmation on each file and folder in the specified folder, before deleting them. The specified folder itself is not deleted.

B.5.4 The Export of COBOL Project Fail

Issue: After a resource is updated with an external editor, changes are not reflected in the workspace.

Solution: Take any of the following actions:

- Select **Window** > **Preferences** > **General** > **Workspace** from the menu bar, and then select "Refresh automatically". When this option is selected, resources in the workspace are automatically synchronized with the corresponding resources in the file system.
- When an external editor is used to update a resource, before executing the export, select the updated resource, or the folder containing it in the **Dependency** or **Structure view**, and then select "Refresh" from the context menu.

B.6 Trace log

Eclipse has the tracing function that output the execution state of Plug-in to logs.

This function facilitates inquiry of causes at the occurrence of troubles.

To use the tracing function in NetCOBOL Studio, select **Window > Preferences** > **General** >**Tracing** > **Enable tracing**.

Please output to log when asked by Fujitsu certified service engineer.

The default select state is off, but it is recommended to check box.

Appendix C Handling of Workspace and Project

This section explains the handling of workspace and project.

C.1 Default workspace

The default workspace is in the following folder. It is created when NetCOBOL Studio is first started.

```
My documents folder (*1) \NetCOBOL Studio V11.1.0(x64)\workspace4.3
```

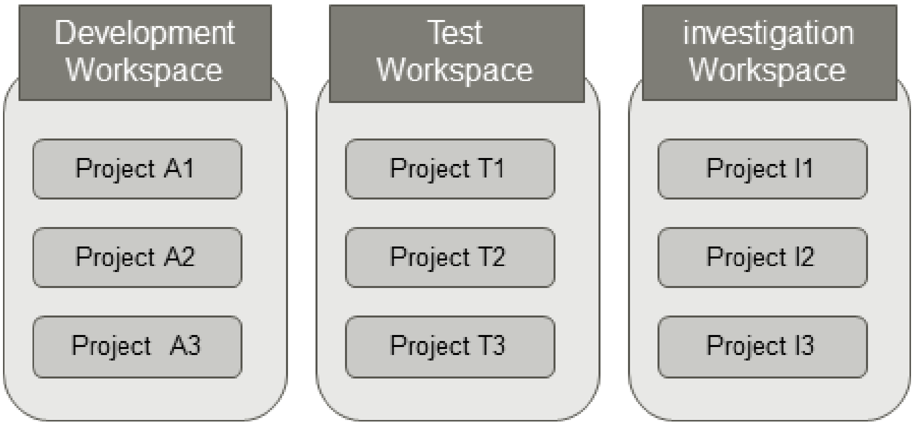
*1: The "My documents" folder is used to save each user's data. The location of the "My documents" folder varies based on the Windows OS.

C.2 Setting and switch method of workspace

Workspace

A workspace is the place where development resources and the user's working state are maintained. Development resources are managed in the workspace as a project. Multiple projects can be created in a workspace. The working state is information such as workbench setting and the break point set in the source file, and is maintained in the workspace.

The workspace can fall into several categories, such as workspace for development, workspace for investigation, and workspace for test.



Note

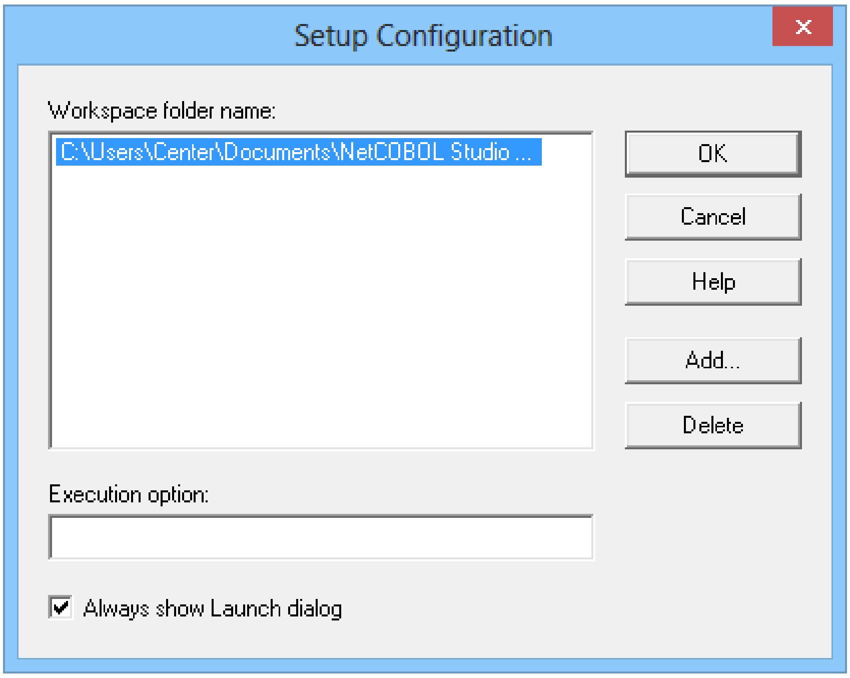
- A workspace can be shared among multiple users or machines, but the same workspace cannot be used among multiple Eclipse bases at the same time.
- A project can be placed out of workspace folder, but do not refer to the same project in multiple workspaces. If a project is used by multiple workspaces, the alteration of project information cannot function normally.

C.2.1 Setting workspace

The Workspace is set using the **Setup Configuration** dialog box according to the following procedures.

1. Select the up arrow > **Apps** > **Fujitsu NetCOBOL V11(x64)** > **NetCOBOL Studio(x64)** from the **Start** menu.
   NetCOBOL Studio Start screen displays.

2. Click the **Setup...** button. The **Setup Configuration** dialog box is displayed.



Setup Configuration

Workspace folder name:

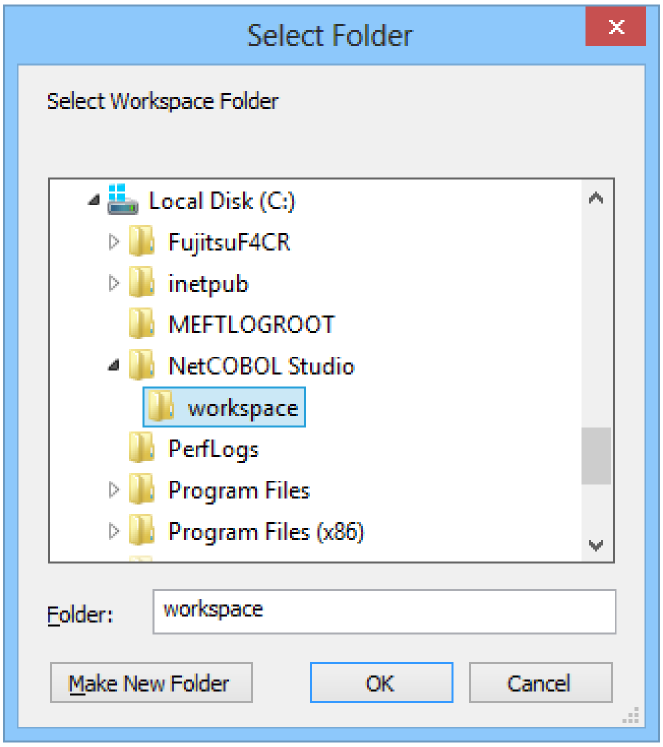
C:\Users\Center\Documents\NetCOBOL Studio ...

Execution option:

☑ Always show Launch dialog

3. Click the **Add** button. The **Select Folder** dialog box is displayed.

4. Select the folder used as the workspace. A "New folder" is made in the selected folder when the **Make New Folder** button is clicked. It can be used by changing the name.



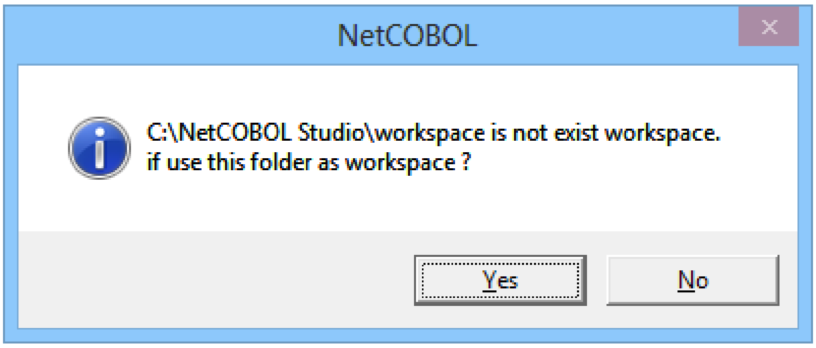
Select Folder

Select Workspace Folder

Folder: workspace

Select a folder, and click the **OK** button. In this example, "C:\NetCOBOL Studio\workspace" is set as the new workspace.

5. Click the **Yes** button since the following message box is displayed.

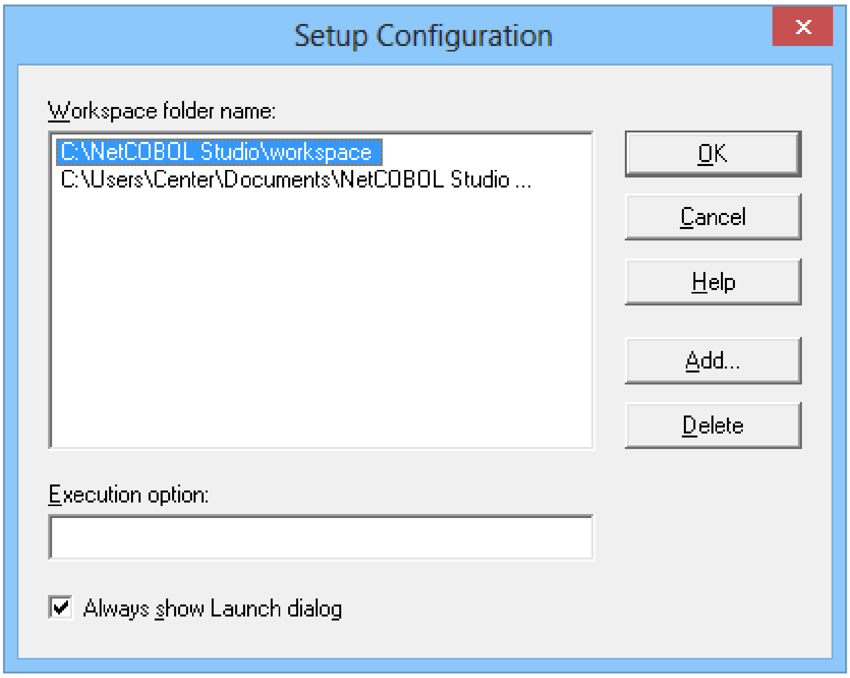


NetCOBOL

C:\NetCOBOL Studio\workspace is not exist workspace.
if use this folder as workspace ?

The list of workspaces is displayed in the "Workspace folder name".

6. Select the Workspace, and click the **OK** button.



Setup Configuration

Workspace folder name:

C:\NetCOBOL Studio\workspace
C:\Users\Center\Documents\NetCOBOL Studio ...

Execution option:

☑ Always show Launch dialog

NetCOBOL Studio Start screen displays.

7. Click the **Run** button on the Start screen.

NetCOBOL Studio is started. The Workspace becomes "C:\NetCOBOL Studio\workspace".

Point

When NetCOBOL Studio starts next time, this setting becomes effective.

C.2.2 Switch of workspace

The Workspace can be switched to another workspace when NetCOBOL Studio is active.

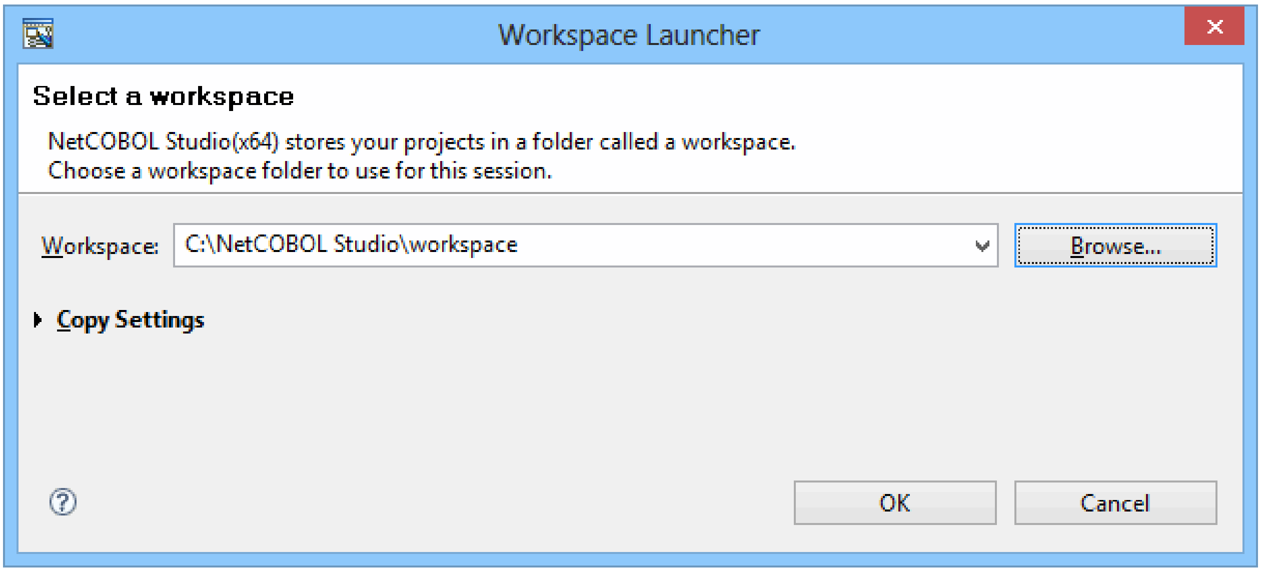
1. Select **File** > **Switch Workspace** > **Other** from the NetCOBOL Studio menu bar.

   The **Workspace Launcher** dialog box is displayed. The present workspace is displayed in "workspace".

2. Enter the switched new workspace name for "Workspace" using the full path.

   The folder can be selected by using the **Browse** button.

Moreover, when the "Workspace" drop-down button is selected, the history of the workspaces used so far is displayed. The Workspace can be selected from the list displayed.



Workspace Launcher

Select a workspace

NetCOBOL Studio(x64) stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\NetCOBOL Studio\workspace

3. Click the **OK** button.

   NetCOBOL Studio is re-started. The Workspace becomes the specified workspace.

C.3 Importing Project

In the following cases, the projects are imported and used.

- When you want to use the project made in Windows 32bit NetCOBOL Studio in Windows 64bit NetCOBOL Studio.

Use the following method to import an existing project in batch mode.

1. Set the workspace of the copy destination, and start NetCOBOL Studio.
2. Select **File** > **Import** from the NetCOBOL Studio menu bar. The **Import wizard** is started.
3. Select **General** > **Existing Projects into Workspace**, and click the **Next** button.
4. Select "Select root directory", and click the **Browse** button. The **Browse For Folder** dialog box is displayed.
5. Select the workspace folder that includes the project, and click the **OK** button.
6. Confirm that the projects are displayed in the "Projects" section, click the **Select All** button. Select "Copy projects into workspace", and click the **Finish** button.

Note

A link error might occur when "Build" is selected for output files that are built on each different platform.

In case of the first time building projects that are imported from NetCOBOL Studio on each different platform, select "Rebuild".

Appendix D Various Specification Formats used in Debugging

This section explains various specification formats used in the **Watch view**.

D.1 Identifier Name

When adding data items for monitoring to the **Watch view**, use the identifier name as data name.

One of the following is the identifier name.

- Identifier
- Condition-name
- Index Name
- Special Register
- Symbolic-constant
- Named Literal
- Function

It is necessary to enclose the identifier name in parenthesis when it contains spaces. See the example below.

```
LIST (DATA01 OF DATA02)
```

Identifier

The identifier can be specified with the same scope of COBOL language specifications in debugging.

In addition, the data described in the factory definition and the object definition can be handled from external, by using the same specification as the original identifier.

Specification Format of Identifier

It is the same as the language specification of COBOL with the exception of the following restrictions for subscript and reference modifier.

- Subscript is specified by the following formats.

Subscript	{Integer | Identifier[{+ | -}Integer]| Index Name[{+ | -}Integer]| Symbolic-constant[{+ | -}Integer]| Named Literal[{+ | -}Integer]}

- Reference modification is specified by the following formats.

Reference Modifier	(High order end character position:[Length])
High order end character position/ Length	{Integer | Identifier[{+ | -}Integer]| Symbolic-constant[{+ | -}Integer]| Named Literal[{+ | -}Integer]}

- Predefined object identifier can be specified except SUPER.
- Identifier is specified by the following formats. Underlined part is the part which is extended originally.

Identifier	[{pointer qualifier | object reference qualifier}] data name [{IN | OF} data name]... [{IN | OF}{file name | report-name }] [({subscript}...)] [reference modifier]
Pointer Qualifier	[(] ... {object reference qualifier] {data name[{IN | OF}data name] ... | (data name [{IN | OF} data name]... ({subscript}...))} | ADDR function} -> [{ data name[{IN| OF}data name] ... | (data name [{IN| OF} data name] ... ({subscript}...))}) ->]
Object Reference Qualifier	[class-name[predefined object identifier SUPER] ::] [object reference item [{IN | OF}data name] ... [({subscript}...)] [predefined object identifier SUPER] ::] ...
Predefined Object Identifier SUPER	AS class-name

- Identifier is specified by the internal name under the class condition.
- Object reference qualifier is specified under the condition that handling data which is described in factory definition or object definition from external. The class-name of object reference qualifier specifies the internal name.
- Under the condition that identifying super-class data, predefined object identifier SUPER specifies the super-class name. Super-class name specifies the internal name or the external name.

Condition-name

Condition-name can be specified with the same scope of COBOL language specifications in debugging.

In addition, the condition-name described in the factory definition and the object definition can be handled from external, by using the same specification as the original condition-name.

Specification Format of condition-name

It is the same as the language specification of COBOL with the exception of the following restrictions for subscript.

- Subscript is specified by the following formats.

Subscript	{Integer | Identifier[{+ | -}Integer]| Index Name[{+ | -}Integer]| Symbolic-constant[{+ | -}Integer]| Named Literal[{+ | -}Integer]}

- Condition-name is specified by the following formats. The underlined part is the part that is extended originally.

Condition-name	[{pointer qualifier | object reference qualifier}] condition-name [{IN | OF} data name]... [{ IN | OF}file name]

	[({subscript}...)
Pointer Qualifier	[( ] ... {object reference qualifier] { data name [{ IN | OF} data name] ... | (data name [{IN | OF} data name] ... ({subscript}...))} | ADDR function} -> [{ data name [{ IN| OF} data name] ... | (data name [{IN| OF} data name] ... ({subscript}...))}) ->]
Object Reference Qualifier	[class-name [predefined object identifier SUPER] ::] [object reference item [{ IN | OF} data name] ... [({subscript }...)] [predefined object identifier SUPER] ::] ...
Predefined Object Identifier SUPER	AS class-name

- Object reference qualifier is specified under the condition that handling condition-name which is described in factory definition or object definition from external. The class-name of the object reference qualifier specifies the internal name.
- Under the condition that identifying super-class data, predefined object identifier SUPER specifies the super-class name. Super-class name specifies the internal name or the external name.

Index-name

Index-name can be specified with the same scope of COBOL language specifications in debugging.

In addition, the index-name described in the factory definition and the object definition can be handled from external, by using the same specification as the original index-name.

Specification Format of Index-name

It is the same as the language specification of COBOL. The underlined part is the part that is extended originally.

Index-name	[object reference qualifier] index-name [{IN | OF} data name]... [{IN | OF} file name]
Object Reference Qualifier	[class-name[predefined object identifier SUPER] ::] [object reference item [{IN | OF}data name] ... [({subscript}...)] [predefined object identifier SUPER] ::] ...
Predefined Object Identifier SUPER	AS class-name

- Object reference qualifier is specified under the condition that handling index-name which is described in factory definition or object definition from external. The class-name of the object reference qualifier specifies the internal name.
- Under the condition that identifying super-class data, predefined object identifier SUPER specifies the super-class name. Super-class name specifies the internal name or the external name.

Special Register

Special register can be specified with the same scope of COBOL language specifications in debugging.

Symbolic-constant

Symbolic-constant can be specified with the same scope of COBOL language specifications in debugging.

Named Literal

Named literal can be specified with the same scope of COBOL language specifications in debugging.

Function

The following functions can be specified in debugging.

Function	{ADDR Function | LENG Function | LENGTH Function}
ADDR Function	FUNCTION ADDR(argument)
LENG Function	FUNCTION LENG(argument)
LENGTH Function	FUNCTION LENGTH(argument)

- The argument of the ADDR function must be an identifier beyond Internal Boolean data item and object reference item.
- The argument for the LENG function and LENGTH function must be either data item, nonnumeric literal, hexadecimal nonnumeric literal or national nonnumeric literal.

D.2 Program name

The specification formats for program name are as follows.

```
{external program name |
[external program name].internal program name |
[class-name] :method-name}
```

- external program name

  Specify program definition (external program).

- [external program name].internal program name

  Specify program definition (internal program). It is considered that an external program name of implied program name is specified when the external program name is omitted. The internal program name referred here is the program that contains the external and internal program.

- [class-name]:method-name

  Specify method definition (factory) or method definition (object). When a user-defined property method is specified, refer to Specification Formats of Property Method. It is considered that class-name of implied program name is specified when the class-name is omitted.

To use a program name that contains a period (.), colon(:) or other special character, specify the external name and internal name with a nonnumeric literal.

Specification Formats of Property Method

When debugging a user-defined property method, specify the method-name using the following formats.

- GET method _ GET _ property name
- SET method _ SET _ property name

Appendix E Transition from Project Manager

This section explains how to transfer COBOL application development environment from NetCOBOL project manager for Windows (x86) to NetCOBOL Studio, and how to use the project organization converting command that is the transference support tool.

E.1 Difference Between Project Manager and NetCOBOL Studio

Project manager and NetCOBOL Studio are both development environment tools for developing COBOL applications. Project manager has provided as COBOL development environment tool in NetCOBOL for Windows(x86) from the older version of NetCOBOL for Windows. Now, more companies are using COBOL alongside of JAVA, and the NetCOBOL Studio IDE allows people to choose a more flexible development environment in terms of easily linking in multiple types of objects. And since the NetCOBOL studio is developed in open-source Eclipse, it is a very flexible and extensible development environment.

Note

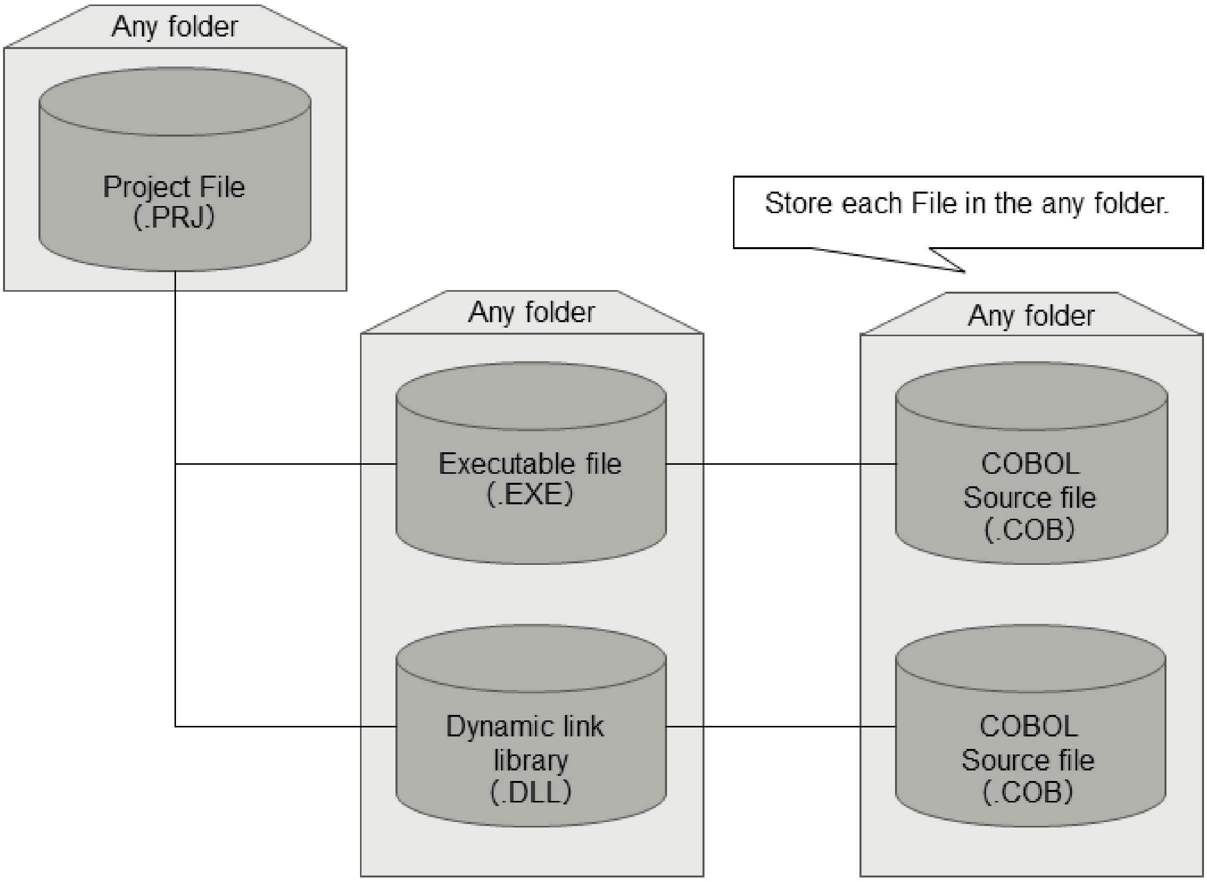
NetCOBOL Studio is the only allowable development IDE for NetCOBOL for Windows(x64). You cannot use the NetCOBOL Project Manager for development in that environment.

Projects developed in different IDEs use different storage methods; thus, projects are stored in NetCOBOL Project Manager and NetCOBOL Studio with different internal formats. Particularly in NetCOBOL Studio, it is imperative to store projects in a directory structure that exactly matches that shown in the IDE workspace. The same is not true with NetCOBOL Project Manager. When you convert from NetCOBOL Project Manager to NetCOBOL Studio, you will need to handle the differences in how the individual components of the project are actually stored.

Project Manager

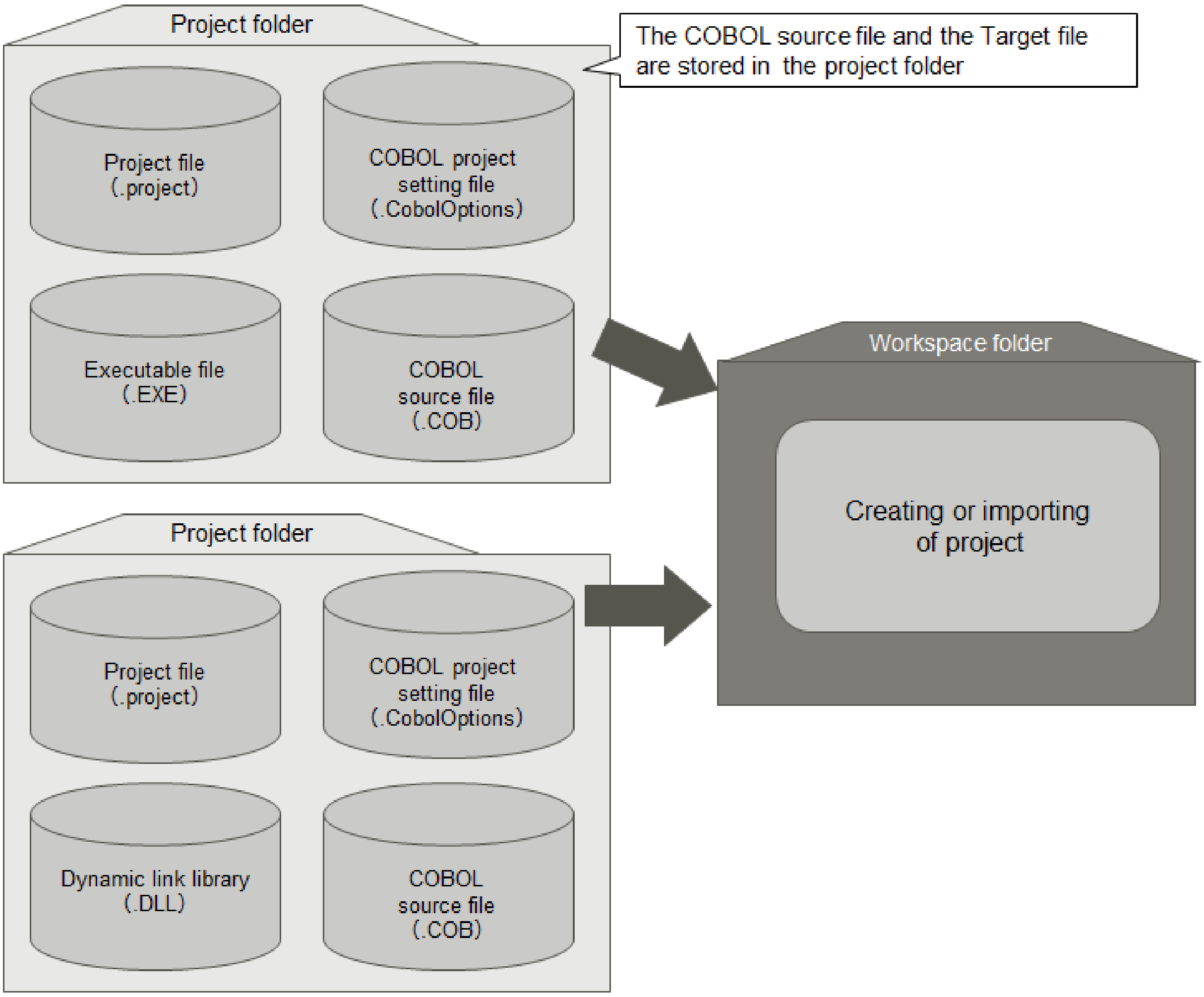
The Project Manager is a tool to manage the resources and the compiler options as a project.

The project manager can register two or more targets (dynamic link library (.DLL) or executable file (.EXE)) in one project file. And the place of the managed resource files can be flexibility decided.



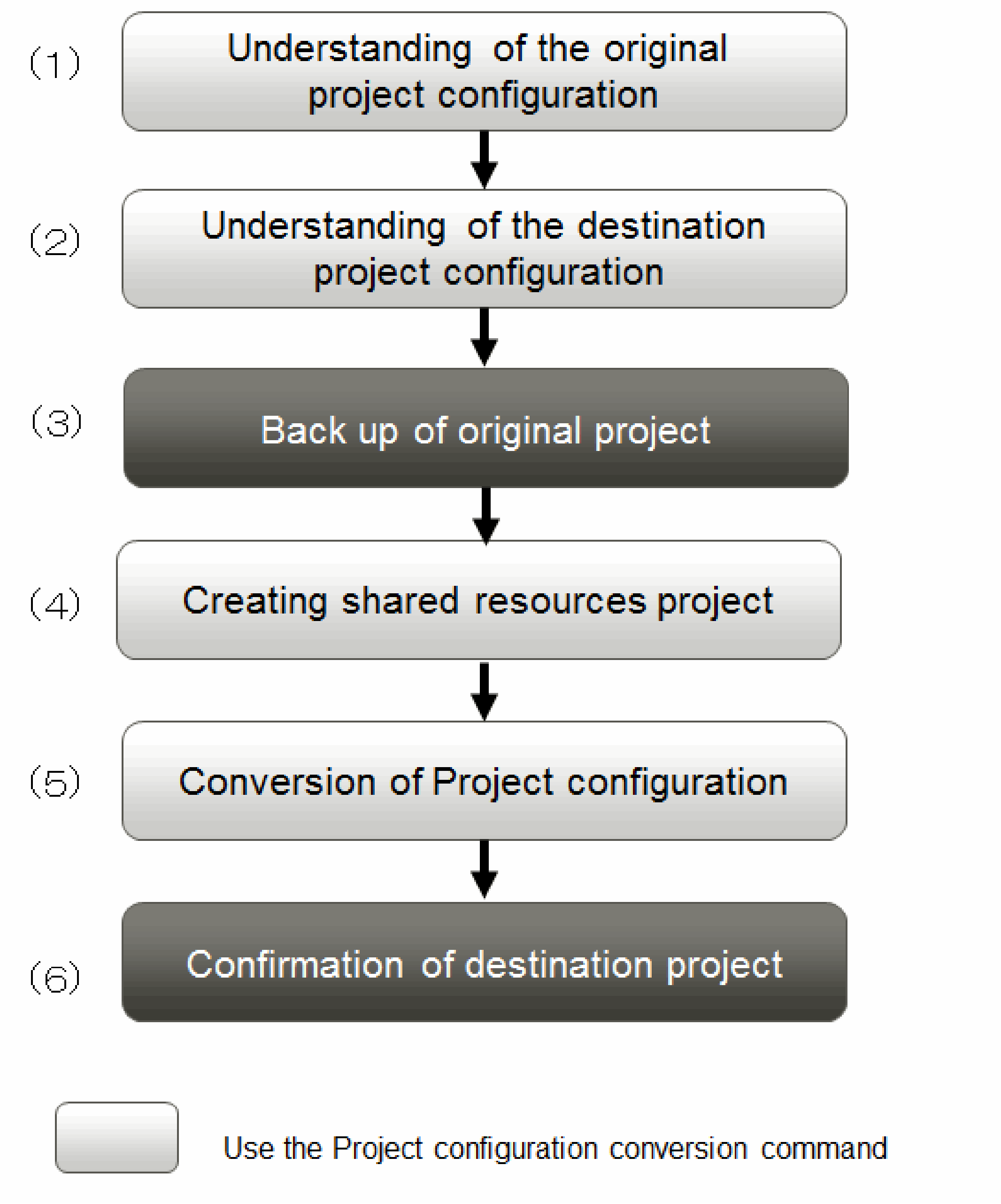
NetCOBOL Studio

Based on Eclipse, an open-source integrated development environment, the COBOL development environment includes assist functions for COBOL program development. In NetCOBOL Studio, folders called Projects are used for creating target files, and managing necessary information such as resources and compile options. It is necessary to configure COBOL source files and target files in the project folder. In addition, it is necessary to create or import a project in the user's workspace in NetCOBOL Studio.



E.2 Project Transition Procedure

The procedure for changing a project in project manager to a project in NetCOBOL Studio is as follows.


(1)
Understanding of the original project configuration
(2)
Understanding of the destination project configuration
(3)
Back up of original project
(4)
Creating shared resources project
(5)
Conversion of Project configuration
(6)
Confirmation of destination project
Use the Project configuration conversion command

(1) Understand the original project configuration

Organize the resources managed by the Project manager project.

(2) Understand the destination project configuration

Become familiar with the resources that are managed in the NetCOBOL Studio project and how they are arranged.

(3) Back up the original project

It is recommended that you back up the conversion source project before using the project configuration conversion command.

(4) Create a shared resources project

Create a COBOL resource project on NetCOBOL Studio, and then copy the shared file that is used by multiple projects, which contains the library and descriptor, into the COBOL resource project.

One method for creating a COBOL resource project is to use the wizard provided by NetCOBOL Studio. Another method is to use the project configuration conversion command.

a. Using NetCOBOL Studio

1. Start NetCOBOL Studio.
2. Select **File** > **New** > **COBOL Resource Project** from menu bar to start the wizard for creating a COBOL resource project.
3. Run the wizard, and follow the instructions to create the OBOL resource project. For details, refer to the "4.2.3 COBOL resource generation wizard".
4. Copy the existing shared COBOL resource into the created COBOL resource project. For details, refer to "4.5.1 Adding the existing COBOL resource.

b. Using the Project configuration conversion command

1. Use the "E.3.2 COBOL Resource Project Writer Module e" of the project configuration conversion command. This function executes the procedure from step 2 to step 4. For details, refer to "E.3.2 COBOL Resource Project Writer Module ".
2. Use the import function of NetCOBOL Studio to import a COBOL resource project that has been created into any workspace.

(5) Convert the Project Organization

Create NetCOBOL Studio COBOL project in each target (exe and dll). Copy the COBOL resources into the COBOL project folder. The compiler option with the path specification changes according to the COBOL project configuration of NetCOBOL Studio.

a. Using NetCOBOL Studio

1. Start NetCOBOL Studio
2. Select **File** > **New** > **COBOL Project** from the menu bar to start the wizard for creating a COBOL project.
3. Follow the wizard instructions to create a COBOL project that corresponds to the target file. For details, refer to "4.2.2 COBOL Project generation Wizard".
4. Add the existing COBOL resource into the COBOL project. For details, refer to "4.4.1 Adding Existing COBOL Resources".
5. Select **File** > **New** > **COBOL Solution Project** from menu bar to start the wizard for creating a COBOL solution project.
6. Follow the wizard instructions to create a COBOL solution project that corresponds to the project manager project file. For details, refer to "4.2.1 COBOL solution generation wizard".
7. Add the COBOL project into the COBOL solution project.
8. Set the build options in the COBOL solution project property dialog box.
9. Deselect the **Enable project specific setting** checkbox in the build page of every COBOL project. When building a COBOL project, build options of the COBOL solution project become effective. For details, refer to "4.3.3 Setting common options for the projects".
10. When using a precompiler, add the precompiler build tool in the property dialog box of the COBOL solution project. For details, refer to "6.2.3 Creating a COBOL program by using the precompiler".

b. The Method of using Project Configuration Conversion Command

a. Use the "Project Organization-converting Function" of the project configuration conversion command. This function does processing from step2 to 10 when above-mentioned Using NetCOBOL Studio. For details, refer to "E.3.3 Usage of Project Configuration Conversion Command (PM2NS Command) ".
Confirm the project configuration conversion command output message. If necessary, change the conversion destination project setting after starting NetCOBOL Studio.

b. Using the Import function of NetCOBOL Studio, import the COBOL solution project and the COBOL project that has been created in any workspace.

(6) Confirm the destination project

Confirm the project organization of NetCOBOL Studio and make sure that it can be built correctly.

When the conversion destination project is converted with the project configuration conversion command, confirm the message that is output at the time of converting and correct the conversion destination project setting as necessary.

E.3 Transition of Project According to Project Configuration Conversion Command

Provide the project configuration conversion command as a transfer support tool in order to convert the project in project manager to the project in NetCOBOL Studio. In the following two functions are parts of the project configuration conversion command.

- Function that converts the project in project manager to the project in NetCOBOL Studio (project organization-converting function).
- Function that summarizes shared files such as library and create COBOL resource project (COBOL resource project created function).

E.3.1 Project Configuration Conversion Function

A NetCOBOL Studio project creates a target. As the project manager, when setting common build options for multiple targets, set the same build options for all COBOL projects, or use a COBOL solution project. For more about COBOL solution projects, refer to "4.1.1 COBOL Solution project".

The project configuration conversion command provides a function that converts the project manager project to a COBOL solution project, and converts every target to a COBOL project.

It is necessary for the following COBOL resource files to exist in the COBOL project folder in the NetCOBOL Studio COBOL project. Therefore, the project configuration conversion command copies those files into the project folder or the subfolder of the project folder.

Table E.1 COBOL resource

File Types	Extent	Conversion Handling
COBOL source file (input of compiler)	.CBL .COB .COBOL	When the COBOL conversion source file exists, create the project folder or the subfolder of the project folder into which to copy it. Whether or not to create a subfolder is decided by the -M option of the project configuration conversion command. For details, refer to "E.3.3 Usage of Project Configuration Conversion Command (PM2NS Command) ".
Library file (input of compiler)	.CBL	When it is not handled as a shared file according to the -M option of the project configuration conversion command, copy it using the following rules. - When it exists in the same folder as the conversion source project file, copy it into the COBOL project folder. - When it exists in the same folder as the conversion source COBOL source file, copy it into the same folder as the COBOL source file. For details of the -M option, refer to "E.3.3 Usage of Project Configuration Conversion Command (PM2NS Command) ".
Descriptor file (input of compiler)	.FFD .PMD .SMD .PXD	Convert it using the same rule as the library file.
Repository file (input of compiler)	.REP	Convert it using the same rule as the library file.
Other files	Any	When it is not handled as a shared file according to the -M option of the project configuration conversion command, copy it into the COBOL project folder if it exists in the same folder as the conversion source project file.
Object file(input of linker)	.OBJ	Copy it using the same rule as the other files.
Library file(input of linker)	.LIB	Copy it using the same rule as the other files.

File Types	Extent	Conversion Handling
Include file (input of precompiler)	Any	Copy it using the same rule as the other files.
Object file(output of compiler)	.OBJ	When the .OBJ file with the same base-name as the COBOL source file exists in the same folder as the COBOL conversion source file, copy it into the same folder as the COBOL source file.
Debugging information file(output of compiler)	.SVD	When the .SVD file with the same base-name as the COBOL source file exists in the same folder as the COBOL conversion source file, copy it into the same folder as the COBOL source file.
SAI file(output of compiler)	.SAI	When the SAI compile options are specified, and the .SAI file with the same base-name as the COBOL source file exists in the same folder as COBOL conversion source file, copy it into the same folder as the COBOL source file.
Target repository file(output of compiler)	.REP	When conversion source target repository file exists, copy it into the same folder as the COBOL source file.
Target file(output of linker)	.DLL .EXE	When the conversion source target file exists, copy it into the project folder.
MAP file(output of linker)	.MAP	When the /MAP options are specified, and /MAP file with the same base-name as the target file exists in the same folder as the conversion source target file, copy it into the COBOL project folder.
Resource script file(output of project manager)	.RC	When About COBOL97 is set, copy the .RC file with the same base-name as the target file into the COBOL project folder.
Resource file(output of project manager)	.RES	When the .RES file with the same base-name as the target file exists, copy it into the same folder as the project file of the conversion source.

In addition, when files with the same name exist in the conversion destination folder and the -F option is not specified, the project configuration conversion command outputs an error and stops conversion.

E.3.2 COBOL Resource Project Writer Module

Use a project relative path specification to refer to shared files such as library in the NetCOBOL Studio COBOL resource project. When using compile options such as LIB and referring to shared files such as library by relative path in project manager, by adding shared files in the NetCOBOL Studio COBOL resource project, compile options can be specified by relative path even in the NetCOBOL Studio project.

The project configuration conversion command summarizes shared files such as library, and provides the function for creating the COBOL resource project. For details, refer to "E.3.3 Usage of Project Configuration Conversion Command (PM2NS Command) ".

E.3.3 Usage of Project Configuration Conversion Command (PM2NS Command)

Use the project configuration conversion command at the NetCOBOL command prompt.

Project Configuration Conversion Function

```
PM2NS [options list] [project manager project file]
```

COBOL Resource Project creation Function

```
PM2NS [options list] -RCOBOL-resource-project-name -Ishared-files-original-folder
```

Table E.2 Description of Options

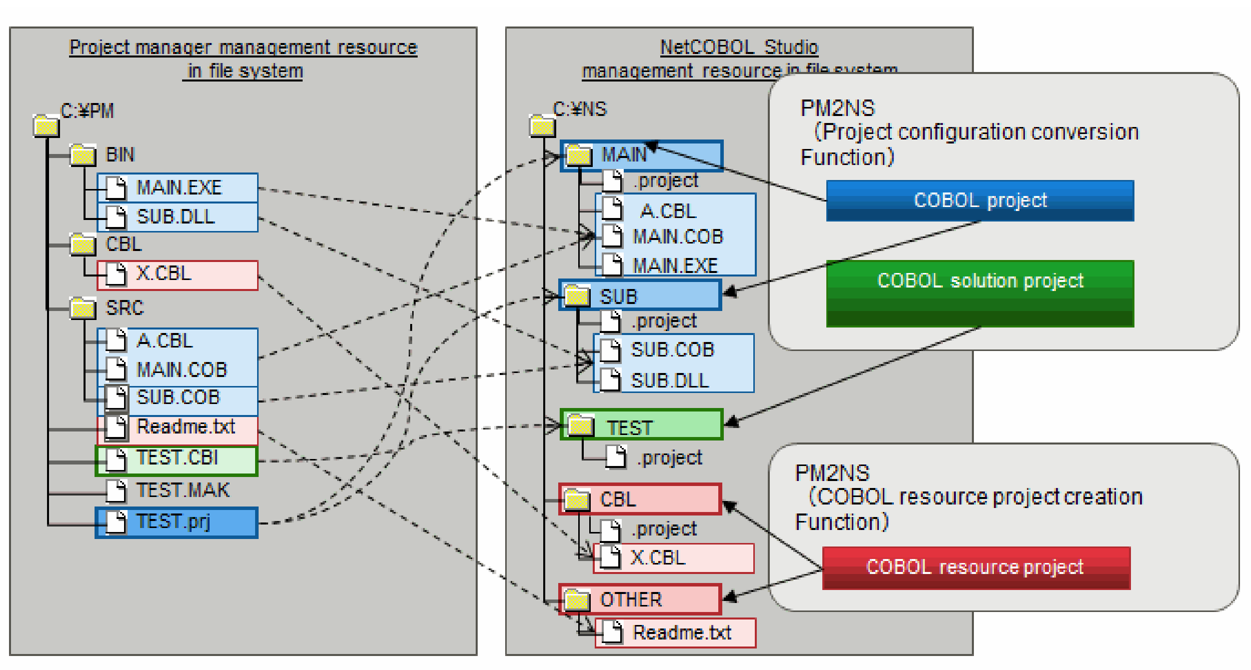
Options	Specify whether or not		Description
	Project configuration conversion Function	COBOL Resource Project creation Function	
-R*COBOL-resource-project-name*		Yes	When creating a COBOL resource project, specify the COBOL resource project name.
-I *shared-files-original-folder*		Yes	-R option must be specified. When creating a COBOL resource project, specify the original shared file path.
-O*output-folder*	Yes	Yes	Specify the project output folder. When options are omitted, the output folder is the current folder. In the output folder, create a project folder used for NetCOBOL Studio, and then create the file or copy the file from the original folder.
-F	Yes	Yes	When an error occurs at the time of converting, output a warning message and continue to convert.
-E*extent*		Yes	When creating a COBOL resource project, specify the extent of shared files that are copied. When options are omitted, copy all files. When multiple extents are specified, separate them with semicolons (;).
-M*INI file*	Yes		Specify INI files that are described as follows. - Path that is specified in compile options and mapping of COBOL resource project - Mapping of subfolder at the time of copying the COBOL source file Formats of INI files are as follows. [COBOLResourceProject] *original-path*=COBOL resource project name ... [COBOLSourceFile] *original-path*=relative path of destination project ... For example, when the conversion source project file is "C:\PM\PRJ", the library file is "C:\PM\A\CBL", the LIB compile option is "..\A\CBL", by specifying the following INI files, convert LIB compile option of COBOL project of conversion destination to"../A_CBL". [COBOLResourceProject] C:\PM\A\CBL=A_CBL
-S{Y|N}	Yes		Specify whether or not COBOL solution project is created at the time of converting. When omit options or specify Y, COBOL solution project is created.

Options	Specify whether or not		Description
	Project configuration conversion Function	COBOL Resource Project creation Function	
			When N is specified, the COBOL solution project is not created. Option information that is set in the conversion source project is set in every option of the conversion destination COBOL project.
-?	Yes	Yes	Output the usage of the PM2NS command.

Yes: it can be specified

E.3.4 Project Conversion Using the Project Configuration Conversion Command

Use the project configuration conversion command to convert the management resource on the file system as shown in the following figure.



In the example above, execute the following command to convert into the project used for NetCOBOL Studio management.

1. Create the COBOL resource project (CBL).

```
 PM2NS -OC:\NS -RCBL -IC:\PM\CBL
```

2. Create the COBOL resource project (OTHER).

```
PM2NS -OC:\NS -Etxt -ROTHER -IC:\PM\SRC -OC:\NS
```

3. Convert the project.

```
PM2NS -OC:\NS C:\PM\TEST.prj
```

- TEST.CBI is compile option file. The content of compile option file is reflected in build option of COBOL solution project.

E.4 Notes

E.4.1 Project that Cannot Be Transferred

In project manager, the project that uses the following functions cannot be transferred to the NetCOBOL Studio project. When an attempt is made to convert using the project configuration conversion command, an error message is output and the conversion stops.

- Multiple pre-compiler (use multiple pre-compiler setting for one target)
- Multistage pre-compiler (use output of pre-compiler as input for another pre-compiler)
- JEF (applications that use JEF options)
- COM server
- CORBA
- OSIV distributed development

Use project manager for NetCOBOL for Windows (x86) as the project for the functions described above, please use project manager of NetCOBOL for Windows (x86).

E.4.2 Project Which Include Functions that are not supported in NetCOBOL Studio

When the conversion source project includes a function that is not supported in NetCOBOL Studio, the project configuration conversion command handles the non-supported function one of two ways.

- Outputs an error message and stops project conversion
  However, when the -F option is specified, after disabling the object function, it outputs a warning message and continues project conversion.
- Outputs a warning message and continues project conversion

Detailed information by function follows.

Function that outputs error message and stops project conversion

When converting a project that includes the following functions, the project configuration conversion command outputs an error message and stops conversion.

However, when specifying the -F option, after disabling the object function, it outputs a warning message and continues conversion.

- An error is generated when the following compile options are specified
  - FLAGSW(GSW), FLAGSW(GSS)
  - LIBEXT
  - FORMEXT
  - SRF(FREE)
- library file creation (create library file as the input of module definition file)

Function that outputs warning message and continues project conversion

When converting a project that includes the following functions, the project configuration conversion command outputs a warning message and continues conversion.

- When %OUTFOLDER% is contained in the pre-compiler parameter
- When the extention of the input file for the pre-compiler is one of the following
  - cobol
  - cob

    - cbl

    - lcai

  When converting a project using the project configuration conversion command, change the extention to "pco" and continue.

- When the extention of the output file for the pre-compiler is none of the following

    - cob

    - cobol

  When converting a project using the project configuration conversion command, change the extention to ".cob" and continue.

- When the base-name of the output file for the pre-compiler is different from the input file.
  When converting a project using the project configuration conversion command, change the base-name of the output file to the base-name of the input file and continue.

- When there are two or more kinds of extentions of the input file for the pre-compiler
  When converting a project using the project configuration conversion command, change the extention to "pco" and continue.

- When there are two or more kinds of extentions of the output file for the pre-compiler
  When converting a project using the project configuration conversion command, change the extention to ".cob" and continue.

- When the following OS-dependent compile options are specified in the build management file on the Unix distributed development application

    - LALIGN

    - FILELIB

    - CODECHECK

    - KANA

  When converting a project using the project configuration conversion command, the command doesn't distinguish server OS. Convert and continue using them in the raw format.

- When server link information can be obtained by a Unix distributed development application

  Server link information cannot be set as the project information. When converting a project using the project configuration conversion command, output server link information in ini file with the following format and continue.

```
[ServerName]
ServerAddress=***
UserName=***
ServerOS=***
HostCode=***
HostClientCodeChange=***
PASVSendMessage=***
```

- When server link information cannot be obtained by a Unix distributed development application
  When converting a project using the project configuration conversion command, output only a warning message and continue.

- When multiple build management files exist by a Unix distributed development application
  When converting a project using the project configuration conversion command, output a warning message about the build management file that has not converted and continue.

- When the following link options used for Solaris are specified in the build management file of a Unix distributed development application

    - pm

    - pc

    - pi

    - Ns

When converting a project using the project configuration conversion command, the command does not distinguish server OS. Convert and continue using them in the raw format.

- When send function information is set by a Unix distributed development application
  When converting a project using the project configuration conversion command, output only a warning message and continue.
- When a reference library name is set by a Unix distributed development application
  When converting a project using the project configuration conversion command, output only a warning message and continue.

Index