

Fujitsu Software NetCOBOL V13.0

Getting Started

Windows(64)

B1WD-3689-01ENZ0(00)
December 2025

Preface

This manual provides an introduction to NetCOBOL. NetCOBOL provides a full-featured development environment for COBOL programs. It allows you to develop COBOL programs that also easily integrate with other languages.

The sample programs shipped with NetCOBOL are intended to give an overview of the capabilities of NetCOBOL. Refer to "NetCOBOL User's Guide" for further details on using NetCOBOL.

Audience

Prior to using NetCOBOL, it is assumed that you have the following knowledge:

- You have some basic understanding as to how to navigate through and use the Microsoft Windows product on your machine.
- You understand the COBOL language from a development perspective.
- If you plan on using Microsoft's Visual Basic development environment, you have spent some time using Visual Basic to get a feel for its interface and capabilities.

Abbreviations

The following abbreviations are used in this manual:

Product Name	Abbreviation
Microsoft(R) Windows Server(R) 2025 Datacenter Microsoft(R) Windows Server(R) 2025 Standard	Windows Server 2025
Microsoft(R) Windows Server(R) 2022 Datacenter Microsoft(R) Windows Server(R) 2022 Standard	Windows Server 2022
Microsoft(R) Windows Server(R) 2019 Datacenter Microsoft(R) Windows Server(R) 2019 Standard	Windows Server 2019
Windows(R) 11 Home Windows(R) 11 Pro Windows(R) 11 Enterprise Windows(R) 11 Education	Windows 11
Microsoft(R) Visual Basic(R) programming system	Visual Basic

In this manual, when all the following products are indicated, it is written as "Windows(x64)" or "Windows."

- Windows Server 2025
- Windows Server 2022
- Windows Server 2019
- Windows 11

Trademarks

- Microsoft, Windows, Windows Server, Visual Basic, and Visual Studio are trademarks of the Microsoft group of companies.
- All other trademarks are the property of their respective owners.

Notation

This manual uses the following conventions:

- Operating procedures are described using Windows 11.

- NetCOBOL Studio menu names and operating procedures described in this manual are based on Eclipse 2024-12 (version 4.34) with the NetCOBOL Studio plugin. However, some images may be from earlier versions if there are no differences in menu names and operating procedures.

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.

The contents of this manual may be revised without prior notice. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Fujitsu Limited.

Publication Date and Edition

Publication Date and Edition	Manual Code
December 2025, Version 1.0	B1WD-3689-01ENZ0(00)

Copyright Notice

Copyright Fujitsu Limited 2012-2025

Contents

Chapter 1 Sample Programs.....	1
1.1 Advance preparation for using NetCOBOL Studio to execute a sample program.....	2
1.1.1 The basic concept of NetCOBOL Studio.....	2
1.1.2 Advance preparation for using sample.....	2
1.1.3 Notes on using the sample programs.....	3
1.2 Sample 1: Data Processing Using Standard Input-Output.....	3
1.2.1 Using NetCOBOL Studio.....	4
1.2.2 Using the COBOL command and the LINK command.....	4
1.2.3 Using the MAKE command.....	5
1.3 Sample 2: Using Line Sequential and Indexed Files.....	5
1.3.1 Using NetCOBOL Studio.....	6
1.3.2 Using MAKE file.....	8
1.4 Sample 4: Screen Input-Output Using the Screen Section.....	8
1.4.1 Using NetCOBOL Studio.....	8
1.4.2 Using MAKE file.....	10
1.5 Sample 5: Calling COBOL Subprograms.....	11
1.5.1 Using NetCOBOL Studio.....	13
1.5.2 Using MAKE file.....	20
1.6 Sample 6: Receiving a Command Line Argument.....	20
1.6.1 Using NetCOBOL Studio.....	21
1.6.2 Using MAKE file.....	23
1.7 Sample 7: Environment Variable Handling.....	23
1.7.1 Using NetCOBOL Studio.....	24
1.7.2 Using MAKE file.....	26
1.8 Sample 8: Using a Print File.....	26
1.8.1 Using NetCOBOL Studio.....	26
1.8.2 Using MAKE file.....	28
1.9 Sample 9: Using a Print File (Advanced usage)	28
1.9.1 Using NetCOBOL Studio.....	34
1.9.2 Using MAKE file.....	36
1.10 Sample 11: Remote Database Access.....	36
1.10.1 Using NetCOBOL Studio.....	37
1.10.2 Using MAKE file.....	39
1.11 Sample 12: Remote Database Access (Multiple row processing).....	40
1.11.1 Using NetCOBOL Studio.....	41
1.11.2 Using MAKE file.....	43
1.12 Sample 13: Calling COBOL from Visual Basic	44
1.12.1 Using NetCOBOL Studio.....	45
1.12.2 Using MAKE file.....	46
1.13 Sample 14: Visual Basic calling COBOL -Simple ATM Example.....	47
1.13.1 Using NetCOBOL Studio.....	49
1.13.2 Using MAKE file.....	52
1.14 Sample 15: Basic Object-Oriented Programming	53
1.14.1 Using NetCOBOL Studio.....	54
1.14.2 Using MAKE file.....	54
1.15 Sample 16: Collection Class (Class Library).....	55
1.15.1 Using NetCOBOL Studio.....	62
1.15.2 Using MAKE file.....	63
1.16 Sample 31: Windows System Function Call.....	64
1.16.1 Using NetCOBOL Studio.....	65
1.16.2 Using MAKE file.....	67
1.17 Sample 32: Starting Another Program.....	67
1.17.1 Using NetCOBOL Studio.....	68
1.17.2 Using MAKE file.....	69

[Index](#).....70

Chapter 1 Sample Programs

The sample programs shipped with NetCOBOL are intended to give an overview of the capabilities of NetCOBOL. For further details on using NetCOBOL, refer to the "NetCOBOL User's Guide." The following table details the sample programs available with NetCOBOL.



The NetCOBOL sample programs are saved in "<NetCOBOL installation folder>\samples". Please copy the sample programs to any folder and then use the sample program.

The following explanation uses "C:\COBOL" as the NetCOBOL installation folder. Change "C:\COBOL" to the NetCOBOL installation folder.

NetCOBOL Sample Programs

- Sample 1
Data Processing Using Standard Input-Output
- Sample 2
Using Line Sequential and Indexed Files
- Sample 4
Screen Input-Output Using the Screen Section
- Sample 5
Calling COBOL Subprograms
- Sample 6
Receiving a Command Line Argument
- Sample 7
Environment Variable Handling
- Sample 8
Using a Print File
- Sample 9
Using a Print File (Advanced usage)
- Sample 11
Remote database access
- Sample 12
Remote database access (multiple row processing)
- Sample 13
Calling COBOL from Visual Basic
- Sample 14
Visual Basic calling COBOL - Simple ATM Example
- Sample 15
Basic Object-Oriented Programming
- Sample 16
Collection Class (Class Library)
- Sample 31
Windows System Function Call
- Sample 32
Starting other programs

Each sample explains the following two methods as a way of operating the exercise program.

- Using NetCOBOL Studio
- Using MAKE file



- When you execute the Sample program by using NetCOBOL Studio, see "[1.1 Advance preparation for using NetCOBOL Studio to execute a sample program.](#)"
- When you use MAKE file, use the NetCOBOL command prompt. On **Start**, select **All > Fujitsu NetCOBOL V13(x64) > NetCOBOL Command Prompt.**

1.1 Advance preparation for using NetCOBOL Studio to execute a sample program

1.1.1 The basic concept of NetCOBOL Studio

Read through "NetCOBOL Studio" in the "NetCOBOL Studio User's Guide" to understand the necessary basic concepts (workspace, perspective, etc.) for using NetCOBOL Studio.

Moreover, NetCOBOL Studio screen is composed of a window for the editor area and two or more information displays. For an explanation of each window, refer to "COBOL perspective", "Editor" or "Debug Perspective" in "NetCOBOL Studio User's Guide."

Automatic build

Automatic build is turned on by default. It can be toggled off or on by selecting "Project" > "Build Automatically" from the NetCOBOL Studio menu bar. When it is checked, it is turned on. For details on automatic build, refer to "Automatic build" in "NetCOBOL Studio User's Guide."

Project folder

The project property is stored in a folder. It is saved in the workspace and when the project is imported, the project folder is created under the workspace folder.

Example: The project folder for SAMPLE01 with a workspace folder of C:\NetCOBOL Studio\workspace:

```
C:\NetCOBOL Studio\workspace\SAMPLE01
```

1.1.2 Advance preparation for using sample

To build, execute, and debug the sample program using NetCOBOL Studio, prepare NetCOBOL Studio and create a project for the sample program in a folder called a workspace.

This document describes the following procedures:

1. [Preparing NetCOBOL Studio](#)
2. [Preparing the workspace](#)
3. [Importing the sample program project into the Eclipse IDE workspace](#)

Preparing NetCOBOL Studio

NetCOBOL Studio becomes available by installing the NetCOBOL Studio plugin into the open source software Eclipse IDE. For details, refer to "Installing Eclipse IDE and NetCOBOL Studio Plugin" in "NetCOBOL Installation Guide."

If the Welcome screen appears when you start the Eclipse IDE, click the Hide icon to close the screen. If the COBOL perspective is not displayed, open it using the following procedure:

1. Click the Open Perspective icon on the toolbar, or select **Window > Perspective > Open Perspective > Other** from the menu bar.
The **Open Perspective** dialog box appears.

2. Select **COBOL** and click **Open**. The COBOL perspective opens.

Preparing the workspace

A "workspace" is a folder that stores various resources, such as projects created in NetCOBOL Studio.

Create a new workspace for the sample programs. For how to create a workspace, refer to "Setting and switch method of workspace" in "NetCOBOL Studio User's Guide."

This manual assumes that the created workspace is "C:\NetCOBOL Studio\workspace".

Because the samples are created in windows-1252, set the default text file encoding for the workspace to "windows-1252" by following these steps:

1. From the menu bar, select **Window > Preferences**. The **Preferences** dialog box appears.
2. Select **General > Workspace**, and then set **Text file encoding** to "windows-1252".
If the Default text file encoding is "windows-1252", select the **Default**.
Otherwise, select **Other** and enter "windows-1252" in the text field.
3. Click **Apply and Close**.

Importing the sample program project into the Eclipse IDE workspace

Import the provided sample program project into the Eclipse IDE workspace using the following procedure:

1. Start the Eclipse IDE in which the NetCOBOL Studio plugin is installed.

Start the Eclipse IDE with the following command. Specify "-cobolarch x64" as a startup option:

```
<Eclipse IDE installation folder>\eclipse.exe -cobolarch x64
```

2. From the menu bar, select **File > Import**. The **Import** wizard starts.
3. Select **General > Existing Projects into Workspace** and click **Next**.
4. Select **Select root directory** and click **Browse**. The **Select Folder** dialog box appears.
5. Select the folder containing the project (the COBOL sample program storage location, here C:\COBOL\Samples\COBOL) and click **Select Folder**.
6. Verify that the COBOL sample program project is displayed in the **Projects** pane, and then click **Select All**. Select **Copy projects into workspace** and click **Finish**.

The sample program project is imported into the NetCOBOL Studio workspace.

1.1.3 Notes on using the sample programs

For each sample program, the NetCOBOL Studio project associated files below are provided.

Please do not edit these project associated files. The application does not run correctly when these files are changed.

- .Settings\org.eclipse.core.resources.prefs
- .CobolOptions
- .project

1.2 Sample 1: Data Processing Using Standard Input-Output

Sample 1 demonstrates using the ACCEPT/DISPLAY function to input and output data. Refer to the "NetCOBOL User's Guide" for details on how to use the ACCEPT/DISPLAY statements.

Function

Inputs an alphabetic character (lowercase character) from the console window, and outputs a word to the console window beginning with the input alphabetic character.

Files Included in Sample 1

- SAMPLE1.COB (COBOL source program)
- MAKEFILE
- COBOL85.CBR

COBOL Statements Used

ACCEPT, DISPLAY, EXIT, IF, and PERFORM statements are used.

1.2.1 Using NetCOBOL Studio

Compiling and Linking the Program

1. NetCOBOL Studio is started by specifying the workspace created for the sample program. See "[Preparing the workspace.](#)"
2. In the Dependency view, select the SAMPLE01 project.

If there is no SAMPLE01 project, import the SAMPLE01 project into the NetCOBOL Studio workspace. See "[Importing the sample program project into the Eclipse IDE workspace.](#)"
3. The structure of the project is as follows.

```
SAMPLE01
├ Source Files
│   └ Sample1.cob
├ Linking Files
└ Other Files
    ├── .settings folder
    ├── COBOL85.CBR
    └ Makefile
```

The build is executed immediately after importing the project when an automatic build is set. In this case, the file (. EXE . OBJ etc.) is generated after the build is displayed in "Other Files". It is set to automatic build by default.

4. On the **Project** menu, click **Build Project** when SAMPLE1.EXE is not created in "Other Files" (When an automatic build is not executed), click **Build Project** On the **Project** menu.

The project is built, and then SAMPLE1.EXE is created.

Debugging

Refer to "Creating a COBOL Program" in "NetCOBOL Studio User's Guide" for the SAMPLE1 debugging procedure using the debugging facility of NetCOBOL Studio.

Program execution

In the Dependency view, select the SAMPLE01 project. On the **Run** menu, click **Run As > COBOL Application**.

When one alphanumerical character is entered, the first word with that character is displayed.

1.2.2 Using the COBOL command and the LINK command

Compiling and Linking the Program

Open a command prompt. Compile and link the sample program using the following command.

```
C:\COBOL\Samples\COBOL\Sample01>COBOL.EXE -M SAMPLE1.COB
C:\COBOL\Samples\COBOL\Sample01>LINK /OUT:SAMPLE1.EXE SAMPLE1.OBJ F4AGCIMP.LIB MSVCRT.LIB
```

1.2.3 Using the MAKE command

The sample program can also be compiled and linked using the nmake command.

```
C:\COBOL\Samples\COBOL\Sample01>nmake
```

Compilation of the sample program is now complete. Verify that SAMPLE1.EXE was created in the same folder in which the sample program is stored.

Executing the Program

In the command prompt or from Windows Explorer, execute SAMPLE1.EXE

Enter a lowercase letter and then press the ENTER key. A word that begins with the input lowercase letter is displayed.

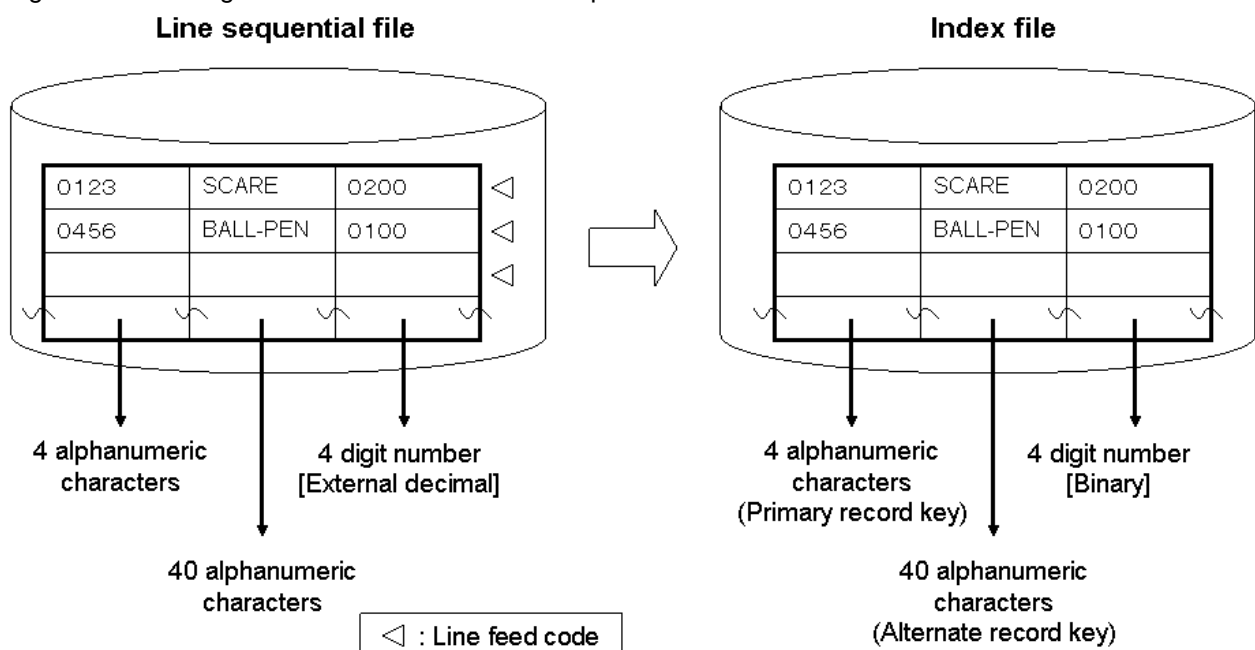
1.3 Sample 2: Using Line Sequential and Indexed Files

Sample 2 demonstrates a program that reads a data file (line sequential file) created with the Editor, and then creates a master file (indexed file). For details on how to use line sequential files and indexed files, refer to the "NetCOBOL User's Guide."

Overview

Reads a data file (line sequential file) that contains product codes, product names, and unit prices, and creates an indexed file with the product code as a primary record key and the product name as an alternate record key.

Figure 1.1 Creating an indexed file from a line sequential file



Files Included in Sample 2

- SAMPLE2.COB (COBOL source program)
- DATAFILE (Data file)
- MAKEFILE
- COBOL85.CBR

COBOL Statements Used

The CLOSE, EXIT, GO TO, MOVE, OPEN, READ, and WRITE statements are used.

1.3.1 Using NetCOBOL Studio

Compiling and Linking the Program

1. NetCOBOL Studio is started by specifying the workspace created for the sample program. See "[Preparing the workspace](#)."
2. In the Dependency view, select the SAMPLE02 project.

If there is no SAMPLE02 project, import the SAMPLE02 project into the NetCOBOL Studio workspace. See "[Importing the sample program project into the Eclipse IDE workspace](#)."

3. The structure of the project is as follows.

```
SAMPLE02
├ Source Files
│   └ Sample2.cob
├ Linking Files
└ Other Files
    ├── .settings folder
    ├── COBOL85.CBR
    ├── Datafile
    └ Makefile
```

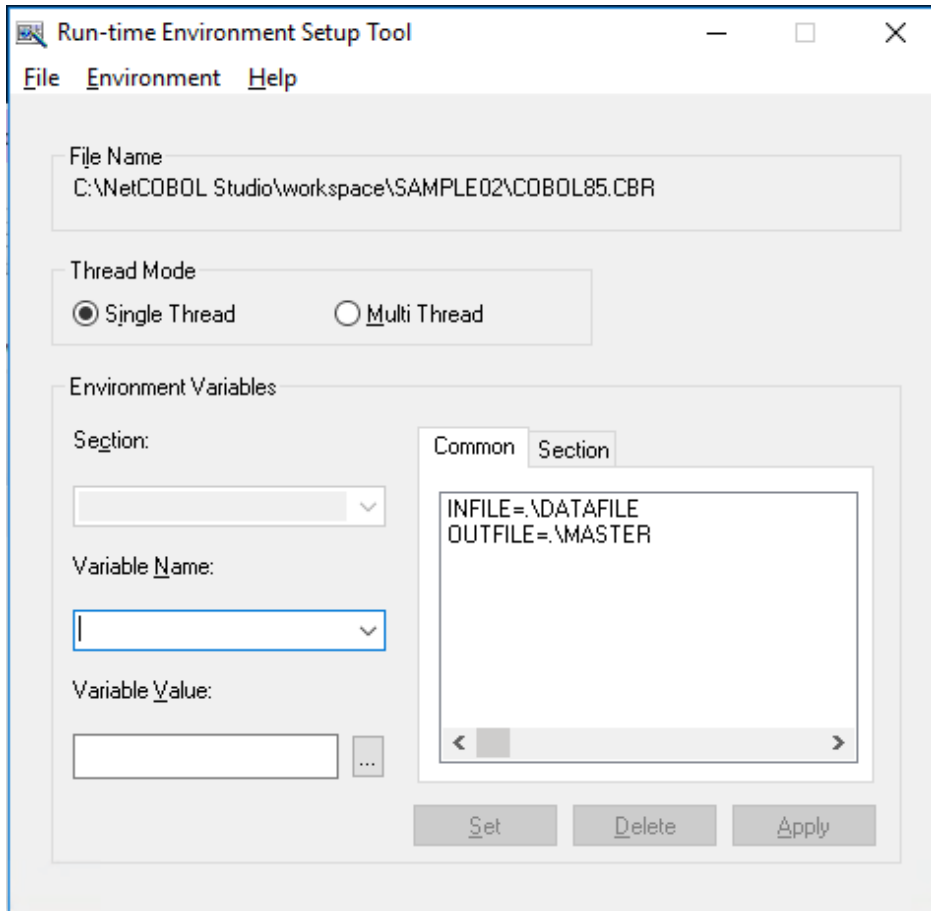
The build is executed immediately after importing the project when an automatic build is set. In this case, the file (. EXE . OBJ etc.) is generated after the build is displayed in "Other Files". It is set to automatic build by default.

4. On the **Project** menu, click **Build Project** when SAMPLE2.EXE is not created in "Other Files" (When an automatic build is not executed).

The project is built, and then SAMPLE2.EXE is created.

Setting Runtime Environment Information

1. Start **Run-time Environment Setup Tool**. The **Run-time Environment Setup Tool** dialog appears.



2. On the **File** menu, click **Open**. The **Select Runtime Initialization file** dialog box appears.
3. Select COBOL85.CBR in the folder that contains the executable program (SAMPLE2.EXE).
4. On the **Common** tab, enter data as shown below:
 - For the file-identifier INFILE, specify the name of the data file (line sequential file) in DATAFILE.
 - For the file-identifier OUTFILE, specify the name of master file (indexed file) in MASTER.

```
INFILE= . \DATAFILE
OUTFILE= . \MASTER
```

If MASTER is specified for OUTFILE, input the data and click **Set**.

5. Click **Apply**. The data is saved in the object initialization file.
6. On the **File** menu, click **Exit** to terminate the run-time environment setup tool.

Information

INFILE and OUTFILE are the file reference identifiers specified for ASSIGN clauses in COBOL programs. The file reference identifier is used to associate a COBOL program and an actual file.

Program execution

1. In the Dependency view, select SAMPLE02 project.

2. On the **Run** menu, click **Run As > COBOL Application**.

Execution result

No termination message is displayed.

After execution is complete, an indexed file (MASTER) with a product code as a key is created in the SAMPLE02 directory. Use Windows Explorer or File Manager to verify that the indexed file was created.

Use the COBOL File Utility to confirm that the indexed file (MASTER) was created correctly. The indexed file record can be browsed using the COBOL File Utility Browsing Records function. Refer to "COBOL File Utility" in "NetCOBOL User's Guide" for details.

1.3.2 Using MAKE file

Compiling and Linking the Program

Open a command prompt. Compile and link the sample program using the following command.

```
C:\COBOL\Samples\COBOL\Sample02>nmake
```

Compilation of the sample program is now complete. Verify that SAMPLE2.EXE was created in the same folder in which the sample program is stored.

Setting Runtime Environment Information

Same as "[1.3.1 Using NetCOBOL Studio](#)".

Executing the Program

In the command prompt or from Windows Explorer, execute SAMPLE2.EXE

Execution result

Same as "[1.3.1 Using NetCOBOL Studio](#)".

1.4 Sample 4: Screen Input-Output Using the Screen Section

Sample 4 demonstrates using the Screen Section (the "screen handling function") to display and accept data. Refer to the "NetCOBOL User's Guide" for details on how to use the screen handling function.

Overview

When an employee's number and name are written to the screen, the program creates an indexed file with the employee's number as a primary record key, and the name as an alternate record key.

Files Included in Sample 4

- SAMPLE4.COB (COBOL source program)
- MAKEFILE
- SAMPLE4.KBD (Key definition file)
- COBOL85.CBR

COBOL Statements Used

The ACCEPT, CLOSE, DISPLAY, EXIT, GO TO, IF, MOVE, OPEN, and WRITE statements are used.

1.4.1 Using NetCOBOL Studio

Compiling and Linking the Program

1. NetCOBOL Studio is started by specifying the workspace created for the sample program. See "[Preparing the workspace](#)."
2. In the Dependency view, select the SAMPLE04 project.

If there is no SAMPLE04 project, import the SAMPLE04 project into the NetCOBOL Studio workspace. See "[Importing the sample program project into the Eclipse IDE workspace](#)."

3. The structure of the project is as follows.

```
SAMPLE04
├ Source Files
│   └ Sample4.cob
├ Linking Files
└ Other Files
    ├── .settings folder
    ├── COBOL85.CBR
    ├── Makefile
    └ Sample4.kbd
```

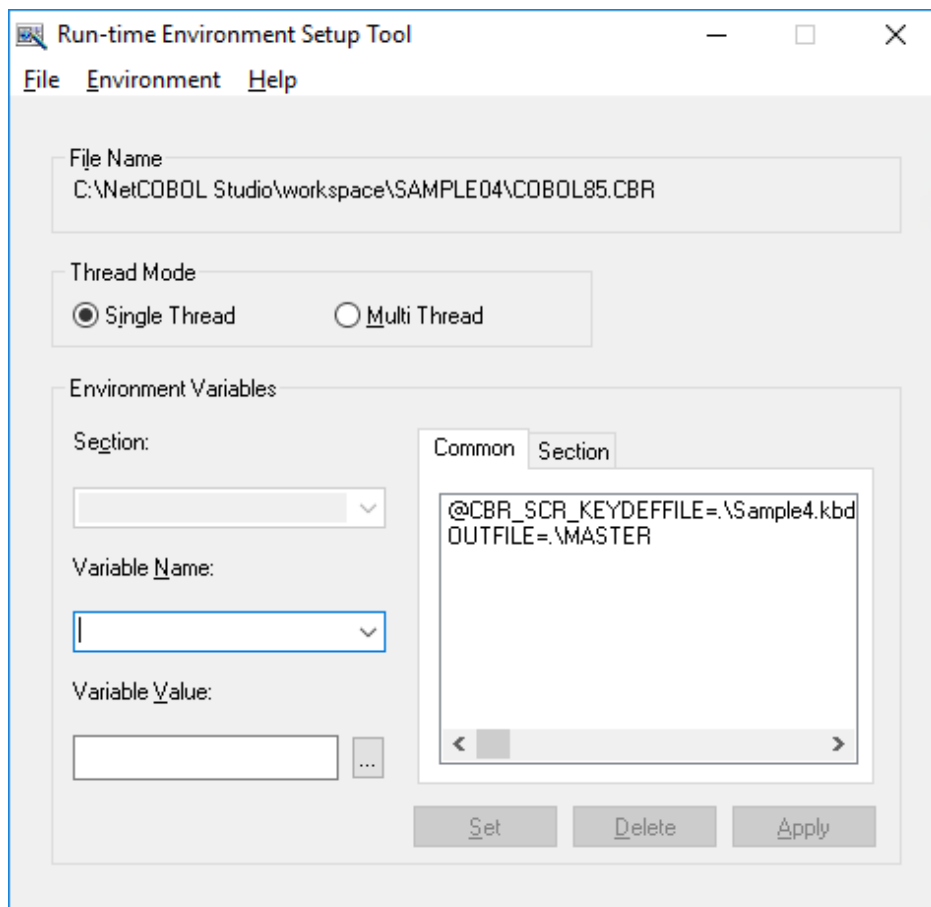
The build is executed immediately after importing the project when an automatic build is set. In this case, the file (. EXE . OBJ etc.) is generated after the build is displayed in "Other Files". It is set to automatic build by default.

4. On the **Project** menu, click **Build Project** when SAMPLE4.EXE is not created in "Other Files" (When an automatic build is not executed).

The project is built, and then SAMPLE4.EXE is created.

Setting Runtime Environment Information

1. Start **Run-time Environment Setup Tool**. The **Run-time Environment Setup Tool** dialog appears.



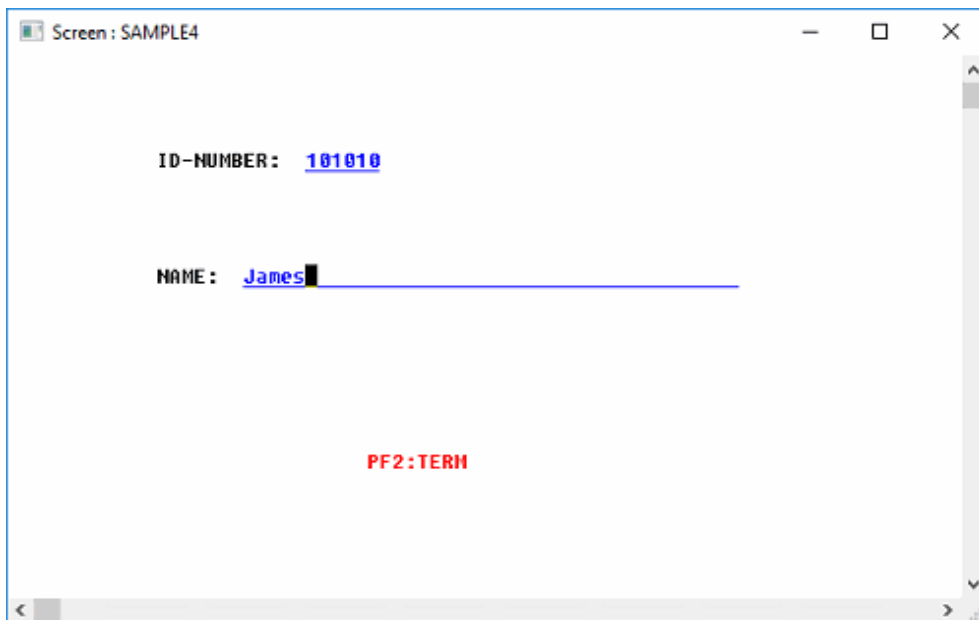
2. On the **File** menu, click **Open**. The **Select Runtime Initialization file** dialog box appears
3. Select COBOL85.CBR in the folder that contains the executable program (SAMPLE4.EXE).
4. Select the Common tab and enter data as shown below:
 - For the file-identifier OUTFILE, specify the master file name (indexed file) in MASTER.
 - For the environment variable @CBR_SCR_KEYDEFFILE, specify SAMPLE4.KBD, which enables the F2 key for use.
5. Click **Apply**. The data is saved in the object initialization file.
6. On the **File** menu, click **Exit** to terminate the run-time environment setup tool.

Program execution

1. In the Dependency view, select SAMPLE04 project.
2. On the **Run** menu, click **Run As > COBOL Application**.

Execution result

The screen for entering an employee's number and name is displayed.



Enter an employee's 6 digit number and name (up to 40 alphanumeric characters), and press the ENTER key. The input data is written to the master file, and the screen is cleared for the input of subsequent data.

To terminate processing, press the F2 key. Use Windows Explorer to confirm that the index file (MASTER) was created in the SAMPLE04 directory with the employee number as the main record key and the employee name as the record sub-key.

1.4.2 Using MAKE file

Compiling and Linking the Program

Open a command prompt. Compile and link the sample program using the following command.

```
C:\COBOL\Samples\COBOL\Sample04>nmake
```

Compilation of the sample program is now complete. Verify that SAMPLE4.EXE was created in the same folder in which the sample program is stored.

Setting Runtime Environment Information

Same as "[1.4.1 Using NetCOBOL Studio](#)".

Executing the Program

Execute SAMPLE4.EXE from a command prompt or from Windows Explorer.

Execution result

Same as "[1.4.1 Using NetCOBOL Studio](#)".

1.5 Sample 5: Calling COBOL Subprograms

Sample 5 demonstrates an application that calls a subprogram from the main program. Sample 5 was created using free format source.

Sample 5 also demonstrates how to pass an argument string to a program and how to display a message box.

Overview

Reads the contents of the master file (indexed file created in Sample 2), stores the records in a work file whose name is provided in the @MGPRM environment variable (a way of passing information to a main program's linkage section), and then passes the work file to a subprogram that prints the records.

The master file stores product codes, product names, and unit prices. The work file name must be specified in the @MGPRM parameter at program execution.

Files Included in Sample 5

- SAMPLE5.COB (COBOL source program)
- S_REC.CBL (COBOL library file)
- MAKEFILE
- COBOL85.CBR
- INSATSU.COB

When NetCOBOL Studio is used, the following are used. The COBOL source program is a variable format.

- SAMPLE05_EXE\SAMPLE5.COB (COBOL source program)
- SAMPLE05_EXE\COBOL85.CBR
- SAMPLE05_DLL\INSATSU.COB
- SAMPLE05_DLL\COBOL85.CBR
- SAMPLE05_LIB\S_REC.CBL (COBOL library file)

COBOL Statements Used

The CALL, DISPLAY, EXIT, GO TO, MOVE, OPEN, READ, and WRITE statements are used.

Using Free Format in a COBOL Source Program

The following is an example of using free format in a COBOL source program.

Column position	1	10	20	30	40	50	60	70
<pre> IDENTIFICATION DIVISION. PROGRAM-ID. SAMPLE5. *> THIS SAMPLE PROGRAM IS IN FREE FORMAT. THE PROGRAM MUST BE *> COMPILED WITH THE SRF COMPILER OPTION. THE SRF COMPILER OPTION *> SPECIFIES THE SOURCE FORMAT TYPE. SRF(FREE,FREE) TELLS THE *> COMPILER THAT THE SOURCE PROGRAM AND COPYBOOKS ARE IN FREE FORMAT ENVIRONMENT DIVISION. CONFIGURATION SECTION. SPECIAL-NAMES. SYSERR IS MESSAGE-DEVICE. : DATA DIVISION. FILE SECTION. FD MASTER-FILE. 01 MASTER-RECORD. 02 GOODS-RECORD. 03 GOODS-CODE PIC X(4). 03 GOODS-NAME PIC X(38). 03 GOODS-PRICE PIC 9(4) BINARY. : PROCEDURE DIVISION USING PARAMETER. *> (1) DETERMINE WORK-FILE NAME IF PARAMETER-LEN = 0 DISPLAY "NOT SPECIFIED PARAMETER."- "PLEASE SPECIFY PARAMETER." UPON MESSAGE-DEVICE GO TO TERM-PROC. : TERM-PROC. EXIT PROGRAM. END PROGRAM SAMPLE5. </pre>								

Note

In the above figure, colons are used to denote sections of source code that have been omitted.

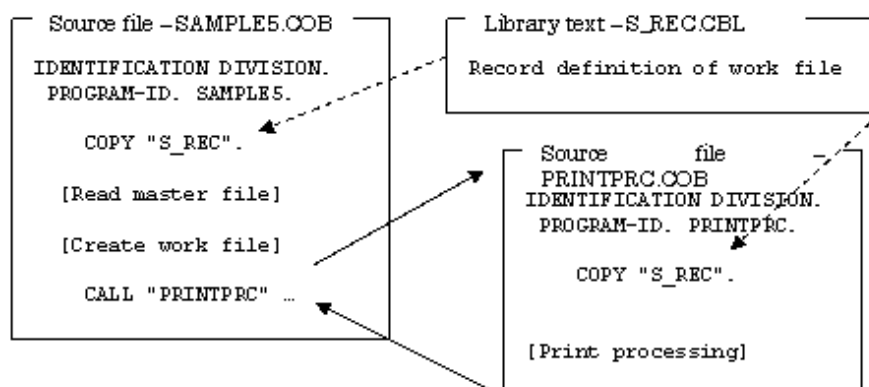
In free format, COBOL statements can be written in any character position on the line. Lines beginning with ">" are treated as comments.

Note

You must specify the SRF compiler option in order to use free format. The SRF compiler option has two parameters; the first specifies the format for the source program and the second specifies the format of copybooks. All copybooks must have the same format type. The available types are FIX, for fixed format source, VAR, for variable format source, and FREE, for free format source.

File Interdependence

The following figure shows the relationship between the source files used in Sample 5.



Prerequisite to Executing the Program

The master file created in Sample 2 is used; therefore, execute the program in Sample 2 before executing Sample 5.

1.5.1 Using NetCOBOL Studio

Compiling and Linking the Program

1. NetCOBOL Studio is started by specifying the workspace created for the sample program. See "[Preparing the workspace](#)."
2. In the Dependency view, the presence of the SAMPLE05, SAMPLE05_EXE, SAMPLE05_DLL and SAMPLE05_LIB projects are confirmed.

If there are no SAMPLE05 projects, import projects of sample program to NetCOBOL Studio workspace. See "[Importing the sample program project into the Eclipse IDE workspace](#)."

3. The structure of the project is as follows.

```
SAMPLE05
├─ SAMPLE05_DLL
│  ├─ Source Files
│  │   └─ Insatsu.cob
│  ├─ Linking Files
│  └─ Other Files
│     ├─ .settings folder
│     └─ COBOL85.CBR
├─ SAMPLE05_EXE
│  ├─ Source Files
│  │   └─ Sample5.cob
│  ├─ Linking Files
│  └─ Other Files
│     ├─ .settings folder
│     ├─ COBOL85.CBR
│     └─ S_rec.cbl
└─ SAMPLE05_LIB
   ├─ .settings folder
   └─ S_rec.cbl
```

The build is executed immediately after importing the project when an automatic build is set. In this case, the file (. EXE . OBJ etc.) is generated after the build is displayed in "Other Files". It is set to automatic build by default.

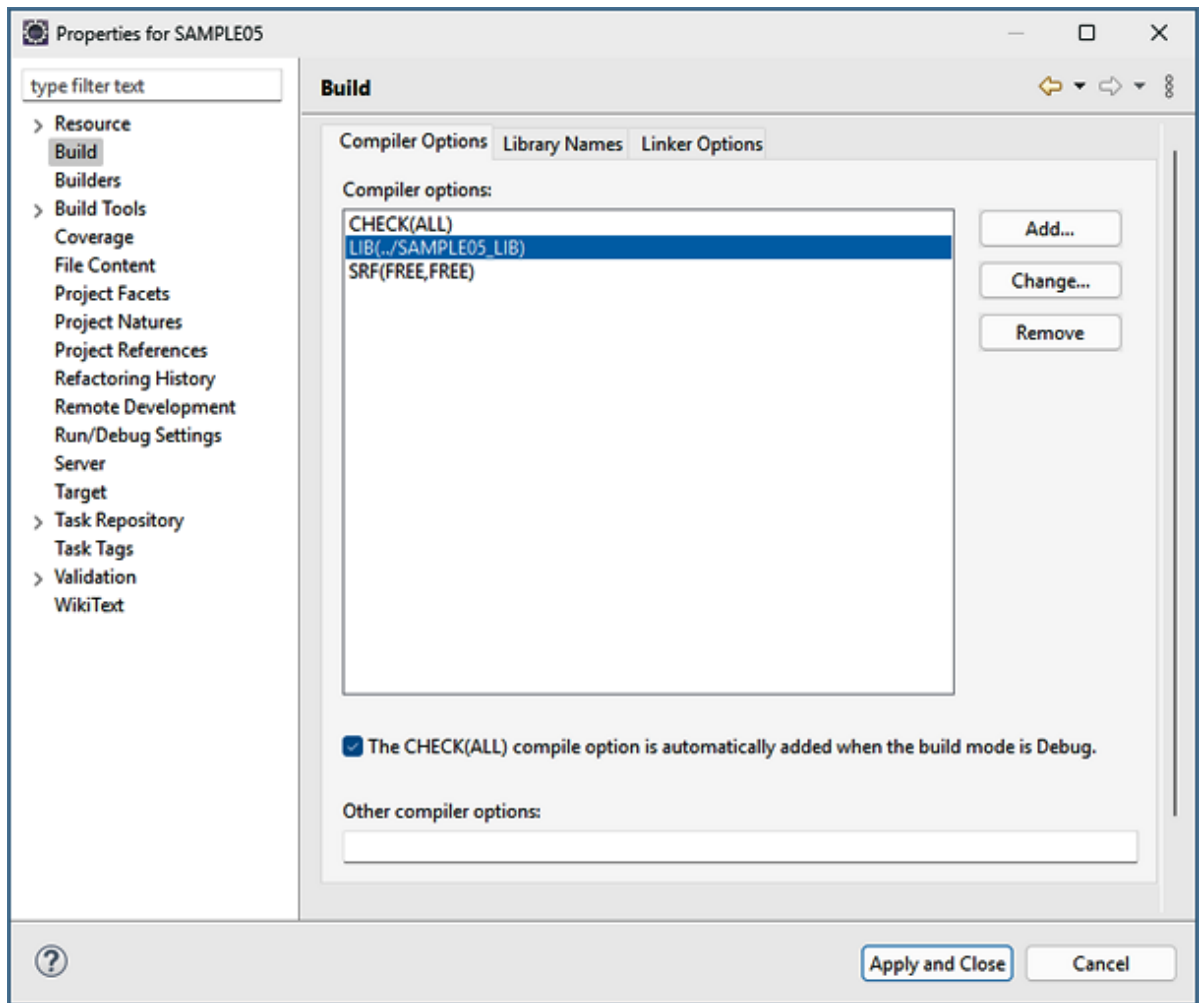
4. Build setting of solution project

Set a project common option to the build page of the SAMPLE05 solution project.

Here, set compile option LIB as a common option.

- a. In the Dependency view, select the SAMPLE05 projects, and chosen the property from the context menu. The Properties dialog box is displayed.

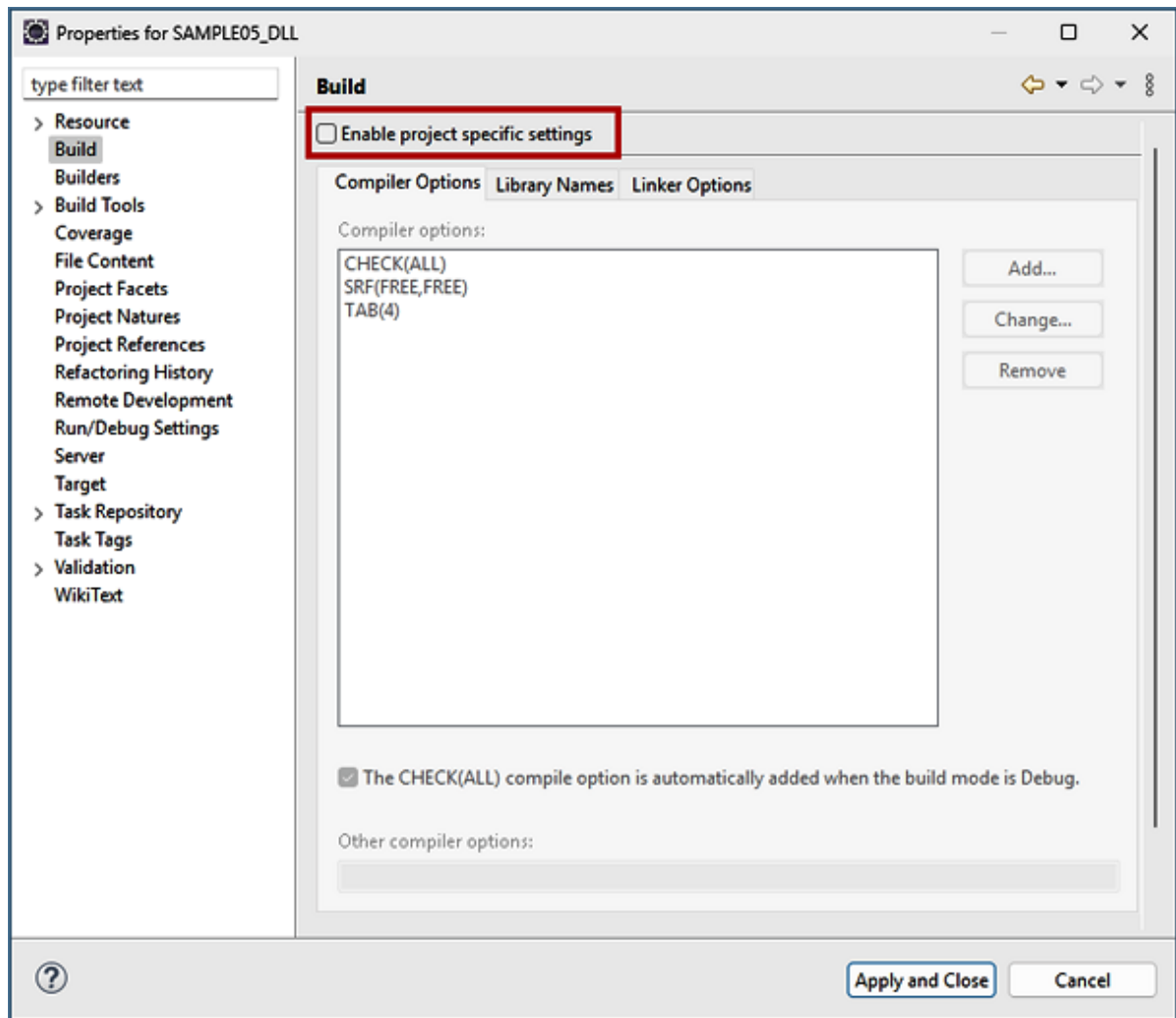
- b. In the left pane, select **Build**. The **Build** page appears. Click the **Compiler Options** tab, confirm the content of the option.



Here, confirm the storage folder of S_REC.CBL is specified for compiler option LIB, and compiler option SRF(FREE,FREE) is specified. Click **Apply and Close**.

5. Build setting of sub program and main program

Display the build page of the SAMPLE05_DLL project and the SAMPLE05_EXE project.



Uncheck the **Enable project specific settings** check box.

6. The library reference in the main program

The main program links library file (INSATSU.lib). This library file is output from the SAMPLE05_DLL project. Confirm "INSATSU.lib" is added to the link file of the SAMPLE05_EXE project on the Dependency view.



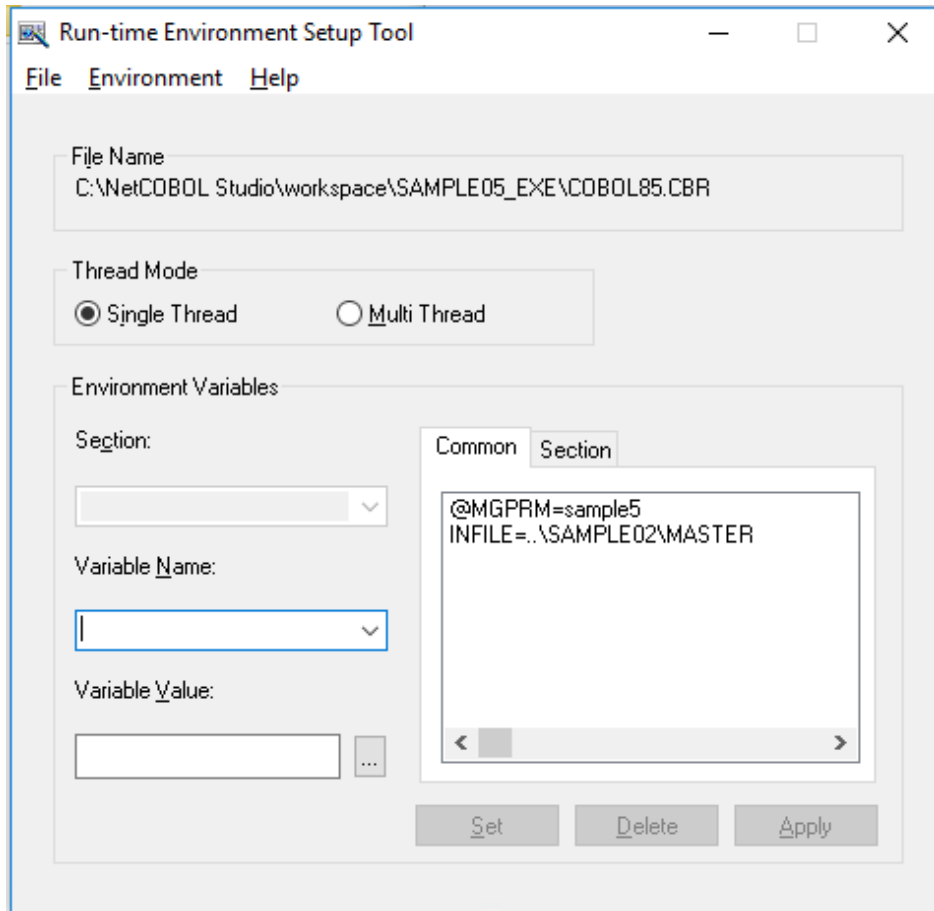
7. Build of solution project

Select the SAMPLE05 project, and chosen the "Rebuild Project" from the context menu.

The solution project is built, and then SAMPLE5.EXE is created.

Setting Runtime Environment Information

1. Start **Run-time Environment Setup Tool**. The **Run-time Environment Setup Tool** dialog appears.



2. On the **File** menu, click **Open**. The **Select Runtime Initialization** file dialog box appears.
3. Select COBOL85.CBR in the folder that contains the executable program (SAMPLE5.EXE).

When the build is done from NetCOBOL Studio, the executable program is made for the SAMPLE05_EXE project folder.

The content of the initialization file for execution is displayed.

4. On the **Common** tab and enter data as shown below:

- For the file-identifier INFILE, specify the path name of the master file (MASTER) created in Sample 2.

```
INFILE= . . \SAMPLE02\MASTER
```

A relative path is a path relative to the current folder.

When you select the **Run** menu > **Run As** > **COBOL Application**, the current folder is a project folder.

- Specify a work file name in the @MGPRM parameter. The string in this parameter is passed to the first linkage section item in the executing program. The work file name can contain up to 8 alphanumeric characters. The extension "TMP" is added to the work file name before the file is created.

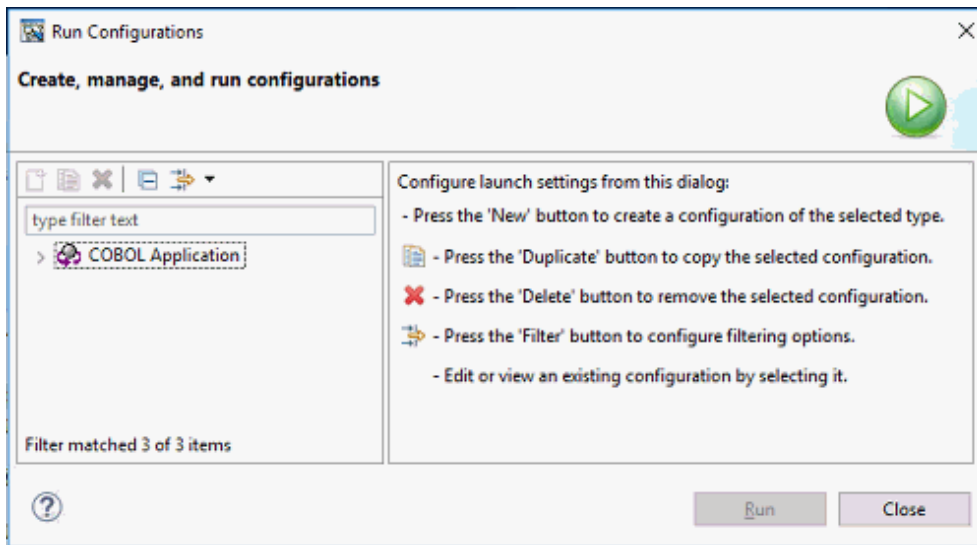
5. Click **Apply**. The data is saved in the object initialization file.
6. On the **File** menu, click **Exit** to terminate the run-time environment setup tool.

Executing the Program

This sample program makes the executable file of the dynamic linking structure. It is necessary to add the storage folder of the dynamic link library (DLL) to environment variable PATH when the DLL is not in the same folder of executable file (EXE) because the DLL of sub-program is loaded by dynamic linker of the system.

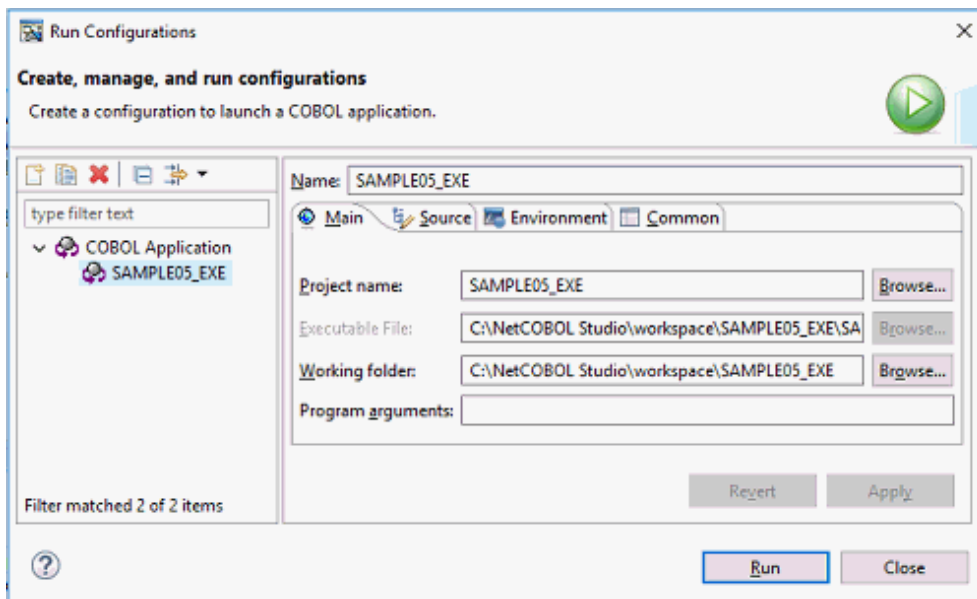
It explains the method of setting environment variable information with NetCOBOL Studio.

1. In the Dependency view, select SAMPLE05_EXE project.
2. On the **Run** menu, click **Run Configurations....** The **Run Configurations** dialog boxes appears.

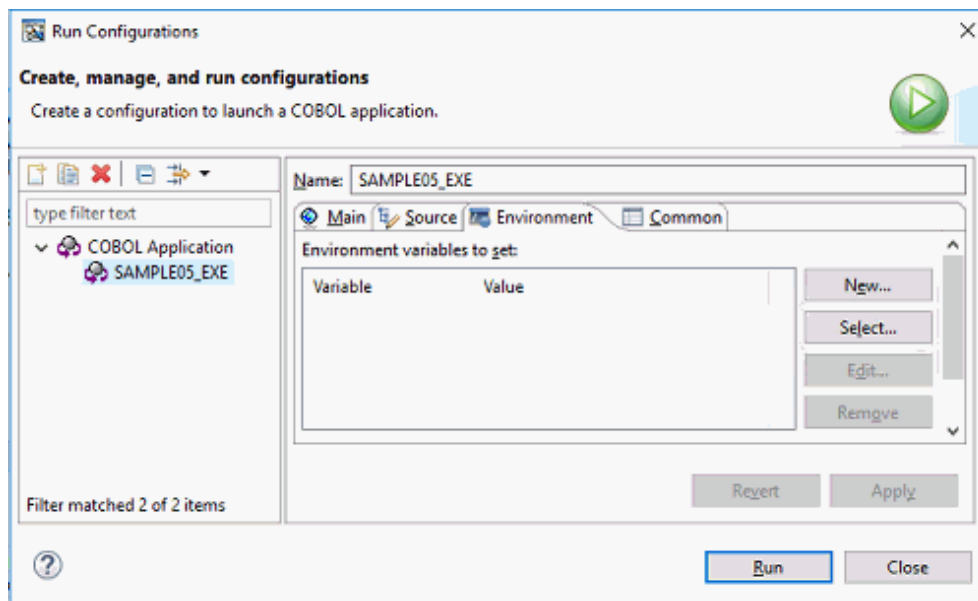


3. In the left pane, select **COBOL Application**, and then click the "New Launch Configuration" () button.

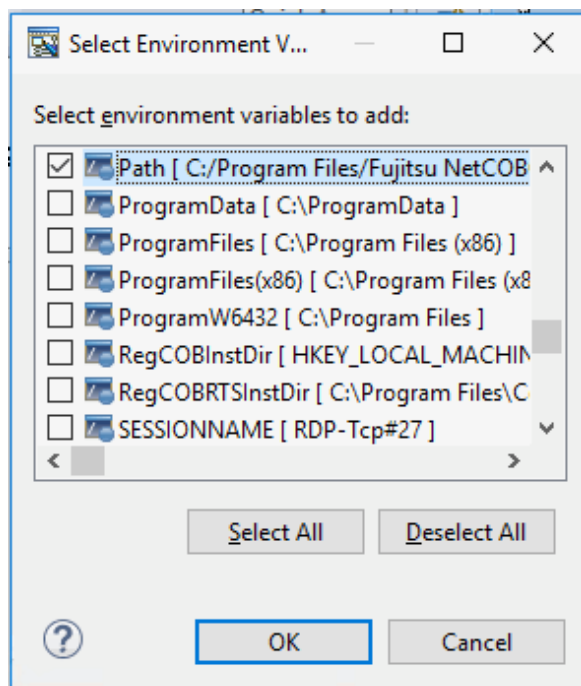
In the right pane, "SAMPLE05_EXE" is displayed in the name, and the configuration information at execution time is displayed.



4. Select the **Environment** tab.

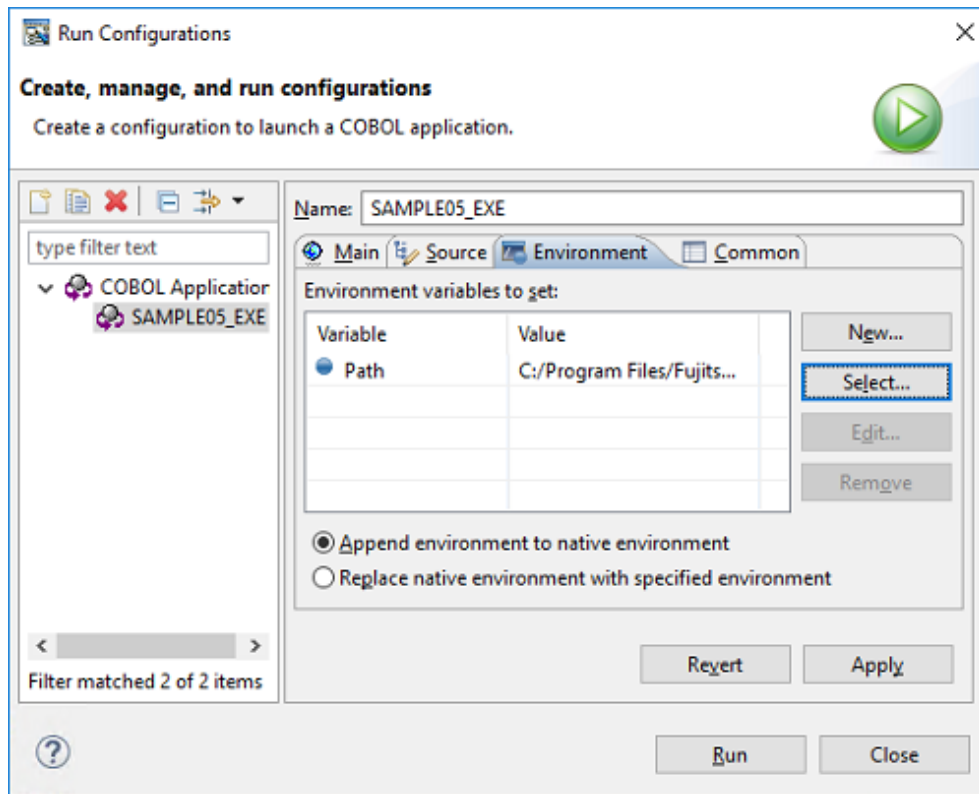


5. Here, the storage folder of INSATSU.DLL is added to environment variable Path. First of all, click **Select...**. The **Select Environment Variables** dialog box is displayed.



6. Check the **Path** check box, and click **OK**.

"Path" is added to the **Environment variables to set** list.



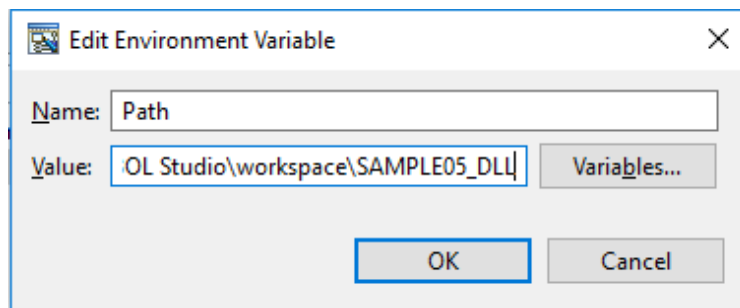
7. It is confirmed that the **Append environment to native environment** button is clicked.

Note

When the "Replace native environment with specified environment" is checked and the following procedure is progressed, the application cannot be correctly executed.

Check the "Append environment to native environment".

8. On the **Environment variables to set** list, select "Path", and click the **Edit...** The **Edit Environment Variable** dialog box appears.



Add the storage folder of SAMPLE05_DLL project to the **Value** field, and then click **OK**.

9. On the **Environment** tab, click **Apply**. The environmental setting of execution time is completed.
10. Click **Run**. SAMPLE5.EXE is executed.

Execution result

The message "GENERATE WORK-FILE=SAMPLE5.TMP" is displayed. Confirm the message contents, and close the message box by clicking **OK** button.

When the program execution ends, master file is printed to the printer which is set as "Set as default printer".

1.5.2 Using MAKE file

Compiling and Linking the Program

Open a command prompt. Compile and link the sample program using the following command.

```
c:\COBOL\Samples\COBOL\Sample05>nmake
```

Compilation of the sample program is now complete. Verify that SAMPLE5.EXE and INSATSU.DLL were created in the same folder in which the sample program is stored.

Setting Runtime Environment Information

Same as "[1.5.1 Using NetCOBOL Studio](#)".

Executing the Program

Execute SAMPLE5.EXE from a command prompt or from Windows Explorer.

The message "GENERATE WORK-FILE=SAMPLE5.TMP" is displayed. Confirm the contents, and then click the OK button to close the message box.

The master file contents are written to the default printer at the completion of program execution.

Execution result

Same as "[1.5.1 Using NetCOBOL Studio](#)".

1.6 Sample 6: Receiving a Command Line Argument

Sample 6 demonstrates a program that receives an argument specified at program execution, using the command line argument handling function (ACCEPT FROM argument-name/argument-value). Refer to "Using ACCEPT and DISPLAY Statements" in "NetCOBOL User's Guide" for details on how to use the command line argument handling function.

Sample 6 also calls an internal program.

Overview

The sample program calculates the number of days from the start date to the end date. The start and end dates are specified as command arguments in the following format:

```
command-name start-date end-date
```

start-date, end-date:

Specify a year, month, and day between January 1, 1900 and December 31, 2172 in the YYYYMMDD format.

Files Included in Sample 6

- SAMPLE6.COB (COBOL source program)
- MAKEFILE
- COBOL85.CBR

COBOL Statements Used

The ACCEPT, CALL, COMPUTE, COPY, DISPLAY, DIVIDE, EXIT, GO TO, IF, MOVE, and PERFORM statements are used.

1.6.1 Using NetCOBOL Studio

Compiling and Linking the Program

1. NetCOBOL Studio is started by specifying the workspace created for the sample program. See "[Preparing the workspace.](#)"
2. In the Dependency view, the presence of the SAMPLE06 project is confirmed.

If there is no SAMPLE06 project, import the SAMPLE06 project into the NetCOBOL Studio workspace. See "[Importing the sample program project into the Eclipse IDE workspace.](#)"
3. The structure of the project is as follows.

```
SAMPLE06
├ Source Files
│   └ Sample6.cob
├ Linking Files
└ Other Files
    ├── .settings folder
    ├── COBOL85.CBR
    └ Makefile
```


The build is executed immediately after importing the project when an automatic build is set. In this case, the file (. EXE . OBJ etc.) is generated after the build is displayed in "Other Files". It is set to automatic build by default.

4. On the **Project** menu, click **Build Project** when SAMPLE6.EXE is not created in "Other Files" (When an automatic build is not executed).

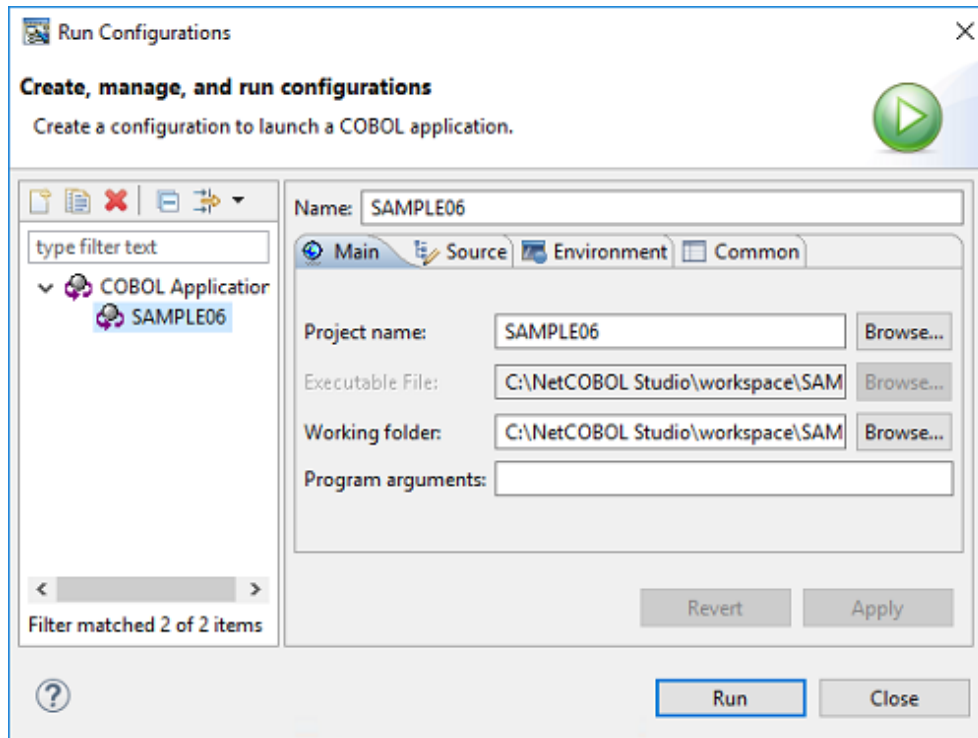
The project is built, and then SAMPLE6.EXE is created.

Executing the Program

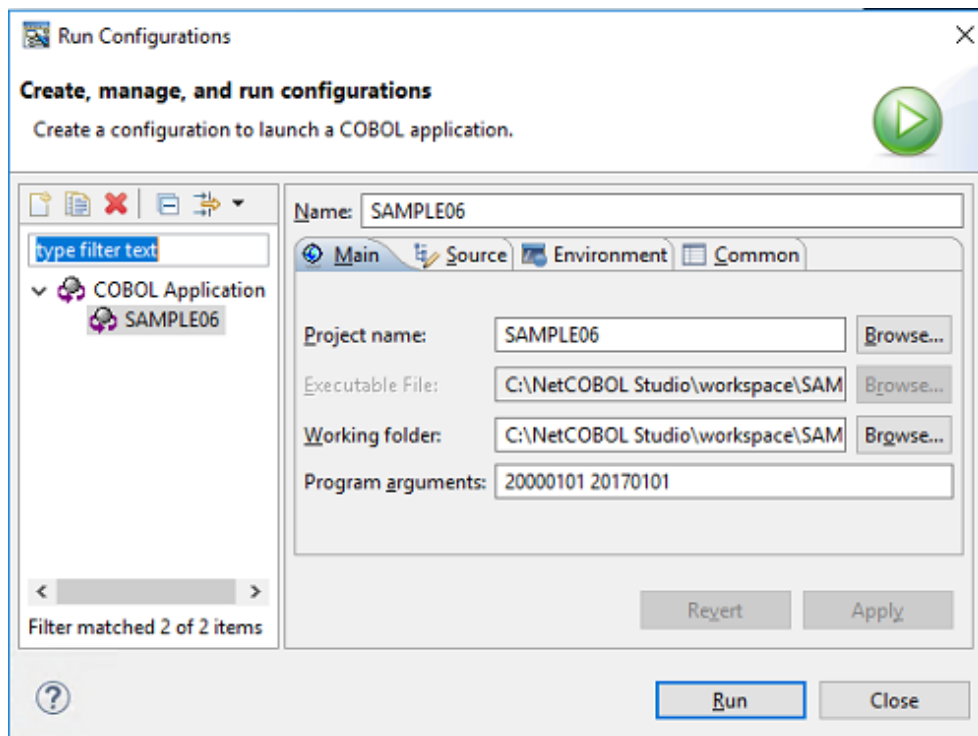
1. On the Dependency view, select the SAMPLE06 project
2. On the **Run** menu, click **Run Configurations....** The **Run Configurations** dialog boxes appears.

3. In the left pane, select **COBOL Application**, and then click the "New Launch Configuration" () button.

In the right pane, SAMPLE06 is displayed in the name, and the configuration information at execution time is displayed.



4. Select the **Main** tab.



In the **Program arguments** field, enter the start-date and end-date.

Example of program argument

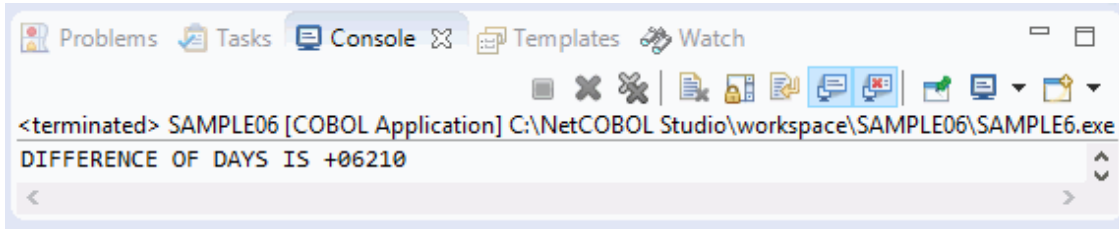
20000101 20170101

5. Click **Apply**, and then click **Run**. SAMPLE06 is started.

Execution result

The output destination of the DISPLAY statement is system console for this sample program.

Days from January 1, 2000 to January 1, 2017 are displayed as follows.



Refer to "Defining the target" in "NetCOBOL Studio User's Guide" when you want to make the output destination of the DISPLAY statement COBOL console.

1.6.2 Using MAKE file

Compiling and Linking the Program

Open a command prompt. Compile and link the sample program using the following command.

```
C:\COBOL\Samples\COBOL\Sample06>nmake
```

Compilation of the sample program is now complete. Verify that SAMPLE6.EXE was created in the same folder in which the sample program is stored.

Executing the Program

Execute "SAMPLE6.EXE start-date end-date" from a command prompt or from Windows Explorer.

```
C:\COBOL\Samples\COBOL\Sample06>SAMPLE6.EXE 20000101 20170101
```

Execution Result

Sample 6 displays the number of days from the specified start date to the specified end date.

```
C:\COBOL\Samples\COBOL\Sample06>SAMPLE6.EXE 20000101 20170101
DIFFERENCE OF DAYS IS +06210

C:\COBOL\Samples\COBOL\Sample06>
```

1.7 Sample 7: Environment Variable Handling

Sample 7 demonstrates a program that changes the value of an environment variable during COBOL program execution, using the environment variable handling function (ACCEPT FROM/DISPLAY UPON environment-name/environment-value). Refer to "Using ACCEPT and DISPLAY Statements" in "NetCOBOL User's Guide" for details on how to use the environment variable handling function.

Overview

The sample program divides a master file (the indexed file created in Sample 2) that contains product codes, product names, and unit prices into two master files according to product codes. The following table shows the division method and the names of the two new master files:

Table 1.1 Division of the master files

Product Code	File Name
Code beginning with 0	master-file-name.A
Code beginning with a non-zero value	master-file-name.B

Files Included in Sample 7

- SAMPLE7.COB (COBOL source program)
- MAKEFILE
- COBOL85.CBR

COBOL Statements Used

The ACCEPT, CLOSE, DISPLAY, EXIT, GO TO, IF, OPEN, READ, STRING, and WRITE statements are used.

1.7.1 Using NetCOBOL Studio

Compiling and Linking the Program

1. NetCOBOL Studio is started by specifying the workspace created for the sample program. See "[Preparing the workspace](#)."
2. In the Dependency view, the presence of the presence of the SAMPLE07 project is confirmed.
If there is no SAMPLE07 project, import the project for the sample program into the NetCOBOL Studio workspace. See "[Importing the sample program project into the Eclipse IDE workspace](#)."
3. The structure of the project is as follows.

```
SAMPLE07
├ Source Files
│   └ Sample7.cob
├ Linking Files
└ Other Files
    ├── .settings folder
    ├── COBOL85.CBR
    └ Makefile
```

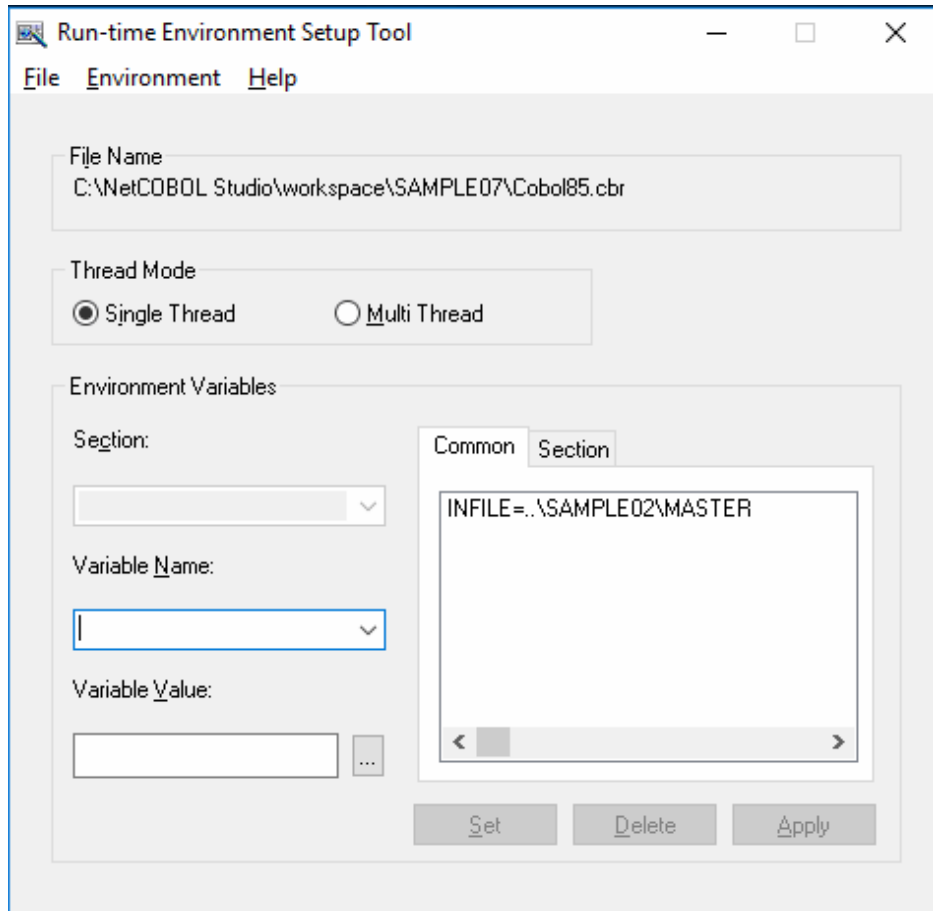
The build is executed immediately after importing the project when an automatic build is set. In this case, the file (. EXE . OBJ etc.) is generated after the build is displayed in "Other Files". It is set to automatic build by default.

4. On the **Project** menu, click **Build Project** when SAMPLE7.EXE is not created in "Other Files" (When an automatic build is not executed).

The project is built, and then SAMPLE7.EXE is created.

Setting Runtime Environment Information

1. Start **Run-time Environment Setup Tool**. The **Run-time Environment Setup Tool** dialog appears.



2. On the **File** menu, click **Open**. The **Select Runtime Initialization** file dialog box appears.
3. Select COBOL85.CBR in the folder that contains the executable program (SAMPLE7.EXE).
4. On the **Common** tab and enter data as shown below:
 - For the file-identifier INFILE, specify the path name of the master file (MASTER) created in Sample 2.

```
INFILE=..\SAMPLE02\MASTER
```

5. Click **Apply**. The data is saved in the object initialization file.
6. On the **File** menu, click **Exit** to terminate the run-time environment setup tool.

Executing the Program

1. In the Dependency view, select SAMPLE07 project.
2. On the **Run** menu, click **Run As > COBOL Application**.



Execute the program in Sample 2 beforehand.

Execution result

The following two files are created in the directory of the master file created in Sample 2:

- MASTER.A: Stores the data of products whose codes begin with 0.
- MASTER.B: Stores the data of products whose codes begin with a non-zero value.

The contents of the newly created master files (MASTER.A and MASTER.B) can be checked with the program in Sample 5 in the same manner as for the master file created in Sample 2.

1.7.2 Using MAKE file

Compiling and Linking the Program

Open a command prompt. Compile and link the sample program using the following command.

```
C:\COBOL\Samples\COBOL\Sample07>nmake
```

Compilation of the sample program is now complete. Verify that SAMPLE7.EXE was created in the same folder in which the sample program is stored.

Setting Runtime Environment Information

Same as "[1.7.1 Using NetCOBOL Studio](#)".

Executing the Program

Execute SAMPLE7.EXE from a command prompt or from Windows Explorer.



Note

Execute the program in Sample 2 beforehand.

Execution result

Same as "[1.7.1 Using NetCOBOL Studio](#)".

1.8 Sample 8: Using a Print File

Sample 8 demonstrates a program that outputs data (input from the console window) to a printer using a print file. Refer to "Printing" in "NetCOBOL User's Guide" for details on using a print file.

Overview

The sample program inputs data of up to 40 alphanumeric characters from the console window, and outputs the data to the printer.

Files Included in Sample 8

- SAMPLE8.COB (COBOL source program)
- MAKEFILE
- COBOL85.CBR

COBOL Statements Used

The ACCEPT, CLOSE, EXIT, GO TO, IF, OPEN, and WRITE statements are used.

1.8.1 Using NetCOBOL Studio

Compiling and Linking the Program

1. NetCOBOL Studio is started by specifying the workspace created for the sample program. See "[Preparing the workspace](#)."

2. In the Dependency view, the presence of the presence of the SAMPLE08 project is confirmed.

If there is no SAMPLE08 project, import the project for the sample program into the NetCOBOL Studio workspace. See "[Importing the sample program project into the Eclipse IDE workspace](#)."

3. The structure of the project is as follows.

```
SAMPLE08
├ Source Files
│   └ Sample8.cob
├ Linking Files
└ Other Files
    ├── .settings folder
    ├── COBOL85.CBR
    └ Makefile
```

The build is executed immediately after importing the project when an automatic build is set. In this case, the file (. EXE . OBJ etc.) is generated after the build is displayed in "Other Files". It is set to automatic build by default.

4. On the **Project** menu, click **Build Project** when SAMPLE8.EXE is not created in "Other Files" (When an automatic build is not executed).

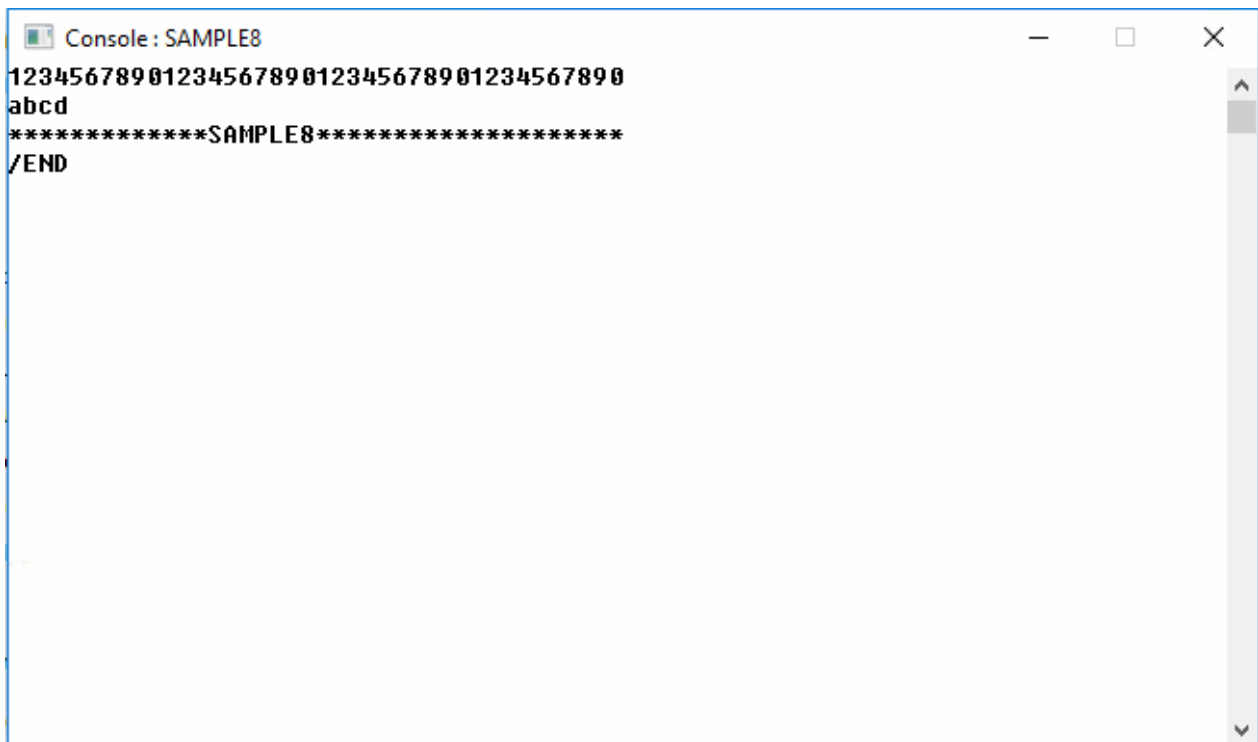
The project is built, and then SAMPLE8.EXE is created.

Executing the Program

1. In the Dependency view, select SAMPLE08 project.
2. On the **Run** menu, click **Run As > COBOL Application**.

Execution result

A console window is displayed. In the console window, enter the data to be printed. Up to 40 characters can be entered at a time.



To terminate the program, press the RETURN key, type **/END** and press the RETURN key again. Click the "OK" button to close the message window.

The input data is written to the printer at program termination.


```
1234567890123456789012345678901234567890
abcd
*****SAMPLE8*****
```

1.8.2 Using MAKE file

Compiling and Linking the Program

Open a command prompt. Compile and link the sample program using the following command.

```
C:\COBOL\Samples\COBOL\Sample08>nmake
```

Compilation of the sample program is now complete. Verify that SAMPLE8.EXE was created in the same folder in which the sample program is stored.

Executing the Program

Execute SAMPLE8.EXE from a command prompt or from Windows Explorer.

Execution result

Same as "[1.8.1 Using NetCOBOL Studio](#)".

1.9 Sample 9: Using a Print File (Advanced usage)

Sample 9 demonstrates the following:

- Using a print file without a FORMAT clause
- Using the I control record to set and change page forms, in combination with Forms Control Buffers (FCBs)
- Using the CHARACTER TYPE clause to control letter size and pitch
- Using the PRINTING POSITION clause to control the layout (line / column)

Refer to "Printing" in "NetCOBOL User's Guide" for details on using "print file 1" and "print file 2".

Overview

The table below describes each of the tasks performed by this sample. The tasks show a number of printing features. There are essentially four elements that give you control over the various printing features:

1. COBOL syntax
 - PRINTING MODE clauses in the SPECIAL-NAMES paragraph.
 - ASSIGN TO PRINTER in the SELECT statement.
 - CHARACTER TYPE and PRINTING POSITION clauses in data definitions.
2. The I-Control Record

A record that you write to the print file using the syntax:

"CTL IS page-ctrl" in SPECIAL-NAMES

and

WRITE I-Control-Record AFTER ADVANCING page-cntl

3. Forms Control Buffers (FCBs)

These are form information buffers stored by the COBOL runtime system, using information defined in environment variables of the form "FCBxxxx=".

4. Environment variables

Environment variables define fonts, FCBs, document names and other printing details.

The table below indicates which of the above are used to provide a particular feature. You will need to read the table, inspect the COBOL code, and consult the chapter on "Printing" in the "NetCOBOL User's Guide" to fully understand all of the features being demonstrated.

Table 1.2 Features demonstrated in Sample 9

Task Description	Detailed features	Controlled by I-Control field / COBOL clause	Related Environment Variable(s)
1a. Prints a page at 6 LPI, 10 CPI on a PowerFORM overlay grid	6 LPI - defined in FCB	I-Control: FCB-NAME (="LT6L")	FCBLT6L=...
	10 CPI	PRINTING MODE x ... AT PITCH ... + CHARACTER TYPE x	
	Letter size paper	I-Control: PAPER-SIZE (="LTR")	
	Impact font	PRINTING MODE x ... WITH FONT GOTHIC ... + CHARACTER TYPE x	@PrinterFontName= (..., Impact)
	Courier New font	PRINTING MODE x ... WITH FONT MINCHOU ... + CHARACTER TYPE x	@PrinterFontName= (Courier New, ...)
	Grid (PowerFORM overlay - KOL6LT6L.OVD)	I-Control: FOVL-NAME (="LT6L") FOVL-R (= 1 - to use overlay on a single page)	FOVLTYPE=KOL6 OVD_SUFFIX=OVD
	Data item position within line	PRINTING POSITION	
	Different character type forms	PRINTING MODE x ... FORM ... +	

Task Description	Detailed features	Controlled by I-Control field / COBOL clause	Related Environment Variable(s)
		CHARACTER TYPE x	
	Document name displayed by Windows	I-Control: DOCUMENT-NAME (=DOC1)	@CBR_DocumentName_ DOC1=<document name string>
1b. Prints a page at 8 LPI, 10 CPI on a PowerFORM overlay grid	8 LPI - defined in FCB	I-Control: FCB-NAME (= "LT8L")	FCBLT8L=...
	10 CPI	PRINTING MODE x ... AT PITCH ... + CHARACTER TYPE x	
	Letter size paper	I-Control: PAPER-SIZE (= "LTR")	
	Impact font	PRINTING MODE x ... WITH FONT GOTHIC ... + CHARACTER TYPE x	@PrinterFontName= (..., Impact)
	Courier New font	PRINTING MODE x ... WITH FONT MINCHOU ... + CHARACTER TYPE x	@PrinterFontName= (Courier New, ...)
	Grid (PowerFORM overlay - KOL6LT8L.OVD)	I-Control: FOVL-NAME (= "LT8L") FOVL-R (= 1 - to use overlay on a single page)	FOVLTYPE=KOL6 OVD_SUFFIX=OVD
	Data item position within line	PRINTING POSITION	
	Different character type forms	PRINTING MODE x ... FORM ... + CHARACTER TYPE x	
2a. Prints letters in font sizes 3, 7.2, 9, 12, 18, 24, 36, 50, 72, 100, 200, and 300 points. On legal-sized paper (After printing a header page)	Document name displayed by Windows	I-Control: DOCUMENT-NAME (=DOC1)	@CBR_DocumentName_ DOC1=<document name string>
	Different font sizes	PRINTING MODE x ... IN SIZE nn POINT ...	

Task Description	Detailed features	Controlled by I-Control field / COBOL clause	Related Environment Variable(s)
The COBOL runtime system automatically calculates the best character pitch fitted to the character size (character pitch specification is omitted).		+ CHARACTER TYPE x	
	Legal size paper	I-Control: PAPER-SIZE ("XXX") FCB-NAME ("LPI6")	@PRN_FormName_XXX =Legal 8 1/2 x 14 in FCBLPI6=...
	Impact font	Default - Gothic font	@PrinterFontName=(..., Impact)
	Shaded background (PowerFORM overlay - KOL6LGLT.OVD)	I-Control: FOVL-NAME ("LGLT") FOVL-R (= 1 - to use overlay on a single page)	FOVLTYPE=KOL6 OVD_SUFFIX=OVD
	Document name displayed by Windows	I-Control: DOCUMENT-NAME (=DOC2)	@CBR_DocumentName_ DOC2=<document name string>
2b. Prints characters at pitches 1, 2, 3, 5, 6, 7.5, 20, and 24 CPI. The COBOL runtime system automatically calculates the best character size fitted to the character pitch (the character size specification is omitted).	Different pitches	PRINTING MODE x ... AT PITCH n CPI ... + CHARACTER TYPE x	
	Legal size paper	I-Control: PAPER-SIZE ("XXX") FCB-NAME ("LPI6")	@PRN_FormName_XXX =Legal 8 1/2 x 14 in FCBLPI6=...
	Impact font	Default - Gothic font	@PrinterFontName=(..., Impact)
	Shaded background (PowerFORM overlay - KOL6LGLT.OVD)	I-Control: FOVL-NAME ("LGLT") FOVL-R (= 1 - to use overlay on a single page)	FOVLTYPE=KOL6 OVD_SUFFIX=OVD
	Document name displayed by Windows	I-Control: DOCUMENT-NAME (=DOC2)	@CBR_DocumentName_ DOC2=<document name string>
2c. Prints characters in Impact, Impact half-size, Courier New, Courier New half size.	Impact font	PRINTING MODE x ... WITH FONT {GOTHIC } {GOTHIC-HANKAKU} ... + CHARACTER TYPE x	@PrinterFontName=(..., Impact)

Task Description	Detailed features	Controlled by I-Control field / COBOL clause	Related Environment Variable(s)
	Courier New font	PRINTING MODE x ... WITH FONT {MINCHOU } {MINCHOU-HANKAKU} ... + CHARACTER TYPE x	@PrinterFontName= (Courier New, ...)
	Full size	PRINTING MODE x ... BY FORM F ... + CHARACTER TYPE x	
	Half size	PRINTING MODE x ... BY FORM H ... + CHARACTER TYPE x	
	Legal size paper	I-Control: PAPER-SIZE ("XXX") FCB-NAME ("LPI6")	@PRN_FormName_XXX =Legal 8 1/2 x 14 in FCBLPI6=...
	Shaded background (PowerFORM overlay - KOL6LGLT.OVD)	I-Control: FOVL-NAME ("LGLT") FOVL-R (= 1 - to use overlay on a single page)	FOVLTYPE=KOL6 OVD_SUFFIX=OVD
	Document name displayed by Windows	I-Control: DOCUMENT-NAME (=DOC2)	@CBR_DocumentName_ DOC2=<document name string>
2d. Prints characters in different form sizes: Em-size, en-size, expanded em- size, expanded en-size, tall em-size, tall en-size, double em-size and double en-size.	Em-size	PRINTING MODE x ... BY FORM F ... + CHARACTER TYPE x	
	En-size	As above with: ... BY FORM H ...	
	Expanded em-size	As above with: ... BY FORM F0201 ...	

Task Description	Detailed features	Controlled by I-Control field / COBOL clause	Related Environment Variable(s)
	Expanded en-size	As above with: ... BY FORM H0201 ...	
	Tall em-size	As above with: ... BY FORM F0102 ...	
	Tall en-size	As above with: ... BY FORM H0102 ...	
	Double em-size	As above with: ... BY FORM F0202...	
	Double en-size	As above with: ... BY FORM H0202...	
	Legal size paper	I-Control: PAPER-SIZE ("XXX") FCB-NAME ("LPI6")	@PRN_FormName_XXX =Legal 8 1/2 x 14 in FCBLPI6=...
	Shaded background (PowerFORM overlay - KOL6LGLT.OVD)	I-Control: FOVL-NAME ("LGLT") FOVL-R (= 1 - to use overlay on a single page)	FOVLTYPE=KOL6 OVD_SUFFIX=OVD
	Document name displayed by Windows	I-Control: DOCUMENT-NAME (=DOC2)	@CBR_DocumentName_ DOC2=<document name string>
2e. Prints a mixture of the above features: font size, pitch, half/full size characters.			

Files Included in Sample 9

- SAMPLE9.COB (COBOL source program)
- KOL6LGLT.OVD (Form overlay pattern)
- KOL6LT6L.OVD (Form overlay pattern)
- KOL6LT8L.OVD (Form overlay pattern)
- COBOL85.CBR
- MAKEFILE

COBOL Statements Used

The ADD, CLOSE, DISPLAY, IF, MOVE, OPEN, PERFORM, STOP, and WRITE statements are used.

1.9.1 Using NetCOBOL Studio

Compiling and Linking the Program

1. NetCOBOL Studio is started by specifying the workspace created for the sample program. See "[Preparing the workspace.](#)"
2. In the Dependency view, the presence of the SAMPLE09 project is confirmed.

If there is no SAMPLE09 project, import the project for the sample program into the NetCOBOL Studio workspace. See "[Importing the sample program project into the Eclipse IDE workspace.](#)"

3. The structure of the project is as follows.

```
SAMPLE09
├ Source Files
│   └ Sample9.cob
├ Linking Files
└ Other Files
    ├── .settings folder
    ├── COBOL85.CBR
    ├── Kol6lgl.t.OVD
    ├── Kol6lt6l.OVD
    ├── Kol6lt8l.OVD
    └ Makefile
```

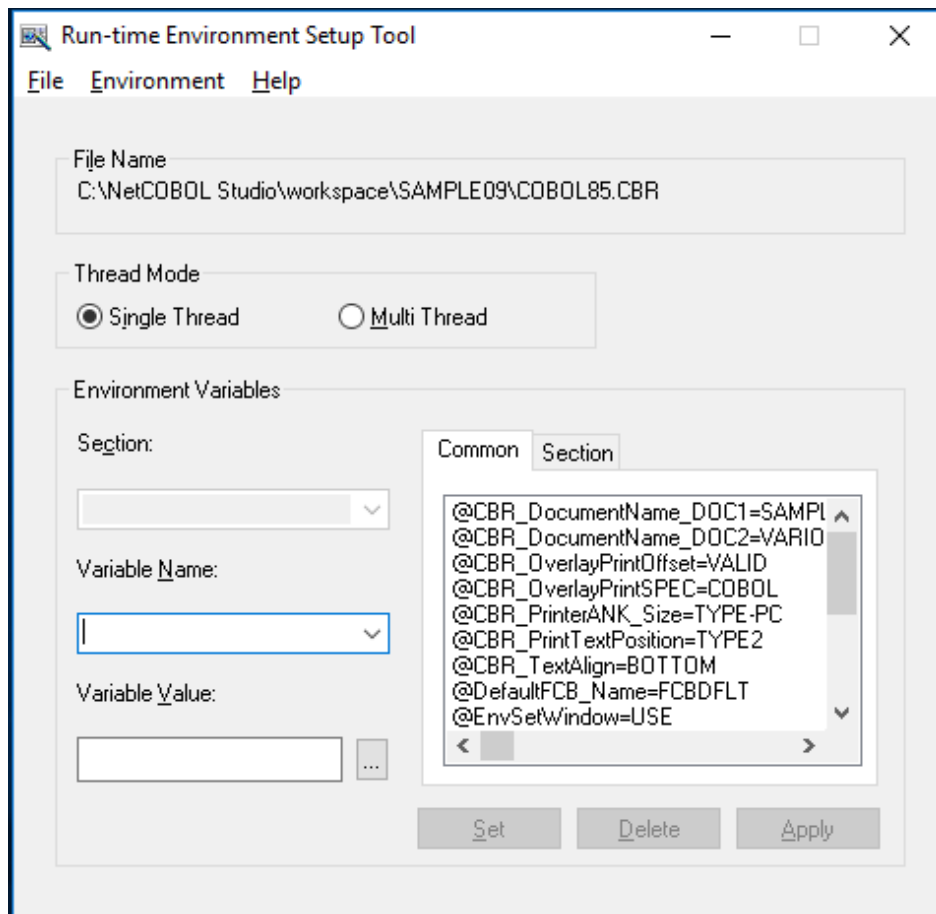
The build is executed immediately after importing the project when an automatic build is set. In this case, the file (.EXE .OBJ etc.) is generated after the build is displayed in "Other Files". It is set to automatic build by default.

4. On the **Project** menu, click **Build Project** when SAMPLE9.EXE is not created in "Other Files" (When an automatic build is not executed).

The project is built, and then SAMPLE9.EXE is created.

Setting Runtime Environment Information

1. Start **Run-time Environment Setup Tool**. The **Run-time Environment Setup Tool** dialog appears.



2. On the **File** menu, click **Open**. The **Select Runtime Initialization** file dialog box appears.
3. Select COBOL85.CBR in the SAMPLE09 folder. The window should look like the above figure.
4. Check the setting of environment variable FOVLDIR in the list of environment variables. If it is not set to your location for the SAMPLE09 folder, change it to that value by:

FOVLDIR= . \

- a. Selecting FOVLDIR in the environment variable list. "FOVLDIR" will be displayed in the Variable Name field, and its current setting in the Variable Value field.
 - b. Use the browse ("...") button to navigate to the SAMPLE09 folder, select any file, and click **OK**. The path and filename are returned to the Variable Value field.
 - c. Delete the last "\" and the file name that follows it from the string in the Variable Value field.
 - d. Click the Set button to set your change in the Section list of environment variables.
 - e. Click the Apply button to save your changes to the COBOL85.CBR file.
5. Click **Apply**. The data is saved in the object initialization file.
 6. On the **File** menu, click **Exit** to terminate the run-time environment setup tool.

Executing the Program

1. In the Dependency view, select SAMPLE09 project.
2. On the **Run** menu, click **Run As > COBOL Application**.

Execution result

The sample pages described in the table "Features demonstrated in Sample 9" above are printed to the default printer.

1.9.2 Using MAKE file

Compiling and Linking the Program

Open a command prompt. Compile and link the sample program using the following command.

```
C:\COBOL\Samples\COBOL\Sample09>nmake
```

Compilation of the sample program is now complete. Verify that SAMPLE9.EXE was created in the same folder in which the sample program is stored.

Setting Runtime Environment Information

Same as "[1.9.1 Using NetCOBOL Studio](#)".

Executing the Program

Execute SAMPLE9.EXE from a command prompt or from Windows Explorer.

Execution result

Same as "[1.9.1 Using NetCOBOL Studio](#)".

1.10 Sample 11: Remote Database Access

Sample 11 extracts data from a database and assigns it to a host variable using the SQL database function.

In normal operation, the database is placed on a server and is accessed by the client via an ODBC driver.

For more information about using ODBC drivers, refer to "Database (SQL)" in "NetCOBOL User's Guide", and the relevant documentation from your database vendor.

To run this sample program in a true distributed configuration, the following products are required:

Client

- ODBC driver manager
- ODBC driver
- Products needed for the ODBC driver

On the server

- Database
- Products needed for accessing the database via ODBC

Overview

The sample program accesses the database on the server and outputs all data stored in the database table "STOCK" to the client console. When all data has been referenced, the link to the database is disconnected.

Files Included in Sample 11

- SAMPLE11.COB (COBOL source program)
- MAKEFILE
- COBOL85.CBR

COBOL Statements Used

The DISPLAY, IF and PERFORM statements are used.

Embedded SQL statements (embedded exception declarations and CONNECT, DECLARE CURSOR, OPEN, FETCH, CLOSE, ROLLBACK, and DISCONNECT statements) are also used.

Prerequisite to Executing the Program

- ODBC is a defined interface that attempts to provide a highly generic API into any database system that provides compliant drivers. Just about every database system available today provides ODBC drivers for a variety of platforms.
- In order to execute this sample, the DBMS product which can be connected via ODBC is installed in server side and make the table named STOCK for the database connected by default.

Make the STOCK table in the format as following.

GNO	GOODS	QOH	WHNO
Binary integer 4 digits	Fixed-length character 10 bytes	Binary integer 9 digits	Binary integer 4 digits

Store the data items shown below in the STOCK table.

GNO	GOODS	QOH	WHNO
110	TELEVISION	85	2
111	TELEVISION	90	2
123	REFRIGERATOR	60	1
124	REFRIGERATOR	75	1
137	RADIO	150	2
138	RADIO	200	2
140	CASSETTE DECK	120	2
141	CASSETTE DECK	80	2
200	AIR CONDITIONER	4	1
201	AIR CONDITIONER	15	1
212	TELEVISION	0	2
215	VIDEO	5	2
226	REFRIGERATOR	8	1
227	REFRIGERATOR	15	1
240	CASSETTE DECK	25	2
243	CASSETTE DECK	14	2
351	CASSETTE TAPE	2500	2
380	SHAVER	870	3
390	DRIER	540	3

- Create the ODBC information file ("DBMSACS.INF" in this sample) using SQLODBC.S.EXE.

1.10.1 Using NetCOBOL Studio

Compiling and Linking the Program

1. NetCOBOL Studio is started by specifying the workspace created for the sample program. See ["Preparing the workspace."](#)

2. In the Dependency view, the presence of the presence of the SAMPLE11 project is confirmed.
3. If there is no SAMPLE11 project, import the project for the sample program into the NetCOBOL Studio workspace. See "[Importing the sample program project into the Eclipse IDE workspace.](#)"
4. The structure of the project is as follows.

```

SAMPLE11
├ Source Files
├   └ Sample11.cob
├ Linking Files
├ Other Files
├   └ .settings folder
├   └ COBOL85.CBR
├   └ Makefile

```

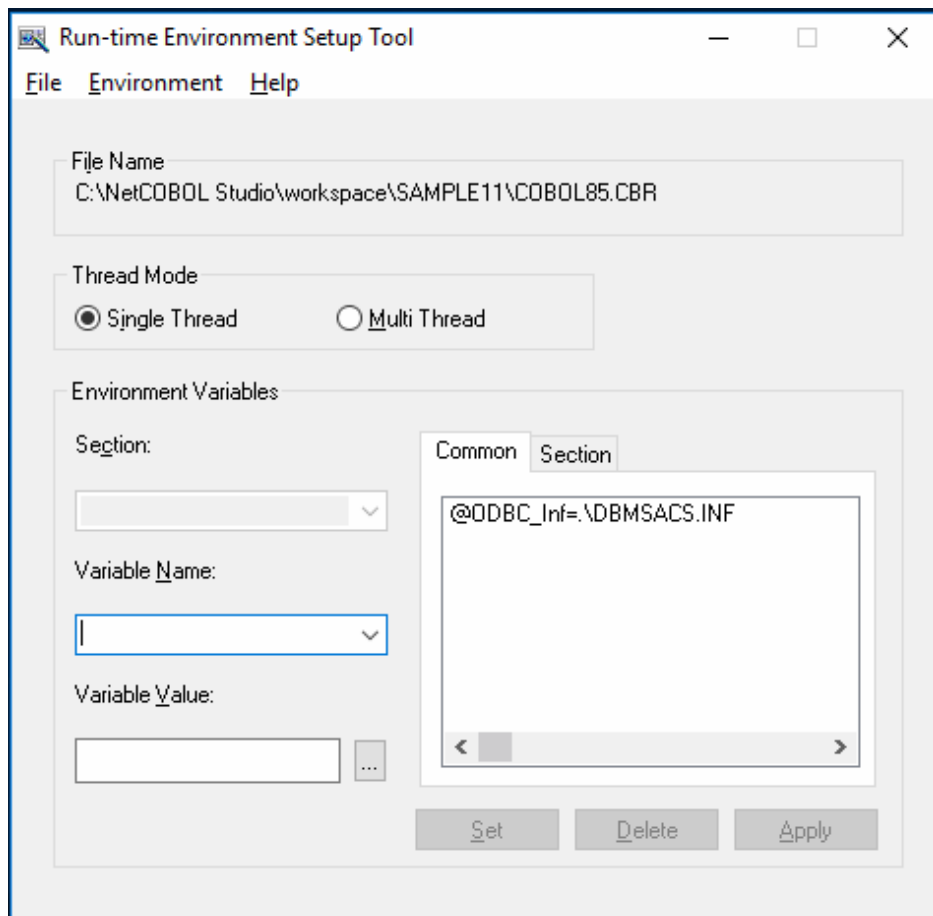
The build is executed immediately after importing the project when an automatic build is set. In this case, the file (. EXE . OBJ etc.) is generated after the build is displayed in "Other Files". It is set to automatic build by default.

5. On the **Project** menu, click **Build Project** when SAMPLE11.EXE is not created in "Other Files" (When an automatic build is not executed).

The project is built, and then SAMPLE11.EXE is created.

Setting Runtime Environment Information

1. Start **Run-time Environment Setup Tool**. The **Run-time Environment Setup Tool** dialog appears.



2. On the **File** menu, click **Open**. The **Select Runtime Initialization** file dialog box appears.
3. Select COBOL85.CBR in the folder that contains the executable program (SAMPLE11.EXE).

4. Select the Common tab and enter data as shown below:
 - Specify DBMSACS.INF for the environment variable @ODBC_Inf.
5. Click **Apply**. The data is saved in the object initialization file.
6. On the **File** menu, click **Exit** to terminate the run-time environment setup tool.

Executing the Program

1. In the Dependency view, select SAMPLE11 project.
2. On the **Run** menu, click **Run As > COBOL Application**.

Execution result

The data extracted from the table is displayed, as shown in the following figure. (Console: SAMPLE11)

```
no.12:
Product number    = +0227
Product name      = REFRIGERATOR
Stock quantiry    = +00000015
Warehouse number  = +0001
no.13:
Product number    = +0240
Product name      = CASSETTE DECK
Stock quantiry    = +00000025
Warehouse number  = +0002
no.14:
Product number    = +0243
Product name      = CASSETTE DECK
Stock quantiry    = +00000014
Warehouse number  = +0002
no.15:
Product number    = +0351
Product name      = CASSETTE TAPE
Stock quantiry    = +00002500
Warehouse number  = +0002

There are 15 data in total
END OF SESSION
```

1.10.2 Using MAKE file

Compiling and Linking the Program

Open a command prompt. Compile and link the sample program using the following command.

```
C:\COBOL\Samples\COBOL\Sample11>nmake
```

Compilation of the sample program is now complete. Verify that SAMPLE11.EXE was created in the same folder in which the sample program is stored.

Setting Runtime Environment Information

Same as "[1.10.1 Using NetCOBOL Studio](#)".

Executing the Program

Execute SAMPLE11.EXE from a command prompt or from Windows Explorer.

Execution result

Same as "[1.10.1 Using NetCOBOL Studio](#)".

1.11 Sample 12: Remote Database Access (Multiple row processing)

Sample 12 shows how two or more lines in a database can be operated using one SQL statement, demonstrating an example of advanced usage of the database (SQL) function.

In this example, a database that exists on a server is accessed from a client, via an ODBC driver. For details of database access using an ODBC driver, refer to "Database (SQL)" in "NetCOBOL User's Guide."

To use this program, the following products are necessary:

Client

- ODBC driver manager
- ODBC driver
- Products necessary for the ODBC driver

Server

- Database



Note

.....
This sample does not function correctly with Microsoft® Access.
.....

- Products necessary for database access using ODBC

Overview

Sample 12 uses the STOCK table of the sample database. Refer to "Sample Database" in "NetCOBOL User's Guide" for details. Sample 12 accesses and disconnects it after the following operation:

- Display of all data items in the database
- Fetch of the GNO value on a row with GOODS value "TELEVISION"
- QOH update on a row with the fetched GNO
- Deletion of lines with GOODS values "RADIO", "SHAVER", and "DRIER"
- Redisplay of all data items in the database

Part of the output result is output to a file by using compiler option SSOUT.

Programs and files in Sample 12

- SAMPLE12.COB (COBOL source program)
- MAKEFILE
- COBOL85.CBR

COBOL functions used in Sample 12

- Remote database access
- Project management function

COBOL statements used

The CALL, DISPLAY, IF, and PERFORM statements are used.

Embedded SQL statements (host variable with multiple rows specified, host variable with a table specified, embedded exception declaration, CONNECT statement, cursor declaration, OPEN statement, FETCH statement, SELECT statement, DELETE statement, UPDATE statement, CLOSE statement, COMMIT statement, ROLLBACK statement, and DISCONNECT statement) are used.

Prerequisite to Executing the Program

In order to execute this sample, the DBMS product which can be connected via ODBC is installed in server side and make the table named STOCK for the database connected by default.

Make the STOCK table in the format as following.

GNO	GOODS	QOH	WHNO
Binary integer 4 digits	Fixed-length character 10 bytes	Binary integer 9 digits	Binary integer 4 digits

Store the data items shown below in the STOCK table.

GNO	GOODS	QOH	WHNO
110	TELEVISION	85	2
111	TELEVISION	90	2
123	REFRIGERATOR	60	1
124	REFRIGERATOR	75	1
137	RADIO	150	2
138	RADIO	200	2
140	CASSETTE DECK	120	2
141	CASSETTE DECK	80	2
200	AIR CONDITIONER	4	1
201	AIR CONDITIONER	15	1
212	TELEVISION	0	2
215	VIDEO	5	2
226	REFRIGERATOR	8	1
227	REFRIGERATOR	15	1
240	CASSETTE DECK	25	2
243	CASSETTE DECK	14	2
351	CASSETTE TAPE	2500	2
380	SHAVER	870	3
390	DRIER	540	3

Then create the ODBC information file using SQLODBCS.EXE.

1.11.1 Using NetCOBOL Studio

Compiling and Linking the Program

1. NetCOBOL Studio is started by specifying the workspace created for the sample program. See ["Preparing the workspace."](#)
2. In the Dependency view, select the SAMPLE12 project.

If there is no SAMPLE12 project, import the project of sample program into the NetCOBOL Studio workspace. See ["Importing the sample program project into the Eclipse IDE workspace."](#)

3. The structure of the project is as follows.

```
SAMPLE12
├ Source Files
│   └ Sample12.cob
├ Linking Files
└ Other Files
    ├── .settings folder
    ├── COBOL85.CBR
    └ Makefile
```

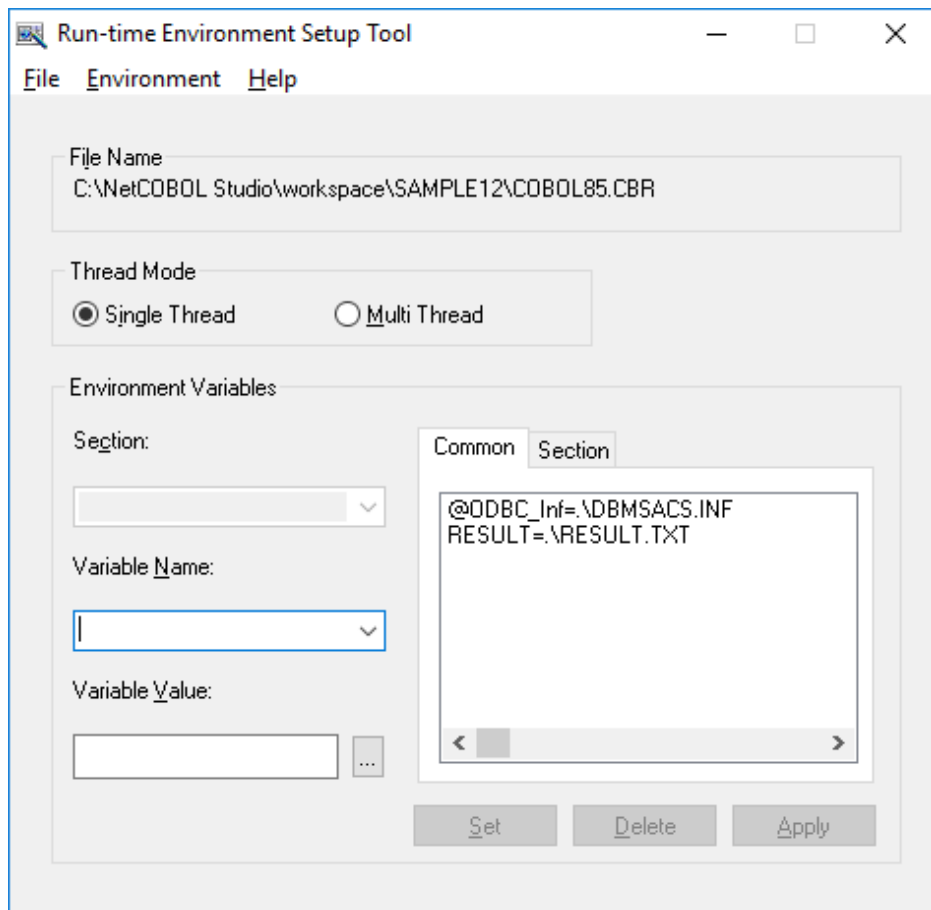
The build is executed immediately after importing the project when an automatic build is set. In this case, the file (. EXE . OBJ etc.) is generated after the build is displayed in "Other Files". It is set to automatic build by default.

4. On the **Project** menu, click **Build Project** when SAMPLE12.EXE is not created in "Other Files" (When an automatic build is not executed), click **Build Project** On the **Project** menu.

The project is built, and then SAMPLE12.EXE is created.

Setting Runtime Environment Information

1. Start **Run-time Environment Setup Tool**. The **Run-time Environment Setup Tool** dialog appears.



2. On the **File** menu, click **Open**. The **Select Runtime Initialization file** dialog box appears.
3. Select COBOL85.CBR in the folder that contains the executable program (SAMPLE12.EXE).
4. On the **Common** tab, enter data as shown below:
 - Specify an ODBC information file name in @ODBC_Inf (ODBC information file specification).
 - Specify a file to save the DISPLAY statement output result in environment variable RESULT.

5. Click **Apply**. The data is saved in the object initialization file.
6. On the **File** menu, click **Exit** to terminate the run-time environment setup tool.

Executing the Program

1. In the Dependency view, select SAMPLE12 project.
2. On the **Run** menu, click **Run As > COBOL Application**.

Execution result

The following is displayed in the COBOL console window. (Console: SAMPLE12)

```
SUCCESSFUL CONNECTION WITH DATABASE.

RECEIVE THE PRODUCT NUMBER WHOSE PRODUCT NAME IS 'TELEVISION'
SET STOCKS OF THE FOLLOWING PROCUCTS DECREASING 10
  TELEVISION -> +0110
  TELEVISION -> +0111
  TELEVISION -> +0212
DELETE THE ROW WHICH HAS PRODUCT NAME IS 'RADIO'.  'SHAVER' OR 'DRIER'.

PROGRAM END
```

The contents of the STOCK table before and after the operation are output in the format shown below to the file assigned to environment variable RESULT.

```
Contents before processing
No.01:
  Product number      = +0110
  Product name        = TELEVISION
  Stock quantity      = +000000085
  Warehouse number    = +0002
                        :
No.19:
  Product number      = +0390
  Product name        = DRIVER
  Stock quantity      = +000000540
  Warehouse number    = +0003
There are 19 data in total.
Contents after processing
No.01:
  Product number      = +0110
  Product name        = TELEVISION
  Stock quantity      = +000000075
  Warehouse number    = +0002
                        :
No.15:
  Product number      = +0351
  Product name        = CASSETTE TAPE
  Stock quantity      = +000002500
  Warehouse number    = +0002
There are 15 data items in total.
```

1.11.2 Using MAKE file

Compiling and Linking the Program

Open a command prompt. Compile and link the sample program using the following command.

```
C:\COBOL\Samples\COBOL\Sample12>nmake
```


Compilation of the sample program is now complete. Verify that SAMPLE12.EXE was created in the same folder in which the sample program is stored.

Setting Runtime Environment Information

Same as "[1.11.1 Using NetCOBOL Studio](#)".

Executing the Program

Execute SAMPLE12.EXE from a command prompt or from Windows Explorer.

Execution result

Same as "[1.11.1 Using NetCOBOL Studio](#)".

1.12 Sample 13: Calling COBOL from Visual Basic

Sample 13 illustrates a COBOL DLL created with NetCOBOL that is called from a Visual Basic(R) application.

This sample program requires the following:

- Microsoft .NET Framework 4 or later

Overview

At initialization, the Visual Basic application calls a subroutine JMPCINT2 that initializes the COBOL runtime environment, ready for a call to a COBOL program.

The Visual Basic form shows a simple equation in which the user enters two numbers on either side of a multiply "*" sign and presses the "=" button. The Visual Basic application passes the two values to the COBOL application, which does the multiplication and returns the result for the Visual Basic code to display.

The Visual Basic application's termination code calls the JMPCINT3 subroutine to close the COBOL runtime environment.

Files Included in Sample 13

- SAMPLE13.COB (COBOL source program)
- VBProj\app.config
- VBProj\AssemblyInfo.vb (Visual Basic Assembly information file)
- VBProj\Sample13.Designer.vb (Visual Basic Designer code file)
- VBProj\Sample13.resX (Visual Basic XML resource file)
- VBProj\sample13.sln (Visual Basic solution file)
- VBProj\Sample13.vb (Visual Basic source code file)
- VBProj\sample13.vbproj (Visual Basic project file)
- MAKEFILE_VB
- MAKEFILE_COBOL
- COBOL85.CBR

Subroutines used in Sample 13

These subroutines are used by Visual Basic to initialize and terminate the COBOL runtime system.

- JMPCINT2
- JMPCINT3

1.12.1 Using NetCOBOL Studio

Compiling and Linking the Visual Basic Program

Open a command prompt. Compile and link the sample program using the following command.

```
C:\COBOL\Samples\COBOL\Sample13>nmake -f MakeFile_VB
```

Compilation of the sample program is now complete. Verify that SAMPLE13.EXE was created.

In the example above, SAMPLE13.EXE is stored as shown below.

```
C:\COBOL\Samples\COBOL\SAMPLE13\VBProj\bin\SAMPLE13.EXE
```

Compiling and Linking the COBOL Program

1. NetCOBOL Studio is started by specifying the workspace created for the sample program. See ["Preparing the workspace."](#)
2. In the Dependency view, select the SAMPLE13 project.

If there is no SAMPLE13 project, import the project for the sample program into the NetCOBOL Studio workspace. See ["Importing the sample program project into the Eclipse IDE workspace."](#)

3. The structure of the project is as follows.

```
SAMPLE13
├ Source Files
│   └ Sample13.cob
├ Linking Files
└ Other Files
    ├── .settings folder
    ├── VBProj folder
    ├── COBOL85.CBR
    ├── Makefile_COBOL
    └ Makefile_VB
```

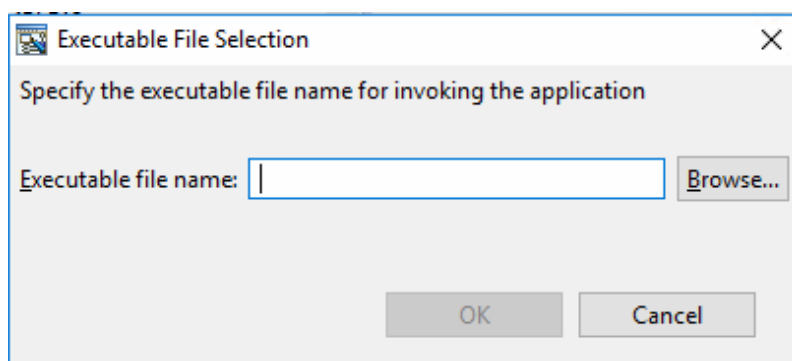
The build is executed immediately after importing the project when an automatic build is set. In this case, the file (. EXE . OBJ etc.) is generated after the build is displayed in "Other Files". It is set to automatic build by default.

4. On the **Project** menu, click **Build Project** when SAMPLE13.DLL is not created in "Other Files" (When an automatic build is not executed).

The project is built, and then SAMPLE13.DLL is created.

Executing the Program

1. Confirm that SAMPLE13.DLL is in a current folder or in the folder set to environment variable PATH.
2. In the Dependency view, select SAMPLE13 project.
3. On the **Run** menu, click **Run As > COBOL Application**. The Executable File Selection dialog appears.

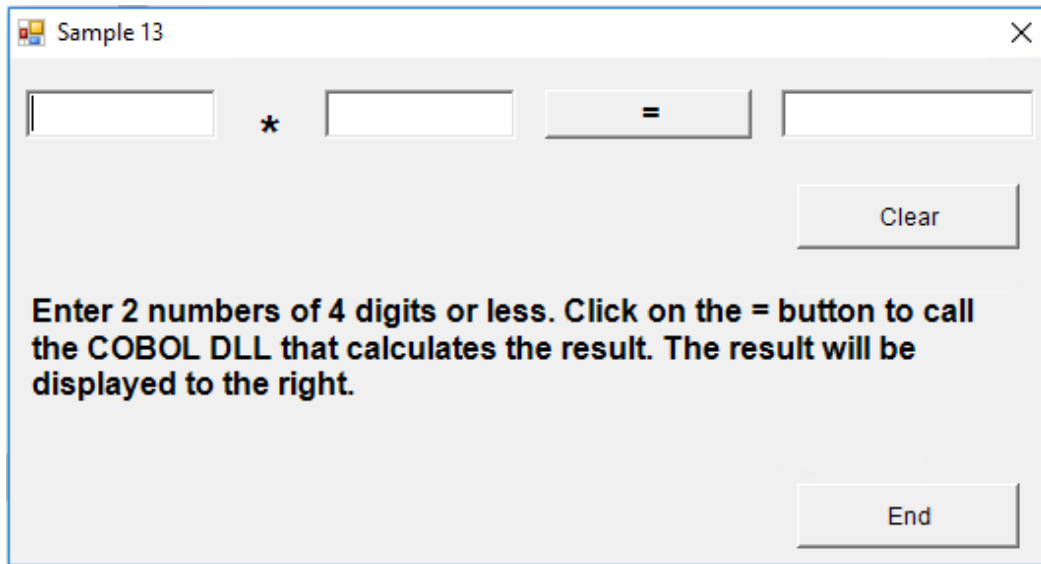


4. Enter the following file as an "Executable file name", and click **OK**.

```
C:\COBOL\Samples\COBOL\SAMPLE13\VBProj\bin\x64\Release\SAMPLE13.EXE
```

Execution result

1. The simple calculator window used by this application is shown in the following figure.



2. To use this form:
Enter a number (up to 4 digits) in each text box to the left of the "=" button.
Press the "=" button.
3. Visual Basic calls COBOL to perform the calculation and format the answer. Visual Basic then displays the answer to the right of the "=" button.

1.12.2 Using MAKE file

Compiling and Linking the Visual Basic Program

Same as "[1.12.1 Using NetCOBOL Studio](#)".

Compiling and Linking the COBOL Program

Open a command prompt. Compile and link the sample program using the following command.

```
C:\COBOL\Samples\COBOL\Sample13>nmake -f Makefile_COBOL
```

Compilation of the sample program is now complete. Verify that SAMPLE13.DLL was created in the same folder in which the sample program is stored.

Executing the Program

Confirm that the SAMPLE13.DLL file is in a current folder or in the folder set to environment variable PATH.

Execute SAMPLE13.EXE from a command prompt or from Windows Explorer.

Execution result

Same as "[1.12.1 Using NetCOBOL Studio](#)".

1.13 Sample 14: Visual Basic calling COBOL -Simple ATM Example

Sample 14 demonstrates Visual Basic calling COBOL by using a simple automatic teller machine (ATM) bank account handling example.

This sample program requires the following:

- Microsoft .NET Framework 4 or later

Overview

This sample program performs the following account functions:

- Opening a new account
- Depositing funds
- Withdrawing funds

The account data, comprising account number, PIN number, name and balance, is saved in an indexed file.

The structure of the indexed file is:

Account number	9(5)	*> (This is the primary record key.)
Password	9(4)	
Name	X(12)	
Deposit	9(9)	

When functions are requested from the "ATM terminal" (user screens), the record data for the account in the indexed file is updated.

Files Included in Sample 14

- VBProj\app.config
- VBProj\AssemblyInfo.vb (Visual Basic Assembly information file)
- VBProj>Error_h.Designer.vb (Visual Basic Designer code file)
Error message box.
- VBProj>Error_h.resX (Visual Basic XML resource file)
Error message box.
- VBProj>Error_h.vb (Visual Basic source code file)
Error message box.
- VBProj\Nyukin.Designer.vb (Visual Basic Designer code file)
Dialog for performing a deposit.
- VBProj\Nyukin.resX (Visual Basic XML resource file)
Dialog for performing a deposit.
- VBProj\Nyukin.vb (Visual Basic source code file)
Dialog for performing a deposit.
- VBProj\Sample14.sln (Visual Basic solution file)
- VBProj\Sample14.vbproj (Visual Basic project file)
- VBProj\Sample14_bas.vb (Visual Basic standard module)
- VBProj\Sample14_frm.Designer.vb (Visual Basic Designer code file)
- VBProj\Sample14_frm.resX (Visual Basic XML resource file)
- VBProj\Sample14_frm.vb (Visual Basic source code file)
- VBProj\Sele.Designer.vb (Visual Basic Designer code file)
Account-handling dialog - shows account number, name and balance, and offers the withdrawal and deposit functions.
- VBProj\Sele.resX (Visual Basic XML resource file)
Account-handling dialog - shows account number, name and balance, and offers the withdrawal and deposit functions.

- VBProj\Sele.vb (Visual Basic source code file)
Account-handling dialog - shows account number, name and balance, and offers the withdrawal and deposit functions.
- VBProj\Sinki.Designer.vb (Visual Basic Designer code file)
Dialog for opening a new account.
- VBProj\Sinki.resX (Visual Basic XML resource file)
Dialog for opening a new account.
- VBProj\Sinki.vb (Visual Basic source code file)
Dialog for opening a new account.
- VBProj\Sinkichk.Designer.vb (Visual Basic Designer code file)
Displays the assigned account number for a new account.
- VBProj\Sinkichk.resX (Visual Basic XML resource file)
Displays the assigned account number for a new account.
- VBProj\Sinkichk.vb (Visual Basic source code file)
Displays the assigned account number for a new account.
- VBProj\Syukin.Designer.vb (Visual Basic Designer code file)
Dialog for performing a withdrawal.
- VBProj\Syukin.resX (Visual Basic XML resource file)
Dialog for performing a withdrawal.
- VBProj\Syukin.vb (Visual Basic source code file)
Dialog for performing a withdrawal.
- K_KEN.COB (COBOL source program)
Retrieves accounts by account number.
- K_SIN.COB (COBOL source program)
Opens a new account.
- K_NYU.COB (COBOL source program)
Adds money deposited to an account.
- K_SYU.COB (COBOL source program)
Subtracts money withdrawn from an account.
- MAKEFILE_VB
MakeFile for Visual Basic program
- MAKEFILE_COBOL
MakeFile for COBOL Program
- COBOL85.CBR

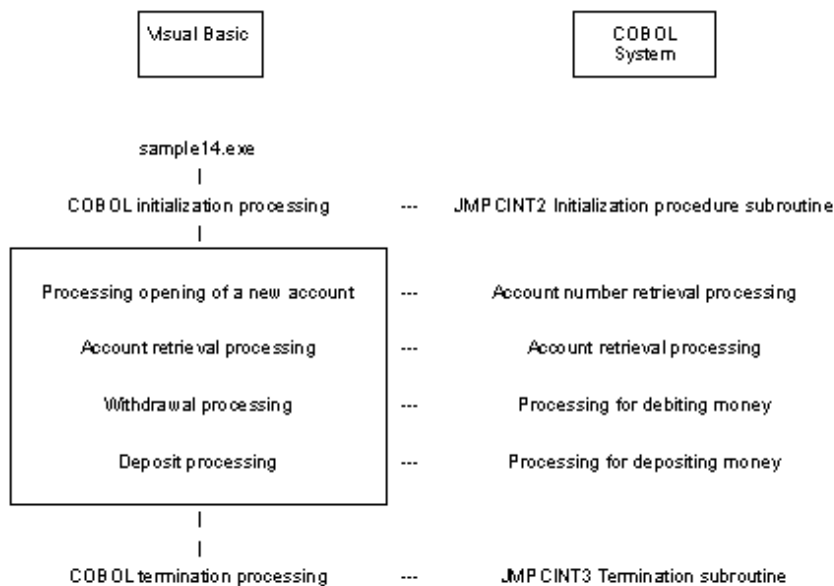
Processing Overview

The Visual Basic application starts, and subroutine JMPCINT2, which initializes the COBOL runtime system, is called when the main form is loaded.

The Visual Basic forms manage the interface with the user - accepting input data, transaction requests and displaying output data and messages. COBOL programs are called to manage the account data in the indexed file.

When the Visual Basic application is closed, it calls subroutine JMPCINT3, which terminates the COBOL runtime.

Figure A.55 shows the structure of the application:



COBOL Statements used in Sample 14

The MOVE, IF, PERFORM, COMPUTE, OPEN, READ, WRITE, REWRITE, CLOSE and EXIT statements are used.

COBOL Runtime System Subroutines

The following routines are used to initialize and terminate the COBOL run-time system.

- JMPCINT2
- JMPCINT3

1.13.1 Using NetCOBOL Studio

Compiling and Linking the Visual Basic Program

Open a command prompt. Compile and link the sample program using the following command.

```
C:\COBOL\Samples\COBOL\Sample14>nmake -f MakeFile_VB
```

Compilation of the sample program is now complete. Verify that SAMPLE14.EXE was created.

In the example above, SAMPLE14.EXE is stored as shown below.

```
C:\COBOL\Samples\COBOL\SAMPLE14\VBProj\bin\x64\Release\SAMPLE14.EXE
```

Compiling and Linking the COBOL Program

1. NetCOBOL Studio is started by specifying the workspace created for the sample program. See ["Preparing the workspace."](#)
2. In the Dependency view, select the SAMPLE14 project.
If there is no SAMPLE14 project, import the project for the sample program into the NetCOBOL Studio workspace. See ["Importing the sample program project into the Eclipse IDE workspace."](#)
3. The structure of the project is as follows.

```
SAMPLE14
├ Source Files
│ └ K_syu.cob
│ └ K_ken.cob
│ └ K_nyu.cob
```

```

|   └─ K_sin.cob
└─ Linking Files
   └─ Other Files
      └─ .settings folder
      └─ VBProj folder
      └─ COBOL85.CBR
      └─ Makefile_COBOL
      └─ Makefile_VB

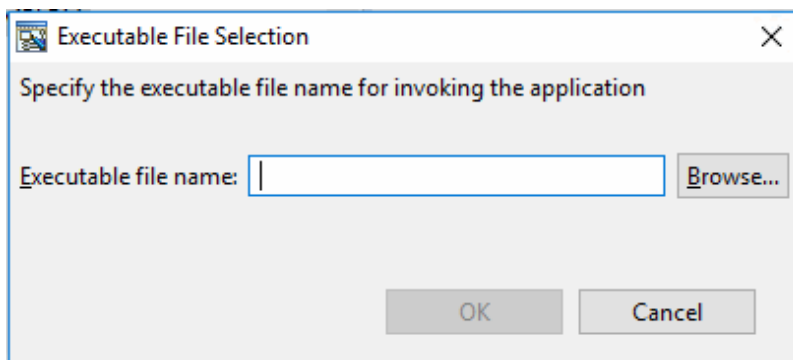
```

The build is executed immediately after importing the project when an automatic build is set. In this case, the file (. EXE . OBJ etc.) is generated after the build is displayed in "Other files". It is set to automatic build by default.

4. On the **Project** menu, click **Build Project** when K_KEN.DLL is not created in "Other Files" (When an automatic build is not executed).
The project is built, and then K_KEN.DLL is created.

Executing the Program

1. Confirm that K_KEN.DLL is in a current folder or in the folder set to environment variable PATH.
2. In the Dependency view, select SAMPLE14 project.
3. On the Run menu, click Run As > COBOL Application. The Executable File Selection dialog appears.

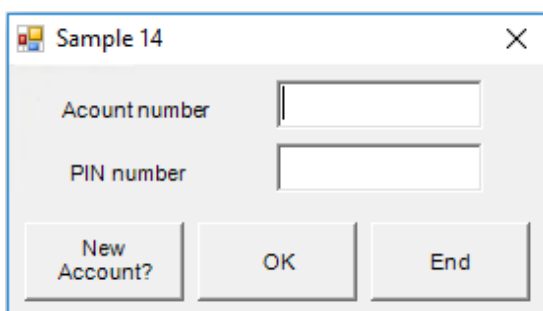


4. Enter the following file as an "Executable file name", and click **OK**.

```
C:\COBOL\Samples\COBOL\SAMPLE14\VBProj\bin\SAMPLE14.EXE
```

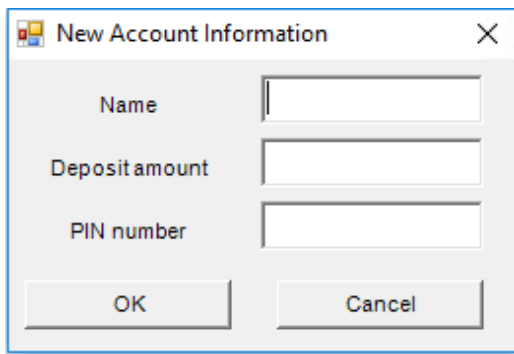
Execution result

The Sample14 dialog box



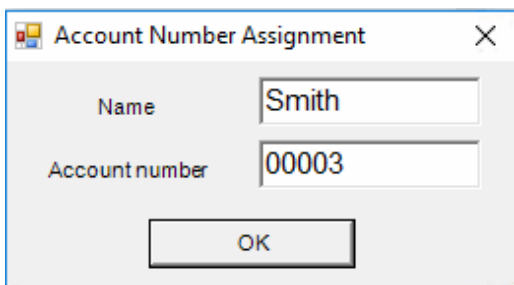
1. Click **New Account?**. The New Account Information dialog appears.
2. Open a new account. Type in a name (such as Smith), an amount (such as 10000) and a PIN number (such as 1234). Click **OK**.
A dialog displays the assigned account number.
3. To terminate the application, click **End**.

The New Account Information dialog box

A screenshot of the 'New Account Information' dialog box. It has a title bar with a close button (X). The dialog contains three text input fields: 'Name', 'Deposit amount', and 'PIN number'. Below these fields are two buttons: 'OK' and 'Cancel'.

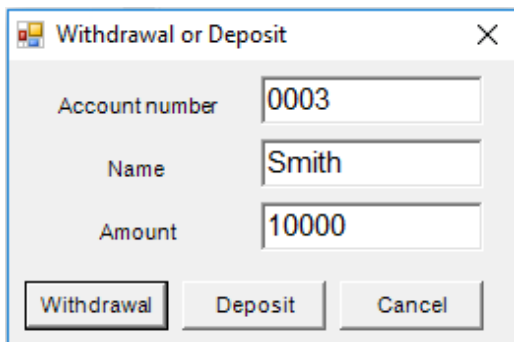
1. Type in a name, an amount and a PIN number. Click **OK**. A new account is made and the account number assignment dialog box is displayed. If an error occurs, an error dialog box is displayed.
2. To cancel new account creation, click **Cancel**.

The Account Number Assignment dialog box

A screenshot of the 'Account Number Assignment' dialog box. It has a title bar with a close button (X). The dialog contains two text input fields: 'Name' (containing 'Smith') and 'Account number' (containing '00003'). Below these fields is a single button: 'OK'.

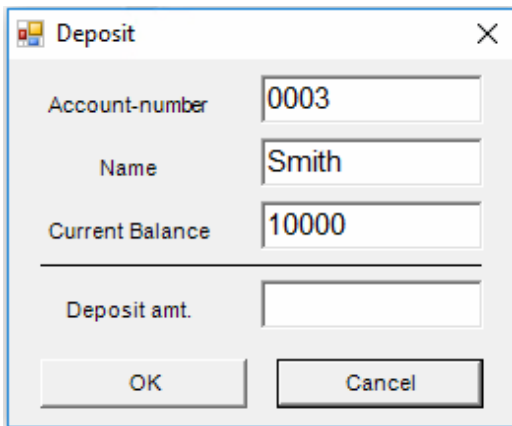
1. Confirm the account number and click **OK**. The Sample14 dialog box is displayed:

The Withdrawal or Deposit dialog box

A screenshot of the 'Withdrawal or Deposit' dialog box. It has a title bar with a close button (X). The dialog contains three text input fields: 'Account number' (containing '0003'), 'Name' (containing 'Smith'), and 'Amount' (containing '10000'). Below these fields are three buttons: 'Withdrawal', 'Deposit', and 'Cancel'.

1. To withdraw, click **Withdrawal**.
2. To deposit, click **Deposit**.
3. To interrupt the application, click **Cancel**.

The Deposit dialog box

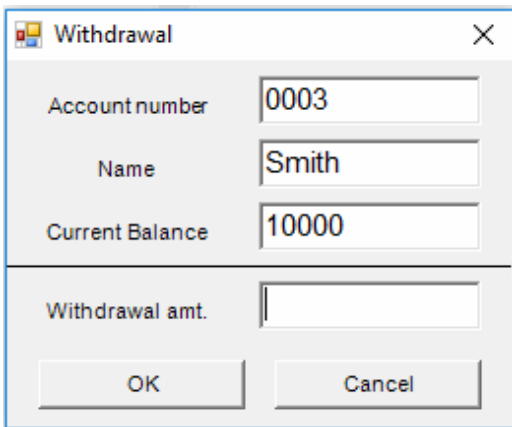
A Windows-style dialog box titled "Deposit" with a close button (X) in the top right corner. It contains three input fields: "Account-number" with the value "0003", "Name" with the value "Smith", and "Current Balance" with the value "10000". Below these fields is a horizontal line, followed by an empty "Deposit amt." input field. At the bottom are two buttons: "OK" and "Cancel".

1. Type in a deposit amount and click **OK**. The Withdrawal or Deposit dialog box appears.

If an error occurs, an **ERROR** message box is displayed.

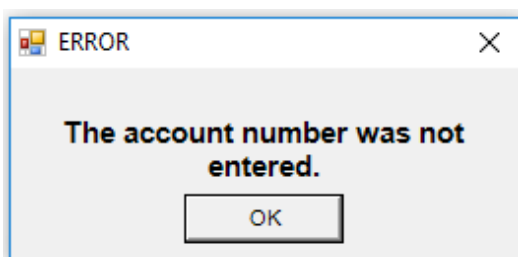
1. To interrupt the application, click the **Cancel**.

The Withdrawal dialog box

A Windows-style dialog box titled "Withdrawal" with a close button (X) in the top right corner. It contains three input fields: "Account number" with the value "0003", "Name" with the value "Smith", and "Current Balance" with the value "10000". Below these fields is a horizontal line, followed by an empty "Withdrawal amt." input field. At the bottom are two buttons: "OK" and "Cancel".

1. Type in a withdrawal amount and click **OK**. The Withdrawal or Deposit dialog box appears. If an error occurs, an **ERROR** message box is displayed.
2. To interrupt the application, click **Cancel**.

The ERROR window

A Windows-style error message box titled "ERROR" with a close button (X) in the top right corner. The main text reads "The account number was not entered." in bold. Below the text is a single button labeled "OK".

1. To confirm the error message, click **OK**.

1.13.2 Using MAKE file

Compiling and Linking the Program

Open a command prompt. Compile and link the sample program using the following command.

```
C:\COBOL\Samples\COBOL\Sample14>nmake -f MakeFile_COBOL
```

Compilation of the sample program is now complete. Verify that K_KEN.DLL and SAMPLE14.EXE were created in the same folder in which the sample program is stored.

Executing the Program

Confirm that K_KEN.DLL is in a current folder or in the folder set to environment variable PATH.

Execute SAMPLE14.EXE from a command prompt or from Windows Explorer.

Execution result

Same as "[1.13.1 Using NetCOBOL Studio](#)".

1.14 Sample 15: Basic Object-Oriented Programming

This program illustrates basic object-oriented programming functions including encapsulation, object generation and method invocation.

Overview

In the sample program, three employee objects are generated. After an object has been generated using the "NEW" method, the "Data-Set" method is invoked to set the data.

Although all of the employee objects have the same form, they have different data (employee numbers, names, departments and sections, and address information). Address information containing postal codes and addresses also belongs to an object.

Upon input of an employee number on the screen, the appropriate "Data-Display" method in the employee object is invoked, and the employee information in the object is displayed.

The employee object invokes the "Data-Get" method of the associated address object to acquire the address information.

The employee object consists of three pieces of data and an object reference to an address object. The structure of the object is transparent to the main program user. However, the user must understand the "Data-Set" and "Data-Display" methods.

The encapsulation of data completely masks the information in the object.

Files Included in Sample 15

- MAIN.COB (COBOL source program)
- MEMBER.COB (COBOL source program)
- ADDRESS.COB (COBOL source program)
- MAKEFILE
- COBOL85.CBR

COBOL Functions used in Sample 15

- Object-oriented programming function
 - Class definition (Encapsulation)
 - Object generation
 - Method invocation
- Project management

Object-Oriented Syntax used in Sample 15

- INVOKE and SET statements

- REPOSITORY paragraph
- Class, object and method definitions

1.14.1 Using NetCOBOL Studio

Compiling and Linking the Program

1. NetCOBOL Studio is started by specifying the workspace created for the sample program. See ["Preparing the workspace."](#)
2. In the Dependency view, select the SAMPLE15 project.

If there is no SAMPLE15 project, import the project for the sample program into the NetCOBOL Studio workspace. See ["Importing the sample program project into the Eclipse IDE workspace."](#)

3. The structure of the project is as follows.

```
SAMPLE15
├ Source Files
│   ├── Address.cob
│   ├── Main.cob
│   └ Member.cob
├ Linking Files
└ Other Files
    ├── .settings folder
    ├── COBOL85.CBR
    └ Makefile
```

The build is executed immediately after importing the project when an automatic build is set. In this case, the file (. EXE . OBJ etc.) is generated after the build is displayed in "Other Files". It is set to automatic build by default.

4. On the **Project** menu, click **Build Project** when MAIN.EXE is not created in "Other Files" (When an automatic build is not executed).
The project is built, and then MAIN.EXE is created.

Executing the Program

1. In the Dependency view, select SAMPLE15 project.
2. On the **Run** menu, click **Run As > COBOL Application**. The Executable File Selection dialog appears.

Execution result

Sample 15 requires no special execution environment information to be set.

```
Please enter the employee number(1 or 2 or 3)
```

The interface is very basic - simply enter an employee number 1, 2 or 3 to display details for that employee. After the details are displayed, enter N to terminate or Y to continue.

```
Please enter the employee number(1 or 2 or 3)
1
NO.---NAME-----BELONGING-----POST---ADDR-----
0001 James Smith           Language group      411-0007 2929 Park Avenue, New York, N
Y
Do you to end?(Y/N)
```

1.14.2 Using MAKE file

Compiling and Linking the Program

Open a command prompt. Compile and link the sample program using the following command.

```
C:\COBOL\Samples\COBOL\Sample15>nmake
```

Compilation of the sample program is now complete. Verify that MAIN.EXE was created in the same folder in which the sample program is stored.

Executing the Program

Execute MAIN.EXE from a command prompt or from Windows Explorer.

Execution result

Same as "[1.14.1 Using NetCOBOL Studio](#)".

1.15 Sample 16: Collection Class (Class Library)

Sample 16 demonstrates the use of a collection class for creating a class library.

This sample can be used to create a class library in an actual program.

This sample covers only the basic operation. An easy-to-use class library can be created by modifying and changing this sample.

Overview

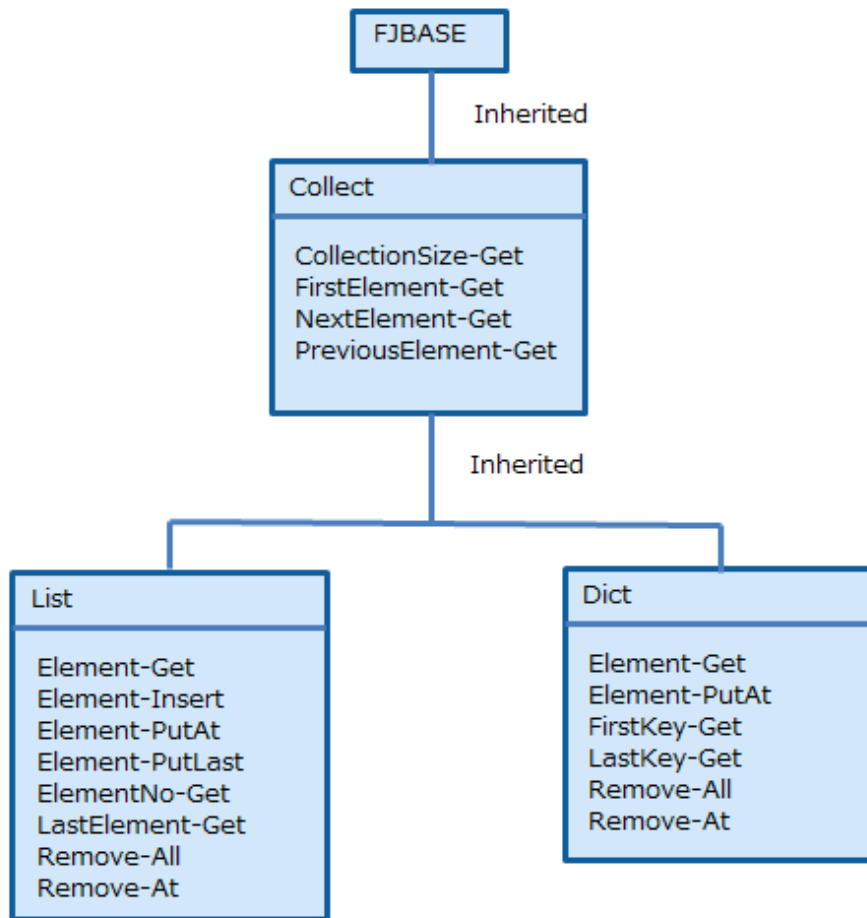
A collection class is the generic name of a class that handles a set of objects — it is used to collectively handle and store many objects.

This sample covers the following classes:

- Collect (Collection)
- Dict (Dictionary)
- List (List)

Class Layers

The following diagram shows the relationships between the class layers in Sample 16.



Note

In addition to the above classes, Sample 16 also includes the classes BinaryTree-Class, DictionaryNode-Class and ListNode-Class. These classes, which are used inside the List and Dict classes, are transparent to the collection class user, and are not explained here.

Collect Class

This is the highest collection class. All collection classes inherit this class.

Collect is an abstract class, and does not create any objects.

Since this class inherits the FJBASE class, all the methods defined in the FJBASE class can be used.

Definitions

```

CLASS-ID. Collect INHERITS FJBASE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
    CLASS FJBASE.
OBJECT.
PROCEDURE DIVISION.
METHOD-ID. CollectionSize-Get.
METHOD-ID. FirstElement-Get.
METHOD-ID. NextElement-Get.
METHOD-ID. PreviousElement-Get.
END OBJECT.
END CLASS Collect.
  
```

CollectionSize-Get method

This method ascertains the number of elements in a set.

Parameter

None

Return value

PIC 9(8) BINARY

Returns the number of elements in a set.

FirstElement-Get method

This method returns the first element in a set.

Parameter

None

Return value

USAGE OBJECT REFERENCE

Returns the first element in a set. If no element exists, NULL is returned.

NextElement-Get method

This method returns the element following the one currently pointed to.

Parameter

None

Return value

USAGE OBJECT REFERENCE

Returns the element following the one currently pointed to. If no following element exists, NULL is returned.

PreviousElement-Get method

This method returns the element immediately preceding the one currently pointed to.

Parameter

None

Return value

USAGE OBJECT REFERENCE

Returns the element immediately preceding the one currently pointed to. If no preceding element exists, NULL is returned.

Dict Class

This class has the following features:

- Each element has a key.
- The key value is unique.
- A key can be used for retrieval.
- The key is used for ordering.

Since this class inherits from the Collect class, all the methods defined in Collect can be used as well.

Definitions

CLASS-ID. Dict INHERITS Collect.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.

```

REPOSITORY.
  CLASS Collect.
  OBJECT.
  PROCEDURE DIVISION.
  METHOD-ID. Element-Get.
  METHOD-ID. Element-PutAt.
  METHOD-ID. FirstKey-Get.
  METHOD-ID. LastKey-Get.
  METHOD-ID. Remove-All.
  METHOD-ID. Remove-At.
  END OBJECT.
END CLASS Dict.

```

Element-Get method

This method returns elements for a specified key.

Parameter

Key:

PIC X(10)

Specifies a key value for an element to be returned.

Return value

USAGE OBJECT REFERENCE

Returns an element for a specified key if it is found, and returns NULL if it is not found.

Element-PutAt method

This method adds an element for a specified key. If an element with the same key already exists, it is replaced by the new element.

Parameters

Key:

PIC X(10)

Specifies the key value of the element to be added or replaced.

Element:

USAGE OBJECT REFERENCE

Specifies the element to be added or replaced.

Return value

None

FirstKey-Get method

This method determines the key value for the first element.

Parameter

None

Return value

PIC X(10)

Returns the key value for the first element. If the number of elements is 0, or if the key for the first element is a blank, a blank is returned.

LastKey-Get method

This method determines the key value for the last element.

Parameter

None

Return value

PIC X(10)

Returns the key value for the last element. If the number of elements is 0, or if the key for the last element is a blank, a blank is returned.

Remove-All method

This method deletes all elements contained in a set.

Parameter

None

Return value

None

Remove-At method

This method deletes an element for a specified key.

Parameter

Key:

PIC X(10)

Specifies the key value for the element to be deleted.

Return value

None

List Class

This class has the following features:

- Elements are arranged in a certain order.
- Allows duplicate elements.

Since this class inherits from the Collect class, all of the methods defined in the Collect class can be used as well.

Definitions

```
CLASS-ID. List INHERITS Collect.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
    CLASS Collect.
OBJECT.
PROCEDURE DIVISION.
METHOD-ID. Element-Get.
METHOD-ID. Element-Insert.
METHOD-ID. Element-PutAt.
METHOD-ID. Element-PutLast.
METHOD-ID. ElementNo-Get.
METHOD-ID. LastElement-Get.
METHOD-ID. Remove-All.
METHOD-ID. Remove-At.
END OBJECT.
END CLASS List.
```

Element-Get method

This method returns the element at a specified location (index).

Parameter

Index:

PIC 9(8) BINARY

Specifies the location of the element to be returned by an integer starting at 1.

Return value

USAGE OBJECT REFERENCE

Returns the specified element. If no element exists at the specified location, NULL is returned.

Element-Insert method

This method adds an element at the specified location (index).

Parameters

Index:

PIC 9(8) BINARY

Specifies the location at which the element is to be added by an integer beginning with 1.

If a value greater than the number of elements plus 1 is specified, no element is added.

Element:

USAGE OBJECT REFERENCE

Specifies the element to be added.

Return value

PIC 9(8) BINARY

Returns the location at which the element was added by an integer beginning with 1. If no element is added, 0 is returned.

Element-PutAt method

This method replaces the element at the specified location (index).

Parameters

Index:

PIC 9(8) BINARY

Specifies the location of the element to be replaced by an integer beginning with 1. If a value greater than the number of elements is specified, no element is replaced.

Element:

USAGE OBJECT REFERENCE

Specifies the element to be replaced.

Return value

Returns the location of the replaced element using an integer beginning with 1.

If no element has been replaced, 0 is returned.

Element-PutLast method

This method adds an element after the last element.

Parameter

Element:

Specifies the element to be added.

Return value

None

ElementNo-Get method

This method checks the location (index) of a specified element.

Parameter

Element:

Specifies the element whose location is checked.

Return value

PIC 9(8) BINARY

Returns the location of the element using an integer beginning with 1.

If the specified element is not found, 0 is returned.

If duplicate elements exist, the first found location is returned.

LastElement-Get method

This method returns the last element.

Parameter

None

Return value

USAGE OBJECT REFERENCE

Returns the last element. If the number of elements is 0, NULL is returned.

Remove-All method

This method deletes all the elements contained in a set.

Parameter

None

Return value

None

Remove-At method

This method deletes the element at the specified location (index).

Parameter

Index:

PIC 9(8) BINARY

Specifies the location of the element to be deleted using an integer starting at 1. If a value greater than the number of elements is specified, no element is deleted.

Return value

Returns the location of the deleted element using an integer beginning with 1. If no element has been deleted, 0 is returned.

Programs and Files in Sample 16

- COLLECT.COB (COBOL source program)
- DICT.COB (COBOL source program)
- LIST.COB (COBOL source program)
- BIN_TREE.COB (COBOL source program)

- D_NODE.COB (COBOL source program)
- L_NODE.COB (COBOL source program)
- MAKEFILE
- COBOL85.CBR

COBOL Functions Used in Sample 16

- Object-oriented programming functions
 - Class definition (Encapsulation)
 - Inheritance
 - Object creation
 - Method calling

Object-Oriented Syntax used in Sample 16

- INVOKE and SET statements
- Object properties
- Method calling
- REPOSITORY paragraphs
- Class , object and method definitions

1.15.1 Using NetCOBOL Studio

Compiling and Linking the Program

1. NetCOBOL Studio is started by specifying the workspace made for the sample program. See "[Preparing the workspace.](#)"
2. In the Dependency view, select the SAMPLE16 project.

If there is no SAMPLE16 project, import the project for the sample program into the NetCOBOL Studio workspace. See "[Importing the sample program project into the Eclipse IDE workspace.](#)"

3. The structure of the project is as follows.

```
SAMPLE16
├ Source Files
│   ├── Collect.cob
│   ├── D_Node.cob
│   ├── Dict.cob
│   ├── L_Node.cob
│   ├── List.cob
│   └ Bin_Tree.cob
├ Linking Files
└ Other Files
    ├── .settings folder
    ├── COBOL85.CBR
    └ Makefile
```

The build is executed immediately after importing the project when an automatic build is set. In this case, the file (. EXE . OBJ etc.) is generated after the build is displayed in "Other Files". It is set to automatic build by default.

4. When compiling terminates, the following files are created.

On the **Project** menu, click **Build Project** when the following files are not created in "Other Files" (When an automatic build is not executed).

- COLLECT.DLL
- COLLECT.LIB
- COLLECT.REP
- DICT.REP
- LIST.REP



Note

Some other files are also created, but they are not required when the class library is used.

Using the Class Library

When the sample class library is used in the program, the following files are required:

For Compiling or Linking

- COLLECT.LIB (Import library)
- COLLECT.REP (Repository library)
- DICT.REP (Repository file)
- LIST.REP (Repository file)

Install the above files to be used into a project that uses the class library.

For Executing

- COLLECT.DLL (DLL file)

1.15.2 Using MAKE file

Compiling and Linking the Program

Open a command prompt. Compile and link the sample program using the following command.

```
C:\COBOL\Samples\COBOL\Sample16>nmake
```

When compiling terminates, the following files are created.

- COLLECT.DLL
- COLLECT.LIB
- COLLECT.REP
- DICT.REP
- LIST.REP



Note

Some other files are also created, but they are not required when the class library is used.

Using the Class Library

When the sample class library to be used is installed in a program, the following files are required:

For Compiling or Linking

- COLLECT.LIB (Import library)
- COLLECT.REP (Repository library)
- DICT.REP (Repository file)
- LIST.REP (Repository file)

Install the above files to be used into a project that uses the class library.

For Executing

- COLLECT.DLL (DLL file)

1.16 Sample 31: Windows System Function Call

Sample 31 demonstrates how to invoke a Windows system function - for this example, a call to create a message box.

Overview

Sample 31 calls the Windows system function "MessageBoxA" to display a message in a message box with YES, NO and Cancel buttons. (Note that an "A" needs to be appended to the function call when the function call contains a character string parameter and you are working in ASCII, as opposed to Unicode data, where the suffix is "W".)

The message box returns a value indicating which button was pressed. This value is returned in the data item specified in the RETURNING phrase.

In a batch file, this return value can be accessed by via ERRORLEVEL, as demonstrated in SAMPLE31.BAT.

```
@echo off
    set msg=Return the value depending on return code from the MessageBox.
:START
    echo %msg%
    set msg=Selected the "Cancel", Restart again.
    start /w MsgBox.exe
    @rem If return code is over 9999 then call the COBOL program again.
    if errorlevel 9999 goto START
    @rem If return code is over 9 then selected the "No".
    if errorlevel 9 goto NG
    echo Selected the "Yes".
    goto END
:NG
    echo Selected the "No".
:END
set msg=
```

Files Included in Sample 31

- MSGBOX.COB (COBOL source program)
- MAKEFILE
- SAMPLE31.BAT (Batch file for start)
- COBOL85.CBR

COBOL Statements Used

- Method of calling C program from COBOL program
- Parameter transfer BY VALUE
- RETURNING phrase of CALL statement

- Special register PROGRAM-STATUS (RETURN-CODE)



- Most Windows system functions (and C routines in general) require that strings be terminated with a null byte (X"00" or LOW-VALUE). This sample shows how you can place these bytes using reference modification.
- The Windows system function names are case sensitive, so be sure to get the case correct, as in "MessageBoxA". Specify compiler option "NOALPHAL" or "ALPHAL(WORD)" to ensure that the COBOL system uses mixed case for the function name.

1.16.1 Using NetCOBOL Studio

Compiling and Linking the Program

1. NetCOBOL Studio is started by specifying the workspace created for the sample program. See "[Preparing the workspace.](#)"
2. In the Dependency view, select the SAMPLE31 project.

If there is no SAMPLE31 project, import the project for the sample program into the NetCOBOL Studio workspace. See "[Importing the sample program project into the Eclipse IDE workspace.](#)"

3. The structure of the project is as follows.

```
SAMPLE31
├ Source Files
│   └ Msgbox.cob
├ Linking Files
└ Other Files
    ├── .settings folder
    ├── COBOL85.CBR
    ├── Makefile
    └ Sample31.bat
```

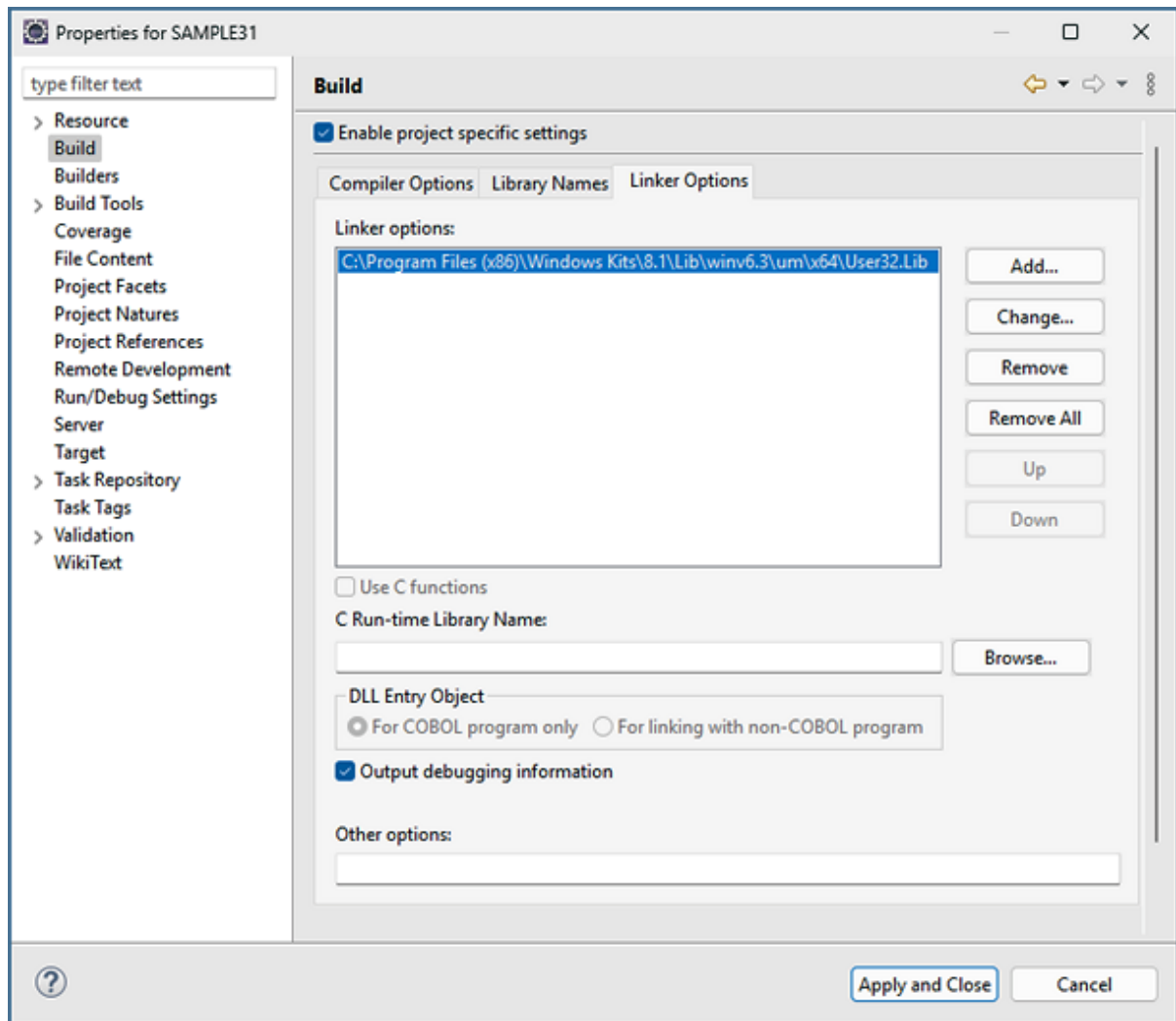
4. The build is executed immediately after importing the project when an automatic build is set. In this case, the file (. EXE . OBJ etc.) is generated after the build is displayed in "Other Files". It is set to automatic build by default.

The properties dialog box is displayed.

5. Link USER32.LIB to use "MessageBoxA" of the Windows system function in this exercise.

To confirm linked libraries, select the SAMPLE31 project from the Dependency view, and then select **Property...** from the context menu. The "Properties for SAMPLE31" dialog box is displayed.

- In the left pane, select **Build**. The **Build** page appears. And, select the **Linker Options** tab.



The storage place of USER32.LIB is set as follows. Please change the setting according to the installation environment of Windows SDK.

```
{NetCOBOL}\User32.Lib
```

- On the **Project** menu, click **Build Project** when MSGBOX.EXE is not created in "Other Files" (When an automatic build is not executed).

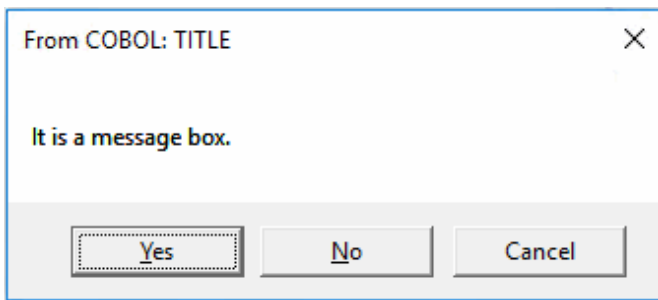
The project is built, and then MSGBOX.EXE is created

Executing the Program

- Open a command prompt, change directories to the SAMPLE31 folder, and execute SAMPLE31.BAT.

```
c:\NetCOBOL Studio\workspace\SAMPLE31>sample31.bat
Return the value depending on return code from the MessageBox
```

2. The following message boxes are displayed. Click one of the buttons.



3. The COBOL program detects which button was pressed and indicates such by displaying a message.

```
c:\NetCOBOL Studio\workspace\SAMPLE31>sample31.bat
Return the value depending on return code from the MessageBox
Selected the "Yes".

c:\NetCOBOL Studio\workspace\SAMPLE31>
```

When the Cancel button is clicked, the program is executed again.

1.16.2 Using MAKE file

Compiling and Linking the Program

Open a command prompt. Compile and link the sample program using the following command.

```
C:\COBOL\Samples\COBOL\Sample31>nmake
```

Compilation of the sample program is now complete. Verify that MSGBOX.EXE was created in the same folder in which the sample program is stored.

Select "Build Project" on "Project" menu bar of NetCOBOL Studio when MSGBOX.EXE is not created in "Other Files" (When an automatic build is not executed).

The project is built, and then MSGBOX.EXE is created.

Executing the Program

Same as "[1.16.1 Using NetCOBOL Studio](#)".

1.17 Sample 32: Starting Another Program

Sample 32 demonstrates a program that starts another program, waits for the started program to terminate, and receives a completion code from the started program via inter-program communications.

Overview

When SAMPLE32 is executed, you are prompted for a program to run. If you enter nothing, the program MSGBOX.EXE from the SAMPLE31 folder is executed. If you specify a program name, you must enter the fully qualified path to the location of the application (or batch file) to execute.

The Windows system function "CreateProcessA" is called specifying this for the argument, and specified program or batch file is started.

If the specified program is successfully started, SAMPLE32 then waits until the specified program terminates and then receives the completion code from the started program.

Files Included in Sample 32

- SAMPLE32.COB(COBOL source program)
- MAKEFILE

- COBOL85.CBR

COBOL Statements Used

- Method of calling a C program from COBOL program
- STDCALL call convention
- Parameter transfer in BY VALUE
- RETURNING phrase of CALL statement
- STORED-CHAR-LENGTH function
- Project Manager

Before Executing the Application

- SAMPLE32 uses MSGBOX.EXE from SAMPLE 31; therefore, please build the MSGBOX application prior to executing SAMPLE32.
- In the following screens, SAMPLE06 is also executed; therefore please build SAMPLE06 prior to executing SAMPLE32.

1.17.1 Using NetCOBOL Studio

Compiling and Linking the Program

1. NetCOBOL Studio is started by specifying the workspace created for the sample program. See ["Preparing the workspace."](#)
2. In the Dependency view, select the SAMPLE32 project.

If there is no SAMPLE32 project, import project for the sample program into the NetCOBOL Studio workspace. See ["Importing the sample program project into the Eclipse IDE workspace."](#)

3. The structure of the project is as follows.

```
SAMPLE32
├ Source Files
│   └ Sample32.cob
├ Linking Files
└ Other Files
    ├── .settings folder
    ├── COBOL85.CBR
    └ Makefile
```

4. The build is executed immediately after importing the project when an automatic build is set. In this case, the file (. EXE . OBJ etc.) is generated after the build is displayed in "Other Files". It is set to automatic build by default.
5. On the **Project** menu, click **Build Project** when SAMPLE32.EXE is not created in "Other Files" (When an automatic build is not executed).

The project is built, and then SAMPLE32.EXE is created.

Executing the Program

1. Execute SAMPLE32.EXE from a command prompt or from Windows Explorer. The following is displayed, waiting for input.

```
Input the path name that execution program.
(If input no character, then execute the MsgBox.EXE of SAMPLE31)
=>
```

2. Input the path and filename of an executable program or batch file. The environment variable PATH is not referenced here; therefore it is necessary to specify a relative path from the SAMPLE32 folder or a fully qualified path name.
3. If nothing is entered, and then MSGBOX.EXE of SAMPLE31 is executed. Press the ENTER key.

4. The completion code of MSGBOX.EXE of SAMPLE31 is displayed, indicating the button that was clicked. In the following screen, the "No" button was clicked.

```
Input the path name that execution program.
(If input no character, then execute the MsgBox.EXE of SAMPLE31)
=>
Execute the program ..\SAMPLE31\MSGBOX.EXE
Succeeded in executing program ..\SAMPLE31\MSGBOX.EXE
Return code from ..\SAMPLE31\MSGBOX.EXE is '00000009'.
```

5. If SAMPLE32 is re-executed and an executable program or batch file name is specified, you are then prompted to enter command line arguments (if any) for the EXE or BAT file, as shown below.

```
Input the path name that execution program.
(If input no character, then execute the MsgBox.EXE of SAMPLE31)
=>..\SAMPLE06\SAMPLE6.EXE
Input the command line arguments.
=>
```

6. SAMPLE06.EXE requires two command line arguments to be specified following the program name.

```
Input the path name that execution program.
(If input no character, then execute the MsgBox.EXE of SAMPLE31)
=>..\SAMPLE06\SAMPLE6.EXE
Input the command line arguments.
=>20000101 20140101
```

7. A message indicating that SAMPLE06 has been started is displayed. The system console is opened and the execution result of SAMPLE06 is output. The completion code of SAMPLE06.EXE is displayed and execution ends.

```
Input the path name that execution program.
(If input no character, then execute the MsgBox.EXE of SAMPLE31)
=>..\SAMPLE06\SAMPLE6.EXE
Input the command line arguments.
=>20000101 20140101
Execute the program ..\SAMPLE06\SAMPLE6.EXE
Succeeded in executing program ..\SAMPLE06\SAMPLE6.EXE
Return code from ..\SAMPLE06\SAMPLE6.EXE is '00000000'.
```

1.17.2 Using MAKE file

Compiling and Linking the Program

Open a command prompt. Compile and link the sample program using the following command.

```
C:\COBOL\Samples\COBOL\Sample32>nmake
```

Compilation of the sample program is now complete. Verify that SAMPLE32.EXE was created in the same folder in which the sample program is stored.

Executing the Program

Same as "1.17.1 Using NetCOBOL Studio".

Index

[Special characters]	
@MGPRM.....	11
[A]	
ACCEPT.....	4,8
ACCEPT FROM.....	23
argument.....	20
[B]	
BY VALUE.....	64
[C]	
CALL.....	11
calling.....	64
CHARACTER TYPE.....	28
class library.....	55
CLOSE.....	6,8
collection class.....	55
command line argument.....	20
COMPUTE.....	20
CONNECT.....	37
[D]	
database access.....	36
DECLARE CURSOR.....	37
DISCONNECT.....	37
DISPLAY.....	4
DISPLAY UPON.....	23
DIVIDE.....	20
[E]	
encapsulation.....	53,62
environment variables.....	23
[F]	
FCB.....	28
FETCH.....	37
Forms Control Buffers.....	28
FOVLDIR.....	35
free format code.....	11
[I]	
I control record.....	28
indexed files.....	5
inheritance.....	62
INVOKE.....	53,62
[J]	
JMPCINT2.....	44
JMPCINT3.....	44
[L]	
line sequential files.....	5
[M]	
message box.....	64
[O]	
object.....	62
object-oriented programming.....	53,55
object creation.....	62
objects generation.....	53
ODBC.....	36
OO COBOL.....	53
OPEN.....	6
[P]	
parameter.....	20
printing.....	26,28
PRINTING MODE.....	28
PRINTING POSITION.....	28
[R]	
READ.....	6
REPOSITORY.....	54,62
RETURNING.....	64
ROLLBACK.....	37
[S]	
Screen Section.....	8
SET.....	53
SQL.....	36,37
STRING.....	24
subprogram.....	11
[V]	
Visual Basic.....	47
[W]	
Windows function calls.....	64
WRITE.....	6