FUJITSU Software
NetCOBOL V11.0

Getting Started

Windows

B1WD-3354-01ENZ0(00)
August 2015

Preface

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.

How This Manual is Organized

This manual contains the following information:

Chapter 1	A Quick Tour
Chapter 2	Developing GUI Applications
Chapter 3	Working with Visual Basic and COBOL
Chapter 4	Using SQL with COBOL
Appendix A	Sample Programs
Appendix B	Tips

Note:

This manual contains product descriptions of some products that have been deprecated. Please refer to the product manual for additional information on product information and usage.

Conventions Used in This Manual

Example of convention	Description
setup	Characters you enter appear in bold.
Program-name	Underlined text indicates a placeholder for information you supply.
ENTER	Small capital letters are used for the name of keys and key sequences such as ENTER and CTRL+R. A plus sign (+) indicates a combination of keys.
…	Ellipses indicate the item immediately preceding can be specified repeatedly.
Edit, Literal	Names of menus and options appear with the initial letter capitalized.
[def]	Indicates that the enclosed item may be omitted.
{ABC|DEF}	Indicates that one of the enclosed items delimited by | is to be selected.
CHECK PARAGRAPH-ID COBOL ALL	Commands, statements, clauses, and options you enter or select appear in uppercase. Program section names, and some proper names also appear in uppercase. Underlined text indicates the default.
DATA DIVISION.	This font is used for examples of program code.

WORKING-STORAGE SECTION. * Get the #INCLUDE file to the data control #INCLUDE "NUMDATA.COB".	
The *form* acts as an application creation window.	Italics are occasionally used for emphasis.
“PowerCOBOL User’s Guide” See Chapter 6, “Creating an Executable Program.”	References to other publications or chapters within publications are in quotation marks.

Related Manuals

- NetCOBOL User’s Guide
- NetCOBOL Language Reference
- NetCOBOL Syntax Samples
- NetCOBOL Debugging Guide
- NetCOBOL CBL Subroutines User’s Guide
- NetCOBOL CGI Subroutines User’s Guide
- NetCOBOL File Access Subroutines User's Guide
- NetCOBOL Getting Started with COM Components
- NetCOBOL ISAPI Subroutines User’s Guide
- NetCOBOL Web Development Tools
- NetCOBOL Web Guide
- NetCOBOL J Adapter Class Generator User's Guide
- PowerBSORT Getting Started
- PowerBSORT Reference
- PowerCOBOL Getting Started
- PowerCOBOL Reference
- PowerCOBOL User’s Guide
- PowerFORM Getting Started
- PowerFORM Runtime Reference

Trademarks

- COBOL/2 is a registered trademark of Micro Focus International Ltd.
- Windows, Windows Server, and Visual Studio are either trademarks or registered trademarks of Microsoft Corporation in the U.S.A. and/or other countries.
- Oracle is a registered trademark of Oracle Corporation.
- NetCOBOL is a trademark or registered trademark of Fujitsu Limited or its subsidiaries in the United States or other countries or in both.
- Adobe, Acrobat, Acrobat Reader, and Acrobat logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

- Other product names are trademarks or registered trademarks of each company.
- Trademark indications are omitted for some system and product names described in this manual.

Security

Since there are Internet-enabled functions in the development environment, it is recommended that you use the NetCOBOL development environment only in an Intranet, and not in the more open Internet environment.

To ensure security when working in an environment that is connected to the Internet, it is important to correctly set up both the applications created with NetCOBOL and their operating environment.

To safeguard resources (such as databases, and input and output files), and definition and information files required for the operation of programs from illegal access and tampering, you need to restrict access to the resources by OS functions and programs. In particular, keep important resources in an intranet environment in which a firewall has been installed.

Although this product offers different communication functions (such as the simple communication interface facility, and the Web subroutines), only the Web subroutines have been designed for use with Internet services. Therefore, only use these other functions in environments that are not connected to the Internet, or in intranet environments in which firewalls have been installed and which have been constructed to prevent security breaches.

If you are using the Web subroutines with NetCOBOL on a Web server, use the Web server authorization apparatus and encryption communication function (SSL) to prevent illegal access or information being leaked or tampered with. Additionally, use the Web server access log to investigate and pursue any incidents of illegal access. For details, refer to the documentation for the Web server you are using.

You need to test applications created with NetCOBOL to ensure that even if malicious or careless data values are provided as input, no important data can be destroyed or sensitive information obtained.

High Risk Activity

The Customer acknowledges and agrees that the Product is designed, developed and manufactured as contemplated for general use, including without limitation, general office use, personal use, household use, and ordinary industrial use, but is not designed, developed and manufactured as contemplated for use accompanying fatal risks or dangers that, unless extremely high safety is secured, could lead directly to death, personal injury, severe physical damage or other loss (hereinafter "High Safety Required Use"), including without limitation, nuclear reaction control in nuclear facility, aircraft flight control, air traffic control, mass transport control, medical life support system, missile launch control in weapon system.

The Customer shall not use the Product without securing the sufficient safety required for the High Safety Required Use. In addition, Fujitsu (or other affiliate's name) shall not be liable against the Customer and/or any third party for any claims or damages arising in connection with the High Safety Required Use of the Product.

Export Regulation

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

Product names

The names of products described in this manual are abbreviated as follows:

Product Name	Abbreviation
Microsoft® Windows Server® 2012 R2 Datacenter Microsoft® Windows Server® 2012 R2 Standard Microsoft® Windows Server® 2012 R2 Essentials Microsoft® Windows Server® 2012 R2 Foundation	Windows Server 2012 R2
Microsoft® Windows Server® 2012 Datacenter Microsoft® Windows Server® 2012 Standard Microsoft® Windows Server® 2012 Essentials	Windows Server 2012

Product Name	Abbreviation
Microsoft® Windows Server® 2012 Foundation	
Microsoft® Windows Server® 2008 R2 Foundation Microsoft® Windows Server® 2008 R2 Standard Microsoft® Windows Server® 2008 R2 Enterprise Microsoft® Windows Server® 2008 R2 Datacenter	Windows Server 2008 R2
Windows® 8.1 Windows® 8.1 Pro Windows® 8.1 Enterprise	Windows 8.1
Windows® 8 Windows® 8 Pro Windows® 8 Enterprise	Windows 8
Windows® 7 Home Premium Windows® 7 Professional Windows® 7 Enterprise Windows® 7 Ultimate	Windows 7

Microsoft Windows products listed in the table above are referred to in this manual as "Windows".

August 2015

Contents

Chapter 1 A Quick Tour

This chapter takes you on a quick tour of the NetCOBOL development environment.

It provides you with:

- An “Overview of the Guided Tour” to help you determine which parts of the tour are most relevant to you.
- “A Guided Tour through NetCOBOL” which takes you through:
    - Using COBOL Project Manager
    - Debugging
    - Using Configuration Management
    - Working with OO COBOL

The chapter focuses on creating COBOL applications. See Chapter 3. Working with Visual Basic for details on developing Visual Basic COBOL applications.

NetCOBOL ships with sample COBOL applications that cover a wide array of NetCOBOL functions. Appendix A describes these sample applications in detail.

NetCOBOL provides NetCOBOL Studio, a development environment based on Eclipse that replaced Project Manager.

For guided tour of NetCOBOL Studio, refer to NetCOBOL Studio User’s Guide Chapter 2: Tutorial.

1.1 Overview of the Guided Tour

With so many features it is not practical to give you a detailed guide through all of NetCOBOL's features and qualities. For example, you need to experience the reliability of the code for yourself, and the only valid test of a compiler's performance is how fast it executes your code. The guided tours in the following sections therefore aim to give you a quick, get-started overview of COBOL Project Manager, then go into depth in just three of the feature areas, namely debugging, configuration management and OO COBOL.

The guided tours will show you the following functions:

Using COBOL Project Manager

Quick tour, showing how to access and use most of the functions in COBOL Project Manager.

- Using a project - opening projects, understanding the project tree, and building trees.
- Editing code - invoking the editor, editing free-format COBOL source, and aids to code readability.
- Building - building, executing, and setting compile and link options.
- Maintaining data files - pointers to, and descriptions of, the data maintenance functions.

Debugging

- Preparing for debugging - setting compiler options and building.
- Invoking the debugger - options available in setting the debugging environment.
- Controlling execution - description of execution control functions and how to access them.
- Breaking execution - setting simple breakpoints, breaking on change of data, and breaking on a condition.
- Monitoring and changing data values - datatips, watching data, and the detailed data dialog.
- Tracing execution path - overview of switching on recording and exploring the execution path.
- Using the recording and playback functions - step-by-step instructions for recording and playing back debugging commands or sessions.
- Debugging with Visual Basic - walk through a dual language debugging session.

Using the configuration management functions

- Overview of PowerGEM structures - context setting for the configuration management environment.
- How to set up a PowerGEM library - step-by-step instructions for setting up a PowerGEM library.
- Using the configuration management functions from Project Manager - how to check-out and check-in files, the effects of check-out, and overview of other CM functions.

Working with OO COBOL

- Brief introduction to OO COBOL - concepts supported and a description of the major constituents of OO COBOL.
- Guided tour of an OO COBOL application - an OO COBOL application viewed from the Project Manager, COBOL editor, Project Browser and Class Browser.

1.2 A Guided Tour through NetCOBOL

Follow the steps outlined in the sections below to acquaint yourself with some of the key features of the NetCOBOL product.

Note

Many of the windows have been resized before taking snapshots so that they do not take up too much space in this document. The windows you see are likely to be different sizes to the windows shown in this tour.

1.2.1 Using COBOL Project Manager

The COBOL Project Manager is an integrated development environment that provides project level management for developing COBOL applications. The Project Manager brings together:

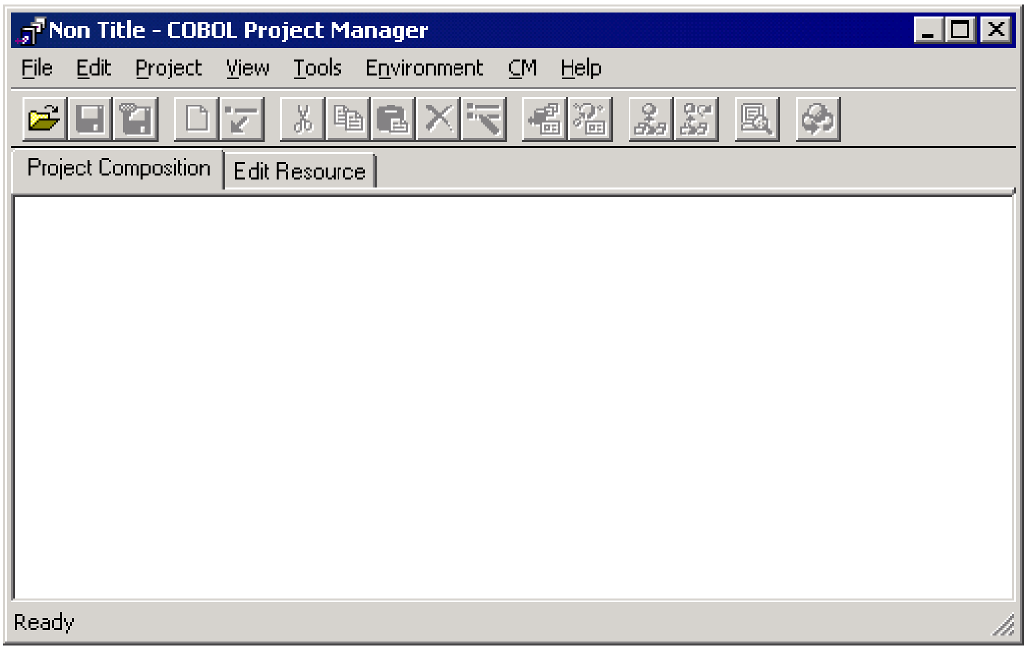
- A sophisticated, yet easy to use source code editor
- A full ANSI 85 and ANSI 74 compliant 32 bit COBOL compiler
- A 32 bit linker
- A sophisticated COBOL source debugger
- Execution facilities
- Project management and Make facilities
- Fujitsu’s File Management Utility
- OO COBOL Class and Project browsers

We will use one of the COBOL samples to illustrate how to access the various functions of COBOL Project Manager.

A. Invoking COBOL Project Manager and Opening a Project

1. From the Windows Start menu select Programs, Fujitsu NetCOBOL V10, COBOL Project Manager.

   The COBOL Project Manager window is displayed.

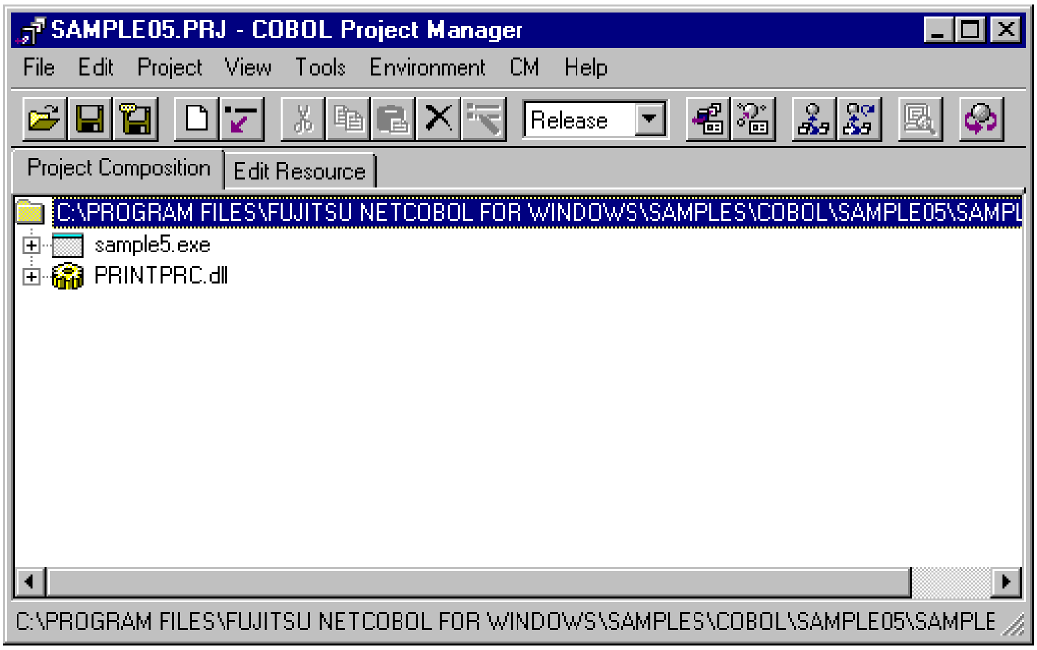

Non Title - COBOL Project Manager
File Edit Project View Tools Environment CM Help
Project Composition
Edit Resource
Ready

2. Click on the Project Open button, on the toolbar and navigate to the folder:

   C:\Program Files\Fujitsu NetCOBOL for Windows\Samples\cobol\Sample05 - assuming that you used the default folder names when you installed the products. Substitute your folder names if you overrode the defaults.

3. Select the file sample05.prj and click Open.

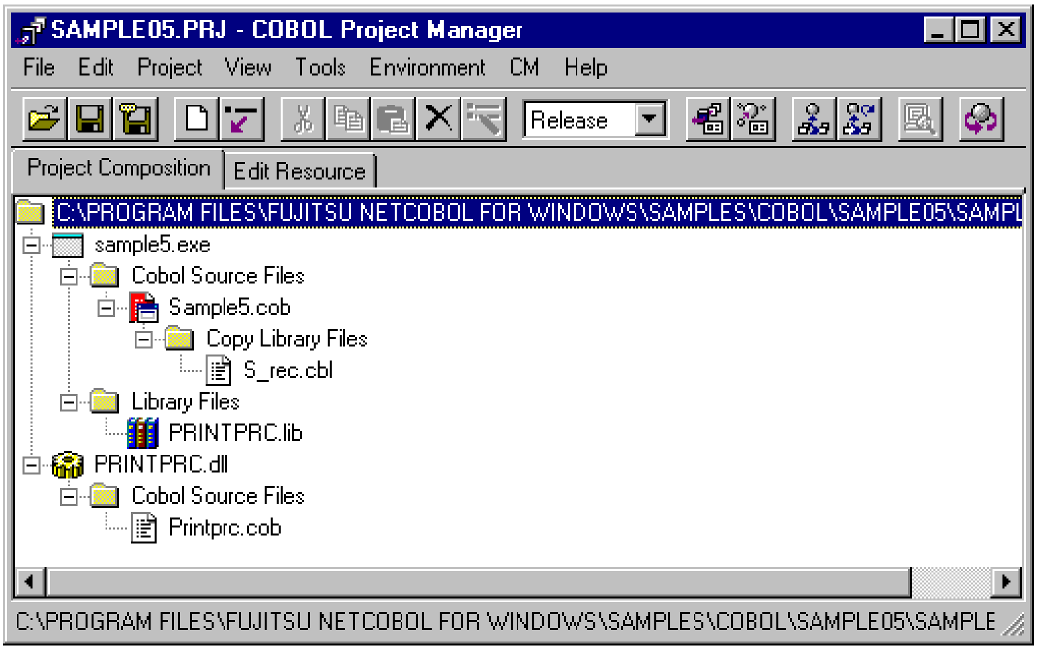
   Project Manager opens the project and displays a contracted project tree:


SAMPLE05.PRJ - COBOL Project Manager
File Edit Project View Tools Environment CM Help
Release
Project Composition
Edit Resource
C:\PROGRAM FILES\FUJITSU NETCOBOL FOR WINDOWS\SAMPLES\COBOL\SAMPLE05\SAMPL
sample5.exe
PRINTPRC.dll
C:\PROGRAM FILES\FUJITSU NETCOBOL FOR WINDOWS\SAMPLES\COBOL\SAMPLE05\SAMPLE

Notice that you can immediately see that there are two parts to this project - an EXE file and a DLL file. You would be correct in deducing that the program within the EXE calls a program within the DLL!

4. Fully expand the project tree by selecting View, Expand All from the menu bar.

   (Or you can repeatedly click on the plus symbols, , in the project tree but that takes longer!). The expanded tree looks like this:



Example

- You quickly have an overview of how the application sources are structured: a main program, SAMPLE5.COB, calls a subprogram, PRINTPRC.COB, and they both share a common copy library, S_REC.CBL.
- The chosen program structure uses a library file PRINTPRC.LIB.
- The different types of file are clearly distinguished by different icons.
- The icon for the main program, SAMPLE5.COB, stands out as it is colored red.
- The project tree is made up of folders and items. Folders contain collections of associated items, and items are assembled from several collections. For example the project folder contains two items, SAMPLE5.EXE and PRINTPRC.DLL. The SAMPLE5.EXE item is made up of two collections: a collection of COBOL source files; and a collection of library files (because the application is a simple one, each collection only contains one item).

5. Right-click on various folders and items in the project tree.

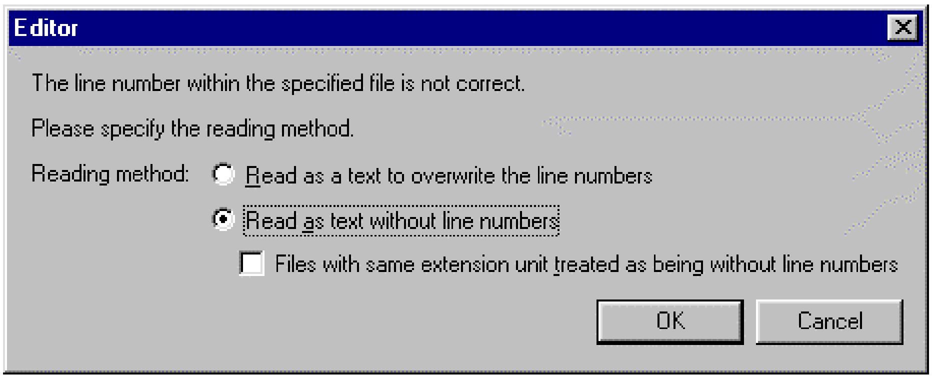
   Project Manager displays a pop-up menu with appropriate functions for the selected folder or item.

   You build project trees by using the New Folder and New/Add File functions.

B. Editing Source Code

1. For example, one of the things you can do using the pop-up menu is edit the source code. Right-click on SAMPLE5.COB and select Edit from the pop-up menu.

   SAMPLE5.COB demonstrates the use of free format coding, and doesn't have line numbers in columns 1-6. The editor therefore displays the dialog below:



The line number within the specified file is not correct.

Please specify the reading method.

Reading method: ○ Read as a text to overwrite the line numbers

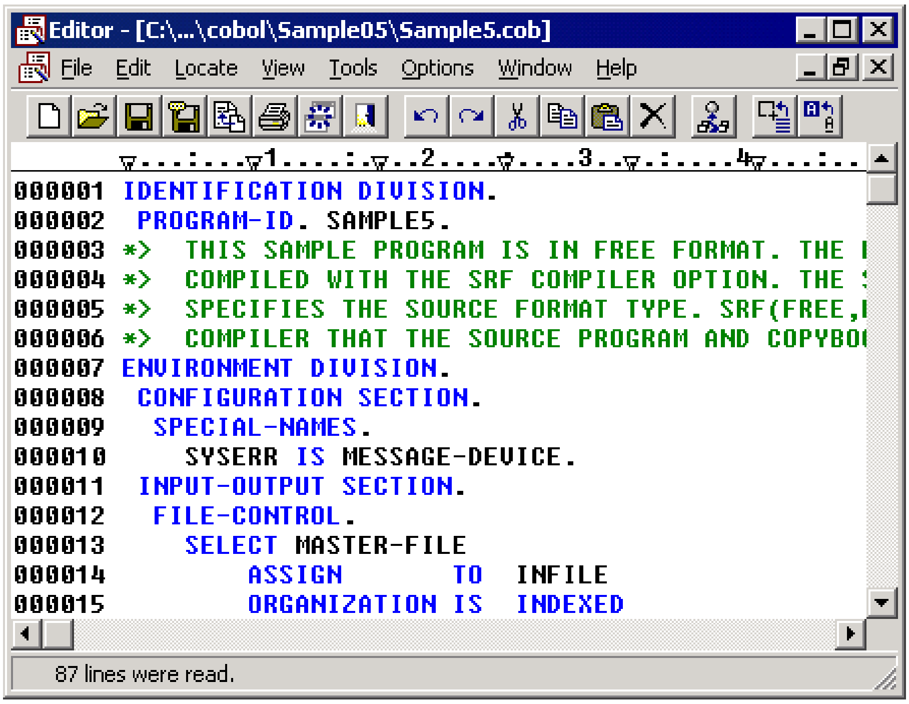
● Read as text without line numbers

□ Files with same extension unit treated as being without line numbers

2. Make sure you select "Read as text without line numbers" and click OK.

   The Editor displays the source code for SAMPLE5.COB.

   Notice that readability of the code is aided by using different colors for comments, reserved words and user-defined words.



```
000001 IDENTIFICATION DIVISION.
000002  PROGRAM-ID. SAMPLE5.
000003 *>   THIS SAMPLE PROGRAM IS IN FREE FORMAT. THE
000004 *>   COMPILED WITH THE SRF COMPILER OPTION. THE
000005 *>   SPECIFIES THE SOURCE FORMAT TYPE. SRF(FREE,
000006 *>   COMPILER THAT THE SOURCE PROGRAM AND COPYBO
000007 ENVIRONMENT DIVISION.
000008  CONFIGURATION SECTION.
000009   SPECIAL-NAMES.
000010     SYSERR IS MESSAGE-DEVICE.
000011  INPUT-OUTPUT SECTION.
000012   FILE-CONTROL.
000013     SELECT MASTER-FILE
000014         ASSIGN        TO   INFILE
000015         ORGANIZATION IS   INDEXED
```

87 lines were read.

We will make a trivial change to the program so that you can see Project Manager's build function in action.

3. Make sure the status bar is visible by selecting View from the Editor menu bar.

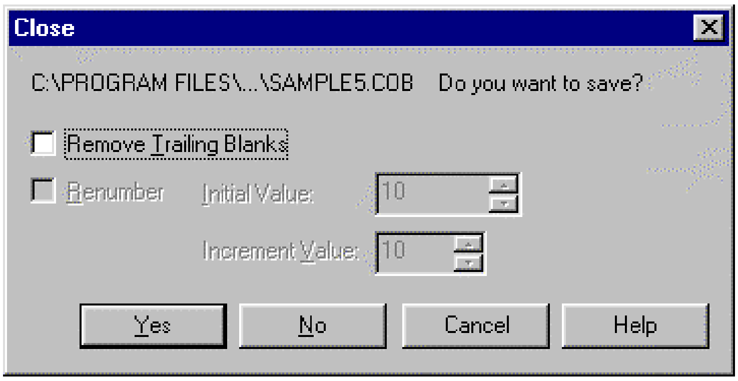
   If there is no check mark by "Status Bar" click on "Status Bar" and the status bar will be displayed at the bottom of the window.

4. Scroll down until you get to line 61 (the line number is displayed in the status bar). Line 61 contains the text:

   DISPLAY "GENERATE WORK-FILE =" WORK-FILE-NAME

5. Insert a space after the "=" sign.

6. Close and save the file by selecting File, Exit Editor from the menu bar and responding "Yes" to the dialog asking if you want to save.



C:\PROGRAM FILES\...\SAMPLE5.COB Do you want to save?

□ Remove Trailing Blanks

□ Renumber Initial Value: 10

Increment Value: 10

C. Building and Executing

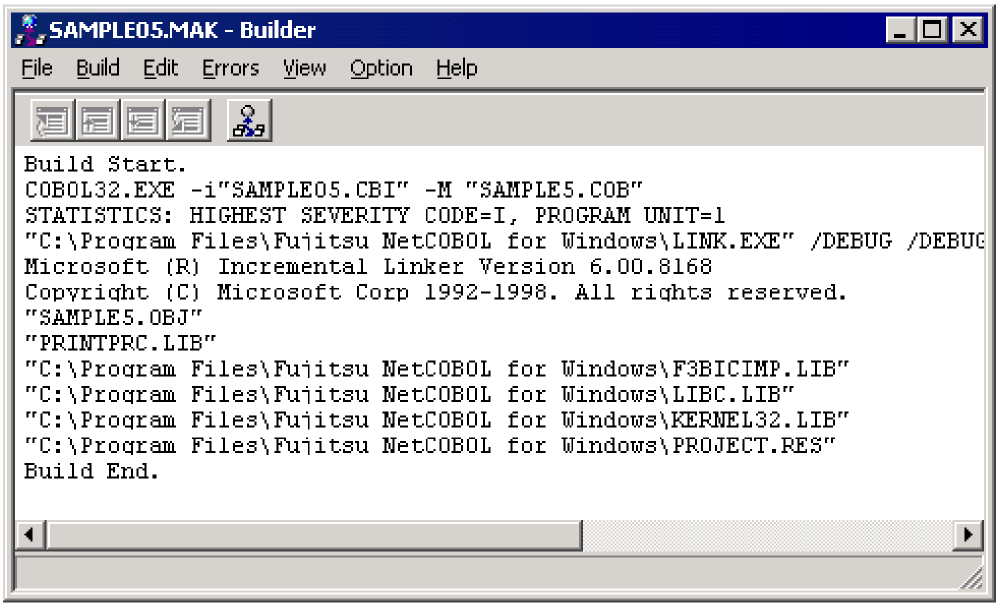
Because the project structure is already in place building is straightforward.

1. Right-click on SAMPLE5.EXE in the project tree, and select Build from the popup menu.

   Project Manager compiles the changed source program and builds the EXE. The build is actually guided by a make file that Project Manager maintains based on the project tree.

2. Any errors in the build are highlighted in red. The Builder tool has functions for navigating from one error to the next. If you have any errors, double click on the error line and the builder puts you into the Editor at the appropriate line. Fix each error and build the program again.

   A successful build has no red error messages:



```
Build Start.
COBOL32.EXE -i"SAMPLE05.CBI" -M "SAMPLE5.COB"
STATISTICS: HIGHEST SEVERITY CODE=I, PROGRAM UNIT=1
"C:\Program Files\Fujitsu NetCOBOL for Windows\LINK.EXE" /DEBUG /DEBUG
Microsoft (R) Incremental Linker Version 6.00.8168
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.
"SAMPLE5.OBJ"
"PRINTPRC.LIB"
"C:\Program Files\Fujitsu NetCOBOL for Windows\F3BICIMP.LIB"
"C:\Program Files\Fujitsu NetCOBOL for Windows\LIBC.LIB"
"C:\Program Files\Fujitsu NetCOBOL for Windows\KERNEL32.LIB"
"C:\Program Files\Fujitsu NetCOBOL for Windows\PROJECT.RES"
Build End.
```

3. Close the Builder window.

   Setting compiler and linker options: Had you wanted to change the compiler or linker options before building, you would have selected SAMPLE05.PRJ in the project tree view and, from the menu bar, selected Project, Option, Compiler (or Linker) Options. These functions display dialogs in which you can set the options. You are offered lists of all the compiler options and dialogs explaining the supported options, so you do not have to remember all those details.

Note

SAMPLE5 prints a single sheet of data to your default printer. If this is likely to cause a problem do not execute the steps 4, 5 and 6.

4. SAMPLE05 relies on a file that is created by SAMPLE02. With our apologies, changes to this guided tour mean the file is not necessarily available at this point. To be sure it is available, close the SAMPLE05 project, open SAMPLE02.PRJ (in the ..\Sample02 folder), Build SAMPLE02 and execute SAMPLE02 (select SAMPLE02.EXE in the project tree and select Project, Execute from the menu bar). Then close SAMPLE02.PRJ and reopen SAMPLE05.PRJ.

5. Now execute the program by selecting SAMPLE5.EXE in the project tree and selecting Project, Execute from the menu bar.

   Project Manager starts the program executing. SAMPLE5 executes displaying the message you edited - check that there is a space after the "=" sign.

6. Click OK in the message box. SAMPLE5 continues to run, printing a single sheet of data to your default printer.

   We will have a closer look at the program in the debugging section.

D. Maintaining Data Files

NetCOBOL provides the COBOL File Utility for maintaining COBOL data files.

1. From the Tools menu, select File Utility.

   Project Manager invokes the COBOL File Utility.

2. Click on the Commands menu of the COBOL File Utility to see the range of functions provided. You can select each of these functions in turn to see exactly what they provide. The following list summarizes the functions ("COBOL files" means sequential, relative or indexed files in fixed or variable length format):

   - Convert: Converts text files to variable length sequential files or variable length sequential files to text files.
   - Load: Loads COBOL files from variable length sequential files.
   - Unload: Creates variable length sequential files from COBOL files.
   - Browse: Provides hex and character display of data in COBOL and text files.
   - Print: Prints data from COBOL and text files.
   - Edit: Provides hex and character editing of data in COBOL and text files.
   - Extend: Provides hex and character input of data to extend (add to) COBOL and text files.
   - Sort: Sorts data in COBOL and text files.
   - Attribute: Displays statistics on indexed files.
   - Recovery: Recovers data from corrupted indexed files.
   - Reorganize: Reorganizes indexed files by deleting blocks of unused data.
   - Copy: Copies files.
   - Delete: Deletes files.
   - Move: Moves files.

   Although some of the functionality of the COBOL File Utility is provided in a basic fashion you can see that most file maintenance needs are covered by the above functions.

   Purchasers of the Enterprise Edition of NetCOBOL also have the Data Converter and Data Editor tools. These provide more sophisticated, COBOL-record-descriptionaware data conversion and editing functions. These tools are provided on the Start Programs menu (NetCOBOL group) or can be added to Project Manager using the Tools, Customize Menu function. Dataedit.exe and Dataconv.exe are stored in the "C:\Program Files\Fujitsu NetCOBOL for Windows\DataTool" folder.

1.2.2 Debugging

This section guides you through a number of the key debugging functions.

A. Preparing for Debugging

Programs are prepared for debugging by compiling with the TEST compiler option.

When you set this compiler option within Project Manager the appropriate linker options are set automatically. The Project Manager has a “Build for Debugging” option on the Project, Option sub-menu that sets all the appropriate compiler and linker options for you.

In Project Manager compiler and linker options are set at the project level - so all files within a project are compiled with the same options.

Check that Sample05 is being built for debugging:

1. From the Project menu select, Option and check that “Build for Debugging” is checked.
2. If it is not checked, click on the menu item.

   If it is checked click on a blank area of the Project Manager window to close the menu.
3. From the Project menu select Build.

   Project Manager builds the project using the newly set compiler and linker options.

   Note that if you didn’t have to check the “Build for Debugging” option, so that nothing has changed since the previous build, the builder simply displays the messages “Build Start” and “Build End” indicating that there was nothing to do.

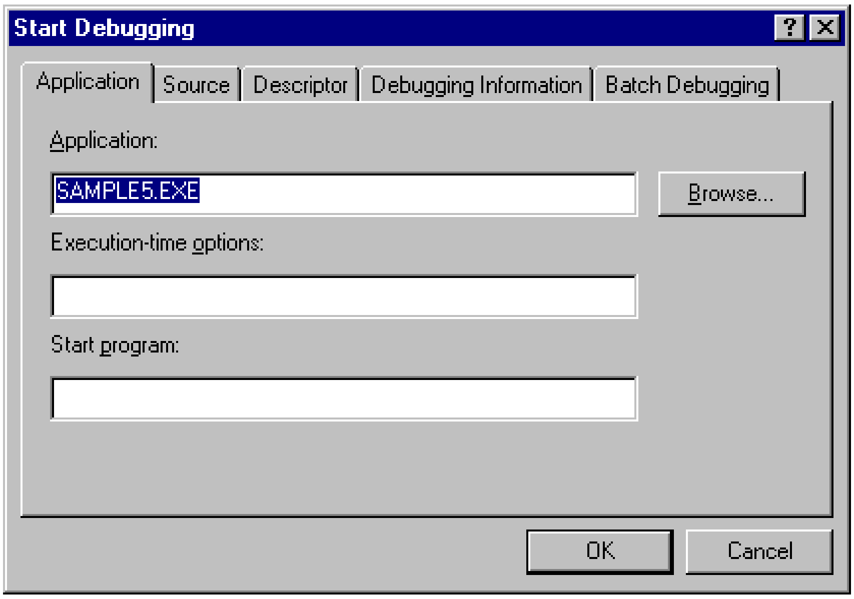
The project is now ready for debugging.

B. Invoking the Debugger

To invoke the debugger:

1. Select SAMPLE05.PRJ in the project tree.
2. From the Project menu select Debug.

   The COBOL Debugger starts and displays the Start Debugging dialog.

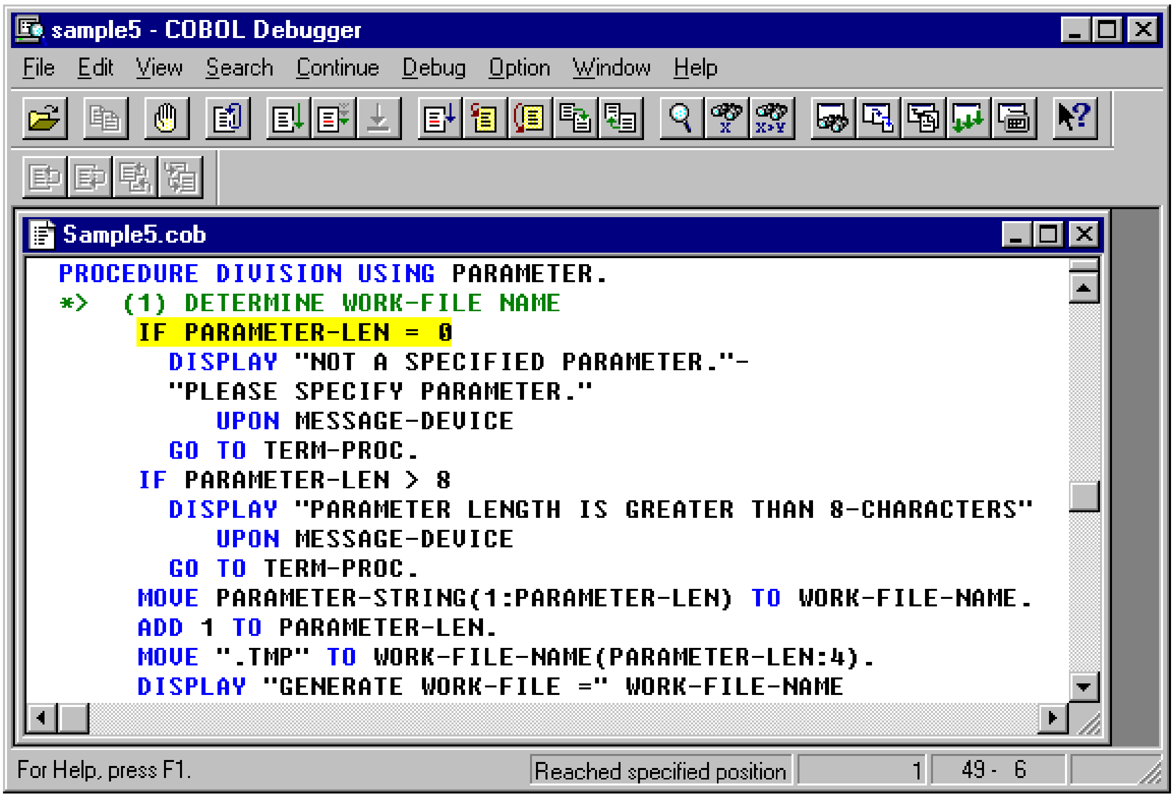


Application: SAMPLE5.EXE

   This dialog gives you the opportunity to specify information that may affect your debugging session, such as: the locations of various source and system files, the option to have the debugging session driven by a command file, and even which program you first want to see in the debugger. Although the defaults will work for you most of the time, the ability to configure this information at start-up is invaluable in certain situations, such as debugging with other languages. (See "Debugging with Visual Basic" below).

3. For this debugging session you do not need to specify any extra information so click OK.

4. The COBOL Debugger displays Sample5.cob. It highlights the first executable line in the procedure division.



```
PROCEDURE DIVISION USING PARAMETER.
*>   (1) DETERMINE WORK-FILE NAME
         IF PARAMETER-LEN = 0
           DISPLAY "NOT A SPECIFIED PARAMETER."-
           "PLEASE SPECIFY PARAMETER."
              UPON MESSAGE-DEVICE
           GO TO TERM-PROC.
         IF PARAMETER-LEN > 8
           DISPLAY "PARAMETER LENGTH IS GREATER THAN 8-CHARACTERS"
              UPON MESSAGE-DEVICE
           GO TO TERM-PROC.
         MOVE PARAMETER-STRING(1:PARAMETER-LEN) TO WORK-FILE-NAME.
         ADD 1 TO PARAMETER-LEN.
         MOVE ".TMP" TO WORK-FILE-NAME(PARAMETER-LEN:4).
         DISPLAY "GENERATE WORK-FILE =" WORK-FILE-NAME
```

For Help, press F1. | Reached specified position | 1 | 49 - 6

Notice that code readability is aided by coloring the source code.

5. You may want to open a number of windows in the debugger so maximize the debugger window by clicking on the maximize button, at the top right of the window.

6. Widen the source display window by dragging the border.

You are now ready to explore the program using the debugging features.

Sections C to F give general descriptions of the functions available with pointers to where these functions can be found. You can experiment with the functions that are of most interest to you.

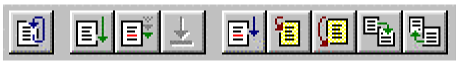
Some powerful features you may want to check are:

- The "Specific Execution" execution control function. (In section C)
- Break on change of data. (In section D)
- Datatips. (In section E)
- Tracing the execution path. (In section F)

C. Controlling Execution

Location

The execution control functions are located on the Continue menu, and a number of them are available on the toolbar:



Hover the mouse pointer over a button to display a tooltip confirming the function of the button.

Description

Re-debug:

Restarts the program debugging session with re-initialized data. Gives the option of clearing or retaining breakpoints, watch data, and passage counts.

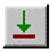
Go:

Runs the program to the next breakpoint, break condition, or the end whichever comes first.

Animate:

Executes the program line by line at a configurable speed.

Break:



Interrupts execution when the program is running or being animated.

Change Start Point to Cursor: (not on toolbar)

Makes the statement containing the cursor the next statement to be executed. (If disabled try stepping one statement.)

Run to Cursor:

Runs the program up to the statement containing the cursor.

Step Into:

Executes one statement. If the statement transfers control to another debuggable program the debugger takes you into that program.

Step Over:

Executes one statement and stops on the next statement in the current program, regardless of whether the current statement transfers control to another debuggable program.

Step Out:

Runs the current program, interrupting execution when control passes to the calling (debuggable) program.

Run to Next Program:

Runs the current program, interrupting execution when control passes to another debuggable program.

Specific Execution: (not on toolbar)

Runs the program up to the next point specified in the Specific Execution Condition.

Specific Execution Condition: (not on toolbar)

Displays a dialog from which you can select a condition for interrupting program execution. Options include stopping at any of 52 verbs, stopping at a specified line number, stopping at the entry or exit of a named program, and stopping where a named file is accessed.

D. Breaking Execution

In addition to specifying points to which execution should run in the execution control functions, there are three other ways of breaking execution:

- Setting breakpoints
- Breaking on change of data
- Breaking on satisfaction of a specified condition

Breakpoints

1. To set a simple breakpoint - right click on the line to contain the breakpoint and select "Set Breakpoint" from the pop-up menu.
2. To set a more complex breakpoint, select Breakpoint from the Debug menu.

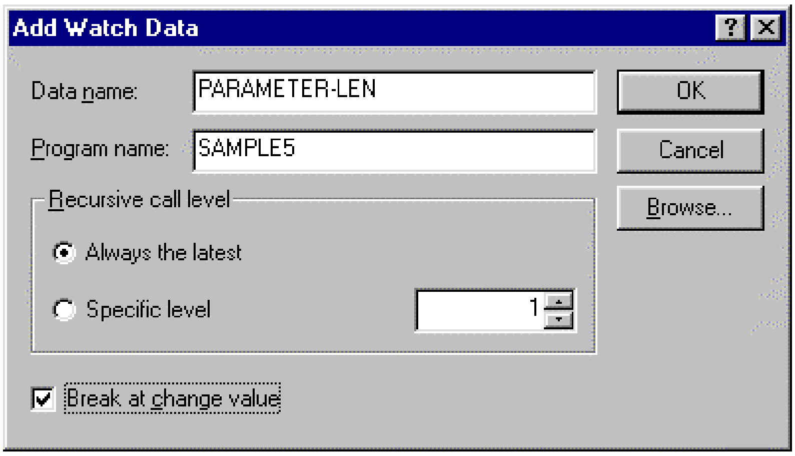
   You can specify that execution should only be interrupted at the line if a condition is also satisfied, or if the line has been executed a particular number of times. You can also maintain a list of breakpoints enabling and disabling them depending on your debugging situation.

Break on Change of Data

To have the debugger interrupt execution when the value of a data item changes:

1. Right-click on the data item name, anywhere that it occurs in the text, and select Add Watch Data from the pop-up menu.

   The debugger displays the Add Watch Data dialog with the data and program name fields already filled in:



Data name: PARAMETER-LEN

Program name: SAMPLE5

Recursive call level

● Always the latest

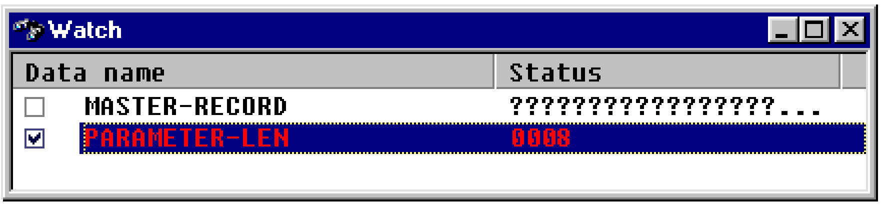
○ Specific level 1

☑ Break at change value

2. Select any options you require, such as “Break at change of value” and click OK.

   The item is added to the Watch window - shown below with two items:

   MASTER-RECORD and PARAMETER-LEN.



	Data name	Status
☐	MASTER-RECORD	????????????????...
☑	PARAMETER-LEN	0008

The check box indicates that any change in PARAMETER-LEN causes a break in execution - the red colored text indicates a break on change of value has just occurred.

The queries in the MASTER-RECORD entry indicate non-character values (in this case LOW-VALUES as the record has not yet been read).

The “···” in the MASTER-RECORD entry indicates there is more data than displayed in the window.

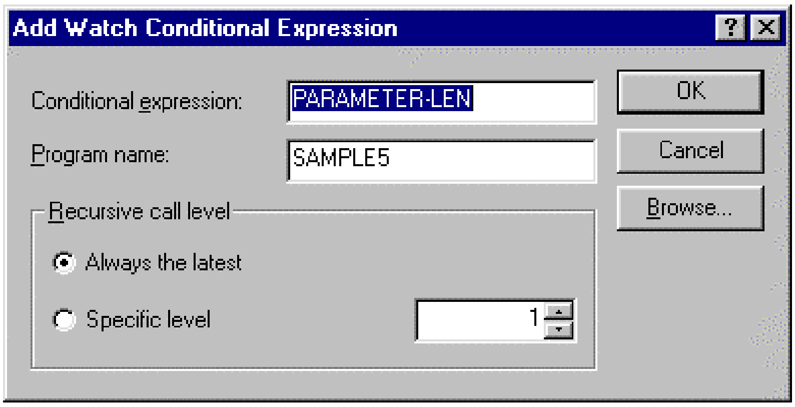
3. You can check and uncheck the checkboxes in the Watch window to monitor different items for changes.

Break on Condition

To have the debugger interrupt execution when a particular condition is satisfied:

1. From the Debug menu, select Add Watch Conditional Expression.

   The debugger displays the Add Watch Conditional Expression dialog with any word that was pointed at entered in the “Conditional expression” field.



Add Watch Conditional Expression

Conditional expression: PARAMETER-LEN

Program name: SAMPLE5

Recursive call level

● Always the latest

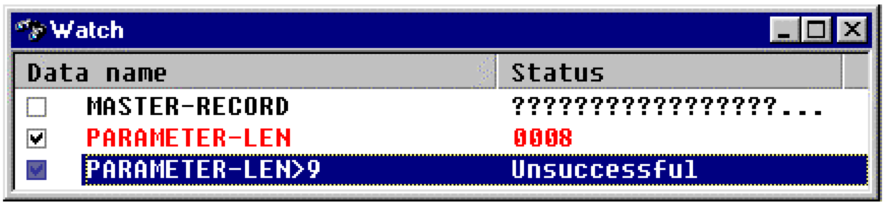
○ Specific level 1

2. Enter a conditional expression in the entry field. (e.g. PARAMETER-LEN>9).

   Note: The Browse function lets you enter the name of a record or group item, then select data items from within that record.

3. Click the OK button.

   The debugger adds the condition to the Watch window:



Watch

	Data name	Status
□	MASTER-RECORD	?????????????????...
☑	PARAMETER-LEN	0008
☑	PARAMETER-LEN>9	Unsuccessful

   It then monitors the condition and interrupts execution when the condition is satisfied.

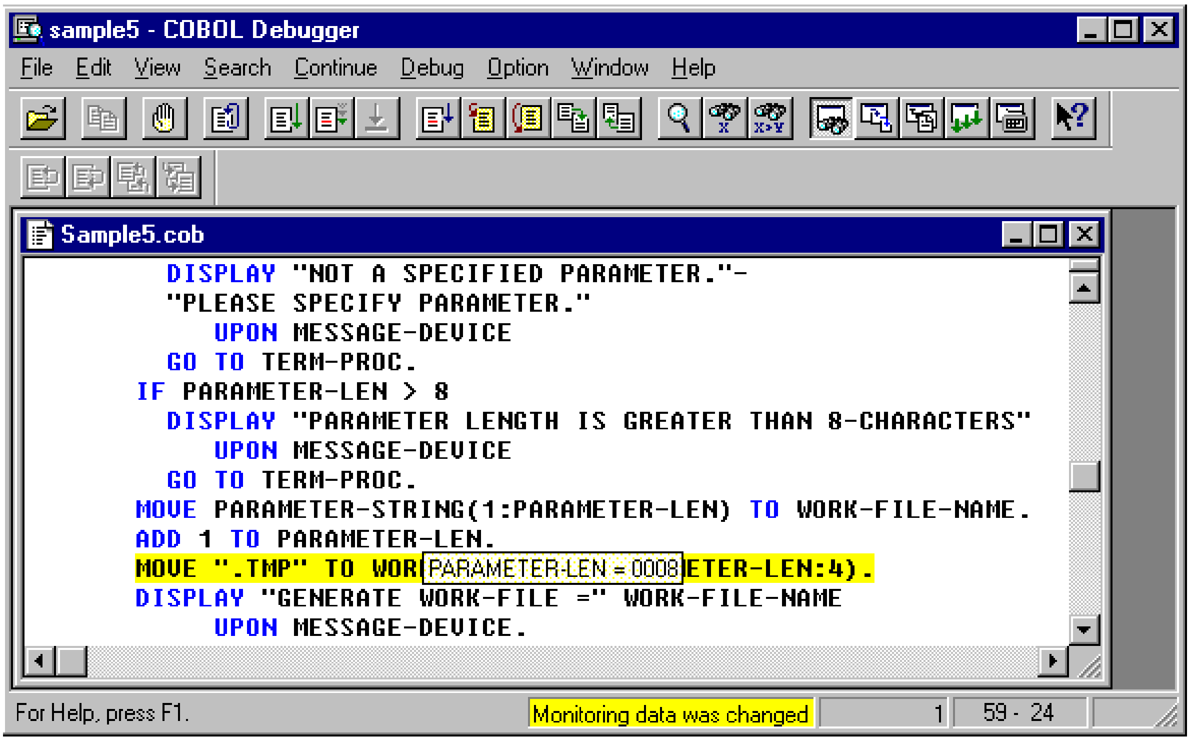
E. Monitoring and Changing Data Values

You can monitor data values by using Datatips or the Watch Data window.

You can change data using the Debug - Data dialog, or the Watch Data window.

Datatips

You can check the value of a data item very quickly by using the datatip feature. Just hover the mouse pointer over the data item name anywhere in the text, and the debugger displays the value.

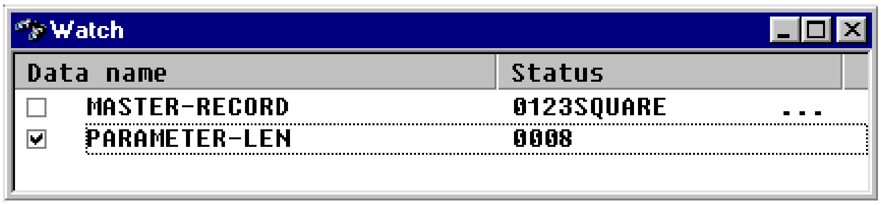


```
  DISPLAY "NOT A SPECIFIED PARAMETER."-
  "PLEASE SPECIFY PARAMETER."
     UPON MESSAGE-DEVICE
  GO TO TERM-PROC.
IF PARAMETER-LEN > 8
  DISPLAY "PARAMETER LENGTH IS GREATER THAN 8-CHARACTERS"
     UPON MESSAGE-DEVICE
  GO TO TERM-PROC.
MOVE PARAMETER-STRING(1:PARAMETER-LEN) TO WORK-FILE-NAME.
ADD 1 TO PARAMETER-LEN.
MOVE ".TMP" TO WOR PARAMETER-LEN = 0008 ETER-LEN:4).
DISPLAY "GENERATE WORK-FILE =" WORK-FILE-NAME
     UPON MESSAGE-DEVICE.
```

Monitoring data was changed

Watch Data Window

The Watch Data window displays a list of items with their values.



Data name	Status
□ MASTER-RECORD	0123SQUARE ...
☑ PARAMETER-LEN	0008

To add an item to the Watch Data window, right click on the item name, and select "Add Watch Data" from the pop-up menu.

Alternatively display the Data dialog, enter the data name and click on the Watch button.

To edit an item in the Watch Data window, just click on the value and enter the new value.

Debug, Data Dialog

The Debug, Data dialog lets you:

- Enter a data name, including names of items in any loaded program.
- For recursive programs you can specify the call level at which to view the data item.
- View the value of the item in character or hexadecimal formats.
- Edit the value.
- Add the item to the Watch Data window.

To display the Debug, Data dialog select Data from the Debug menu.

F. Tracing the Execution Path

The debugger can record the execution path so that you can check how execution reached a particular point.

To start the debugger recording the execution path select Trace from the Debug menu.

You can then run the code to a breakpoint or to the point at which a condition is satisfied.

The Backward Trace functions (Previous Statement , Next Statement , Previous Procedure , Next Procedure) help you explore the route that execution took to reach that point.

Use View, Toolbars, to display the Backward Trace toolbar.

G. Using the Recording and Playback Functions.

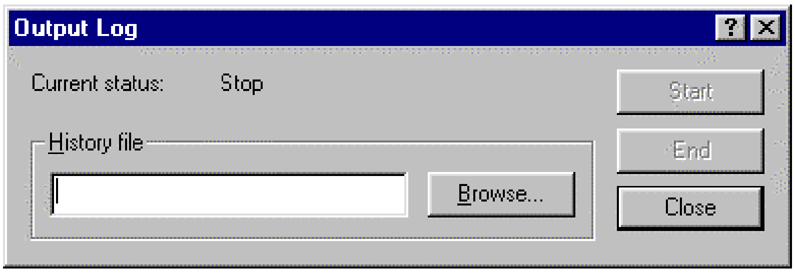
The debugger lets you record complete debugging sessions or parts of debugging sections. This can let you automate repetitive tasks or testing procedures.

The following instructions record and playback a very simple sequence but they demonstrate the principles required to set up much more complex debug recordings.

We will record some steps, and play the recorded file while recording a new file.

1. If you have been debugging using Sample5, select Continue, Redebug to initialize the debugging session. If you have not been debugging Sample5 start the debugger with Sample5.
2. From the Option menu select Output Log.

   The debugger displays the Output Log dialog:

   Current status: Stop

   The History file is the file to which the debugger writes a record of the operations performed with the results of those operations.
3. Enter "Recording1" for the name of the history file.
4. Click on the Start button to start recording.
5. Click on the Close button to close the Output Log dialog window.
6. Click on the Step Into button, two times to execute two statements.
7. Right click on the data name "WORK-FILE-NAME" and select Data from the popup menu.

   The debugger displays the Data dialog.

   There is nothing in WORK-FILE-NAME - it is all spaces (which you can check by switching to Hexadecimal - an ASCII space is "20" in hex).
8. Click on the Output Log button in the Data dialog.

   The debugger writes information about WORK-FILE-NAME to the log file.
9. Click on the Close button to close the Data dialog.
10. From the Option menu, select Output Log, and click on the End button.

    This stops the debugger logging each operation.
11. Click on the Close button to close the Output Log dialog.

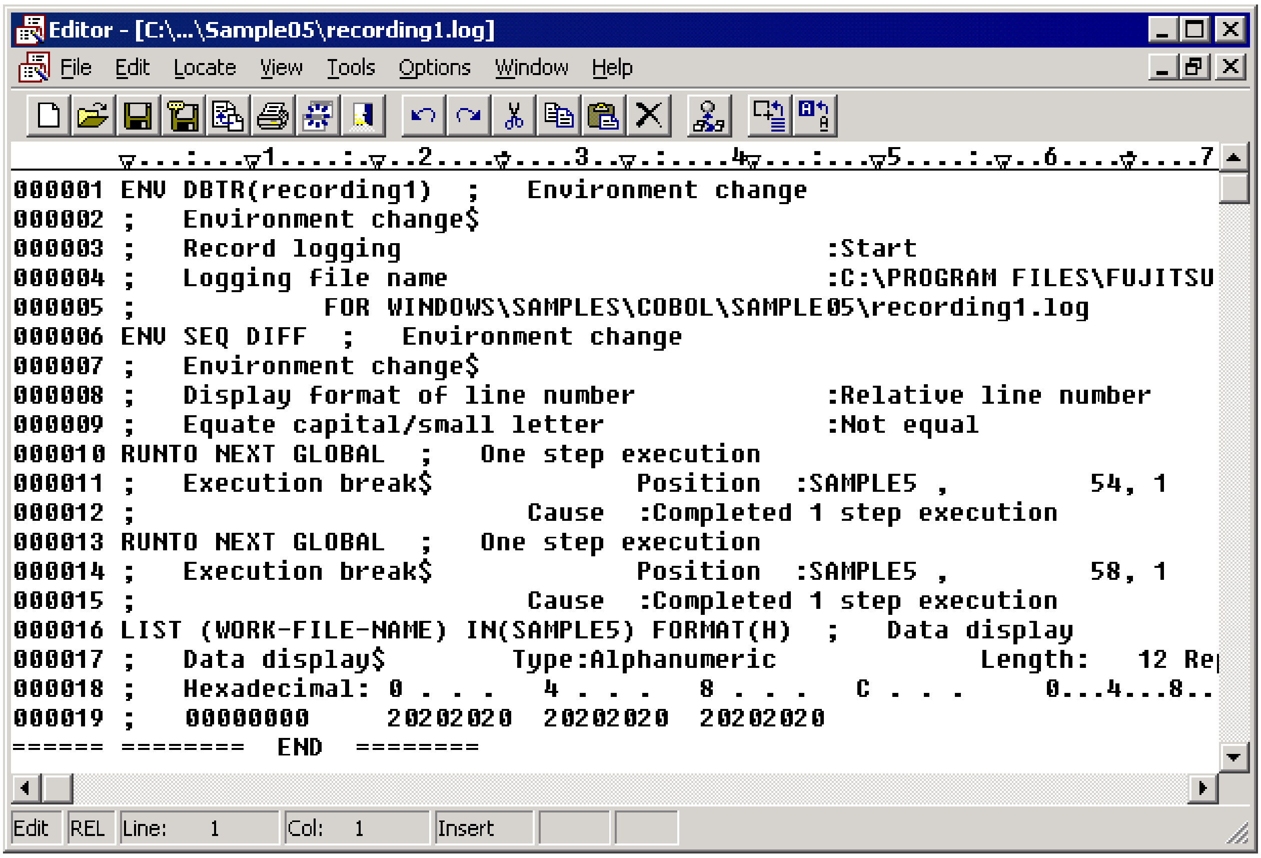
So, to recap, you have recorded stepping two statements, and checking the value of WORK-FILE-NAME.

1. Now open the Recording1.log file by starting Windows Explorer (right click on the Start button in the Windows task bar, and select Explore from the pop-up menu) and navigating to the folder:

   C:\Program Files\Fujitsu NetCOBOL for Windows\Samples\cobol\Sample05

   Then double click on the file "Recording1.log" to bring up the file in the COBOL Editor.

Your recording should look something like this:


Editor - [C:\...\Sample05\recording1.log]
File Edit Locate View Tools Options Window Help
Edit
REL
Line: 1
Col: 1
Insert

```
000001 ENV DBTR(recording1)  ;   Environment change
000002 ;   Environment change$
000003 ;   Record logging                          :Start
000004 ;   Logging file name                       :C:\PROGRAM FILES\FUJITSU
000005 ;            FOR WINDOWS\SAMPLES\COBOL\SAMPLE05\recording1.log
000006 ENV SEQ DIFF  ;  Environment change
000007 ;   Environment change$
000008 ;   Display format of line number           :Relative line number
000009 ;   Equate capital/small letter             :Not equal
000010 RUNTO NEXT GLOBAL  ;  One step execution
000011 ;   Execution break$          Position  :SAMPLE5 ,        54, 1
000012 ;                      Cause  :Completed 1 step execution
000013 RUNTO NEXT GLOBAL  ;  One step execution
000014 ;   Execution break$          Position  :SAMPLE5 ,        58, 1
000015 ;                      Cause  :Completed 1 step execution
000016 LIST (WORK-FILE-NAME) IN(SAMPLE5) FORMAT(H)  ;  Data display
000017 ;   Data display$         Type:Alphanumeric          Length:   12 Re
000018 ;   Hexadecimal: 0 . . .   4 . . .   8 . . .   C . . .    0...4...8..
000019 ;    00000000     20202020  20202020  20202020
====== ========  END  ========
```

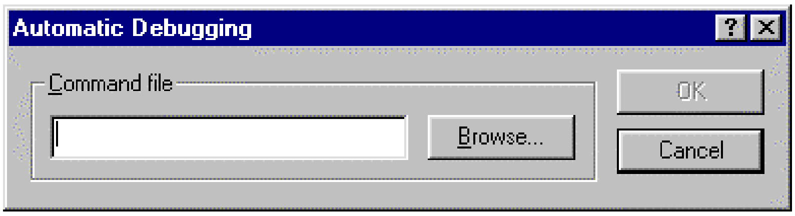
Notice that the log file consists essentially of two types of lines - debug commands or operations (ENV, RUNTO, LIST), and comment lines starting with ";". The comments describe the commands and display results of the commands. Because the results are output as comments you can use the log file as input to the debugger - it ignores the comments and executes the commands.

So now we will execute the log file.

1. Close the Editor and return to the debugger by selecting "sample5 - COBOL Debugger" from the Windows task bar.
2. Start another output log by selecting Option, Output Log from the menu bar, and entering "Recording2" as the History file name. Click on the Start button, then Close the Output Log window.

   By writing an output log when playing back debugger commands you can later inspect the results of those commands.
3. From the Option menu select Automatic Debugging.

   The debugger displays the Automatic Debugging dialog.



   The command file is any file containing debug commands. We will use the first log file we created.
4. Click on the Browse button to display a list of available command files.

   The debugger displays all files with a .log extension.

   If necessary navigate to the folder containing your log file.

5. Select Recording1.log and click on Open.

   You return to the Automatic Debugging dialog with the file name in the Command file entry field.

6. Click on OK to run the command file.

   The debugger displays an information message that it is ignoring the DBTR option in the command file. (The ENV, DBTR option starts history files, and is not supported when running command files - we'd normally edit the log file to remove this option.)

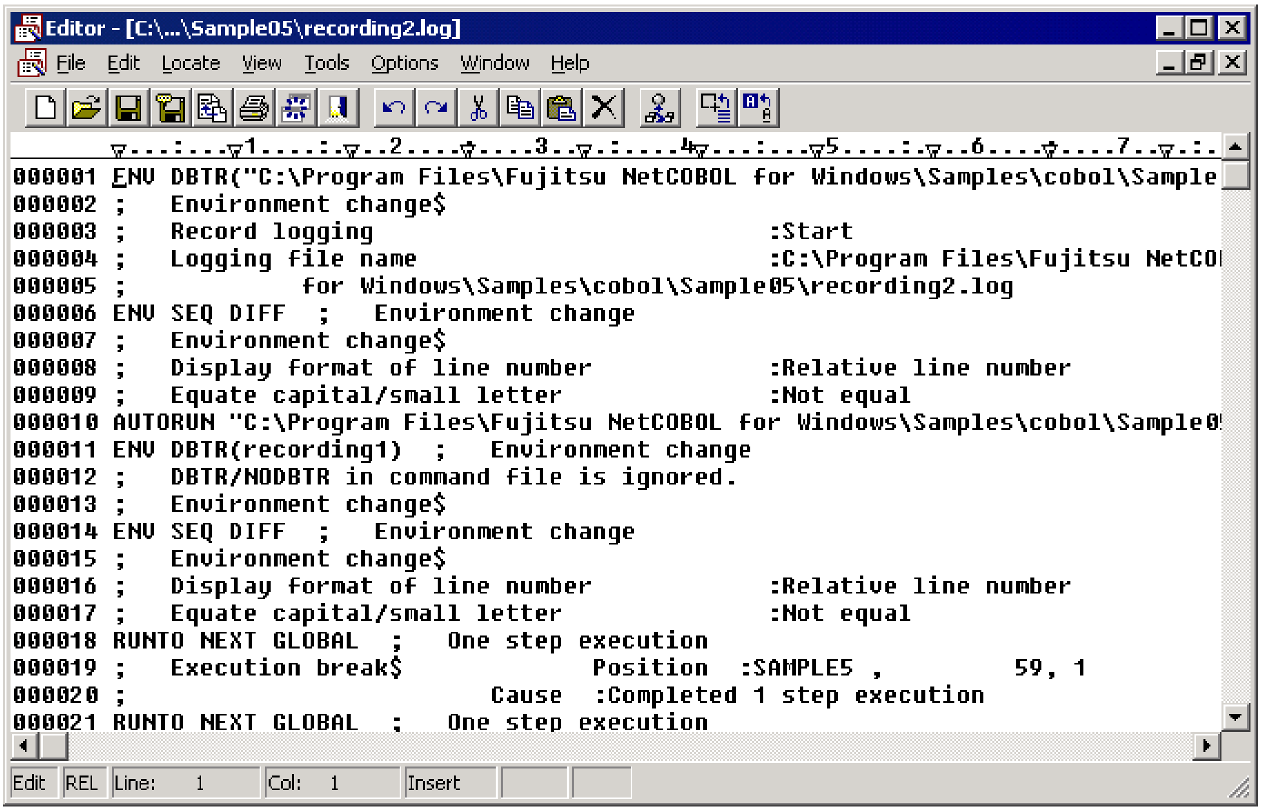
7. Respond OK to the message.

   The debugger executes the commands in the Recording1.log file (two steps and one check of data item contents).

8. Stop writing to the Recording2 output log by selecting Option, Output Log from the menu bar, and clicking on End.

9. Now open the Recording2.log file by returning to Windows Explorer (select Exploring from the Windows task bar) and double clicking on the Recording2.log file to bring it up in the COBOL Editor.

   Your recording should look like this:



```
000001 ENV DBTR("C:\Program Files\Fujitsu NetCOBOL for Windows\Samples\cobol\Sample
000002 ;   Environment change$
000003 ;   Record logging                          :Start
000004 ;   Logging file name                       :C:\Program Files\Fujitsu NetCO
000005 ;            for Windows\Samples\cobol\Sample05\recording2.log
000006 ENV SEQ DIFF  ;   Environment change
000007 ;   Environment change$
000008 ;   Display format of line number           :Relative line number
000009 ;   Equate capital/small letter             :Not equal
000010 AUTORUN "C:\Program Files\Fujitsu NetCOBOL for Windows\Samples\cobol\Sample0
000011 ENV DBTR(recording1)  ;   Environment change
000012 ;   DBTR/NODBTR in command file is ignored.
000013 ;   Environment change$
000014 ENV SEQ DIFF  ;   Environment change
000015 ;   Environment change$
000016 ;   Display format of line number           :Relative line number
000017 ;   Equate capital/small letter             :Not equal
000018 RUNTO NEXT GLOBAL  ;   One step execution
000019 ;   Execution break$           Position  :SAMPLE5 ,         59, 1
000020 ;                      Cause   :Completed 1 step execution
000021 RUNTO NEXT GLOBAL  ;   One step execution
```

This has a couple of differences from Recording1.log:

- First it shows the Recording1.log file being run (the AUTORUN command).
- Second, the results of each of the commands (RUNTO, and LIST) are different because the commands were executed at a different point in the program.

You can see how, in a few steps, long or highly repetitive debug operations can be automated using the logging (recording) and automatic debugging (playback) features.

The full set of debug commands is documented in the debugger help system in the Reference, Line Commands section. This provides you with the option of coding sequences of debug commands using a text editor, or of generating command sequences automatically.

H. Debugging with Visual Basic

NetCOBOL is designed to work well with other languages. A good demonstration of this ability is to see the COBOL and Visual Basic debuggers working side by side. The following steps take you through a simple program made up of Visual Basic code calling a COBOL DLL. They give you all the information necessary to set up a dual language debugging session.

The example assumes that you have Microsoft Visual Basic 5.0 or later installed.

Setup the Environment Variable to Invoke the Debugger

Because, in this example, Visual Basic is going to be calling COBOL, you need to ensure that an environment variable (@CBR_ATTACH_TOOL) is set so that the COBOL Debugger is started when the COBOL code is invoked.

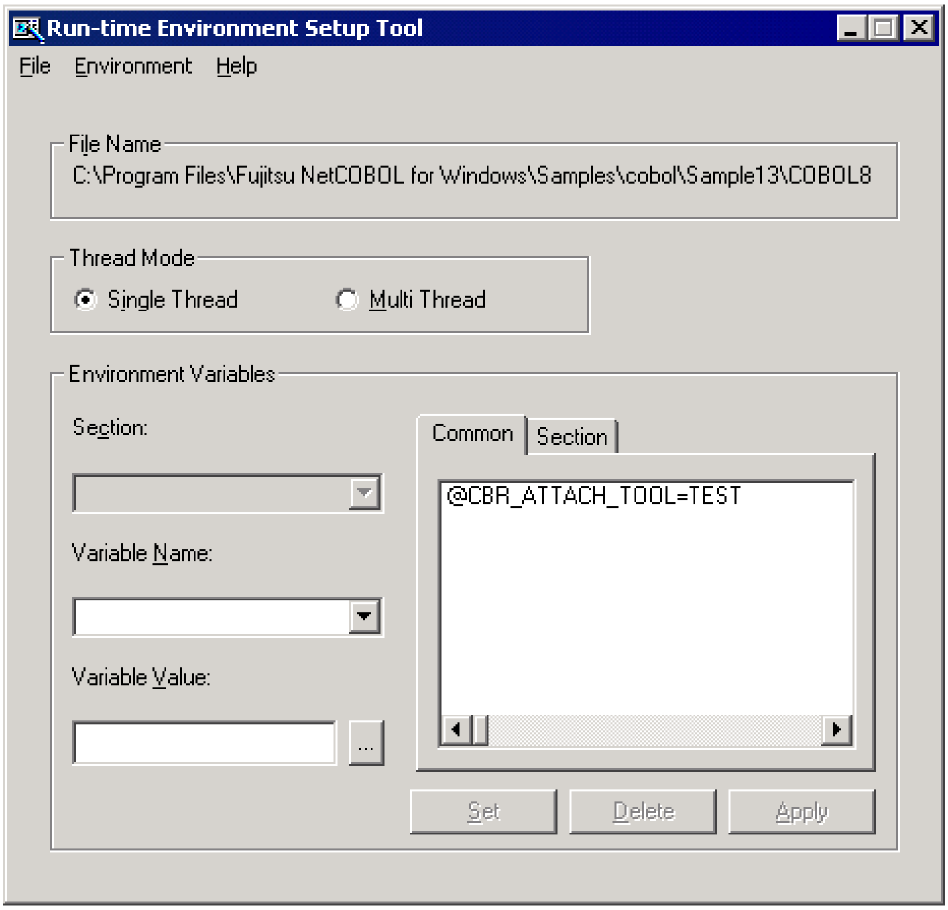
1. From COBOL Project Manager’s Tools menu select “Run-time Environment Setup Tool”.
2. Open the file COBOL85.CBR in the Sample13 folder. If it doesn’t exist, create it by navigating to the folder and entering “COBOL85.CBR” as the file to open.
3. Select the Section tab.
4. Select SAMPLE13 from the Section drop-down list.

   If there is nothing in the list enter “SAMPLE13”.
5. If @CBR_ATTACH_TOOL=TEST is not displayed in the Section list box:

   Select @CBR_ATTACH_TOOL from the Variable Name drop-down list.

   Enter “TEST” into the Variable Value field.

   Click on Set to add the variable to the section list.



File Name: C:\Program Files\Fujitsu NetCOBOL for Windows\Samples\cobol\Sample13\COBOL8

Thread Mode: ● Single Thread ○ Multi Thread

@CBR_ATTACH_TOOL=TEST

6. Click on Apply to write this information to the COBOL85.CBR file.

   Exit from the tool.

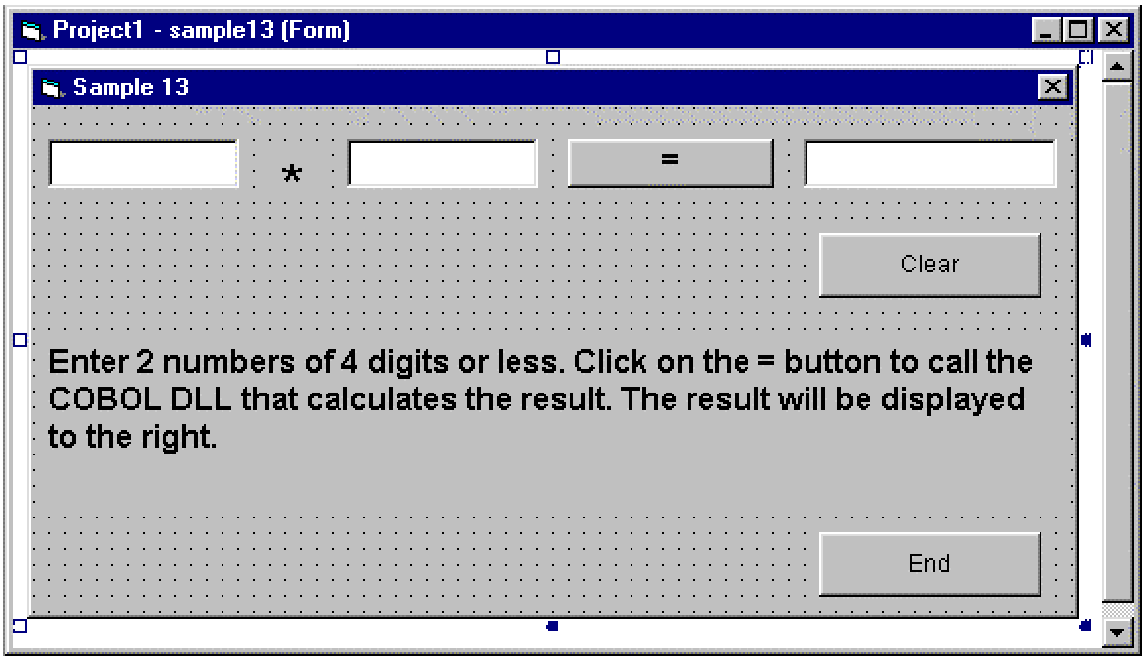
Start Debugging from Visual Basic

1. Start Visual Basic. (The instructions are based on Visual Basic 6.0, if you are using a different version, you may need to adapt them to your version.)

2. Open the Visual Basic project:

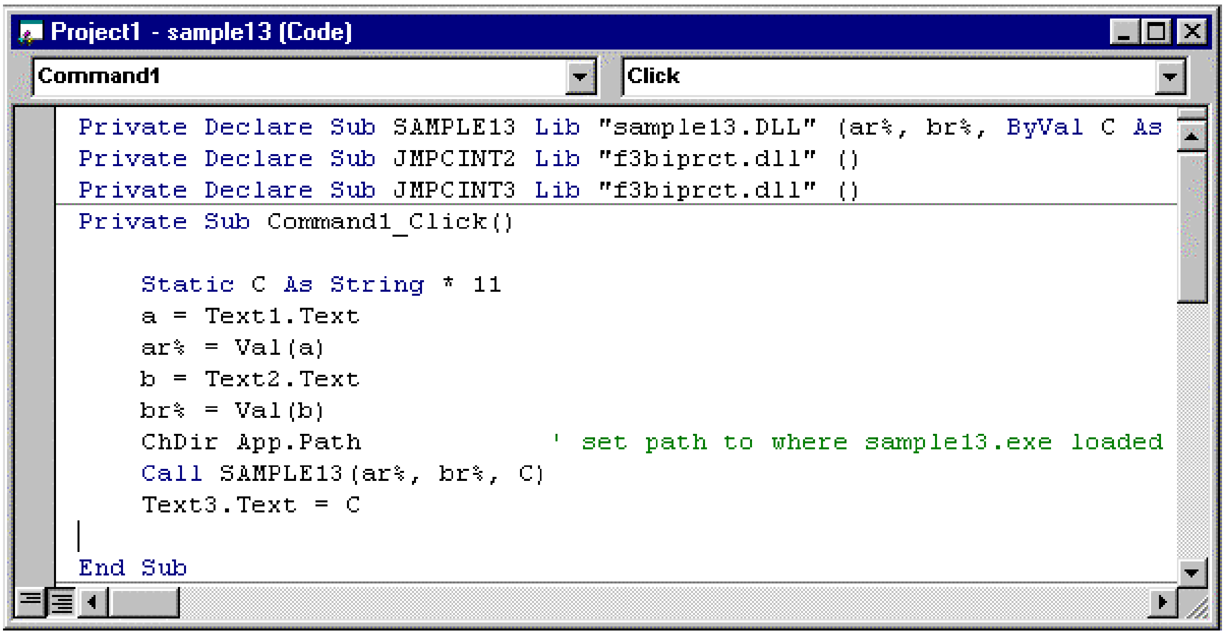
   C:\Program Files\Fujitsu NetCOBOL for Windows\Samples\cobol\Sample13\sample13.vbp

3. Look at the form and code of this sample to get a feel for what it does.


Project1 - sample13 (Form)
Sample 13
*
=
Clear
Enter 2 numbers of 4 digits or less. Click on the = button to call the COBOL DLL that calculates the result. The result will be displayed to the right.
End

The form accepts two numbers and displays the result of multiplying those numbers when the "=" button (Command1) is pressed.

The Command1 code invokes the Sample13 DLL created using NetCOBOL:



```
Private Declare Sub SAMPLE13 Lib "sample13.DLL" (ar%, br%, ByVal C As
Private Declare Sub JMPCINT2 Lib "f3biprct.dll" ()
Private Declare Sub JMPCINT3 Lib "f3biprct.dll" ()
Private Sub Command1_Click()

    Static C As String * 11
    a = Text1.Text
    ar% = Val(a)
    b = Text2.Text
    br% = Val(b)
    ChDir App.Path            ' set path to where sample13.exe loaded
    Call SAMPLE13(ar%, br%, C)
    Text3.Text = C

End Sub
```

There are two other pieces of code you should note, although we will not discuss them further in this guide. They are the calls to JMPCINT2 and JMPCINT3. These calls initialize and close the COBOL run-time system.

1. Execute the Visual Basic code by repeating the Debug, Step Into function (available on the Visual Basic Debug menu, and the Debug toolbar).

2. When the Sample13 dialog displays, enter two numbers and click on the "=" button.

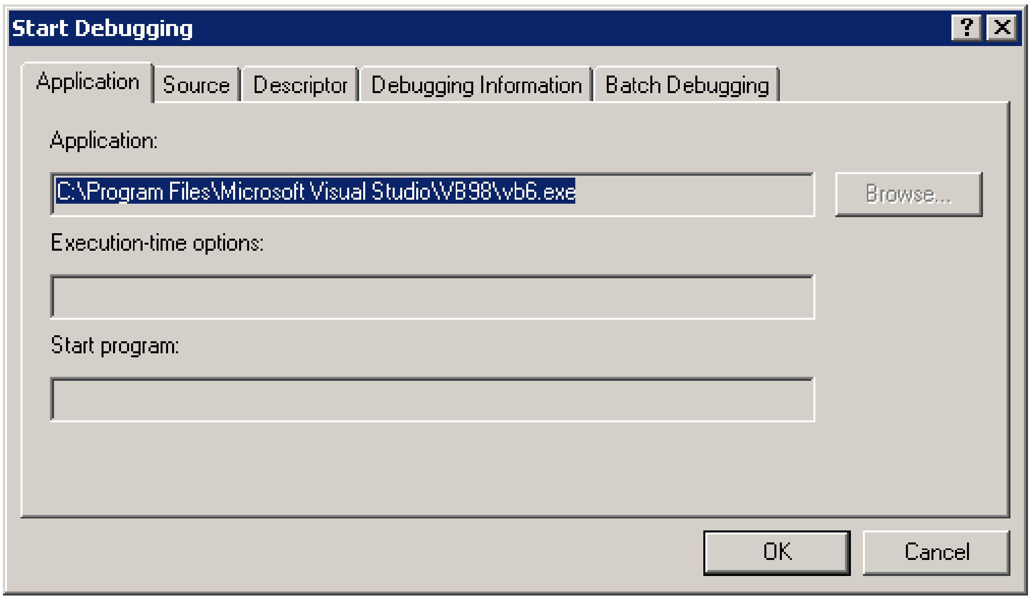
3. Continue stepping up to, and into, the statement:

   Call SAMPLE13 (ar%, br%, C)

Debug the COBOL Code

1. Control passes to the COBOL code. The runtime system recognizes that the COBOL debugger should be invoked (because the @CBR_ATTACH_TOOL environment variable is set) so starts the debugger.

   The debugger displays the Start Debugging window.



Application:

C:\Program Files\Microsoft Visual Studio\VB98\vb6.exe

Execution-time options:

Start program:

2. The debugger displays "VB6.EXE" as the application being executed. This is the application to which the debugger attaches. You need to be aware of this because you cannot close the debugger until you close Visual Basic.

   You need to confirm two locations to the debugger: on the Source and Debugging Information tabs.

3. Select the Source tab and click on the Browse button by the "Source file storage folders" entry field.

   The debugger displays the "Browse for Folder" dialog.

4. From the folder list select

   C:\Program Files\Fujitsu NetCOBOL for Windows\Samples\cobol\sample13

5. Select the Debugging Information tab and click on the Browse button.

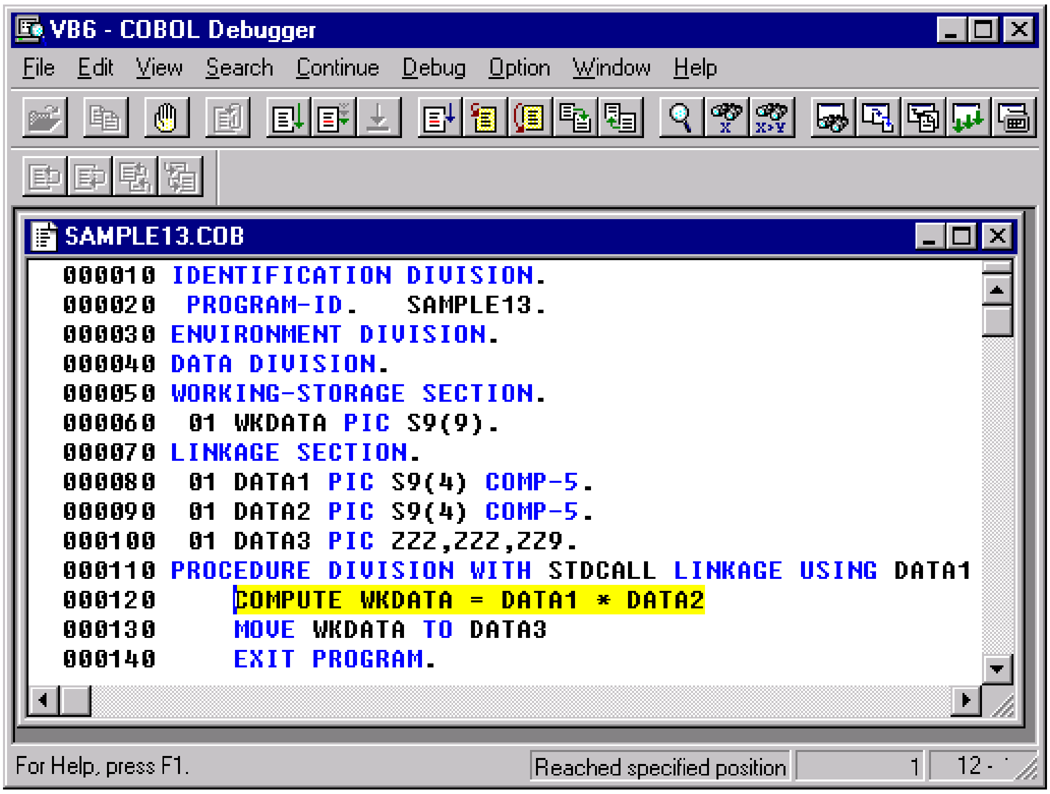
   The debugger displays the "Browse for Folder" dialog.

6. Again select:

   C:\Program Files\Fujitsu NetCOBOL for Windows\Samples\cobol\Sample13

7. You have now ensured that the debugger can find the COBOL source files and debugging information.

8. Click on OK to display the source in the COBOL debugger.



```
000010 IDENTIFICATION DIVISION.
000020  PROGRAM-ID.    SAMPLE13.
000030 ENVIRONMENT DIVISION.
000040 DATA DIVISION.
000050 WORKING-STORAGE SECTION.
000060  01 WKDATA PIC S9(9).
000070 LINKAGE SECTION.
000080  01 DATA1 PIC S9(4) COMP-5.
000090  01 DATA2 PIC S9(4) COMP-5.
000100  01 DATA3 PIC ZZZ,ZZZ,ZZ9.
000110 PROCEDURE DIVISION WITH STDCALL LINKAGE USING DATA1
000120     COMPUTE WKDATA = DATA1 * DATA2
000130     MOVE WKDATA TO DATA3
000140     EXIT PROGRAM.
```

For Help, press F1.   Reached specified position   1   12 -

You can use all the COBOL debugging functions to debug this program. We will simply step through its three statements.

9. On the COBOL debugger toolbar, click the Step Into button, three times.

   The debugger steps through the code. When the EXIT PROGRAM statement is executed, control returns to the Visual Basic debugger (the Visual Basic button on the task bar may flash to indicate you need to switch control to that window).

10. Continue stepping through the Visual Basic code, and select the Sample 13 window to see the result displayed.

Through this simple example you have seen how complex mixed Visual Basic and COBOL applications can be debugged effectively.

1.2.3 Working with OO COBOL

A. Brief Introduction to OO COBOL

The "NetCOBOL User's Guide" contains a full introduction to the Fujitsu OO COBOL implementation. The following comments are intended for those who only want the minimum of information before looking at an actual OO COBOL application. The comments assume a familiarity with OO concepts and terminology.

Fujitsu OO COBOL provides: class/object models, information hiding (data encapsulation, object properties), modularization (methods, including prototype methods), single and multiple inheritance, context-dependent functions (polymorphism), conformance checking, static and dynamic binding, garbage collection, and exception handling.

All these features are provided by very natural extensions to the COBOL language.

In OO COBOL:

- Classes are made up of a Factory object and Class objects.
- The Factory object contains data and methods required either for creating and maintaining objects, or for objects that only require one instance per class.
- The Class objects contain the data and methods that define the entity that the class models.
- Methods can have their own local data as well as procedure code.
- Methods are invoked using the INVOKE statement.

- Each of the main constructs "CLASS-ID", "FACTORY", "OBJECT" and "METHODID" are defined using appropriate combinations of the four standard COBOL divisions (IDENTIFICATION, ENVIRONMENT, DATA and PROCEDURE).
- Class definitions are stored in REPOSITORIES.
- Non-OO COBOL applications can invoke OO programs and OO code can call non-OO programs.

B. Guided Tour of an OO COBOL Application

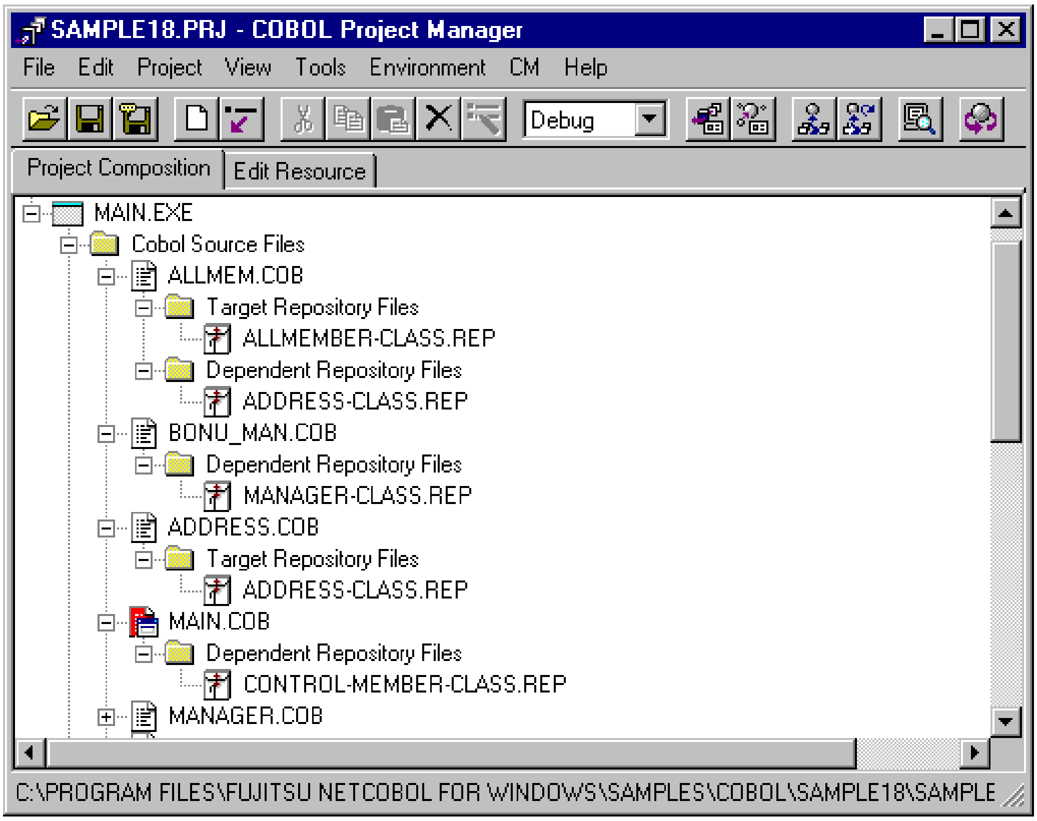
This tour uses one of the OO COBOL samples, Sample18, to give you quick views of the support for OO COBOL within NetCOBOL. You will look at Sample18 in the Project Manager, in the source editor, in the Project Browser and in the Class Browser.

First we'll look at Sample18 within the COBOL Project Manager. Remember that Project Manager presents an application build structure along with access to most of the NetCOBOL development tools.

1. Start COBOL Project Manager (from the Windows task bar select Start, Programs, Net COBOL, COBOL Project Manager).
2. From the Project Manager menu bar select File, Open and navigate to the:

   C:\Program Files\Fujitsu NetCOBOL for Windows\Samples\cobol\Sample18 folder.
3. Select and Open sample18.prj.
4. Expand the MAIN.EXE, Cobol Source File, ALLMEM.COB, BONU_MAN.COB, ADDRESS.COB, and MAIN.COB nodes by clicking on the plus, icons.

   The sample18 project tree looks like this:



```
⊟ MAIN.EXE
  ⊟ Cobol Source Files
    ⊟ ALLMEM.COB
      ⊟ Target Repository Files
          ALLMEMBER-CLASS.REP
      ⊟ Dependent Repository Files
          ADDRESS-CLASS.REP
    ⊟ BONU_MAN.COB
      ⊟ Dependent Repository Files
          MANAGER-CLASS.REP
    ⊟ ADDRESS.COB
      ⊟ Target Repository Files
          ADDRESS-CLASS.REP
    ⊟ MAIN.COB
      ⊟ Dependent Repository Files
          CONTROL-MEMBER-CLASS.REP
    ⊞ MANAGER.COB
```

C:\PROGRAM FILES\FUJITSU NETCOBOL FOR WINDOWS\SAMPLES\COBOL\SAMPLE18\SAMPLE

The tree is similar to those for standard COBOL applications. It shows that MAIN.EXE is built from several COBOL source files (ALLMEM.COB, BONU_MEM.COB, etc.) and uses .DLLs defined in COLLECT.LIB.

The tree is different from those for standard COBOL applications in that repositories have been added to the tree. These nodes show the repositories that are either created by a particular source file (target repositories) or that are referenced by the source file (dependent repositories).

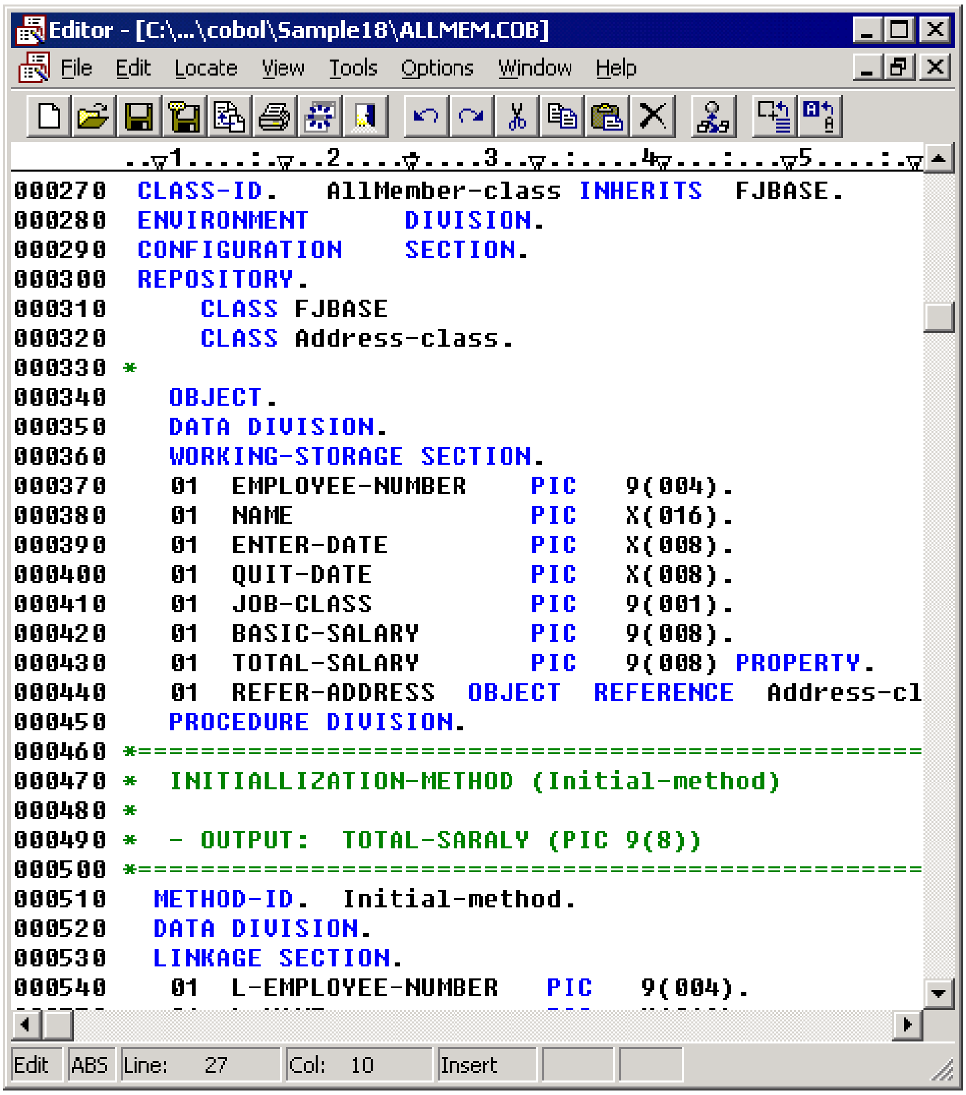
Once you specify an application's source structure you can have Project Manager figure out the repository information by using the "Repository File Search" function on Project Manager's Edit menu.

The repository information helps identify the function of the different source files. For example:

- MAIN.COB is a standard COBOL program that uses an OO class system for its functions. It therefore references several different classes.
- ADDRESS.COB is a class definition, hence it has a target repository. It only inherits from the FJBASE root class so it does not have any dependent repositories.
- ALLMEM.COB has a target repository and is therefore a class definition. It references an item in the ADDRESS class so has a dependent repository.

Now we'll take a quick look at some OO COBOL source.

5. Double-click on ALLMEM.COB.
6. Scroll past the initial (green) comments to the (blue and black) COBOL source.



```
000270 CLASS-ID.   AllMember-class INHERITS  FJBASE.
000280 ENVIRONMENT       DIVISION.
000290 CONFIGURATION     SECTION.
000300 REPOSITORY.
000310     CLASS FJBASE
000320     CLASS Address-class.
000330 *
000340   OBJECT.
000350   DATA DIVISION.
000360   WORKING-STORAGE SECTION.
000370   01  EMPLOYEE-NUMBER     PIC   9(004).
000380   01  NAME                PIC   X(016).
000390   01  ENTER-DATE          PIC   X(008).
000400   01  QUIT-DATE           PIC   X(008).
000410   01  JOB-CLASS           PIC   9(001).
000420   01  BASIC-SALARY        PIC   9(008).
000430   01  TOTAL-SALARY        PIC   9(008) PROPERTY.
000440   01  REFER-ADDRESS  OBJECT  REFERENCE  Address-cl
000450   PROCEDURE DIVISION.
000460 *===================================================
000470 *  INITIALLIZATION-METHOD (Initial-method)
000480 *
000490 *  - OUTPUT:  TOTAL-SARALY (PIC 9(8))
000500 *===================================================
000510  METHOD-ID.  Initial-method.
000520  DATA DIVISION.
000530  LINKAGE SECTION.
000540   01  L-EMPLOYEE-NUMBER    PIC   9(004).
```

Note

- The CLASS-ID paragraph. This is actually part of the IDENTIFICATION DIVISION but the IDENTIFICATION DIVISION header is optional so you will often see a new code structure starting with its identifying paragraph (CLASS-ID, FACTORY, OBJECT, or METHOD-ID).
- The INHERITS clause in the CLASS-ID paragraph indicating the class this class inherits from - in this case the system base class FJBASE.

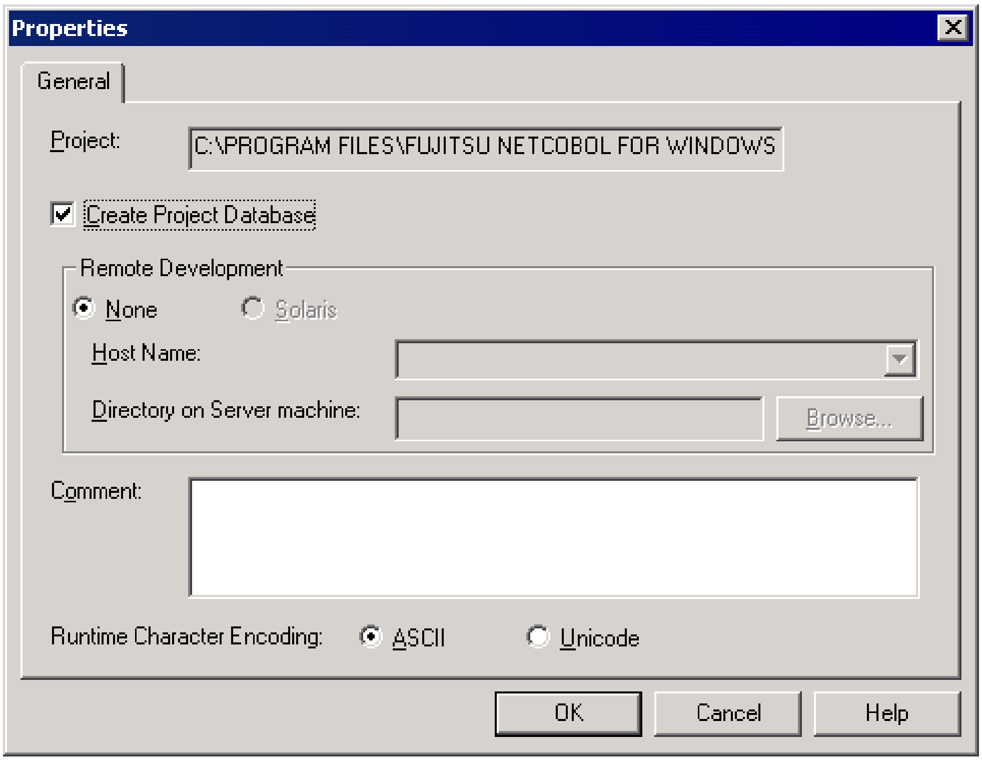
- The methods starting with METHOD-ID, ending with END METHOD, containing data and procedure code.
- At the end of the source file, the END OBJECT and END CLASS, delimiters.

Although some new elements have been introduced, the general structure and layout should be familiar to COBOL programmers - particularly those who have experience with nested programs.

1. Close the editor and return to COBOL Project Manager.

   When developing an OO COBOL application, people within a project group will likely want to review relationships between source and class elements, without having to release the classes to the whole development organization. The Project Browser tool has been provided for this purpose.

2. First indicate that a project information database should be created, by selecting Properties from the menu bar and confirming that "Create Project Database" is checked on the Properties dialog.



Properties

General

Project: C:\PROGRAM FILES\FUJITSU NETCOBOL FOR WINDOWS

☑ Create Project Database

Remote Development

● None ○ Solaris

Host Name:

Directory on Server machine:

Comment:

Runtime Character Encoding: ● ASCII ○ Unicode

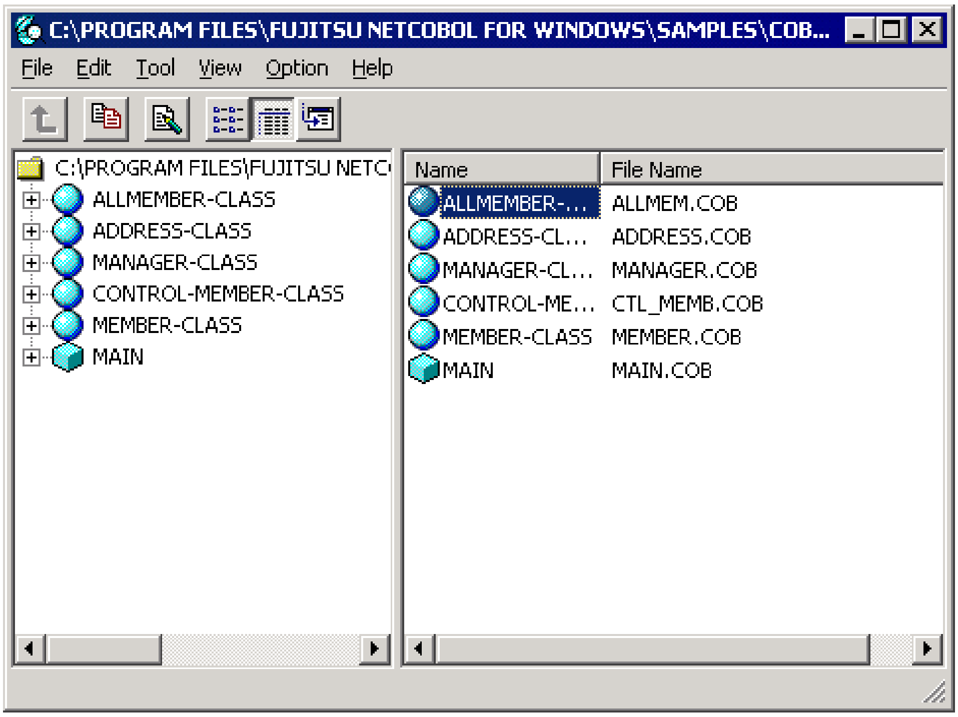
   If it is not checked, click on the "Create Project Database" check box.

3. Build the project information database by right-clicking on SAMPLE18.PRJ in the project tree, and selecting Build from the pop-up menu.

   Assuming that no other updates have been made to the project, Project Manager does no compiles or links, but just displays a "Terminating database file creation" message.

4. You can invoke the Project Browser by selecting, Tools, Project Browser from the Project Manager menu bar.

   Project Manager invokes the Project Browser tool with the SAMPLE18 project already loaded:



C:\PROGRAM FILES\FUJITSU NETCOBOL FOR WINDOWS\SAMPLES\COB...
File
Edit
Tool
View
Option
Help

C:\PROGRAM FILES\FUJITSU NETC
- ALLMEMBER-CLASS
- ADDRESS-CLASS
- MANAGER-CLASS
- CONTROL-MEMBER-CLASS
- MEMBER-CLASS
- MAIN

Name	File Name
ALLMEMBER-...	ALLMEM.COB
ADDRESS-CL...	ADDRESS.COB
MANAGER-CL...	MANAGER.COB
CONTROL-ME...	CTL_MEMB.COB
MEMBER-CLASS	MEMBER.COB
MAIN	MAIN.COB

   The left pane provides a tree view of the classes being used by the project.

   The right pane provides details about the item selected in the left pane.

5. Expand one of the nodes in the left-hand pane, for example the ALLMEMBERCLASS.

   The Project Browser expands the tree to show classes that inherit from this class, referenced classes, methods, properties and inherited methods. See the Project Browser Help for a description of the different icons. Sharp icons denote items defined within the selected class, faded icons denote items defined outside the class (generally these are inherited items).

6. Select different items in the left-hand pane and note the information that is presented in the right-hand pane, such as arguments required for methods and data descriptions of properties.

   Notice that the Project Browser includes non-OO elements, like the MAIN program - an essential part of the project, but not part of the OO system.

   Now we'll switch to the Class Browser for a slightly different view on this application.

7. Close the Project Browser and return to the COBOL Project Manager, where we will create a class information database.

   A class information database combines information from several class repositories for display by the Class Browser.

8. From the Project Manager menu bar, select Tools, Class Database, Options.

   Project Manager displays the dialog titled "Set the Class Database Generation Options". Its purpose is for you to list all the folders containing repositories that you wish to include in your class information database, and for you to specify the name and location for the class information database.

   We will only specify the Sample18 folder for source repositories.

9. Click on the Add button in the "Repository folder" group box.

   Project Manager displays the "Browse for Folder" dialog.

   (Note - this dialog starts browsing at the desktop as it assumes that your repositories will often be located on a central server.)

10. Navigate to the:

    C:\Program Files\Fujitsu NetCOBOL for Windows\Samples\cobol\Sample18 folder, select it, and click OK.

The sample18 folder is added to the list.

11. Click on the Browse button in the "Class database file" group box and navigate to the:

    C:\Program Files\Fujitsu NetCOBOL for Windows\Samples\cobol\Sample18 folder. Then enter "samples" in the "File name" entry field of the Open window, and click on the Open button.

    A class information database called "samples.cid" will be created in the sample18 folder - normally you would place this file in a central location accessible to all developers.

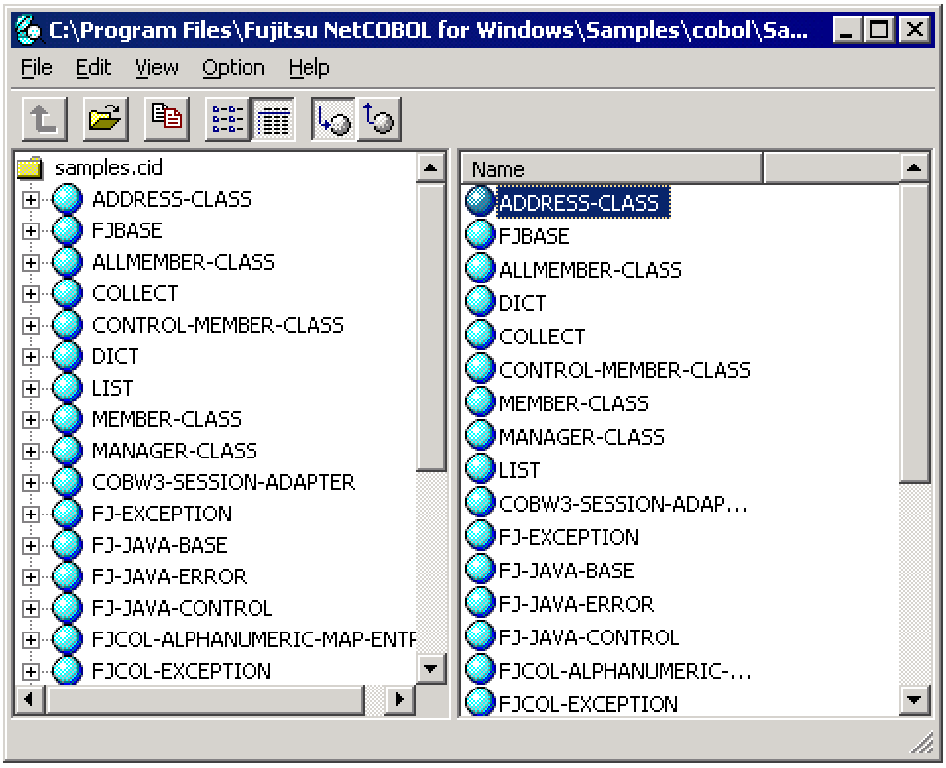
12. Click OK to close the "Set the Class Database Generation Options" dialog.

13. Create the class information database by selecting Tools, Class Database, Create from the Project Manager menu bar.

    Project Manager creates the class information database and displays a message saying "Class database created successfully".

14. You can now invoke the Class Browser by selecting Tools, Class Browser from the Project Manager menu bar.

    Project Manager invokes the Class Browser, displaying information from the Samples class information database.



The Class Browser has a similar appearance to the Project Browser with the following differences:

- Class Browser only displays OO class information.
- Classes within class browser can come from many different projects and application areas.
- Class Browser does not display source file name information.
- You can select whether Class Browser should expand classes up or down the class hierarchy.

15. Experiment with browsing the Sample18 classes. Note how the Class Browser helps you understand the relationships between the classes and the interfaces available in each class.

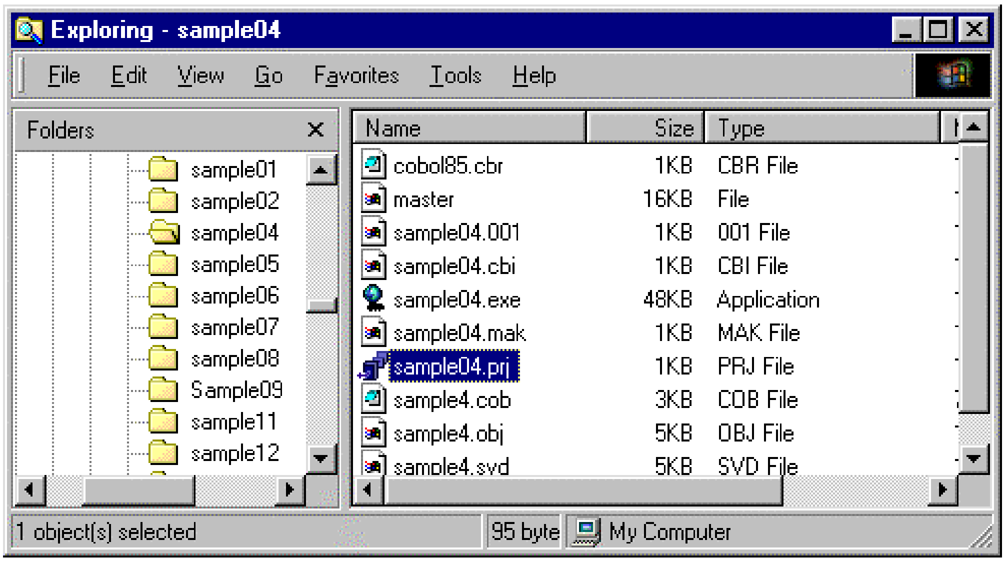
For more information on OO COBOL consult chapters 14 - 17 in the "NetCOBOL User's Guide".

1.2.4 Invoking COBOL Project Manager from Windows Explorer

COBOL Project Manager can be invoked not only from the Start, Programs menu (as discussed previously) but also from Windows Explorer.

You do this by double clicking the mouse on a project file and the COBOL Project Manager starts, displaying the project you selected.

For example, double click on SAMPLE4.PRJ in the Windows Explorer window.



Name	Size	Type
cobol85.cbr	1KB	CBR File
master	16KB	File
sample04.001	1KB	001 File
sample04.cbi	1KB	CBI File
sample04.exe	48KB	Application
sample04.mak	1KB	MAK File
sample04.prj	1KB	PRJ File
sample4.cob	3KB	COB File
sample4.obj	5KB	OBJ File
sample4.svd	5KB	SVD File

This starts COBOL Project Manager with SAMPLE4.PRJ loaded.

1.2.5 Converting Version 3 Projects

If you open a P-STAFF Version 3 project with NetCOBOL or Fujitsu COBOL (Version 4 or later), COBOL Project Manager will convert the project into the current project format.

This is true whether you open the project from COBOL Project Manager or from Windows Explorer.

1.2.6 Other Options

COBOL Project Manager provides many other functions such as:

- Invoking various tools individually from the Tools menu including the compiler, linker, debugger and editor.
- Integrating pre-compilers into your system.
- Working with CORBA objects (Object Director).
- Configuring certain parts of your environment.

For details of these functions see the on-line help for COBOL Project Manager and the “NetCOBOL User’s Guide”.

Chapter 2 Developing GUI Applications

This chapter sets the context for developing GUI applications in COBOL. It is descriptive in nature, providing a background for those who like to know more details.

2.1 A Brief History

It is generally agreed that COBOL programs make up anywhere from 50-75% of the world’s current business applications. Today COBOL programmers need a modern, powerful, and easy to use development environment that fully exploits the Microsoft Windows family of operating systems for both development and production.

The Microsoft Windows family of operating systems has become the preferred end user platform for the vast majority of PC users. A number of pitfalls arose, however, as traditional COBOL programmers from the mainframe world attempted to embrace this exciting new platform.

Primarily, the intense time and complexity of low level Windows API (application programming interface) programming prevents most developers from creating sophisticated graphical user interfaces using traditional 3GL (3rd generation language) compilers and tools.

Two approaches have been developed to deal with this problem. The first is to extend the COBOL language itself to encompass GUI capability directly in the language. This approach offers the advantage of using a single COBOL development environment, and keeping the entire application in native COBOL. Fujitsu’s flagship PowerCOBOL product is the best example of such an approach.

The second approach is to separate the user interface from the actual COBOL application and have it managed by a wholly separate facility. Some COBOL vendors have attempted to deliver such a combination, but these have fallen short for a variety of reasons. Primarily, these environments have proven to be highly unstable, lacking in many features, and very proprietary in nature. Additionally, these environments have been in a constant state of flux as vendors attempted to deliver better stability and a wider array of functionality. The results have typically been less than successful for developers.

In recognizing these challenges, Fujitsu has developed the world’s premier COBOL GUI and client/server application development environment -- Fujitsu’s flagship PowerCOBOL product line.

Fujitsu has also recognized the need for a world class COBOL development product that will integrate, coexist and exploit Microsoft’s Visual Basic development environment.

If you want to develop full-fledged GUI applications, you should explore Fujitsu’s PowerCOBOL product. If you have the Professional or Enterprise editions of NetCOBOL, you have probably already installed the product. If you have NetCOBOL Standard edition, you can check the NetCOBOL web site (www.netcobol.com) for more details. You may additionally use Microsoft’s Visual Basic (version 4.0 or later) with this product to develop GUI applications.

Visual Basic was an attempt to bring out a 4GL-like development environment that would allow programmers and even non-programmers to develop GUI applications in the Windows environment. Users could visually ‘paint’ the user interface, and then create the rest of the application by writing small snippets of code using a proprietary scripting language that was loosely based upon the BASIC programming language.

Initially, Visual Basic was lacking for serious business application development. Microsoft has continued to evolve Visual Basic over the years into a serious development environment. Today it is a strong development tool for a wide number of professional developers who want to rapidly create GUI applications for the Windows family of platforms.

For the users of COBOL applications, however, Visual Basic presents a number of substantial challenges:

- Having no relationship to COBOL, Visual Basic presents an alien development environment to COBOL programmers. They must learn an entirely new development methodology and language.
- Visual Basic does not provide all of the facilities found in the COBOL language.
- Prior to NetCOBOL, Visual Basic could not easily be inter-mixed with COBOL programs to combine the best of both worlds, and to allow the re-use of existing COBOL programs.

Many COBOL programmers have requirements to re-host applications (move COBOL applications from another platform such as the mainframe) to the PC environment. At the same time they want to re-design traditional text-based interfaces to take advantage of GUI.

While some COBOL vendors attempted to provide ‘bridges’ to allow Visual Basic to intermingle with COBOL, these approaches typically fell short.

One of the major inhibiting factors in inter-mixing other vendor’s COBOL with Visual Basic related to the fact that each had it own separate and complex run-time system. Mixing two very proprietary run-time systems is a substantial undertaking. Most legacy run-time systems were architected and developed by their respective vendors long before PC’s, Windows and GUI’s were even invented.

Fujitsu Addresses the COBOL Need

Over the years Microsoft has recognized the need to support COBOL applications and has made multiple attempts to provide products to the COBOL community. Their original COBOL compiler was replaced by Micro Focus’ COBOL/2 product in 1988. The COBOL/2 product suffered from many of the aforementioned pitfalls, and Microsoft withdrew it from the market in June of 1993.

Fujitsu recognized this dilemma and set out to build a next generation COBOL development environment from the ground up. A special emphasis was placed upon the design of the run- time system to alleviate the above noted problems.

Additionally, Fujitsu developed a tool to aid in the conversion of COBOL/2 applications to NetCOBOL (while COBOL is a standard language, COBOL/2 contained a significant number of proprietary extensions).

2.2 NetCOBOL

NetCOBOL is a complete COBOL development environment that allows you to create standalone COBOL applications and/or COBOL components for use with Microsoft Visual Basic and Visual C++ applications.

The COBOL Project Manager is a 32-bit project-oriented development environment that provides integrated access to an editor, compiler, interactive debugger, execution environment, and other supporting tools. A sophisticated data file editor is also included. It supports all COBOL data file types, and provides editing, conversion, printing, sorting, reorganizing and recovery capabilities.

For previous users of Micro Focus’ or Microsoft’s COBOL/2, dialect conversion products are available to aid in converting COBOL programs using some of the Micro Focus proprietary extensions to run under NetCOBOL. See the www.adtools.com Web site for more details.

Fujitsu PowerBSORT OCX is also included. PowerBSORT OCX is callable from any Windows application that supports OCX’s (for example, Microsoft Visual Basic).

PowerBSORT OCX online help is provided along with sample applications for Visual Basic 4.0 and 5.0. The online help and sample files are automatically installed along with PowerBSORT OCX.

2.3 Event-driven Programming

In order to begin developing GUI COBOL applications in the Windows environments, it is important to understand the concept of event -driven programming.

While the term ‘event driven’ may be somewhat new in the computer industry, the concept has been around for a long time, and chances are that if you are a COBOL programmer, you already understand it.

When COBOL programmers first began developing mainframe-based applications that interacted with users (e.g. non-batch style applications), they were constrained to simple line-oriented ACCEPT/DISPLAY syntax.

Later, on-line transaction monitoring systems such as IBM’s CICS and IMS/DC enhanced this capability to handle full screen-oriented interaction (as opposed to line oriented). Some PC COBOL vendors later enhanced the COBOL ACCEPT/DISPLAY verbs in non-standard ways to provide full screen interaction.

While one could argue that none of these techniques qualified as event-driven programming, they were in fact, a primitive form of this approach.

Event-driven programming breaks the user interface portion of a GUI application (screen I/O, keyboard I/O, mouse I/O, and a few other possibilities), into a series of possible events.

These events include user actions such as pressing a key on the keyboard, moving the mouse, clicking the mouse on a screen object such as a button, holding the mouse button down and dragging the mouse to move a window, and so forth.

Writing a proper Windows GUI application entails creating all of the potential windows and objects that the user may interact with (including output objects as well), determining all of the possible events that may occur, and then creating a link between each potential event and your application code to handle the event.

What you aim to end up with is an application that registers its user interface with Windows (in much the same approach as CICS and IMS/DC applications register themselves in the mainframe world). Once the application is registered, Windows recognizes events and sends event driven messages to the application for resolution.

The registration and event handling processes are based upon the Windows message queue paradigm.

The Windows Message Queue

The Windows message queue is an exceptionally complex topic, and will be discussed only briefly to give you a basic understanding.

Much like CICS or IMS/DC (via VTAM) on a mainframe, Windows controls all input from the screen, mouse and keyboard. It must then implement a technique to manage all of this and ensure that any event that takes place is routed to the proper application. For applications that wish to update system I/O resources such as the screen, Windows also needs a mechanism to enable applications to alert it (Windows) to do so.

Windows provides these services by implementing a message queue. Every Windows application must register itself with Windows when it activates. As the user interacts with the system, messages are routed into each application as appropriate. Because Windows supports multiple applications running concurrently, this proves to be a very busy part of the operating system.

This means that any Windows application registers itself with Windows, and then goes into a recurring ‘message loop’ - simply reading in messages sent to it by the operating system and reacting as appropriate. This continues until the application receives whatever it has defined to be a ‘terminate application’ type of message and then it terminates itself.

It’s important to understand that an application can easily receive thousands of Windows event messages in a short period of time. For example, when a user moves the mouse around the screen, multiple messages are generated to alert the application whose window(s) the mouse passes over. These messages not only alert the application that the mouse has moved over the application’s window, but additionally provide information such as the exact location (x, y coordinates) where the mouse moved - even though the user did not press any of the mouse buttons!

If your application does not contain code to handle a certain message then it may lock up or be abended. Fortunately, a common programming technique is to provide a catchall routine that simply ignores all messages other than the ones you’ve defined.

Hopefully, you can begin to envision the Windows application environment where messages are being sent to applications as the user interacts with the system. The applications perform actions based upon the messages received and dispatch messages back to Windows to perform user interface operations (for example, close a window, create a new window, update the screen with the new data being passed along, etc.).

In a low level Windows application, developers are forced to write extensive amounts of code to deal with all of the possible events that may occur, manage the internal message loop, and take care of a great deal of tedious work. This work may even include being forced to re-paint the screen because another application’s window was placed on top of your application’s window. (Windows does not automatically keep track of the layering of application windows and will not automatically restore the prior contents of a window that was briefly overlaid.)

How PowerCOBOL and Visual Basic Simplify Application Development

One of the goals of Fujitsu’s PowerCOBOL and Microsoft’s Visual Basic is to effectively hide from the developer the tedious task of managing the Windows message queue and an application’s message loop.

Using PowerCOBOL or Visual Basic, a developer need only paint the application’s windows and contained objects (for example, buttons, list boxes, text fields, etc.). PowerCOBOL users may program these windows in intuitive COBOL code, while Visual Basic developers define (in a high level scripting language) how they are to interact with the user and the application. The developer then decides which potential events he or she cares about (for example, a button being pushed by a mouse click), and writes the code to handle each such event.

PowerCOBOL actually generates a great deal of the COBOL source code for GUI events and places it automatically into the COBOL program.

This approach frees the programmer from the tedious nature of low level Windows API message queue programming in order to concentrate on higher level aspects of the application.

Both PowerCOBOL and Visual Basic generate code for the application to handle any potential events that the developer forgets or simply does not care to define event code for.

Because some of this code can be quite tedious and complex (for example, re-sizing a window and re-positioning all of its objects), both PowerCOBOL and Visual Basic become great allies to the application developer who needs to field solid, fully functional GUI applications.

When developing these types of applications using Visual Basic, users must be very familiar with Visual Basic’s scripting language, which is very extensive and can be quite complex at times.

Additionally, developers sometimes run into development scenarios where Visual Basic may not provide a certain capability or provides it in a somewhat tedious manner versus another programming language such as ‘C’ or COBOL.

Lastly, developers often have legacy applications available that they would like to encompass in a GUI application. These legacy applications are often written in something other than Visual Basic script. This means that Visual Basic needs to be able to call programs written in languages other than its own scripting language. This is exactly where NetCOBOL comes into play.

By providing the capability to associate potential events in a Visual Basic controlled application to COBOL programs, the development environment is greatly extended. COBOL programmers can write sophisticated GUI applications with a minimal amount of knowledge of Visual Basic’s scripting language. They may additionally re-use existing COBOL code with little or no change within a Visual Basic application as well.

2.4 Using NetCOBOL

Creating COBOL applications to run standalone or under Visual Basic is a flexible and straightforward process. Developers typically follow the steps below:

- Application analysis and design.
- For GUI applications, creation and testing of the user interface using Visual Basic’s development environment.
- Creation and testing of the COBOL component(s) of the application using the COBOL Project Manager development environment.
- Testing and continued development of the overall application that is run under the initial control of Visual Basic when it is a GUI application.

It is worth noting that both Visual Basic and NetCOBOL offer extensive dynamic development environments, making it easy to move between the two when creating and testing applications.

Additionally, both environments provide extensive development support tools to aid the process.

NetCOBOL standalone .EXE files (executables) or .DLL files (dynamic link libraries) do not have to be physically linked into a Visual Basic application and may be called dynamically at run- time by Visual Basic.

Doing away with the requirement to link all of this together allows you to work offline on either end of the application. That is, you may work on and test the user interface under Visual Basic, or work on the COBOL component(s) in the COBOL Project Manager without having first gone through Visual Basic.

Because of the dynamic nature of the two development environments, you do not necessarily have to follow the above steps in order. You could first develop and test the COBOL components and then design the user interface in Visual Basic. The important point is that when it comes time to execute the overall application, you should have the interface and the COBOL components available.

Chapter 3 Working with Visual Basic and COBOL

This chapter teaches you how to mix COBOL and Visual Basic. It starts by walking you through your first Visual Basic COBOL application it then precedes to the details of integrating COBOL with Visual Basic.

3.1 Creating Your First Visual Basic COBOL Application

In this section, you will learn how to create a COBOL application that uses Visual Basic as a graphical front end.

It is assumed that you have already installed Visual Basic, version 5.0 or higher on your machine. If you have not, please do so before continuing this section.

It is also assumed that you have gone through the Visual Basic Tutorial entitled “Your First Visual Basic Application,” located in the “Visual Basic Programmer’s Guide.” This will help you understand the terminology used later in this chapter.

An Overview of the Visual Basic COBOL Application

The application you are going to develop will be a simple graphical window with a text box and two command buttons.

It will be named HICOBOL (short for "Hello from COBOL").

The text box will be assigned a text value by Visual Basic. When selected, one of the command buttons will call your HICOBOL program and pass two parameters.

One parameter is a numeric value. The COBOL program will set this value to 100 each time, but it will be ignored by the Visual Basic part of the application (this parameter is defined to illustrate the technique). You may enhance the application on your own to use this value as appropriate.

The second parameter is a character string. The COBOL program will always set this to "Hello from COBOL". When the COBOL program exits, this value will be returned to Visual Basic and the graphical text box will be updated with this new string to illustrate the interaction between Visual Basic and COBOL.

This will happen each time a user clicks on the command button. The second command button will cause the graphical text box to be reset to display a different string.

How NetCOBOL Interacts with Visual Basic

NetCOBOL was created to work well with other language environments from a runtime perspective.

Most programming API’s (application programming interfaces) available on the Windows platforms support a C-style programming interface. This includes not only operating system services such as GUI interface components, but also additionally a vast array of other programming interfaces. These include database systems, client/server messaging and transaction systems, and even end user applications.

The arrival of NetCOBOL has brought seamless integration with operating system services and other languages to COBOL developers. Visual Basic integration is but one example of the excellent mixed language programming support found in NetCOBOL.

The integration point between NetCOBOL and Visual Basic is the creation of a DLL (dynamic link library) file from NetCOBOL.

Instead of producing a standalone executable (EXE) file within the COBOL development environment, you instead produce a DLL file. You can think of a DLL file as a dynamically callable COBOL subprogram, which does not have to be physically linked to a program that calls it and may even pass data back and forth.

To create a proper DLL, the linker needs to be given the name of the DLL to be produced, and the names to export.

Exported names are the names of entry points contained within the DLL file which are to be made public to allow other applications to call them after the DLL has been identified to the operating system.

You are going to create a simple DLL file called HICOBOL.DLL, which will contain a single exported entry point - the program name. The program name entry point will cause a calling program to enter the program at its first executable statement in the PROCEDURE DIVISION.

Once you have written the HICOBOL.COB program and have created the HICOBOL.DLL executable, you will exit NetCOBOL and enter Visual Basic to design the graphical interface.

While in Visual Basic, you will create a graphical user interface (GUI) for your COBOL application, which will call the HICOBOL.DLL program. Data will be passed back and forth as well.

To identify the NetCOBOL DLL to the Visual Basic program, you will code three simple Visual Basic DEFINE statements. One is for the NetCOBOL run-time load program. Another is for the NetCOBOL run-time unload program. The final DECLARE statement is for the HICOBOL.DLL executable you are going to create.

In order to get a NetCOBOL DLL program to work with Visual Basic, you need to first issue a single call from Visual Basic to ensure that the COBOL run-time support is properly loaded.

You then code a single Visual Basic statement to call the HICOBOL.DLL program and pass data to it.

Finally, you code a call to unload the COBOL run-time support when finished.

That's all there is to it.

From an architectural standpoint, when you develop future, more complex Visual Basic COBOL applications you will arrange things slightly differently than in the following example. The main reason for this is that you only need to load the NetCOBOL run-time once at the start of your application, and issue the unload call just before you exit your Visual Basic application.

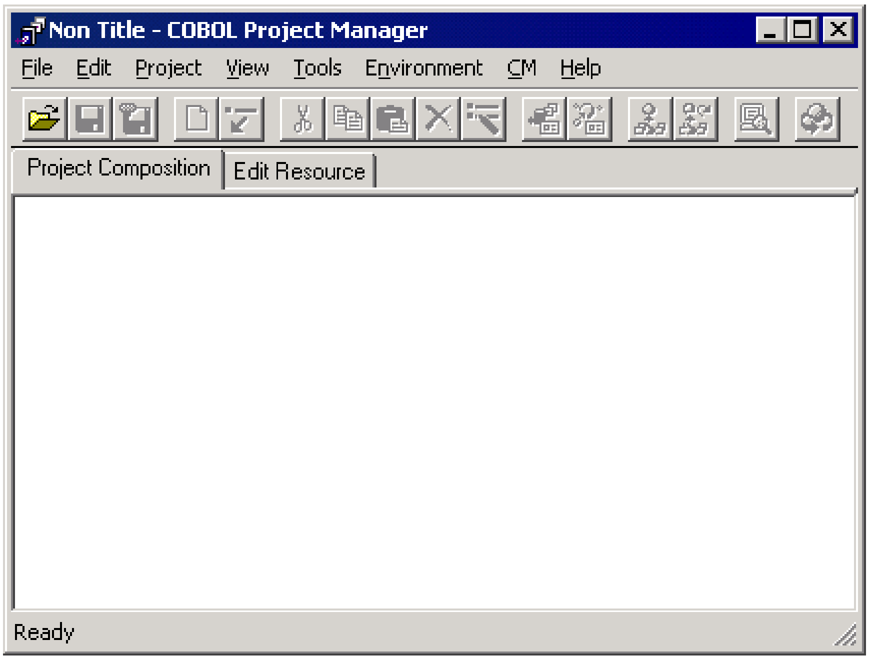
To keep the example simple and straightforward, you are going to code all three calls together (load the COBOL run-time, call HICOBOL, unload the COBOL run-time).

Developing the COBOL Portion of the Application

In this section you are going to develop the HICOBOL.DLL executable which will be called by Visual Basic.

You are going to use the COBOL Project Manager to assist you.

1. Start the COBOL Project Manager; the following window appears:


Non Title - COBOL Project Manager
File Edit Project View Tools Environment CM Help
Project Composition
Edit Resource
Ready

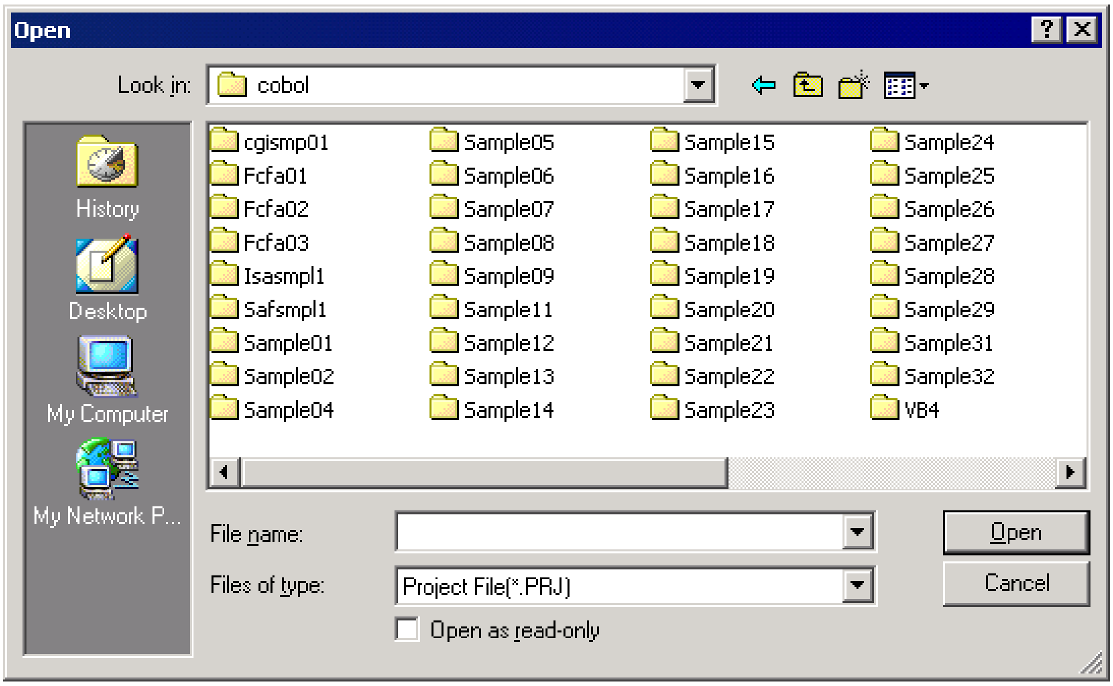
2. From the Project menu, select Open Project.

   Project Manager displays the Open dialog.

   We will use this dialog to create a new folder (sub-directory) to store the application you are about to develop.

3. Navigate to the

   C:\Program Files\Fujitsu NetCOBOL for Windows\Samples\Cobol folder.



Look in: cobol

File name:

Files of type: Project File(*.PRJ)

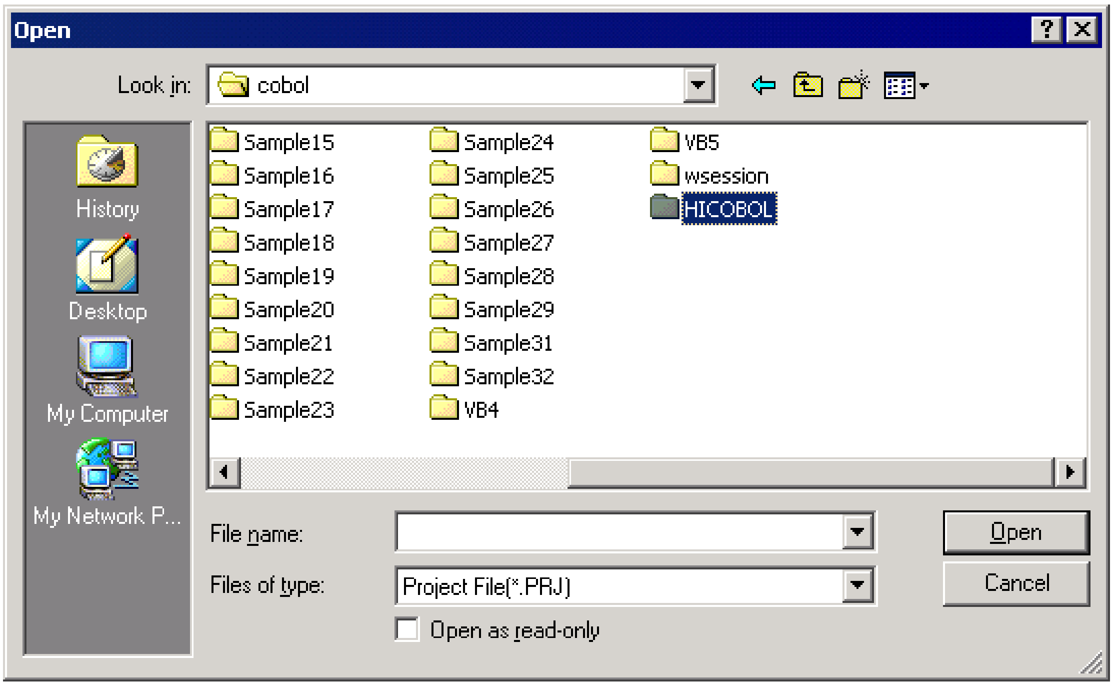
□ Open as read-only

4. Locate and click on the Create New Folder button to the upper right of the dialog.

This creates a new folder in the Samples folder, called “New Folder”, with the text already selected.

5. Change the name of the folder to HICOBOL, by typing:

HICOBOL, and click on ENTER.



Look in: cobol

File name:

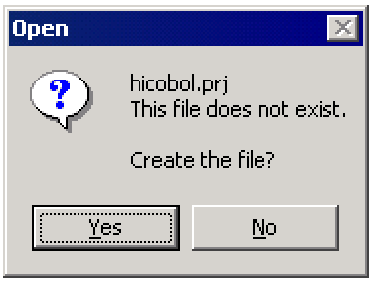
Files of type: Project File(*.PRJ)

□ Open as read-only

6. Double click on the new folder Hicobol, to switch to that folder.

7. In the File Name field, enter the name of the new project, HICOBOL, and click on the Open button.

   Project Manager displays the following message:

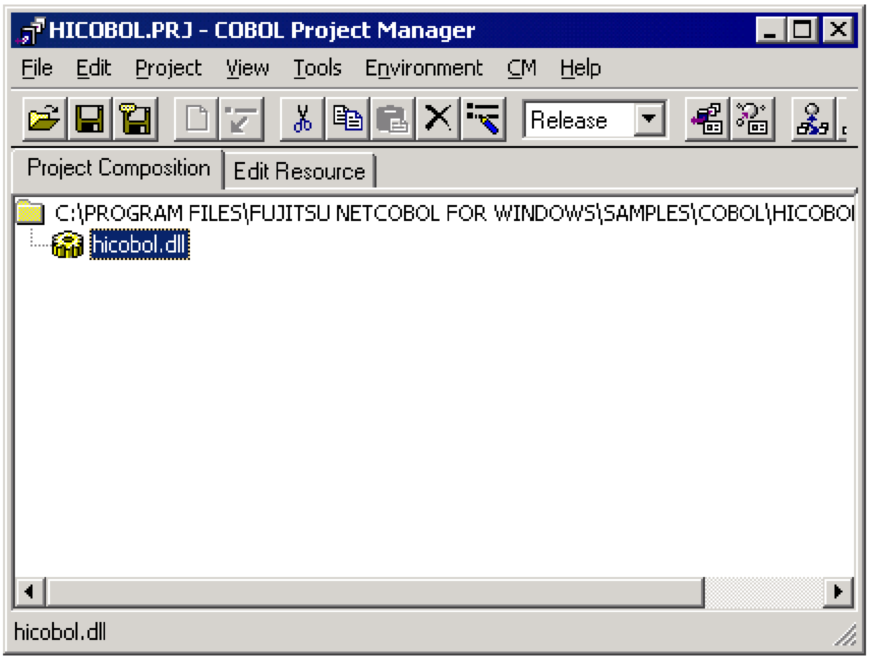


Open

hicobol.prj
This file does not exist.

Create the file?

8. Click on the Yes button, and the project file is created.
9. Right click on the project name to display a pop-up menu and select “New File” to create a new target file.
10. Type over the file name (NewFile.EXE) with HICOBOL.DLL. The COBOL Project Manager window should now look like this:



HICOBOL.PRJ - COBOL Project Manager

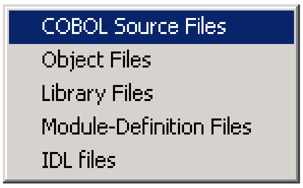
C:\PROGRAM FILES\FUJITSU NETCOBOL FOR WINDOWS\SAMPLES\COBOL\HICOBOI
hicobol.dll

hicobol.dll

Because we are not planning to create any other executables, you will now define the files that will be used to create this DLL.

11. Right-click the mouse on HICOBOL.DLL and select Create Folder from the popup menu.

    Project Manager displays the Create Folder sub-menu:



This menu offers the types of files upon which HICOBOL.DLL might depend. You need to create just one folder - for the COBOL source file.

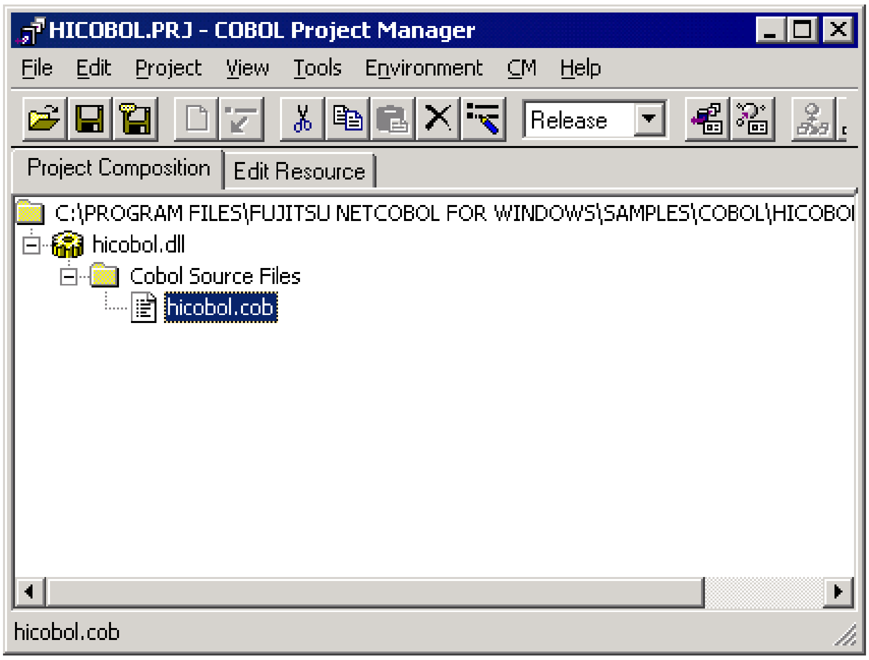
12. Select “COBOL Source File”.

    Project Manager adds a “COBOL Source File” folder to the project tree.

13. Right click on “COBOL Source File” and select “New File” from the pop-up menu.

14. Overtype “NewFile.COB” with “HICOBOL.COB”.

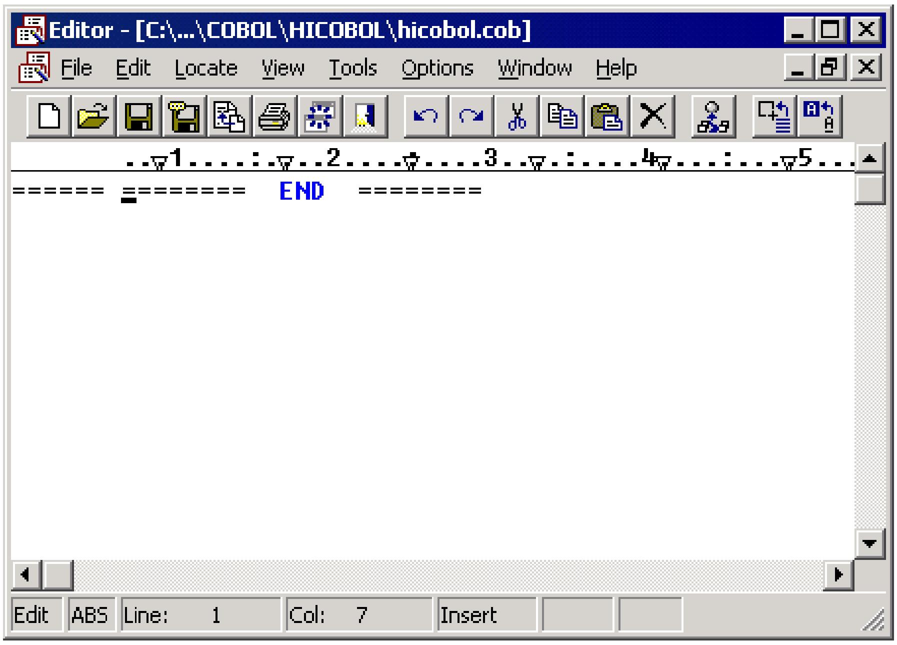
    The project tree should now look like this:


HICOBOL.PRJ - COBOL Project Manager
File Edit Project View Tools Environment CM Help
Release
Project Composition
Edit Resource
C:\PROGRAM FILES\FUJITSU NETCOBOL FOR WINDOWS\SAMPLES\COBOL\HICOBOL
hicobol.dll
Cobol Source Files
hicobol.cob
hicobol.cob

    You are now ready to create the COBOL source program.

15. Within the COBOL Project Manager, double-click the mouse on the HICOBOL.COB file name.

    This brings up a blank Editor window as shown below:

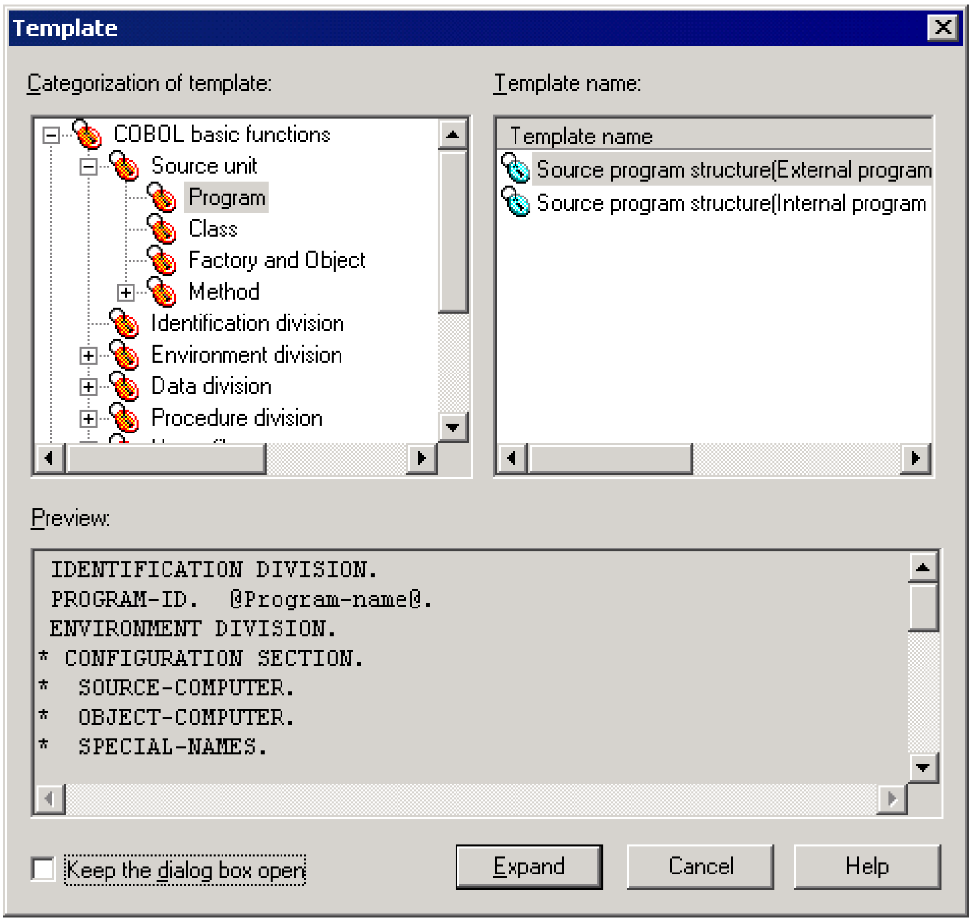

Editor - [C:\...\COBOL\HICOBOL\hicobol.cob]
File Edit Locate View Tools Options Window Help
..▽1....:.▽..2....✣....3..▽.:....4▽...:...▽5...
====== ======== END ========
Edit
ABS
Line: 1
Col: 7
Insert

    You are now ready to construct the HICOBOL.COB source program. The quickest way to do this is to use the Editor’s template function.

16. From the Edit menu select Template.

    The Editor displays the Template dialog.

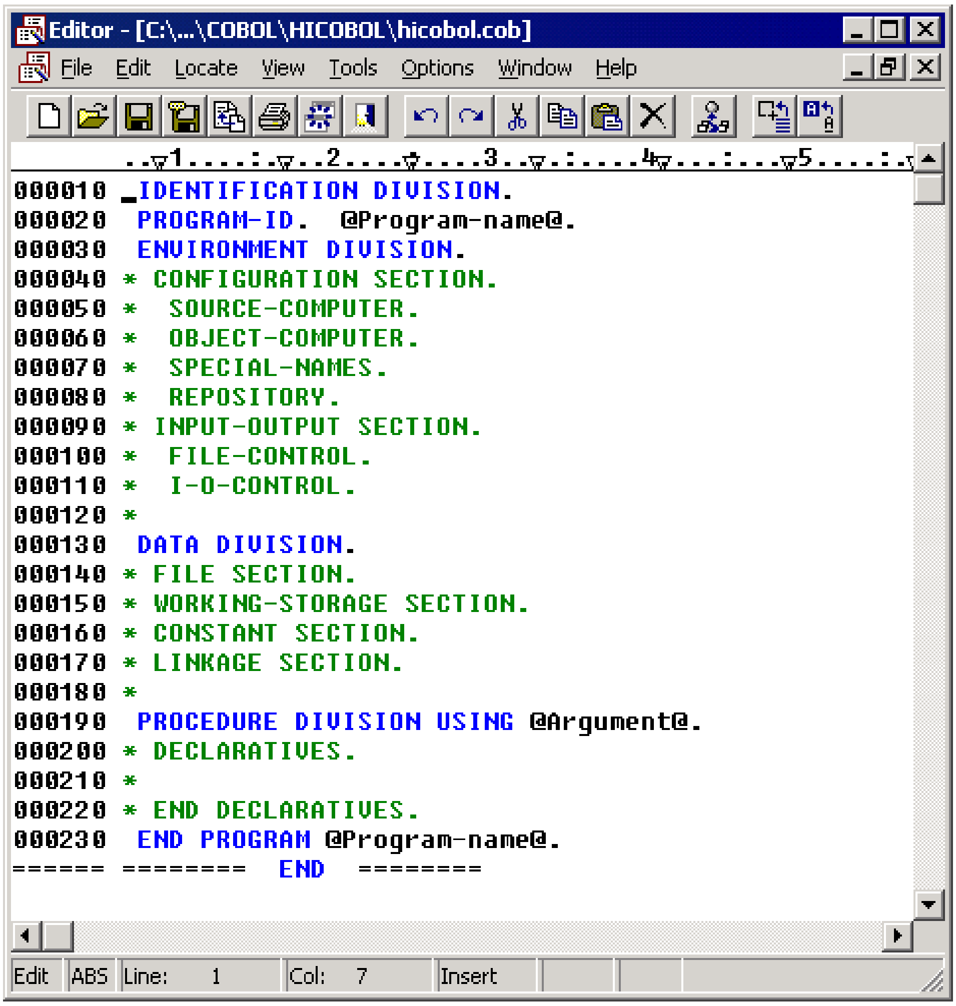
17. Expand the COBOL basic functions, Source unit nodes in the Categorization of template tree:



```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  @Program-name@.
 ENVIRONMENT DIVISION.
* CONFIGURATION SECTION.
*  SOURCE-COMPUTER.
*  OBJECT-COMPUTER.
*  SPECIAL-NAMES.
```

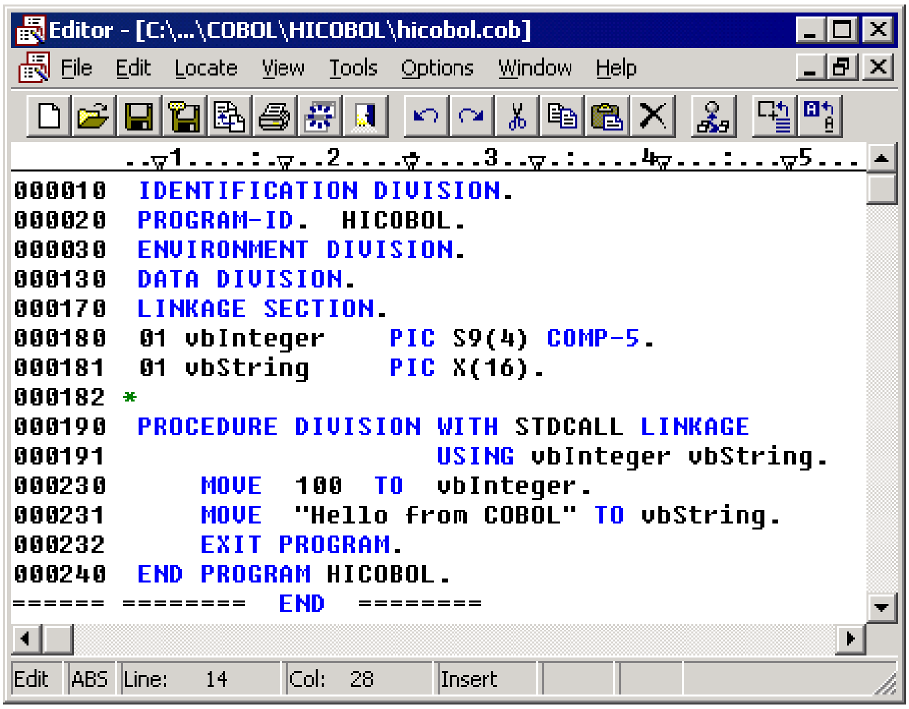
18. Select “Source program structure (External program definition)” from the list of Template name and click the Expand button. The Editor inserts the template code.

19. You can now see the code that has been inserted in the Editor:



```
000010 IDENTIFICATION DIVISION.
000020 PROGRAM-ID.  @Program-name@.
000030 ENVIRONMENT DIVISION.
000040 * CONFIGURATION SECTION.
000050 *  SOURCE-COMPUTER.
000060 *  OBJECT-COMPUTER.
000070 *  SPECIAL-NAMES.
000080 *  REPOSITORY.
000090 * INPUT-OUTPUT SECTION.
000100 *  FILE-CONTROL.
000110 *  I-O-CONTROL.
000120 *
000130  DATA DIVISION.
000140 * FILE SECTION.
000150 * WORKING-STORAGE SECTION.
000160 * CONSTANT SECTION.
000170 * LINKAGE SECTION.
000180 *
000190  PROCEDURE DIVISION USING @Argument@.
000200 * DECLARATIVES.
000210 *
000220 * END DECLARATIVES.
000230  END PROGRAM @Program-name@.
====== ========   END  ========
```

20. Uncomment the LINKAGE SECTION header, delete other unwanted comment lines, and insert the text shown below to create the HICOBOL program:


Editor - [C:\...\COBOL\HICOBOL\hicobol.cob]
File Edit Locate View Tools Options Window Help
Edit ABS Line: 14 Col: 28 Insert

```
000010  IDENTIFICATION DIVISION.
000020  PROGRAM-ID.  HICOBOL.
000030  ENVIRONMENT DIVISION.
000130  DATA DIVISION.
000170  LINKAGE SECTION.
000180  01 vbInteger     PIC S9(4) COMP-5.
000181  01 vbString      PIC X(16).
000182 *
000190  PROCEDURE DIVISION WITH STDCALL LINKAGE
000191                     USING vbInteger vbString.
000230      MOVE   100  TO  vbInteger.
000231      MOVE   "Hello from COBOL" TO vbString.
000232      EXIT PROGRAM.
000240  END PROGRAM HICOBOL.
====== ========    END   ========
```

If you examine the HICOBOL program, you will understand what it does quite easily. Note that it contains a LINKAGE SECTION and thus needs two parameters passed to it, each time it is called.

The vbInteger parameter is always set to the value of 100. The vbString parameter is always set to the value of "Hello from COBOL".

These parameters will be defined in the Visual Basic application and passed to the COBOL program. They are passed by reference - meaning that the COBOL program gains access to the data areas in the Visual Basic program. The COBOL program updates the values and returns control to the Visual Basic program.

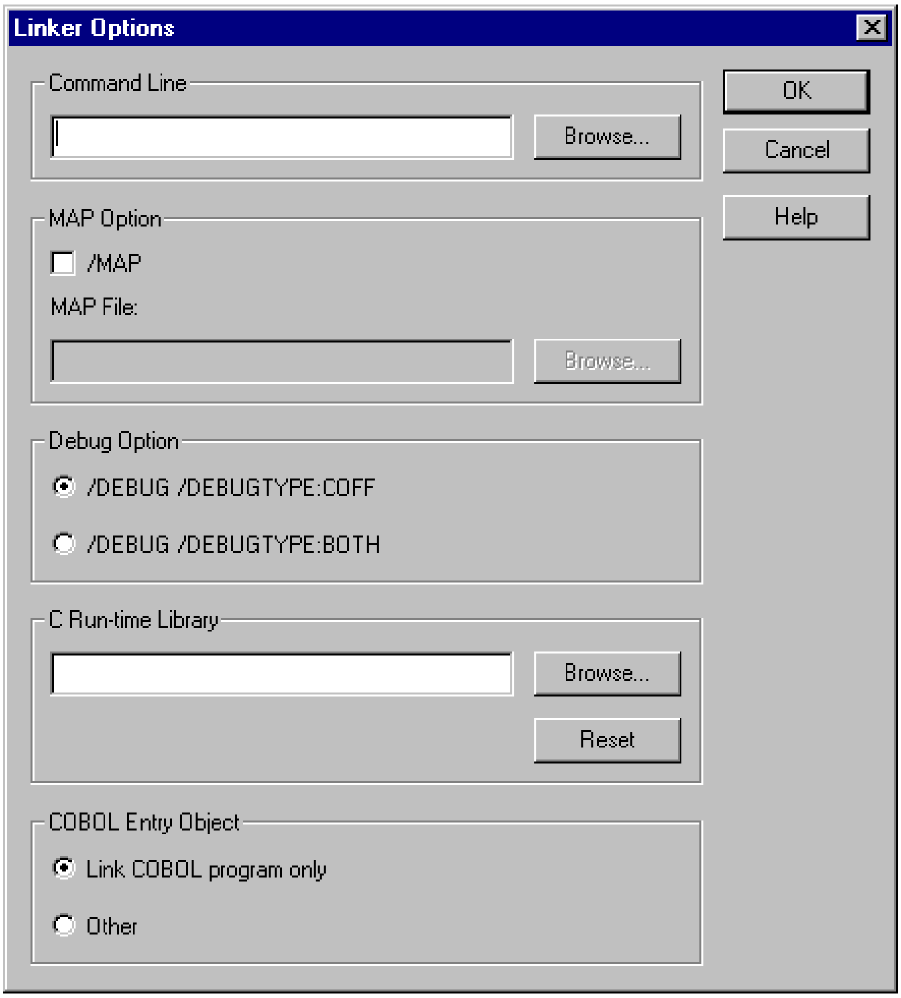
21. Select Save from the File menu to save the program.

22. Select Close from the File menu to close the Editor.

You are nearly ready to build the HICOBOL.DLL executable.

23. Return to the COBOL Project Manager window, select HICOBOL.PRJ in the project tree and, from the Project menu, select Options, Linker Options.

    Project Manager displays the Linker Options dialog:



Command Line

MAP Option

☐ /MAP

MAP File:

Debug Option

● /DEBUG /DEBUGTYPE:COFF

○ /DEBUG /DEBUGTYPE:BOTH

C Run-time Library

COBOL Entry Object

● Link COBOL program only

○ Other

24. In the Command Line entry field type:

    /EXPORT:HICOBOL

    This tells the linker to make the HICOBOL entry point available as an external reference in the DLL file.

25. Click on OK.

26. From the Project menu select Build.

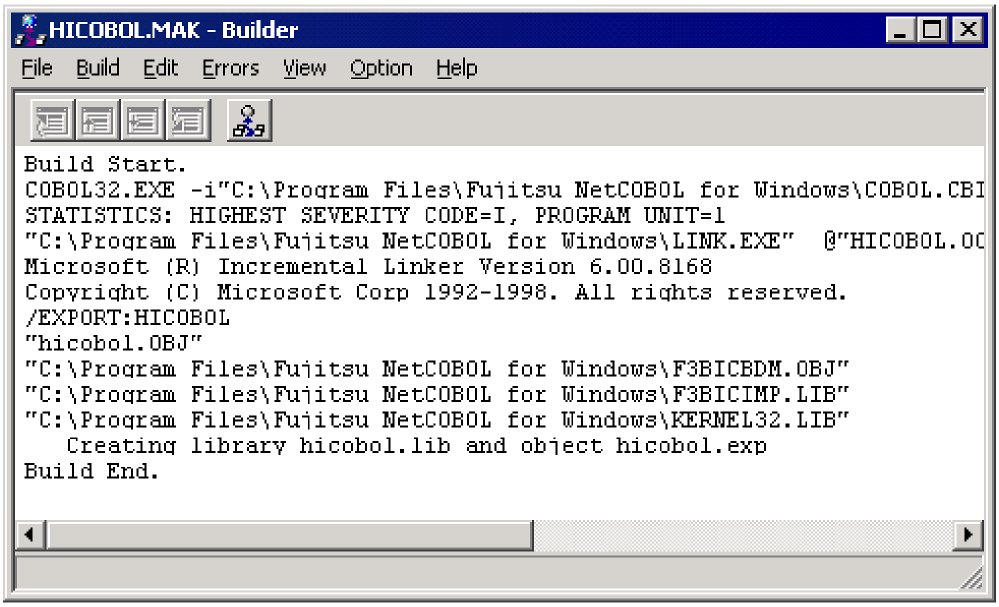
    Project Manager compiles and, if there are no compile errors, links the program into a DLL file.

27. If you receive any compile errors, they will be displayed in red in the Builder window that is used to display compiler and linker messages.

    - Double-click on an error line to be taken to the location of the error in the editor.
    - Any errors should be typos so check your code carefully, correct it and save the file.
    - Return to the COBOL Project Manager window and again select Build from the Project menu.

    Project manager recognizes that the source file has been updated since the last build and recompiles and relinks the appropriate parts of the project (in this case all the parts).

Once you have a successful build (or rebuild) of your application, the following is displayed in the Builder window:



```
Build Start.
COBOL32.EXE -i"C:\Program Files\Fujitsu NetCOBOL for Windows\COBOL.CBI
STATISTICS: HIGHEST SEVERITY CODE=I, PROGRAM UNIT=1
"C:\Program Files\Fujitsu NetCOBOL for Windows\LINK.EXE"  @"HICOBOL.OC
Microsoft (R) Incremental Linker Version 6.00.8168
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.
/EXPORT:HICOBOL
"hicobol.OBJ"
"C:\Program Files\Fujitsu NetCOBOL for Windows\F3BICBDM.OBJ"
"C:\Program Files\Fujitsu NetCOBOL for Windows\F3BICIMP.LIB"
"C:\Program Files\Fujitsu NetCOBOL for Windows\KERNEL32.LIB"
   Creating library hicobol.lib and object hicobol.exp
Build End.
```

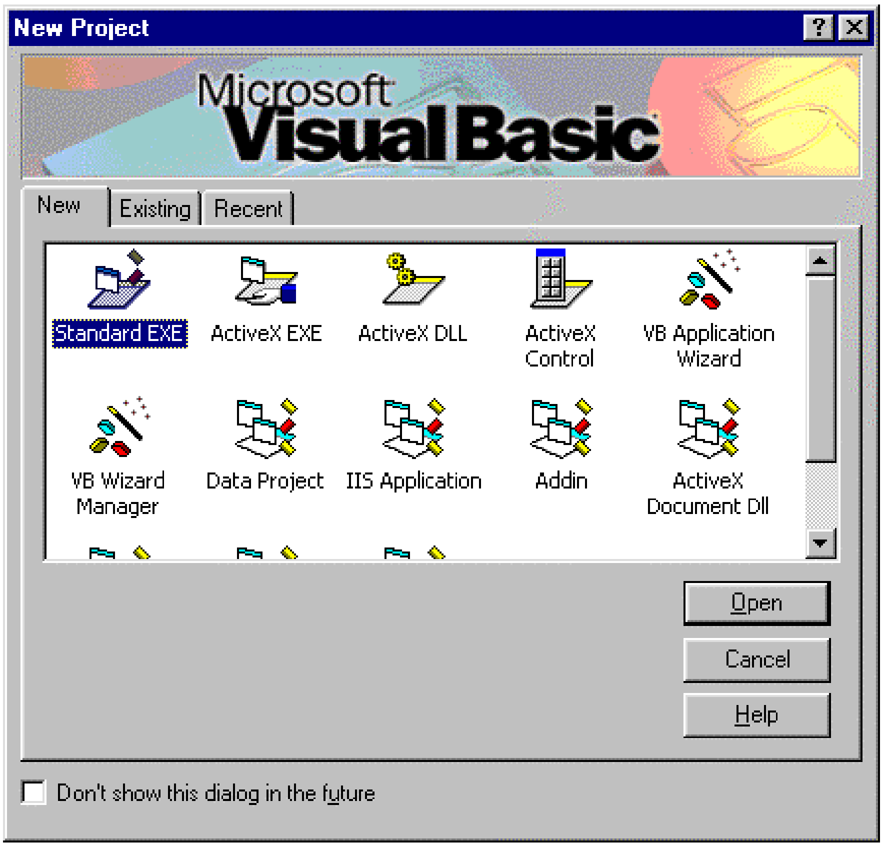
You have now completed the COBOL development side of this application.

Developing the GUI Interface Using Visual Basic

You will now use Visual Basic version 4.0 or higher to develop the graphical user interface for the HICOBOL application. The snapshots are taken using Visual Basic 6.0.

1. Bring up the Visual Basic development environment. (Select Visual Basic from the Windows Start, Programs menu.)

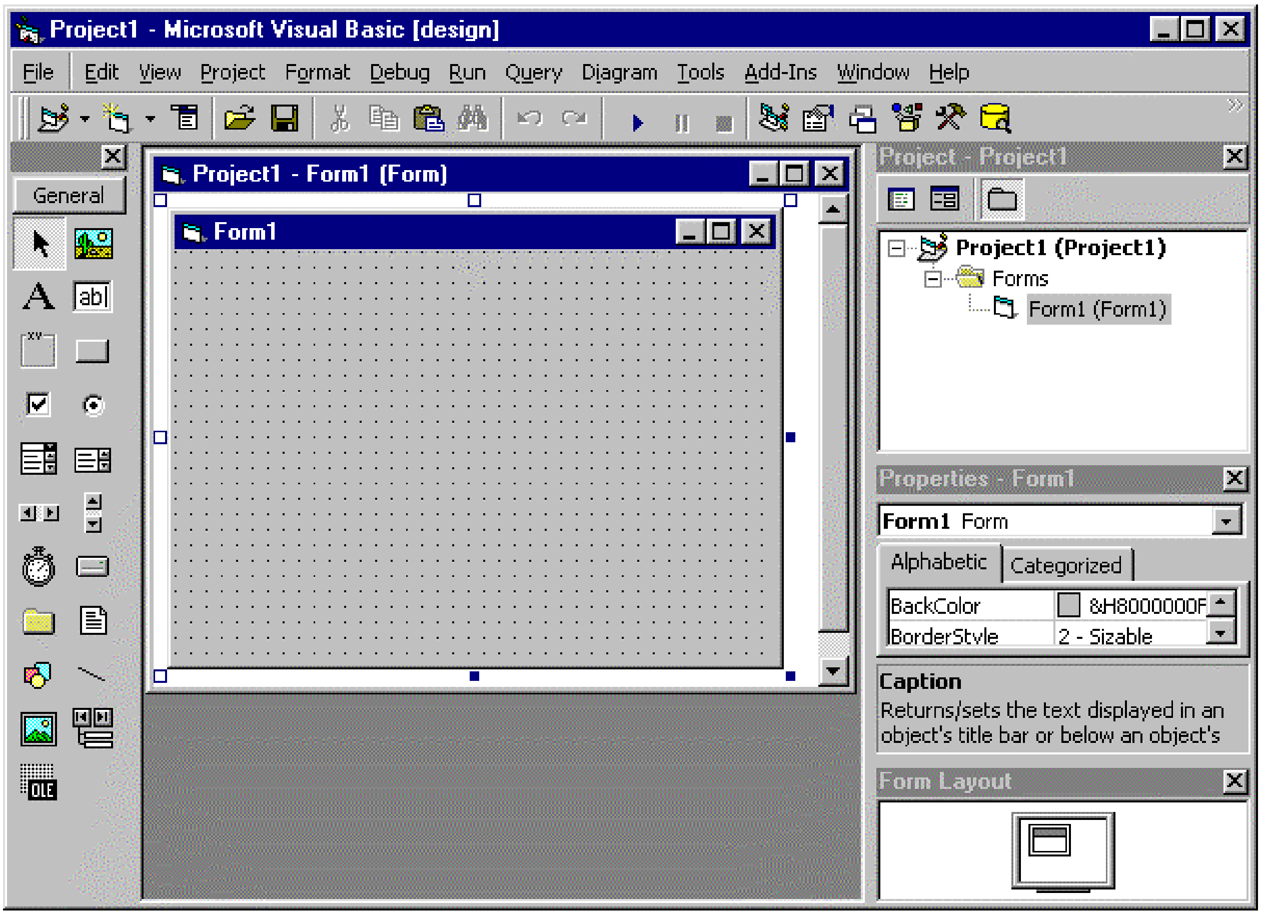
   Visual Basic displays a window like this:



   This window asks you to specify which type of application you are going to develop. You are going to develop a Standard EXE so:

2. Select Standard EXE and click on the Open button.

   Visual Basic displays the following window (or one like it, depending on which display options you have set):



BackColor &H8000000F
BorderStyle 2 - Sizable

Caption
Returns/sets the text displayed in an object's title bar or below an object's

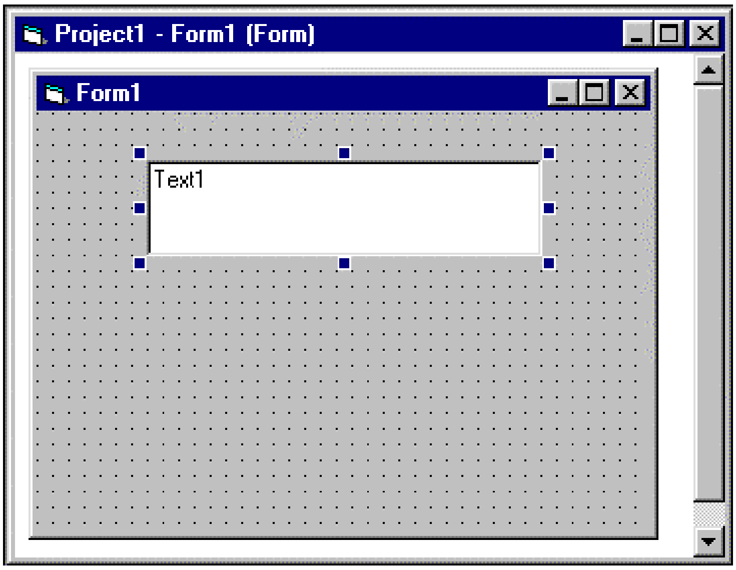
   You should already be familiar with this environment. If not, it is strongly suggested that you go through the Visual Basic “Your First Visual Basic Application” tutorial in “Getting Started with Visual Basic” (or “Visual Basic Programmer’s Guide” - earlier versions).

   In the center of this window is an object named Form1. This will become our main application window. You are going to create a text box and two command buttons on this form.

Create the Text Box:

1. Move the mouse to the toolbox (the “object palette”) on the left-hand side of the window and click once on the text box icon.

2. Move the mouse over to the Form1 grid. Starting near the upper left hand corner of the grid, hold the left mouse button down, and drag the mouse down a bit and to the right to draw a text box to look something like this:

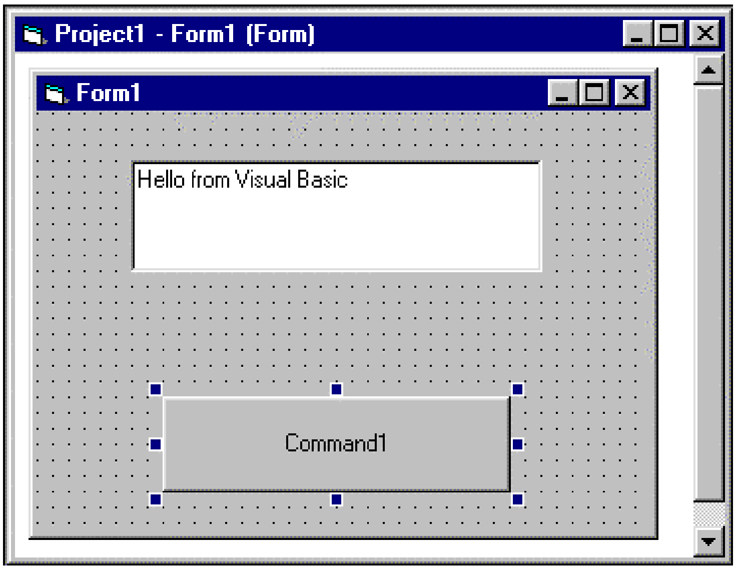


In the Properties window, enter "Hello from Visual Basic" in the Text property value.

You will notice that it instantly changes in the text box on Form1.

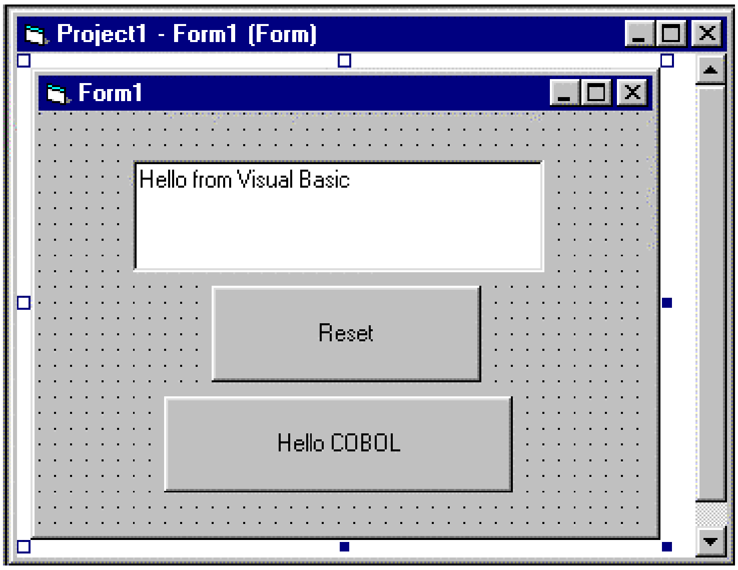
Create the Command Buttons:

1. Click on the command button icon, in the toolbox.
2. Move the mouse pointer near the bottom left of the Form1 window and draw a command button by holding the left mouse button down and dragging it.



3. Change the Caption value (the text displayed on the command button) from Command1 to Hello COBOL, in the Properties window.
4. You may reposition the command button or the text box by selecting either with the mouse and dragging the selected object to the position you desire on the form.
5. Now place a second command button on Form1 between the text box and the first command button by repeating steps 3 to 5 above. Change the caption value to “Reset”.

Form1 should now look like this:



Create the Programming Logic in Visual Basic

Now that you have created the application's main window (Form1) and its associated objects, it's time to wire this all together by creating some program logic to define the interaction of the user interface.

Visual Basic makes this process simple and straightforward in the way it has implemented the event-driven programming paradigm.

In these steps you will take care of the following tasks:

- Writing the actual Visual Basic Script (programming language) to handle events and to call the supporting COBOL application.
- Declaring the components of the supporting COBOL application to Visual Basic.
- Executing the overall application under the control of Visual Basic.

You will begin by writing the program logic.

The first requirement you will take care of is to implement the functionality for the Reset push button. The behavior that is desired is to have the text box string reset with a standard value each time a user clicks this command button.

Implementing the Reset Button

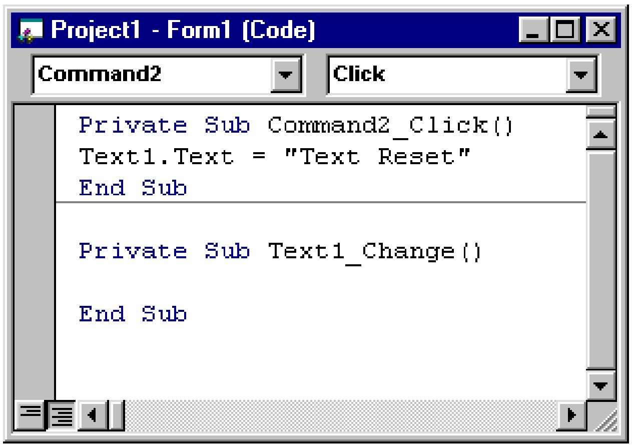
1. Move the mouse pointer over the Reset command button. Double click on this button.

   This brings up a program code window to create a Visual Basic subroutine that will be executed every time a user clicks on the Reset button. Part of the subroutine code is created automatically (its definition and exit).

   The cursor is automatically placed at the proper spot in the new subroutine to begin entering commands.

2. Enter the following line of code, which sets the text string to a new value:

   Text1.Text = "Text Reset"



```
Private Sub Command2_Click()
Text1.Text = "Text Reset"
End Sub

Private Sub Text1_Change()

End Sub
```

3. Click the mouse on the Close button (the X in the upper right hand corner) of the program code window you just typed in.

Implementing the Hello COBOL Button

The Hello COBOL command button is where most of the action in this application will take place. When a user clicks on this button, Visual Basic will call the HICOBOL.DLL executable and pass two parameters to it.

The COBOL program will update both parameters and return control to the Visual Basic program. Visual Basic will take the value of the string parameter (which the COBOL program will change to "Hello from COBOL") and update the text box value with it.

You need to perform the following steps at this point:

1. Move the mouse over the Hello COBOL command button and double click on it to open up a program code window.
2. Enter the following code in the new code window between the Private Sub and End Sub statements:

```
Dim vbInteger as Integer
Dim vbString as String * 16
Call JMPCINT2
Call HICOBOL (vbInteger, vbString)
Text1.Text = vbString
Call JMPCINT3
```

Let's look at what each of the above noted statements does at run-time:

Dim vbInteger as Integer - This defines a numeric data item named vbInteger.

Dim vbString as String * 16 - This defines a character string of 16 characters named vbString.

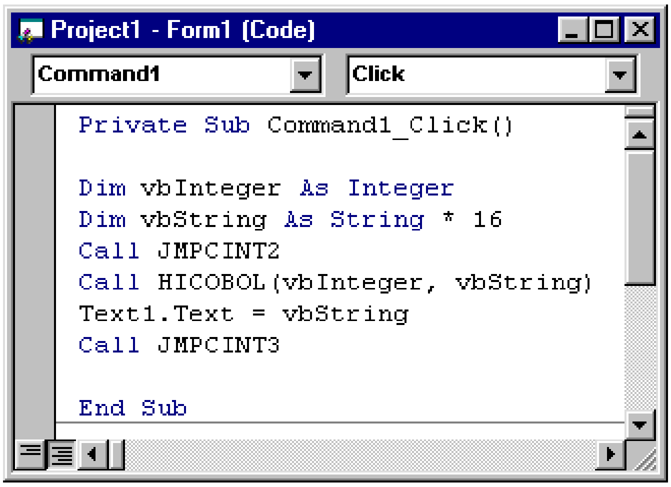
Call JMPCINT2 - This calls the supplied NetCOBOL routine which initializes the COBOL run-time support. It need only be called once per application.

Call HICOBOL (vbInteger, vbString) - This calls the COBOL DLL file you previously created, and passes the two parameters to it.

Text1.Text = vbString - This command instructs Visual Basic to update the Text string in the Text1 text box with the value contained in vbString, which will be returned from the HICOBOL COBOL program.

Call JMPCINT3 - This calls the supplied NetCOBOL routine which unloads (exits) the COBOL run-time support. It need only be called once per application.

Your Code window should look like this:

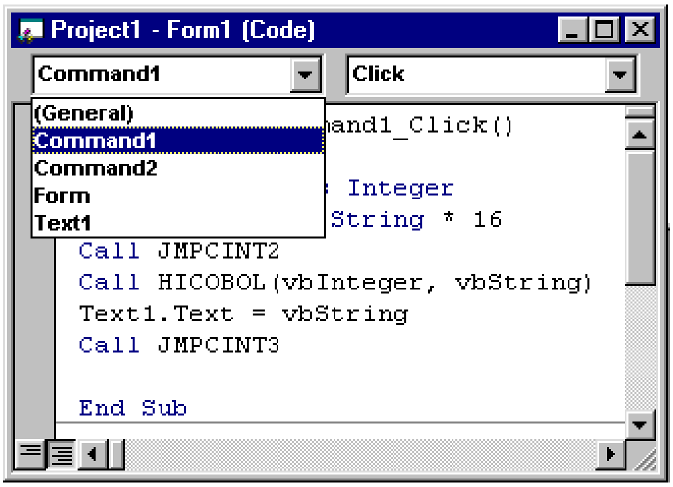


```
Private Sub Command1_Click()

Dim vbInteger As Integer
Dim vbString As String * 16
Call JMPCINT2
Call HICOBOL(vbInteger, vbString)
Text1.Text = vbString
Call JMPCINT3

End Sub
```

Defining the COBOL Components to Visual Basic

The final step that needs to be performed is to define to Visual Basic the external COBOL executables that are going to be referenced in this application. Do the following:

1. While the program code window shown above is still open, click on the small down arrow to activate the drop down box that currently has the value Command1. This box is located in the upper left of the project code window. It should appear as follows:



```
                  and1_Click()

                : Integer
                 String * 16
Call JMPCINT2
Call HICOBOL(vbInteger, vbString)
Text1.Text = vbString
Call JMPCINT3

End Sub
```

This drop down list box allows you to navigate through your program source code for each event subroutine you have defined. You need to go to the General section of your program to create declarations for the external COBOL executables.

2. Click on (General).

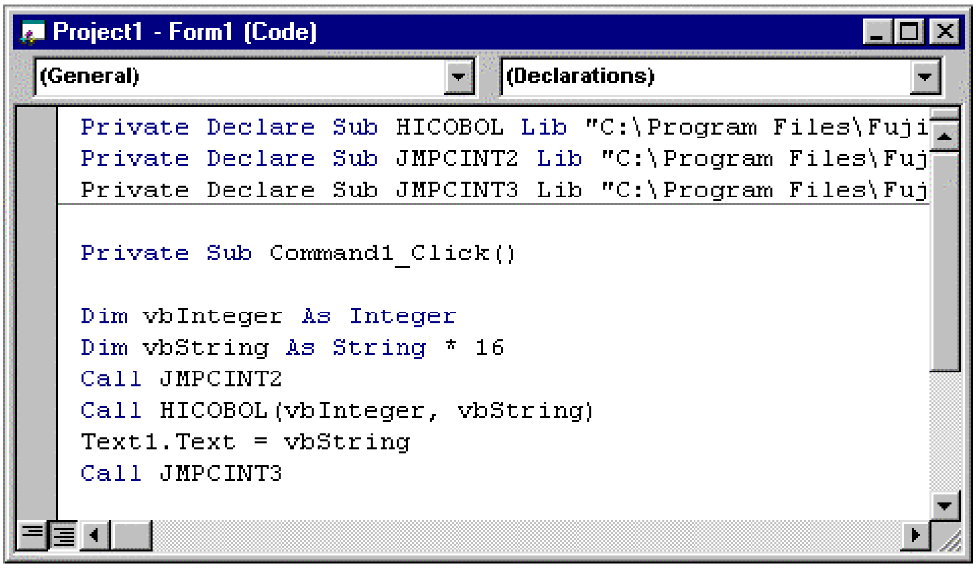
3. Enter the following 3 statements in any order (as one long string each) in the program code window in the General Section area:

```
Private Declare Sub HICOBOL Lib "C:\Program Files\Fujitsu NetCOBOL for Windows\SAMPLES\COBOL
\HICOBOL\HICOBOL.DLL" (vbInteger as Integer, ByVal vbString as String)
Private Declare Sub JMPCINT2 Lib "C:\Program Files\Fujitsu NetCOBOL for Windows\F3BIPRCT.DLL" ()
Private Declare Sub JMPCINT3 Lib "C:\Program Files\Fujitsu NetCOBOL for Windows\F3BIPRCT.DLL" ()
```

The three statements above have been wrapped to multiple lines because of the page size of this manual. They should be entered into the program code window as a single line each (not wrapped).

These statements define to Visual Basic where the external COBOL modules that will be referenced at run-time reside. Be sure to alter the path settings above when entering these if you changed the default installation directory when you installed NetCOBOL to reflect your machine’s installation location.

Your Visual Basic code should now look like this:



```
Private Declare Sub HICOBOL Lib "C:\Program Files\Fuji
Private Declare Sub JMPCINT2 Lib "C:\Program Files\Fuj
Private Declare Sub JMPCINT3 Lib "C:\Program Files\Fuj

Private Sub Command1_Click()

Dim vbInteger As Integer
Dim vbString As String * 16
Call JMPCINT2
Call HICOBOL(vbInteger, vbString)
Text1.Text = vbString
Call JMPCINT3
```

4. Close the program code window.

You have now completed all of the steps required to create this sample application.

You should save the Visual Basic portion you've just created.

1. Select Save Project from the File menu.

   You will be prompted to save the form.

   Change the Form name from Form1.frm to HICOBOL.frm, and save it in the HICOBOL folder (within the "C:\Program Files\Fujitsu NetCOBOL for Windows\samples\cobol" folder).

2. Select Save Project again, to save the project.

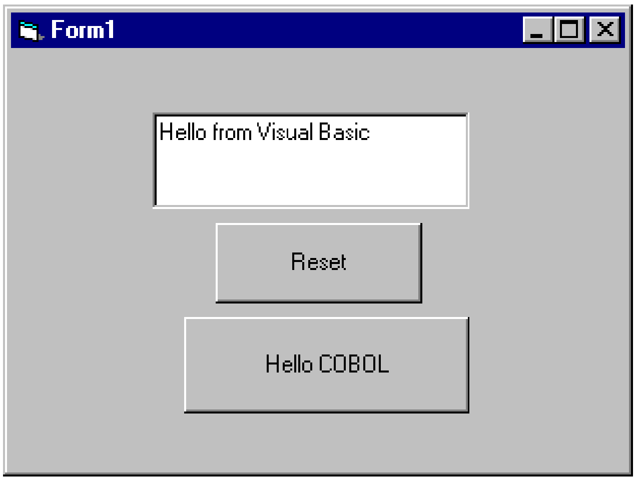
   Name it HICOBOL.vbp and save it.

Executing the HICOBOL Application under Visual Basic

You are now ready to execute the HICOBOL application.

1. From the Run menu of Visual Basic, select Start.

   The HICOBOL graphical window you just created appears:



Hello from Visual Basic

2. Click on the Reset button.

   The text in the Text box changes to "Text Reset".

3. Click on the Hello COBOL command button.

   Visual Basic issues a call to the HICOBOL.DLL executable.

   The program updates the parameters, which then update the text box in your graphical window to read "Hello from COBOL".

4. Click on the Reset button.

   The text in the Text box returns to “Text Reset”.

5. Close the window to exit the application.

If you want to create a standalone EXE for your application, use the Make command under the Visual Basic File menu.

Note that Visual Basic applications require you to distribute the Visual Basic Run-time DLL along with your EXE and the COBOL components required.

3.2 Integrating COBOL Programs with Visual Basic

This section contains detailed information regarding the integration of NetCOBOL programs with Visual Basic applications.

COBOL DLL Interaction with Visual Basic

To access any COBOL routines from Visual Basic, you need to first declare the COBOL PROGRAM-ID or the COBOL ENTRY routine using the Basic DECLARE statement with the special LIB keyword to specify the path name of the COBOL DLL. These declarations can be made in the declaration section of any form module or in the global module, and must be declared as Private. You can then call these routines from your Visual Basic code like any other function call.

When a Visual Basic program calls a COBOL routine, you should call JMPCINT2 before calling the first COBOL routine, and call JMPCINT3 after calling the last COBOL routine. JMPCINT2 is a subroutine that initializes the COBOL Run Time Environment and JMPCINT3 is a subroutine that exits the COBOL Run Time Environment.

Calling COBOL routines without calling JMPCINT2 and JMPCINT3 may degrade performance because the COBOL Run Time Environments will be initialized and exited on every COBOL routine called.

Visual Basic Declarations of COBOL Modules

```
Private Declare Sub PROG Lib "c:\mycobol.dll" (vbInteger as Integer, ByVal vbString as String)
Private Declare Sub JMPCINT2 Lib "C:\Program Files\Fujitsu NetCOBOL for Windows\F3BIPRCT.DLL" ()
Private Declare Sub JMPCINT3 Lib "C:\Program Files\Fujitsu NetCOBOL for Windows\F3BIPRCT.DLL" ()
```

NOTE: The above Declare statements should all be on a single line, they have beenwrapped in this text to fit the width of the page.

Visual Basic Call to a COBOL Program

```
Sub Form_Click ( )
Dim vbInteger as Integer
Dim vbString as string * 15
Call JMPCINT2  ‘ Initialize COBOL Runtime Environment
Call PROG (vbInteger, vbString)
Call JMPCINT3  ‘ Terminate COBOL Runtime Environment
End Sub
```

COBOL LINKAGE SECTION and PROCEDURE DIVISION

```
Identification Division.
Program-ID. "PROG".
Data Division.
Linkage Section.
01 vbInteger pic s9(4) comp-5.
01 vbString pic x(15).
Procedure Division with STDCALL Linkage Using vbInteger, vbString.
```

```
     move 100 to vbInteger
     move "NetCOBOL" to vbString
     exit program.
```

Parameter Passing Between Visual Basic and NetCOBOL

Visual Basic treats a COBOL DLL as a “black box.” It only needs to know the COBOL program's LINKAGE SECTION parameter list to pass data to and from the COBOL DLL. Parameters can only be passed from Visual Basic to COBOL BY REFERENCE, except strings which must be passed using the ByVal keyword in the DECLARE statement. Passing parameters BY REFERENCE is the default in Visual Basic.

Parameter names need not be the same between Visual Basic and COBOL, however, attribute, length, and number of corresponding data items must be identical.

Visual Basic declarations must exactly match the COBOL parameter lists defined in the USING statement of the COBOL PROCEDURE DIVISION or ENTRY statements. No parameter checking is done because the two are separate compilation units.

Visual Basic incorporates a rich assortment of data types, some of which are currently not supported by NetCOBOL. These data types include variable-length strings, Variants, and objects.

Similarly not all COBOL data types are supported by Visual Basic.

Visual Basic Strings

All strings passed between Visual Basic and COBOL must be declared as fixed-length strings and passed using the ByVal keyword in the DECLARE statement. To avoid possible memory corruption, ensure that all strings passed between Visual Basic and COBOL occupy the same size.

```
Dim vbString as string * 15  ‘ Equivalent to PIC X(15)
```

Corresponding Visual Basic and COBOL Data Types

When passing parameters, associate the data types as follows:

The table below lists the classes and encoding forms.

Visual Basic		COBOL
Type Name	Storage Size	PICTURE Clause
Boolean (16bit)	2 Bytes	S9(4) COMP-5
Boolean (32bit)	4 Bytes	S9(9) COMP-5
Byte	1 Byte	X
Currency	8 bytes	S9(10)V9(4) COMP-5
Double	8 Bytes	COMP-2
Integer	2 Bytes	S9(4) COMP-5
Long	4 Bytes	S9(9) COMP-5
Single	4 Bytes	COMP-1
String	1 Byte per Character	X(n)

Passing Arrays

Visual Basic and COBOL can pass numeric arrays. This works because numeric array data is always laid out sequentially in memory. A COBOL routine, if given the first element of an array, has access to all of its elements.

Passing the Currency Data Type

The Currency data type in Visual Basic equates to

```
PIC S9(14)V9(4) COMP-5.
```

You can pass Currency parameters if the receiving Linkage Section item has this picture clause.

A Visual Basic declaration of a COBOL subroutine receiving the Currency data type would look like the following:

```
Private Declare Sub CSPRINT Lib "c:\CSPRINT.dll" (cc As Currency)
Private Declare Sub JMPCINT2 Lib "c:\Program Files\Fujitsu NetCOBOL for Windows\f3biprct.dll" ()
Private Declare Sub JMPCINT3 Lib "c:\Program Files\Fujitsu NetCOBOL for Windows\f3biprct.dll" ()
```

NOTE: The above Declare statements should all be on a single line, they have been wrapped in this text to fit the width of the page.

CSPRINT is the COBOL subroutine; JMPCINT2 and JMPCINT3 are required for mixed language applications including Visual Basic and COBOL.

The code leading to, and including the subroutine call would look like this:

```
Dim cc As Currency
cc = 100.0001
Call CSPRINT(cc)
```

The COBOL program processing the COMP-5 equivalent of Currency would look the following:

```
000010 IDENTIFICATION DIVISION.
000020 PROGRAM-ID. CSPRINT.
000030 ENVIRONMENT DIVISION.
000061 DATA DIVISION.
000101 LINKAGE SECTION.
000102 01 CC PIC S9(14)V9(4) COMP-5.
000110 PROCEDURE DIVISION WITH STDCALL USING CC.
000150      ADD 1 TO CC.
000160 EXIT PROGRAM.
```

NOTE: If the data type is converted (for instance, if it is moved to a PACKED-DECIMAL field, or DISPLAYed), then the value of the data may be truncated.

Returning Control and Exiting Programs

To return control from COBOL to Visual Basic, execute the EXIT PROGRAM statement. When the EXIT PROGRAM statement is executed, control returns immediately to the calling program.

Compiling and Linking the COBOL Programs

To build the COBOL DLL, you must tell the linker which entry points are to be made external (i.e. those entry points that will be called from Visual Basic). You can do this by:

1. Specifying the /ENTRY:entry-name linker option for each entry point, or:
2. Creating a module definition file (.DEF) that lists the attributes of the DLL library. You can modify the example below, changing the “PROG” to match the COBOL PROGRAM-ID (or other ENTRY name that you call).

   ```
   LIBRARY "dll-name"
   EXPORTS PROG
   ```

You then save this file as “prog-name.DEF”, where prog-name is the name of your program.

Add it to your project by right clicking on the DLL file node and select New Folder, Module Definition File, from the pop-up menu. You then add the .DEF file to that folder.

Chapter 4 Using SQL with COBOL

This chapter describes the steps required to setup your environment to access SQL databases using ODBC. It first gives you an overview of ODBC and the general mode of working with SQL databases; then goes into the details of setting up your environment. In particular it covers:

- Defining SQL databases.
- Defining an ODBC data source.
- Setting up NetCOBOL runtime information.

4.1 Introduction

NetCOBOL applications can contain sophisticated processing logic that interacts with SQL database systems. The technology that is used to interface to these databases is known as ODBC (Open Database Connectivity), which provides a standard and generic mechanism for any number of applications to access SQL data in a relational matter.

The following sections deal specifically with how to set up the interface between a NetCOBOL SQL program and SQL databases. It concludes with some tips identifying potential runtime problems with NetCOBOL SQL applications.

Note that the samples and screen pictures are from a Microsoft Windows 2000 environment. You may experience some differences on Windows XP, Windows Vista, Windows Server 2003, and Windows Server 2008 systems, but these should be easy to determine.

Typical SQL Program Structure

When you create a NetCOBOL program using embedded SQL that will access one or more SQL databases, your application logic typically goes through the following steps in the following order:

a. Connect to an SQL database

b. Declare a cursor (for multiple Selects if desired)

c. Open the cursor (or if no cursor is desired, simply execute an SQL SELECT statement).

d. Perform any other desired SQL operations on the data, such as fetching from an open cursor, reading, writing, deleting and updating any specific data.

e. Closing any open cursors.

f. Transaction processing, such as committing or rolling back changes.

g. Disconnecting from SQL databases.

In NetCOBOL, you code your programs using embedded SQL statements and host variables. For information on writing SQL code in NetCOBOL programs, refer to “Chapter 19. Database (SQL)” of the “NetCOBOL User’s Guide”. This chapter covers the information you need to know to set up an environment that supports such programs.

Note that when creating NetCOBOL SQL applications you can access any number of SQL databases available to you within a single application.

4.2 Setting up the SQL Execution Environment

In order for a NetCOBOL SQL application to execute successfully, you must ensure that three separate setup oriented tasks have taken place prior to execution:

1. Define the SQL databases and tables.
2. Define ODBC data sources.
3. Setup NetCOBOL runtime information.

This section gives an overview of these three tasks. The following sections then step you through the details.

Define the SQL Databases and Tables

The SQL database(s) and table(s) must typically be defined prior to execution. Note that the SQL language itself is divided into two separate categories - DDL (Data Definition Language) and DML (Data Manipulation Language). DDL is used to create and delete (drop) tables and apply certain access privileges. DML is used to manipulate data and includes reading, writing, deleting and updating tables, among other things.

NetCOBOL applications generally are DML specific. NetCOBOL does not allow you to code embedded DDL in a COBOL program. You may, however, write NetCOBOL applications using dynamic SQL, which contains DDL statements. See “Using Dynamic SQL” in “Chapter 19. Database (SQL)” of the “NetCOBOL User’s Guide” for more information on dynamic SQL in NetCOBOL programs.

Once your SQL database(s) and table(s) have been set up properly, you must additionally ensure that you have been granted appropriate access and that you are aware of any required user id(s) and password(s) required for accessing the SQL data you desire. If you are unfamiliar with this process, you should consult with your SQL database administrator.

Define ODBC Data Sources

After your SQL database(s) and table(s) have been set up properly, you must define an ODBC data source to be used by your NetCOBOL application. It is important to understand that ODBC acts as a sort of “middle man” between your NetCOBOL application and the actual SQL database(s). The embedded SQL statements contained in your NetCOBOL application are actually compiled into ODBC API (Application Programming Interface) calls to the ODBC software. ODBC then takes care of interfacing to the actual SQL database(s).

The advantage to using ODBC from NetCOBOL is that you are thus able to write a single COBOL program that can access any number of different SQL databases, as this connection is made at runtime. Switching a NetCOBOL application from one vendor’s SQL database system to another is as easy as switching the ODBC data source - no changes are required to the COBOL program itself; it does not have to be recompiled or linked.

The details of creating an ODBC data source are described in the section “Defining an ODBC Data Source” below.

Setup NetCOBOL Runtime Information

Once your SQL database(s) and table(s) have been set up properly, and you have created a proper ODBC data source for your application, you must set up NetCOBOL specific runtime information. This step connects the NetCOBOL application to the proper ODBC data source and contains other information about the SQL database environment you wish to access. This information is contained in a NetCOBOL information (.inf) file which is pointed to at runtime. See the section “Setting Up the COBOL Runtime Information” below for details.

4.3 Setting up SQL Databases and Tables

Because there are a wide array of different SQL database systems from a variety of different vendors it is not feasible to give detailed instructions in this chapter. See your SQL database administrator if you have any questions.

It is worth noting, however, that one of the most common problems that developers experience is not being able to access a SQL database because security and permissions have not been set up correctly.

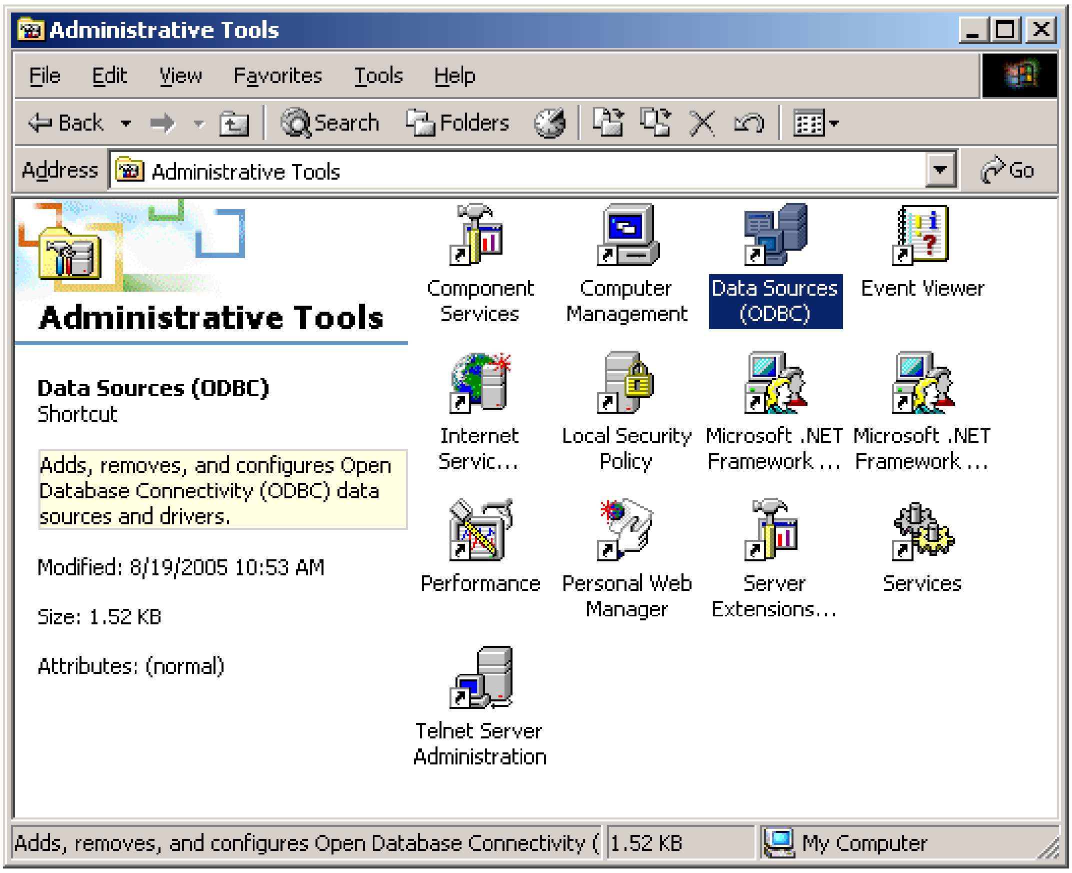
If you receive unexplained errors at runtime when you attempt to access an SQL database, it is a good idea to check the user id and password you are using. Ensure that they have the appropriate permissions set up in the SQL database system to allow you access to the database and tables you are using.

4.4 Defining an ODBC Data Source

Creating an ODBC data source is a simple and straightforward process. You first check that you have a 32 bit ODBC manager installed, then use the Data Source Administrator to create the ODBC data source. The following topics take you through these steps.

Checking for a 32 Bit ODBC Manager

To check for a 32 bit ODBC manager, you access the Windows Control Panel by clicking on the Windows Start button, moving the mouse over the Settings option and left clicking on the Control Panel option. Then you access Administrative Tools by clicking Administrative Tools icon on Windows Control Panel. Look in your Administrative Tools and ensure that you have a Data Sources (ODBC) icon as shown in Figure below:



Data Sources (ODBC)
Shortcut

Adds, removes, and configures Open Database Connectivity (ODBC) data sources and drivers.

Modified: 8/19/2005 10:53 AM

Size: 1.52 KB

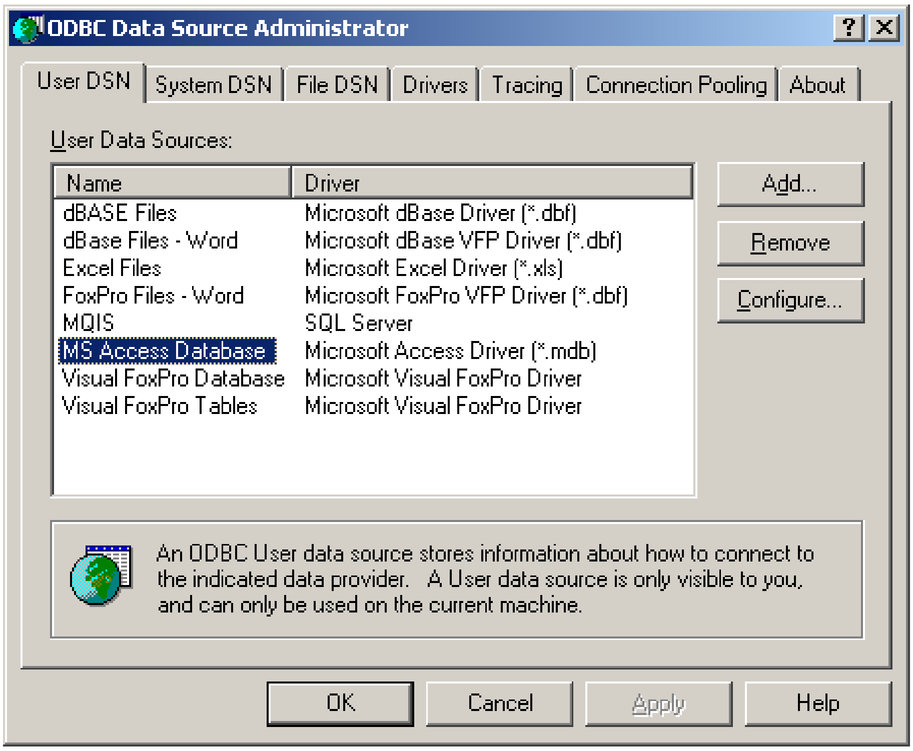
Attributes: (normal)

Adds, removes, and configures Open Database Connectivity (| 1.52 KB

If you do not have Data Sources (ODBC) installed on your Windows machine, check with your software administrator about obtaining it and installing it properly.

An Overview of the ODBC Data Source Administrator

Double click on the Data Sources (ODBC) icon to bring up the ODBC Data Source Administrator as shown in Figure.



User Data Sources:

Name	Driver
dBASE Files	Microsoft dBase Driver (*.dbf)
dBase Files - Word	Microsoft dBase VFP Driver (*.dbf)
Excel Files	Microsoft Excel Driver (*.xls)
FoxPro Files - Word	Microsoft FoxPro VFP Driver (*.dbf)
MQIS	SQL Server
MS Access Database	Microsoft Access Driver (*.mdb)
Visual FoxPro Database	Microsoft Visual FoxPro Driver
Visual FoxPro Tables	Microsoft Visual FoxPro Driver

An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you, and can only be used on the current machine.

Note that your Data Source Administrator may appear differently depending on the Windows operating system that you are running and the version of ODBC you have installed.

There are three tabs in this window for accessing User DSN's, System DSN's, and File DSN's.

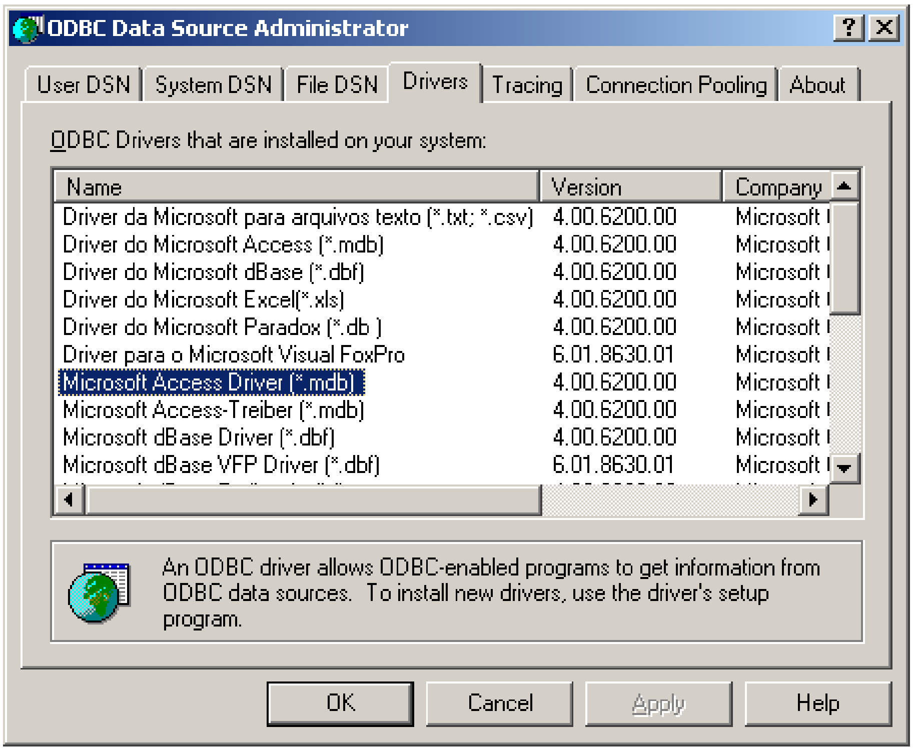
A User DSN (Data Source Name) will only be visible to the same User ID who was logged onto the current machine and previously created the DSN. For example, if you log onto your Windows machine with a user ID of "Fred" create a new ODBC User DSN, log off and log back on as "James", you will not see or have access to the just-created User DSN.

If you had instead logged on as Fred and created a System DSN, any subsequent user who logged onto the same machine, regardless of their user ID would both see and be able to access the newly created DSN.

While User and System DSN's are specific to the current machine only, File DSN's may be shared by other users who have the same ODBC drivers installed on their machines.

You can create ODBC DSN's for any COBOL application using any of these three tabs, but remember the ramifications if you change user ID's or wish to export the DSN to another machine.

Prior to creating a new DSN, you should click on the Drivers tab and ensure the ODBC database driver for the actual type of database you wish to access is installed. In the example below, we will set up a new DSN for connecting to a Microsoft SQL Server database. Note in the Drivers panel shown in Figure 4.3 below that a SQL Server driver is indeed installed as required.



ODBC Drivers that are installed on your system:

Name	Version	Company
Driver da Microsoft para arquivos texto (*.txt; *.csv)	4.00.6200.00	Microsoft
Driver do Microsoft Access (*.mdb)	4.00.6200.00	Microsoft
Driver do Microsoft dBase (*.dbf)	4.00.6200.00	Microsoft
Driver do Microsoft Excel(*.xls)	4.00.6200.00	Microsoft
Driver do Microsoft Paradox (*.db)	4.00.6200.00	Microsoft
Driver para o Microsoft Visual FoxPro	6.01.8630.01	Microsoft
Microsoft Access Driver (*.mdb)	4.00.6200.00	Microsoft
Microsoft Access-Treiber (*.mdb)	4.00.6200.00	Microsoft
Microsoft dBase Driver (*.dbf)	4.00.6200.00	Microsoft
Microsoft dBase VFP Driver (*.dbf)	6.01.8630.01	Microsoft

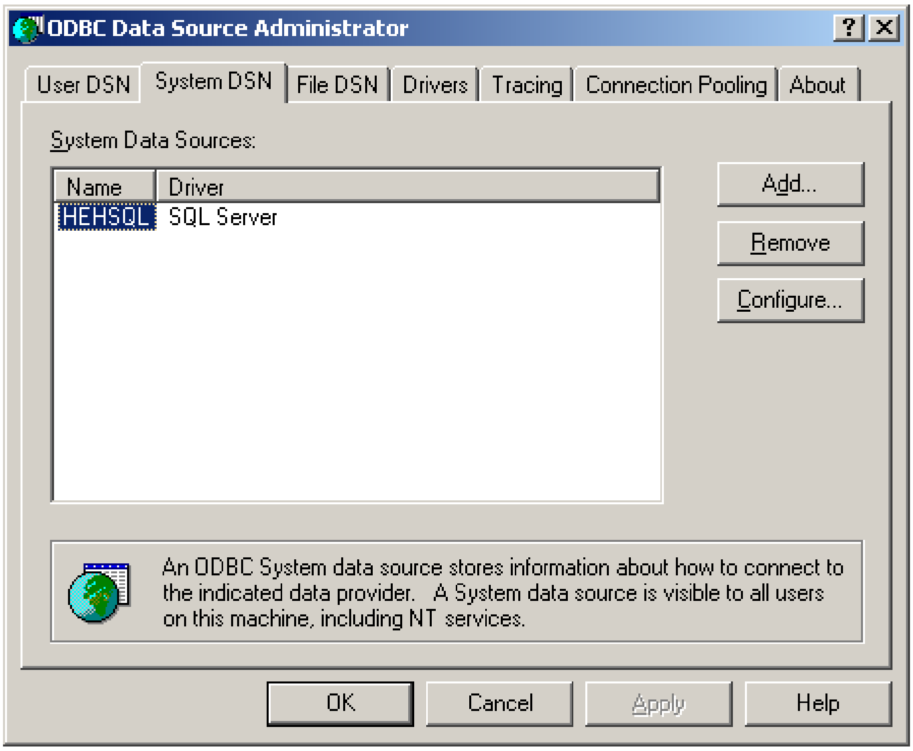
An ODBC driver allows ODBC-enabled programs to get information from ODBC data sources. To install new drivers, use the driver's setup program.

If your system does not contain the required ODBC driver for the type of SQL database system you wish to access, check with your SQL database administrator about obtaining and installing the current driver.

Once you are satisfied that you have a proper driver installed, click back on one of the three DSN tabs.

Creating a Data Source Name (DSN)

In the example below, a new System DSN will be created. The System DSN tab appears as shown in Figure.



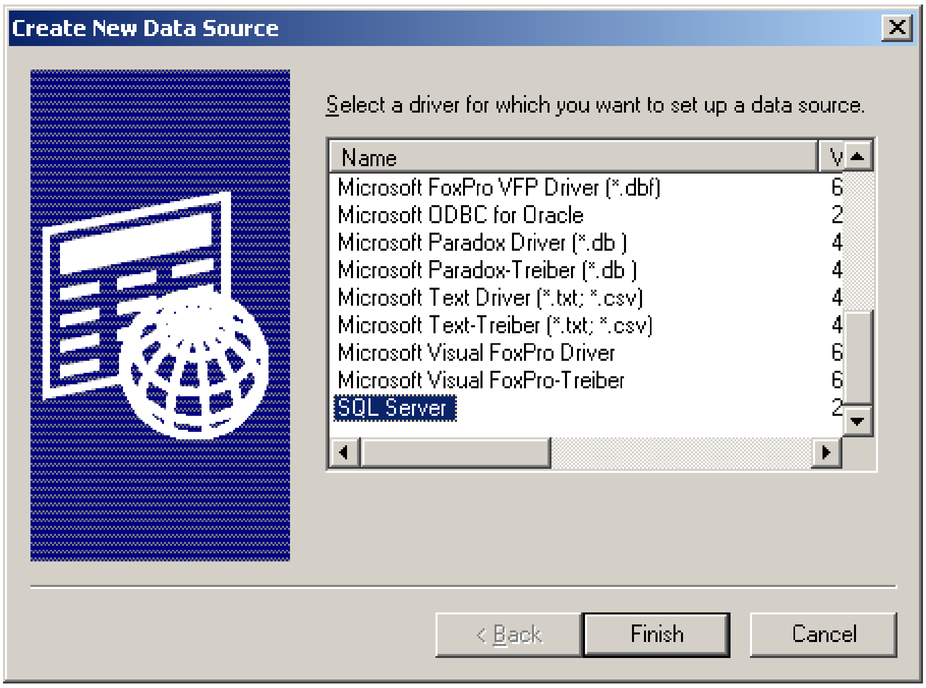
System Data Sources:

Name	Driver
HEHSQL	SQL Server

An ODBC System data source stores information about how to connect to the indicated data provider. A System data source is visible to all users on this machine, including NT services.

To create a new ODBC data source:

1. Click on the Add button. A dialog box appears as shown in Figure.

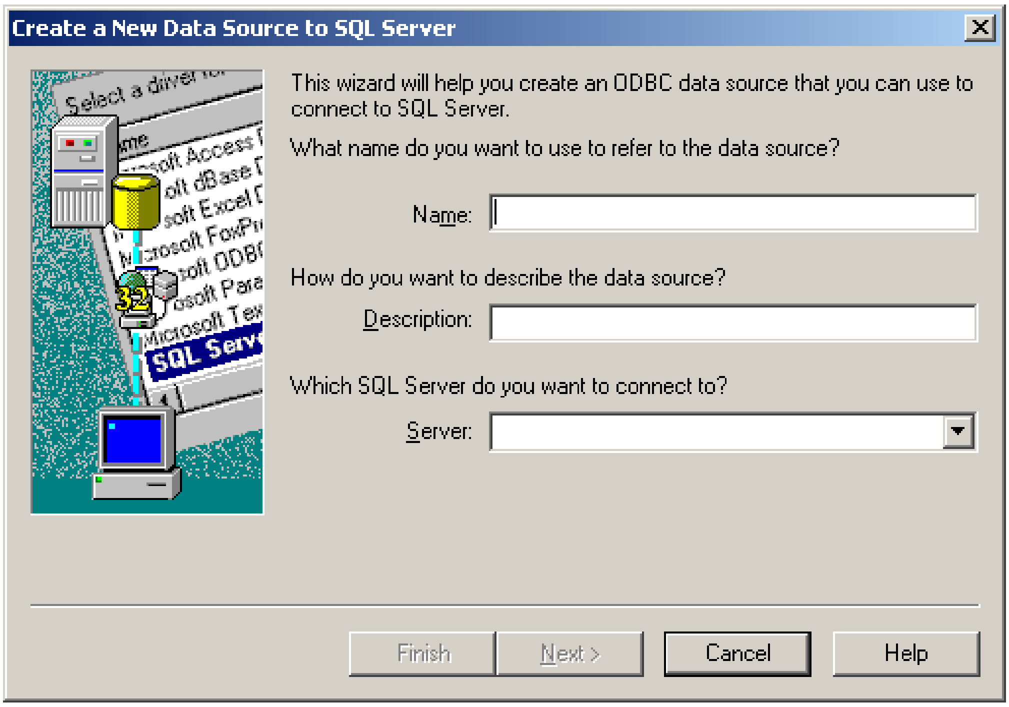


Select a driver for which you want to set up a data source.

Name	V
Microsoft FoxPro VFP Driver (*.dbf)	6
Microsoft ODBC for Oracle	2
Microsoft Paradox Driver (*.db)	4
Microsoft Paradox-Treiber (*.db)	4
Microsoft Text Driver (*.txt; *.csv)	4
Microsoft Text-Treiber (*.txt; *.csv)	4
Microsoft Visual FoxPro Driver	6
Microsoft Visual FoxPro-Treiber	6
SQL Server	2

2. Move down the list of available ODBC drivers and select the one for which you wish to create the ODBC data source (in this example we will pick SQL Server).

3. Click on the Finish button. A dialog box appears as shown in Figure.



Create a New Data Source to SQL Server

This wizard will help you create an ODBC data source that you can use to connect to SQL Server.

What name do you want to use to refer to the data source?

Name:

How do you want to describe the data source?

Description:

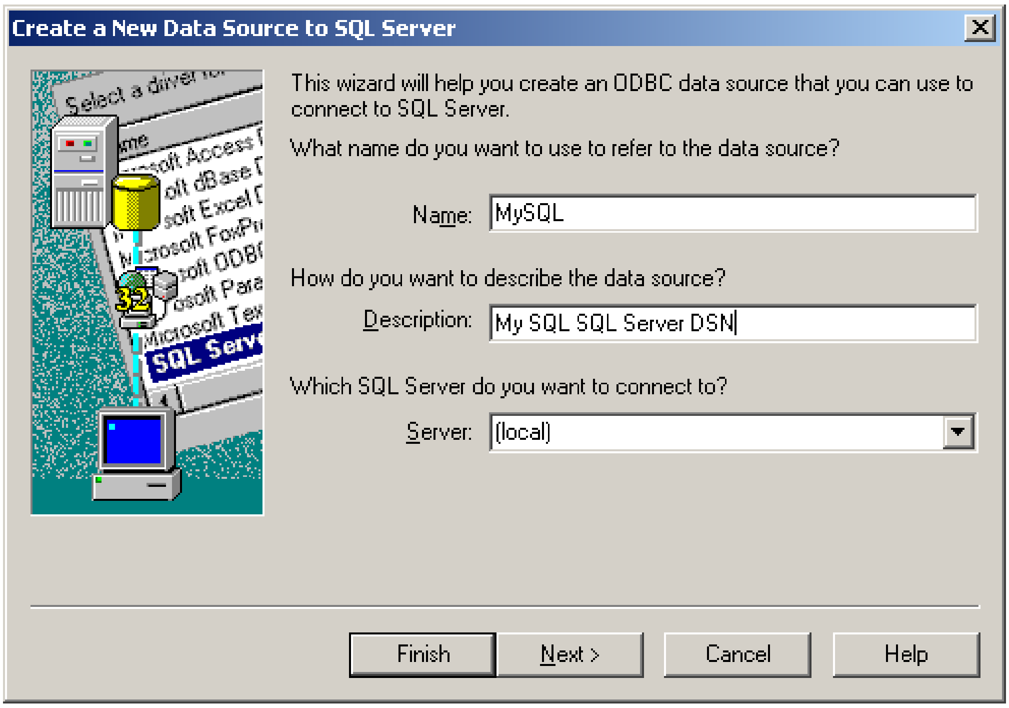
Which SQL Server do you want to connect to?

Server:

4. In the Name field, enter the name you wish to use to access this DSN. In this example, we will use "MySQL".

5. In the Description field (which is optional), enter a meaningful description.

6. In the Server field, you can enter "(local)" if the database system resides on the same system you are developing your application on, or enter the name of any valid database server on your network. In this example, we will enter "(local)" to indicate that the SQL Server resides on the same server as the application being developed as shown in Figure.



Create a New Data Source to SQL Server

This wizard will help you create an ODBC data source that you can use to connect to SQL Server.

What name do you want to use to refer to the data source?

Name: MySQL

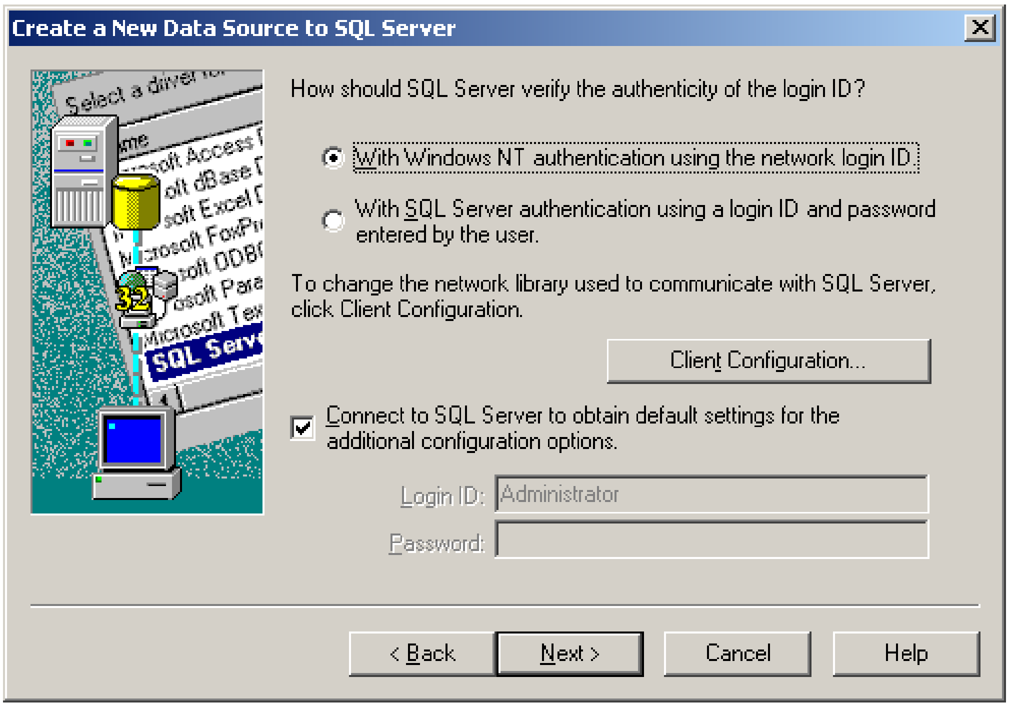
How do you want to describe the data source?

Description: My SQL SQL Server DSN

Which SQL Server do you want to connect to?

Server: (local)

7. Now click on the Next button to bring up the dialog box shown in Figure.



Create a New Data Source to SQL Server

How should SQL Server verify the authenticity of the login ID?

● With Windows NT authentication using the network login ID.

○ With SQL Server authentication using a login ID and password entered by the user.

To change the network library used to communicate with SQL Server, click Client Configuration.

☑ Connect to SQL Server to obtain default settings for the additional configuration options.

Login ID: Administrator

Password:

8. This is an important step in the process regarding security and user privileges.

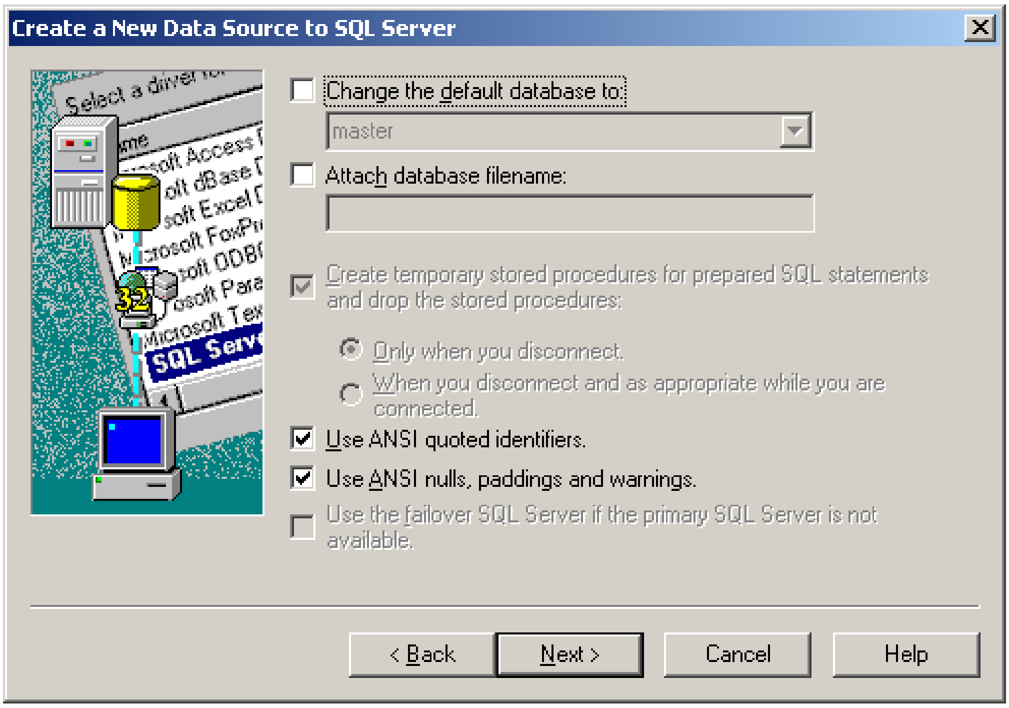
The default for this is to login to the SQL Server you have previously specified using your Windows user ID and password (With Windows 2000 authentication using the network login ID).

The second option is “With SQL Server authentication using a login ID and Password entered by the user”. If you wish to have the user of this data source log in to SQL Server directly, you should select this second option. If you do so, you will be able to specify a Login ID and Password. In this example, we will do this and change the User ID to “sa” and leave the password blank.

Note that the Client Configuration button is for advanced users and will allow you to check certain driver information and to change the default network protocol to be used. You generally should not have to use this facility.

9. Once you have changed the authenticity option and altered the User ID and Password, click on the next button.

   Because we did not change the selection for the “Connect to SQL Server to obtain default settings for the additional configuration options” checkbox, the ODBC DSN setup process will connect to SQL Server and allow us to specify some additional information. A dialog box appears as shown in Figure.



Create a New Data Source to SQL Server

□ Change the default database to:

master

□ Attach database filename:

☑ Create temporary stored procedures for prepared SQL statements and drop the stored procedures:

● Only when you disconnect.

○ When you disconnect and as appropriate while you are connected.

☑ Use ANSI quoted identifiers.

☑ Use ANSI nulls, paddings and warnings.

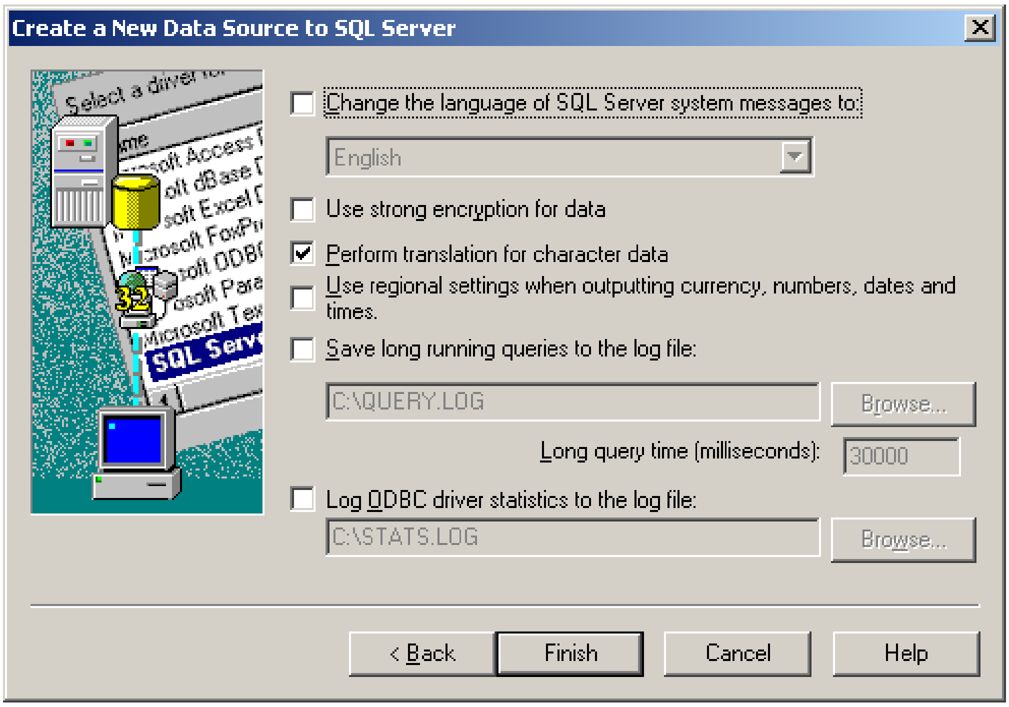
□ Use the failover SQL Server if the primary SQL Server is not available.

   This dialog box allows you to change certain installation specific options, and to specify how to create and deal with temporary stored procedures. It also allows you to change the default database to be accessed. In this case, we will change the default database to be accessed to the Pubs database.

10. Select the check box “Change the default database to:”.

11. Select the Pubs database from the dropdown list that is enabled.

12. Click on the Next button and a subsequent dialog box appears as shown in Figure.



Create a New Data Source to SQL Server

□ Change the language of SQL Server system messages to:

English

□ Use strong encryption for data

☑ Perform translation for character data

□ Use regional settings when outputting currency, numbers, dates and times.

□ Save long running queries to the log file:

C:\QUERY.LOG

Long query time (milliseconds): 30000

□ Log ODBC driver statistics to the log file:

C:\STATS.LOG

Once again some implementation specific options are made available to you.

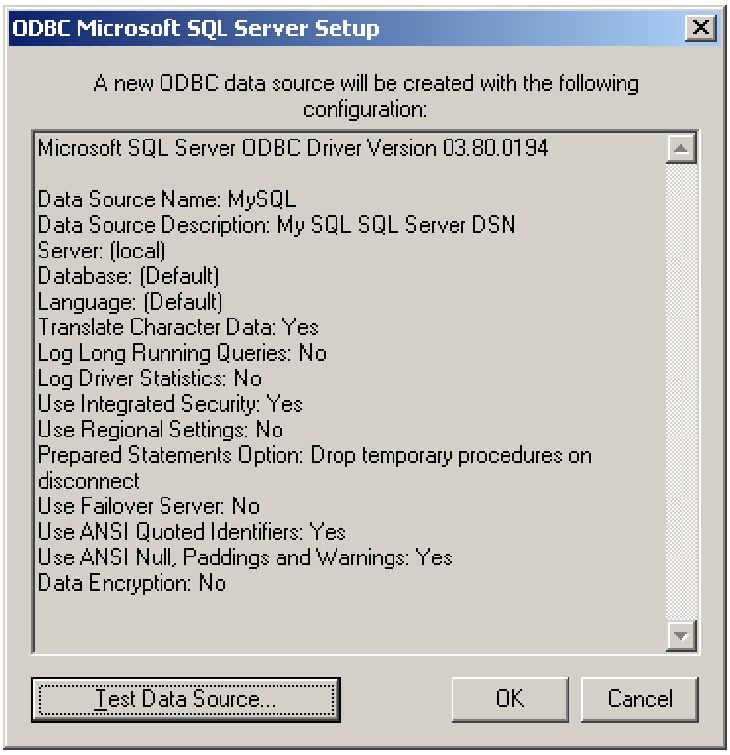
This dialog box allows you to turn on some tracing options and to specify where you want the trace information to be output.

The "Save long running queries to the log file:" option allows you to identify long running queries so that you may examine them later. This is generally used for performance tuning.

"Log ODBC driver statistics to the log file:" option allows you to have ODBC statistical information recorded at run time. This can be useful in debugging ODBC specific problems. You may also turn a tracing option on or off at any time directly in the main ODBC Data Source Administrator window under the Tracing tab.

13. When you have finished with the configuration of the new ODBC DSN, you should click on the Finish button.

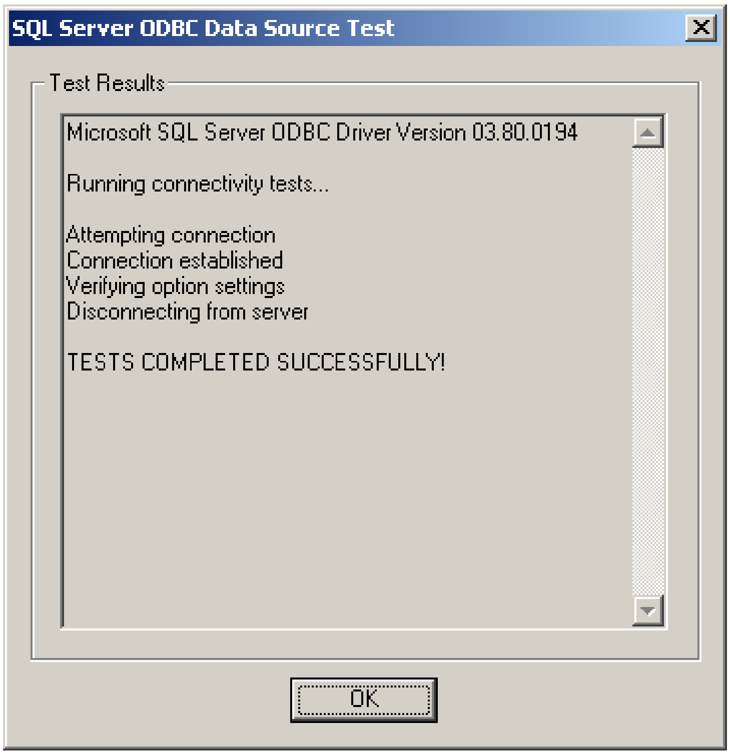
    You will be presented with a dialog box containing an overview of the information associated with the new ODBC DSN you’ve just created as shown in Figure. More importantly, there is a Test Data Source button available. Note that some versions of the ODBC software do not provide this option, so don’t be alarmed if you do not have it.



A new ODBC data source will be created with the following configuration:

```
Microsoft SQL Server ODBC Driver Version 03.80.0194

Data Source Name: MySQL
Data Source Description: My SQL SQL Server DSN
Server: (local)
Database: (Default)
Language: (Default)
Translate Character Data: Yes
Log Long Running Queries: No
Log Driver Statistics: No
Use Integrated Security: Yes
Use Regional Settings: No
Prepared Statements Option: Drop temporary procedures on
disconnect
Use Failover Server: No
Use ANSI Quoted Identifiers: Yes
Use ANSI Null, Paddings and Warnings: Yes
Data Encryption: No
```

14. If you have it available, go ahead and click on the Test Data Source button. This connects to the SQL Server specified and tries out the data source. If it tests correctly, you will be presented with the dialog box shown in Figure.



```
Microsoft SQL Server ODBC Driver Version 03.80.0194

Running connectivity tests...

Attempting connection
Connection established
Verifying option settings
Disconnecting from server

TESTS COMPLETED SUCCESSFULLY!
```

If you do not have the Test Data Source option available in your version of ODBC, you can use a generic query tool such as Microsoft’s Query (MS Query) to quickly try out a new data source.

Once you have successfully set up a new ODBC DSN for your NetCOBOL application, you are ready to set up the NetCOBOL specific runtime configuration.

4.5 Setting Up the COBOL Runtime Information

For the purpose of demonstrating a NetCOBOL application and how to set it up, we will use a simple single-program application that reads three fields from the Employee table of the Pubs database that comes with SQL Server. This project is named “MySQL”. It consists of a single .exe file named “MySQL.EXE”, which is created from the following COBOL program:

```
Identification Division.
Program-ID. MySQL
Environment Division.
Data Division.
Working-Storage Section.
01 Loop-Flag Pic 9 Value 0.
    EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 EmpID Pic X(9) Value Spaces.
01 FName Pic X(20) Value Spaces.
01 LName Pic X(30) Value Spaces.
01 SQLSTATE Pic X(5) Value Spaces.
    EXEC SQL END DECLARE SECTION END-EXEC.
Procedure Division.
    Move 0 To Loop-Flag
              SQLSTATE.
    EXEC SQL
              CONNECT TO Default
    END-EXEC.
    EXEC SQL
              DECLARE CUR1 CURSOR FOR
                 SELECT emp_id, fname, lname FROM employee
    END-EXEC.
    EXEC SQL
              OPEN CUR1
    END-EXEC.
    If SQLSTATE Not = Zeroes
       Display "Open Cursor Error! SQLSTATE: ", SQLSTATE
       Exit Program
    End-If.
    Move 0 To Loop-Flag
    Perform Until Loop-Flag Not = 0
       EXEC SQL
                 FETCH CUR1 INTO :EmpID,
                                 :FName,
                                 :LName
       END-EXEC
       IF SQLSTATE = Zeroes
          Display EmpID, " ", FName, " ", LName
       Else
          Move 1 To Loop-Flag
          EXEC SQL CLOSE CUR1 END-EXEC
       End-IF
    End-Perform.
    EXEC SQL
               DISCONNECT DEFAULT
    END-EXEC.
    Exit Program.
```

Prior to running a NetCOBOL SQL application, you must set up the SQL database(s) and table(s), ensure that your security and authorization information is properly configured and available to you, and set up an ODBC DSN.

Once you have accomplished these tasks, you must make this information available to your NetCOBOL SQL application at run time. This is done in 2 separate steps:

1. Create a NetCOBOL specific information file (.inf file) which will contain required information.
2. Point the application at the appropriate .inf file by creating/adding it to a COBOL85.CBR file.

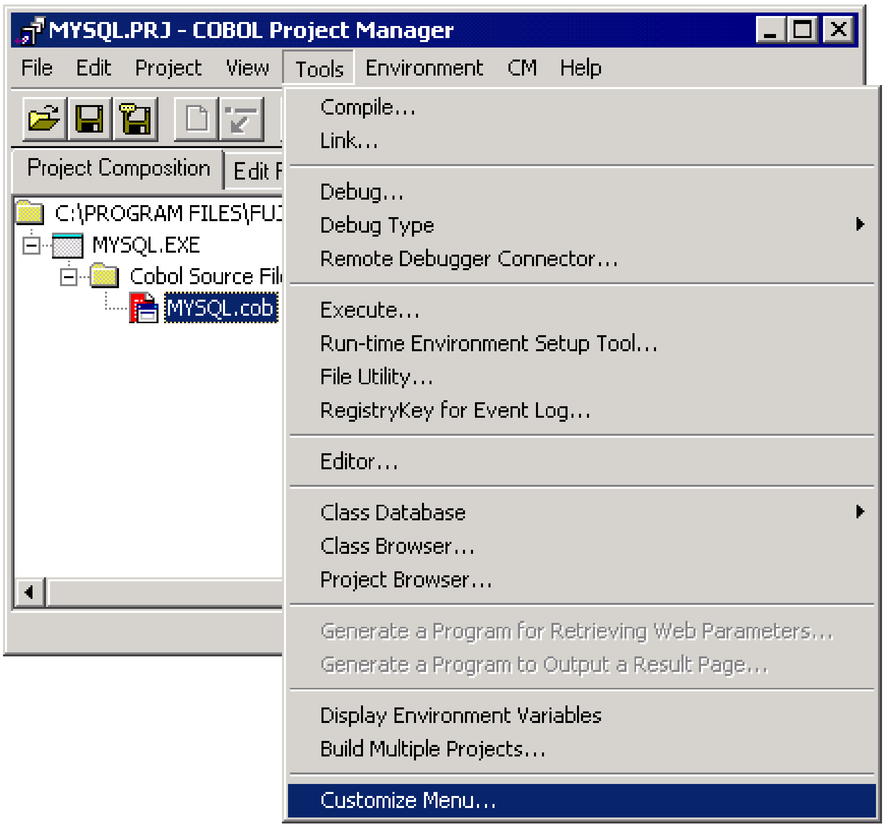
You create a .inf file using the NetCOBOL utility named “SQLODBCS.EXE”. This utility will help you specify the required parameters and write them out in the proper format. It also encrypts the Password value specified. You can view a .inf file in a text editor, but you cannot modify any password fields because the are encrypted. However, you may modify other fields in a text editor.

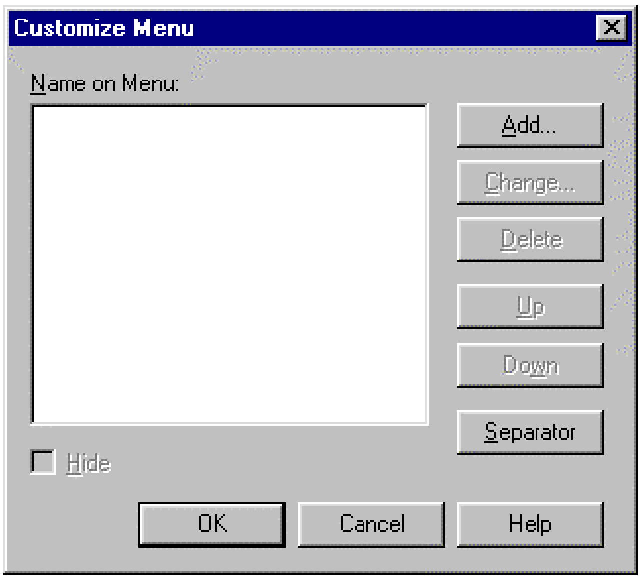
If you have not done so already, you may want to take advantage of the COBOL Project Manager’s configuration features and add the SQLODBCS.EXE utility into the Tools pull down menu. If you do not care to do this, you will have to manually find and execute this utility each time you wish to execute it.

Adding SQLODBCS.EXE to the Tools Menu

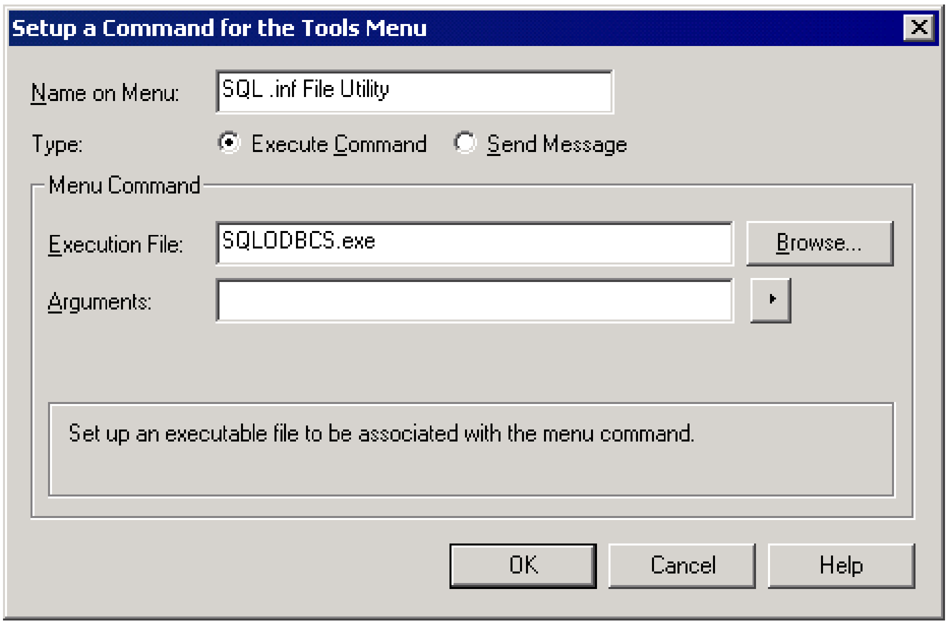
To add the SQLODBCS.EXE utility to the Tools pull down menu of the COBOL Project Manager, perform the following steps:

1. Bring up the COBOL Project Manager. Click on the Tools pull-down menu and select the Customize Menu option as shown in Figure.


MYSQL.PRJ - COBOL Project Manager
File
Edit
Project
View
Tools
Environment
CM
Help
Project Composition
Edit F
C:\PROGRAM FILES\FUJ
MYSQL.EXE
Cobol Source Fil
MYSQL.cob
Compile...
Link...
Debug...
Debug Type
Remote Debugger Connector...
Execute...
Run-time Environment Setup Tool...
File Utility...
RegistryKey for Event Log...
Editor...
Class Database
Class Browser...
Project Browser...
Generate a Program for Retrieving Web Parameters...
Generate a Program to Output a Result Page...
Display Environment Variables
Build Multiple Projects...
Customize Menu...



2. Click on the Add button to add a new menu item. Project Manager displays a dialog box that allows you to enter the name of the executable file to be associated with the new menu item. Fill in the fields as shown in Figure.



Name on Menu: SQL .inf File Utility

Type: ● Execute Command ○ Send Message

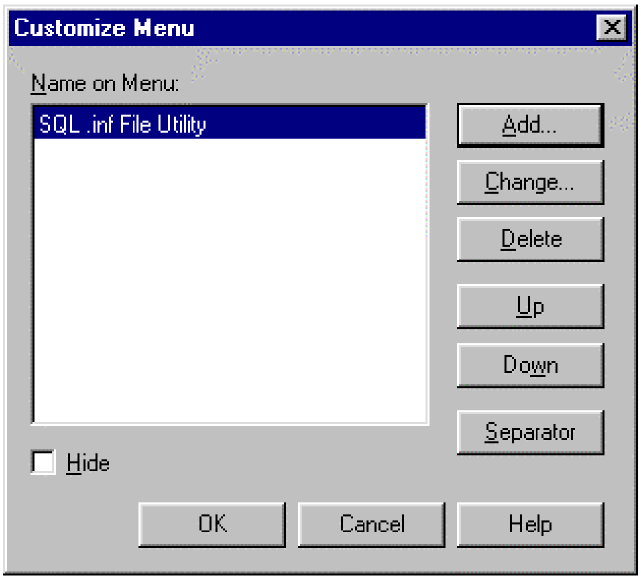
Execution File: SQLODBCS.exe

Arguments:

Set up an executable file to be associated with the menu command.

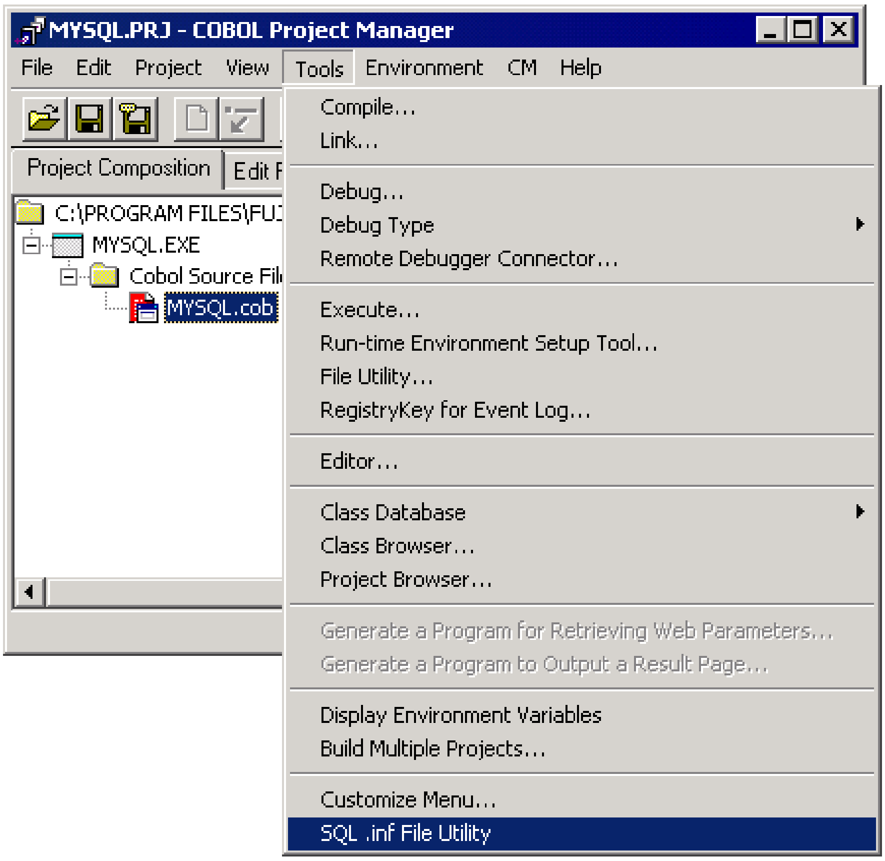
3. Click on the OK button.

   Project Manager returns you to the Customize menu dialog box displaying the information about the newly added item as shown in Figure.



4. Click on the OK button.

   The SQLODBCS utility has now been added to the Tools pull-down menu of the COBOL Project Manager. When you click on the Tools pull-down menu it should appear as shown in Figure.

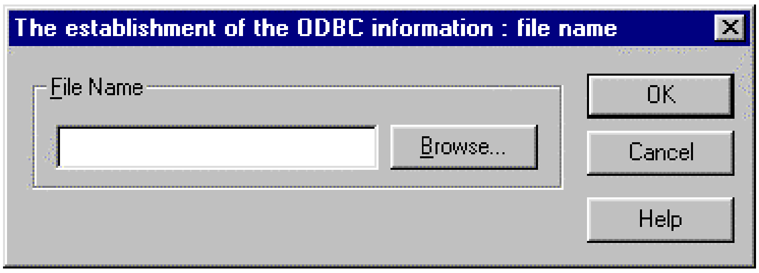


Creating a .INF File

To create a .inf file for a NetCOBOL SQL application:

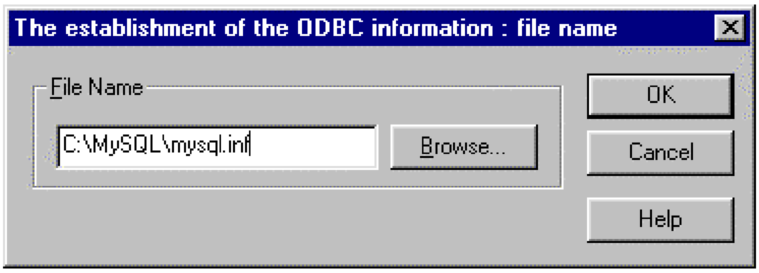
1. Select the SQL .inf File Utility from the Tools menu (or find the SQLODBCS.EXE utility in the "C:\Program Files\Fujitsu NetCOBOL for Windows" folder and execute it).

   The utility displays a file name dialog box to allow you to specify the name of the .inf file you wish to create or modify as shown in Figure.



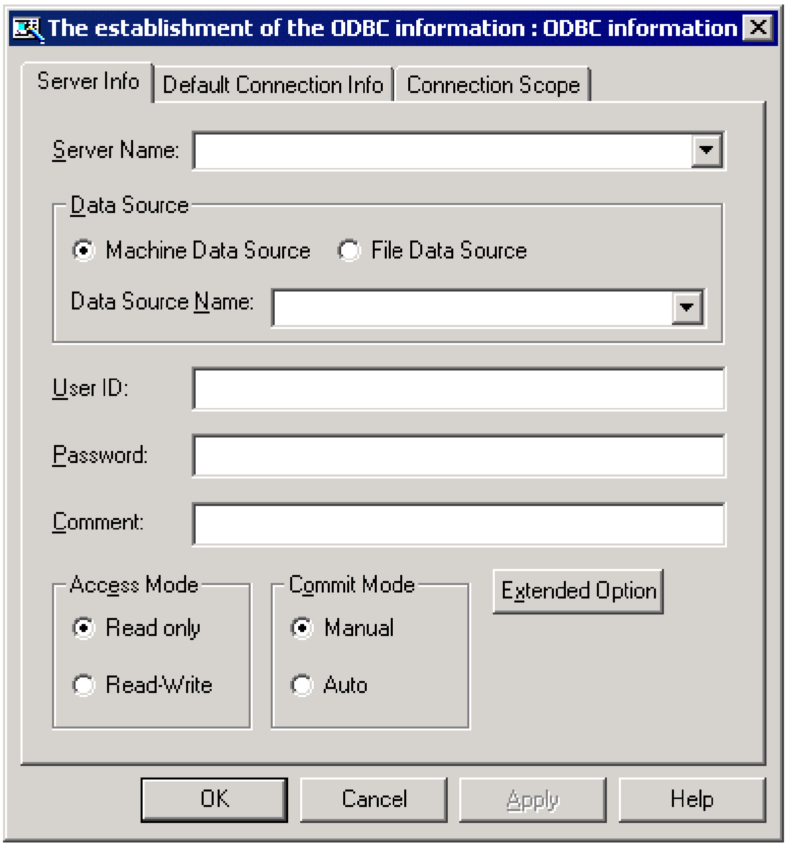
   Note that it is important to fully specify the drive and path of the .inf file you wish to create or modify, to ensure it gets saved to the proper location. If you fail to do this, your new .inf file may be saved to a different directory than where you expect to find it later. For our example, we are going to create a .inf file named C:\MYSQL\MySQL.inf.

2. Enter the name of the .inf file you wish to create as shown in Figure.



File Name: C:\MySQL\mysql.inf

3. Click on the OK button, respond Yes to the message asking you if you wish to create a new file, and OK to the message confirming that a file was created. The SQLODBCS utility window appears as shown in Figure.



Server Name:

Data Source: ● Machine Data Source ○ File Data Source

Data Source Name:

User ID:

Password:

Comment:

Access Mode: ● Read only ○ Read-Write

Commit Mode: ● Manual ○ Auto

In order to understand how to specify parameters for a .inf file in the SQLODBCS utility, you need to understand about connections to servers from a NetCOBOL application.

To establish a connection to an actual SQL database in a NetCOBOL program (as shown in the example COBOL program above), you must code an SQL CONNECT statement such as:

```
EXEC SQL
   CONNECT TO 'FRED'
END-EXEC.
```

In the above code example, the NetCOBOL program is attempting to connect to a server name “FRED”. This is a bit misleading as “FRED” is not the name of an actual SQL or 2000 Server. Instead, “FRED” is a symbolic name to be associated with an entry in a NetCOBOL .inf file.

Remember that when you created an ODBC Data Source in the ODBC Data Sources administrator, you specified the actual SQL Server to connect to. This connection is managed by ODBC and it does not make sense for a COBOL program to specify a physical server, as conflicts could arise with ODBC Data Sources.

The name “FRED” therefore simply refers to an entry in the .inf file, which will contain other parameters, including the name of the actual ODBC data source to use at runtime. Another reason for this level of abstraction from the physical server name is to allow you to set up multiple entries for the same physical server. You might have many different ODBC data sources defined for a variety of different databases and/or tables that exist on a single SQL Server. You would thus require a method to be able to specify these different ODBC data sources.

Another important point is the ability to connect to a default server. The code for this looks like:

```
EXEC SQL
   CONNECT TO DEFAULT
END-EXEC.
```

In the above case, “DEFAULT” is a special symbolic name. It allows you to abstract the name of the symbolic connection out of the program. You can specify a DEFAULT server entry in the SQLODBCS utility to point at any data source (of course you could also make ‘FRED’ your default entry and always connect to ‘FRED’ as well). You can thus think of “DEFAULT” as just another symbolic entry in the .inf file. Simply notice that “DEFAULT” does not require quotes around it like other symbolic names in an SQL CONNECT statement.

Now that you understand symbolic server names and how they are used in SQL CONNECT statements, take a look back at the SQLODBCS window as shown in Figure.

Notice that there are two areas for entering parameters. Near the bottom is an area entitled “Default Connection Information”. You must enter the name of a default server, a User ID and a Password. These three fields are required and will be used at runtime. You may remember that you specified a User ID and Password in the ODBC data source when you created it under the ODBC Data Sources Administrator. Note that NetCOBOL will ignore these, however, and will instead always use the User ID and Password you specify in the SQLODBCS utility (the User ID and Password stored in the .inf file you create). Note that the Password will be encrypted and not viewable.

You can override the User ID and Password from within the actual COBOL program in the CONNECT Statement. For example, if you code the following:

```
EXEC SQL
   CONNECT TO ‘FRED’
  USER ‘ID0001/PW01’
END-EXEC.
```

The value “ID0001” will be used as the User ID, and the value “PW01” will be used as the password when connecting to the actual database server, regardless of the User ID and Password currently specified in the .inf file.

If you code:

```
EXEC SQL
   CONNECT TO ‘FRED’
  USER ‘ID0001’
END-EXEC.
```

The value “ID0001” will be used for the User ID, but since you did not specify a Password, the Password currently specified in the .inf file will be used when connecting to the actual database server.

In neither case will the User ID or Password specified in the actual ODBC data source be used - NetCOBOL will always override this.

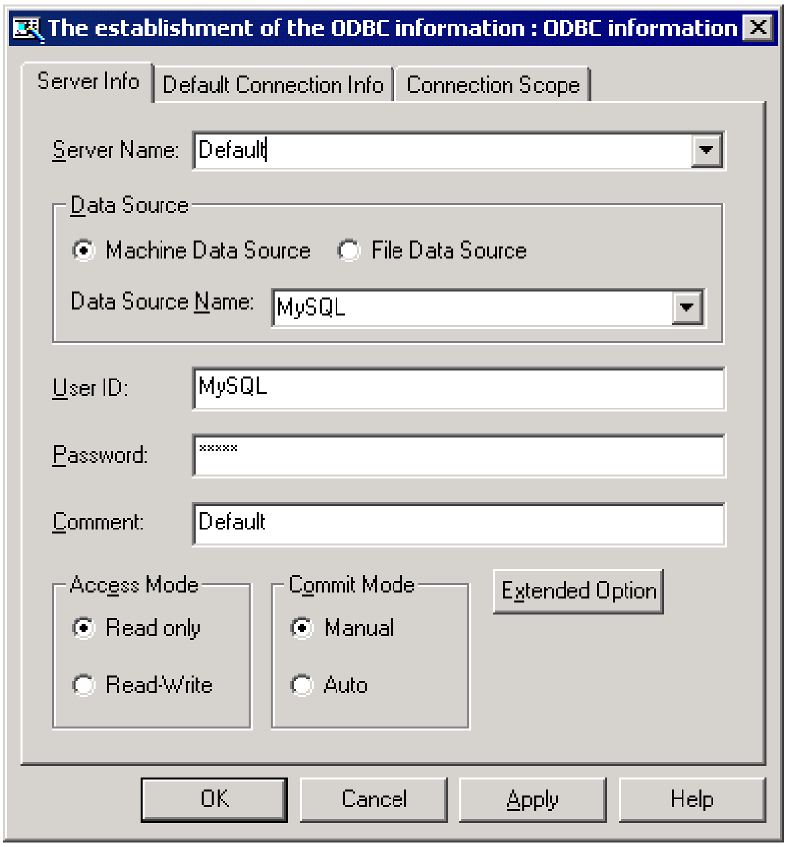
This means that you must specify a User ID and Password in the SQLODBCS utility for any .inf file you wish to create. If you leave these out, the SQLODBCS utility will refuse to create your .inf file. If you manually delete these entries in a .inf file, you will receive an error at runtime.

Our simple SQL COBOL application noted above connects to DEFAULT as shown in the code below.

```
EXEC SQL
   CONNECT TO DEFAULT
END-EXEC.
```

We now continue with the steps for creating the .inf file.

4. To create the appropriate .inf file, enter the parameters shown in Figures. If you are using multithreading or object oriented programs, see the SQLODBCS Help for the appropriate Connection Scope settings.



The establishment of the ODBC information : ODBC information

Server Info | Default Connection Info | Connection Scope

Server Name: Default

Data Source: ● Machine Data Source ○ File Data Source

Data Source Name: MySQL

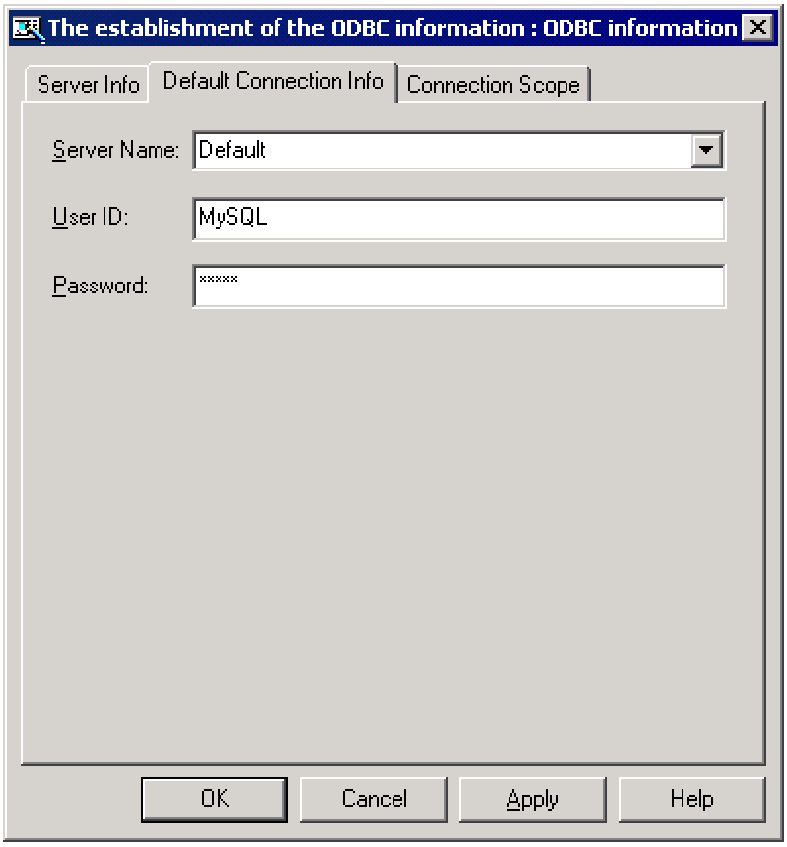
User ID: MySQL

Password: *****

Comment: Default

Access Mode: ● Read only ○ Read-Write

Commit Mode: ● Manual ○ Auto

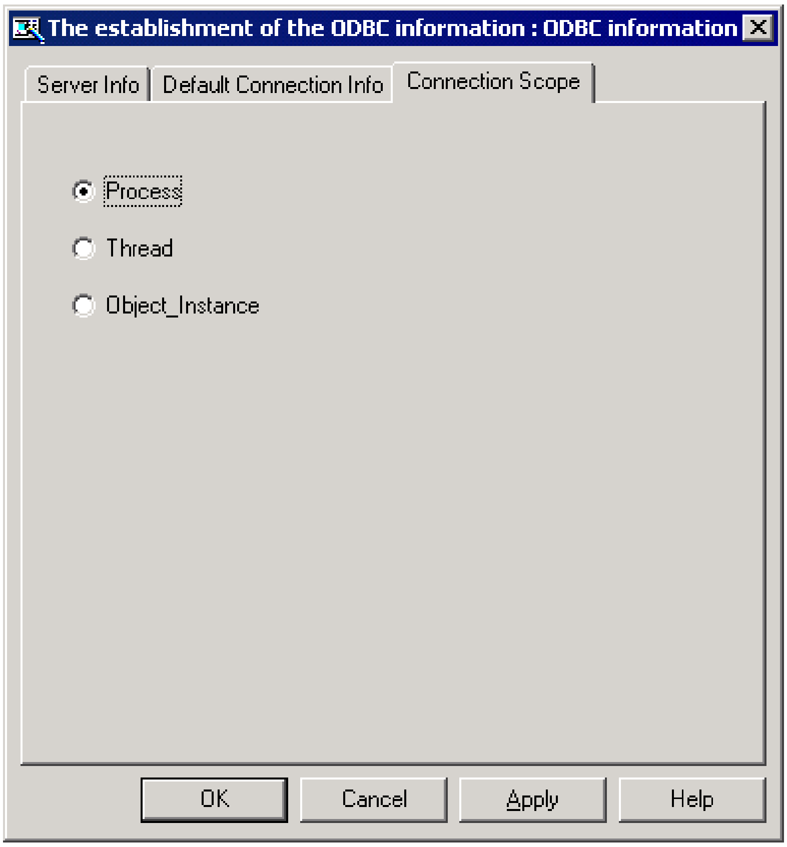


The establishment of the ODBC information : ODBC information

Server Name: Default

User ID: MySQL

Password: *****



The establishment of the ODBC information : ODBC information

● Process

○ Thread

○ Object_Instance

5. Click on the Apply button to create the .inf file and write it to disk.

   The .inf file created by the above parameters will contain the following entries:

```
[SERVER LIST]
Default=Default
[Default]
@SQL_DATASRC=MySQL
@SQL_USERID=MySQL
@SQL_PASSWORD=BEINKAOHLEPJENBABBMM
@SQL_ACCESS_MODE=READ_ONLY
@SQL_COMMIT_MODE=MANUAL
@SQL_CONCURRENCY=READ_ONLY
@SQL_ODBC_CURSORS=USE_DRIVER
@SQL_QUERY_TIMEOUT=0
[SQL_DEFAULT_INF]
@SQL_SERVER=Default
@SQL_USERID=MySQL
@SQL_PASSWORD=BEINKAOHLEPJENBABBMM
[CONNECTION_SCOPE]
@SQL_CONNECTION_SCOPE=PROCESS
```

The Server List contains a line indicating that the server specified as the Default connection (for any program trying to connect to DEFAULT) will be the symbolic entry “Default” (Default=Default).

Following this is the entry information for the symbolic server name “Default”. Note that Entry headings are designated with square brackets around them. This entry contains the name of the ODBC data source to be used (@SQL_DATASRC=MySQL), the access mode to be used (@SQL_ACCESS_MODE=READ_ONLY), and the commit mode to be used (@SQL_COMMIT_MODE=MANUAL).

Following the entry for the symbolic server “Default” is an entry for the default information to be used (SQL_DEFAULT_INF). This contains the name of the default symbolic server and a User ID and Password to be used to access it. Note that the Password value has been encrypted.

Setting up the Runtime Environment Information

Once you have set up your SQL databases and tables, created the appropriate ODBC data source, and created the appropriate .inf file, you are ready to execute your application.

You must, however, ensure that you point the NetCOBOL runtime at the appropriate .inf file. By finding the .inf file, the NetCOBOL runtime will have all of the information needed to interface with ODBC at runtime to access the database tables your application references.

To point the runtime to the appropriate .inf file, you need to create a COBOL85.CBR file (or add a new entry for your application to an existing COBOL85.CBR file). You should create this file in the directory in which your main application executable (.exe) file resides.

It should include the following lines of code:

```
[MYSQL]
@ODBC_Inf=c:\mysql\mysql.inf
```

The entry “[MYSQL] designates the name of the main program. The line “@ODBC_Inf=c:\mysql\mysql.inf” tells the runtime which .inf file to use and where it is located. That’s all there is to it.

Instead of editing the COBOL85.CBR file using a text editor, you can use the Runtime Environment Setup Tool, available from COBOL Project Manager’s Tools menu. Keys to using this tool are that:

- The file you want to edit/create is COBOL85.CBR.
- The Section you want to edit/create is MySQL.
- You can select @ODBC_Inf from the pull-down list.
- You can use the browse (···) button to select the .inf file.
- You MUST click Set to add this variable to the list.
- You MUST click Apply to save the setting.

4.6 Potential Execution Errors Caused by Incorrect Setups

There are a number of errors that may occur at runtime if you do not set up the ODBC environment for a NetCOBOL SQL application properly.

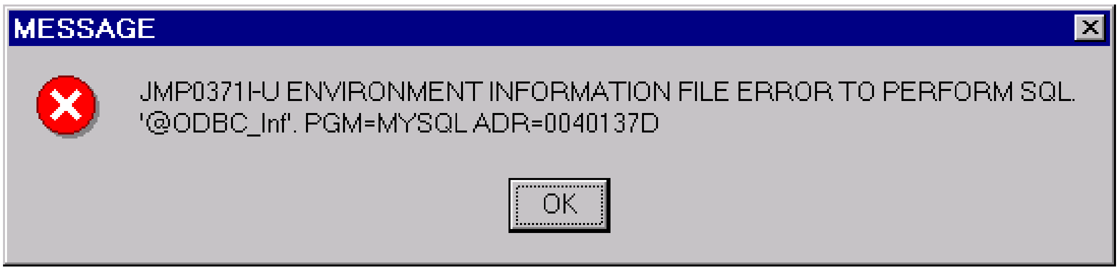
For Example, if you code an erroneous SQL statement, within an EXEC SQL ···. ENDEXEC statement, the NetCOBOL compiler may catch the error and issue an error message.

In some cases, however, the error will not be determined until runtime. Some of the most common examples are listed below.

1. A common runtime error comes from not specifying to the Runtime Environment the name of a valid .inf file. For example, in the above example application, if you forget to specify the line:

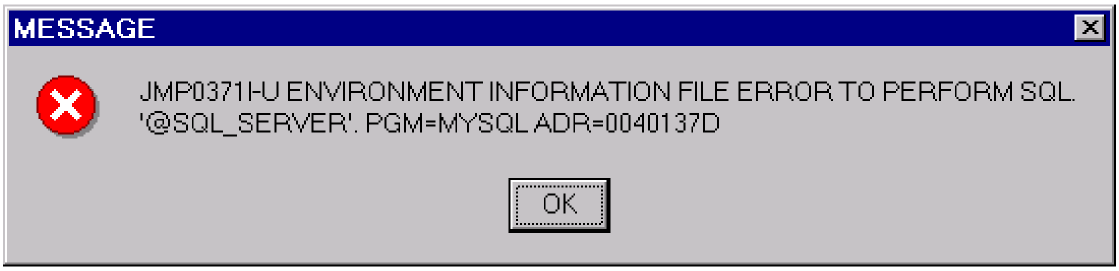
```
@ODBC_Inf=c:\mysql\mysql.inf
```

In COBOL85.CBR, you will receive an error dialog box as shown in Figure. This error occurs whenever no @ODBC_inf parameter is specified.



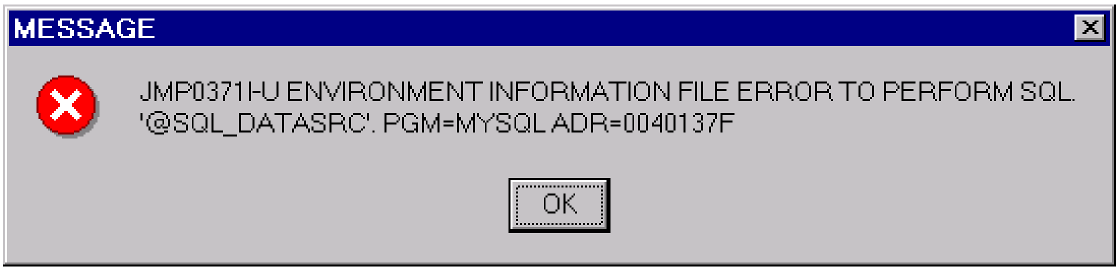
JMP0371I-U ENVIRONMENT INFORMATION FILE ERROR TO PERFORM SQL. '@ODBC_Inf'. PGM=MYSQL ADR=0040137D

2. If you specify an @ODBC_inf parameter and it is found, but it contains the name of a file that does not exist or cannot be found, you receive the error dialog box shown in Figure.



JMP0371I-U ENVIRONMENT INFORMATION FILE ERROR TO PERFORM SQL. '@SQL_SERVER'. PGM=MYSQL ADR=0040137D

3. If your COBOL program attempts to connect to a server that is not specified in your .inf file, you will receive the error dialog shown in Figure.



JMP0371I-U ENVIRONMENT INFORMATION FILE ERROR TO PERFORM SQL. '@SQL_DATASRC'. PGM=MYSQL ADR=0040137F

Capturing Other Errors

There are other errors that will not produce an Error Message dialog box. In fact, they will only produce errors by returning non-zero values in SQLSTATE and/or SQLCODE. You may notice that the simple application above only checks SQLSTATE. You will find two additional data items useful if you add them to your EXEC SQL BEGIN DECALRE SECTION END-EXEC group. They are SQLCODE and SQLMSG. They are defined as follows:

```
   EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 EmpID Pic X(9) Value Spaces.
01 FName Pic X(20) Value Spaces.
```

```
01 LName Pic X(30) Value Spaces.
01 SQLSTATE Pic X(5) Value Spaces.
01 SQLCODE Pic S9(9) Comp-5 Value Zeroes.
01 SQLMSG Pic X(254) Value Spaces.
   EXEC SQL END DECLARE SECTION END-EXEC.
```

SQLMSG is of particular use. The ODBC driver will place descriptive error messages in this variable. For example, if you code the name of an ODBC data source that does not exist in the “@SQL_DATASRC=” parameter in your .inf file, you will not receive an error dialog box. Instead you will receive a value of IM002 in SQLSTATE. This is not very helpful. But if you look at the value contained in SQLMSG, it will be:

```
[Microsoft] [ODBC Driver Manager] Data source name not found and no default driver specified.
```

There are some SQL syntax errors that will not be caught by the NetCOBOL compiler and will only show up at runtime. For example, if you code an erroneous Where clause in a Select statement such as:

```
EXEC SQL
     DECLARE CUR1 CURSOR FOR
        SELECT emp_id, fname, lname FROM employee
        Where emp_id Not NULL
END-EXEC.
```

Note that “Not NULL” is invalid SQL Code. When this statement is executed at runtime, you will receive a value of 37000 in SQLSTATE (again, not very helpful), but SQLMSG will contain:

```
[Microsoft] [ODBC SQL Server Driver] [SQL Server] Incorrect syntax near the keyword ‘Null’ .
```

It is thus a good idea to always include SQLMSG in your COBOL program and to interrogate it whenever error messages occur. You may wish to display SQLMSG when debugging after key SQL statements in your program.

Appendix A Sample Programs

The sample programs shipped with NetCOBOL are intended to give an overview of the capabilities of NetCOBOL and COBOL Project Manager. Refer to the “NetCOBOL User’s Guide” for further details on using NetCOBOL. The following table details the sample programs available with NetCOBOL.

Table A.1 NetCOBOL Sample Programs

Sample	Purpose of sample	Functions Used
Sample01	Data Processing Using Standard Input-Output	- ACCEPT/DISPLAY (console window) - Debugger
Sample02	Using Line Sequential and Indexed Files	- Line sequential file (reference) - Indexed file (creation
Sample03	Screen Input-Output Operation Using the Presentation File Function	Sample 3 is not available in the English version of NetCOBOL.
Sample04	Screen Input-Output Using the Screen Section	- Screen handling - Indexed file (reference)
Sample05	Calling COBOL Subprograms	- Inter-program communication - Library fetch - ACCEPT/DISPLAY (message box) - Print file - Indexed file (reference) - Line sequential file (creation) - Passing parameters for execution - Free format
Sample06	Receiving a Command Line Argument	- Fetching command line arguments - Internal program
Sample07	Environment Variable Handling	- Environment variable handling
Sample08	Using a Print File	- Print file - ACCEPT/DISPLAY (console window)
Sample09	Using a Print File (Advanced usage)	- Print file (changing print form, letter form, and layout)
Sample10	Using a Print File with FORMAT clause	Sample 10 is not available in the English version of NetCOBOL.
Sample11	Remote database access	- Embedded SQL statements (reference)
Sample12	Remote database access (multiple row processing)	- Embedded SQL statements (reference, update and delete)
Sample13	Calling COBOL from Visual Basic	- Visual Basic - COBOL subprograms
Sample14	Visual Basic calling COBOL -Simple ATM Example	- Visual Basic - COBOL subprograms
Sample15	Basic Object-Oriented Programming	- Class definition (encapsulation) - object generation - method invocation

Sample	Purpose of sample	Functions Used
Sample16	Collection Class (Class Library)	- Class definition (encapsulation) - object generation - method invocation
Sample17	Object-Oriented Cobol (Aggregation, Singleton, and Iteration)	- Using Class Library - Multiple inheritance - polymorphism
Sample18	Advanced Object-Oriented Programming	- Using Class Library - Multiple inheritance - polymorphism
Sample19	Object Persistence (Indexed File)	- polymorphism - Indexed file
Sample20	Object Persistence (Database)	- polymorphism - Remote database access (ODBC)
Sample21	Multithread Programming	- Multithread programming - Index file (creation, reference, update, and rewrite) - External data/file - Data locking - COBOL ISAPI Subroutine
Sample22	Multithread Programming (Advanced usage)	- Multithread programming - COBOL ISAPI Subroutine - Event log (output of user definition information) - External data/file - Data locking - Sharing data between threads
Sample23	COM Program to Control Excel (Late Binding)	- COM Client functions (Late binding)
Sample24	COM Program to Control Excel (Early Binding)	- COM Client functions (Early binding)
Sample25	Creating a COBOL COM Server Program	- COM Server functions - Remote Database Access - *COM-ARRAY class
Sample26	Using the COM Linkage-COBOL Server Program (COBOL Client)	- Using COBOL COM Server - COM Client Functions - *COM-ARRAY class - Screen handling
Sample27	Using the COM Linkage-COBOL Server Program (ASP Client)	- Using COBOL COM Server - ASP client
Sample28	Transaction Management with COM Linkage-MTS	- COM Server functions

Sample	Purpose of sample	Functions Used
		- Object context object(MTS) - Remote Database Access - *COM-ARRAY class
Sample29	Inter-application Communication Function	- Inter-application communication
Sample30	Using UNICODE	Sample 30 is not available in the English version of NetCOBOL.
Sample31	Windows System Function Call	- Calling Windows API’s - STDCALL calling convention - BY VALUE parameters - RETURNING phrase of a CALL statement - PROGRAM-STATUS special register
Sample32	Starting other programs	- Calling Windows API’s - STDCALL calling convention - STORED-CHARACTER-LENGTH function

A.1 Using the Sample Program Projects

There are project files for each sample COBOL application. You can compile and link any of them through the following steps:

1. Select the project file (by double clicking on it) under the appropriate samples directory. For instance, the project file for Sample 1 will be on the drive where you installed the product, under

   %install_path%\Samples\cobol\Sample01; the filename is sample01.prj

   (%install_path% is “C:\Program Files\Fujitsu NetCOBOL for Windows” in default).

2. Once you have loaded the project file into COBOL Project Manager, you can build (compile and link) the programs in the project by selecting Build from the Project menu, or by pressing the F7 key. Refer to the section on Using Project Manager in Chapter 1, “A Quick Tour” for more details as well as the “NetCOBOL User’s Guide.”

3. All sample projects have the TEST compiler option set, so once you build the projects you can monitor their execution under the COBOL Debugger by selecting Debug from the Project menu in COBOL Project Manager. See the description of debugging in Chapter 1, “A Quick Tour” for more details as well as the “NetCOBOL Debugging Guide.”

4. To execute the program select Execute from the Project menu. You need to have the Project or the .EXE file selected in the project tree for the Execute function to be enabled.

The description for each sample usually includes instructions to step you through one or more aspects of the program and include different demonstrations of using the tools that come with NetCOBOL.

A.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 beginning with the input alphabetic character to the console window.

Files Included in Sample 1

- SAMPLE1.COB (COBOL source program)
- SAMPLE01. PRJ (COBOL project file)
- SAMPLE01. CBI (COBOL compilation option file)

COBOL Statements Used

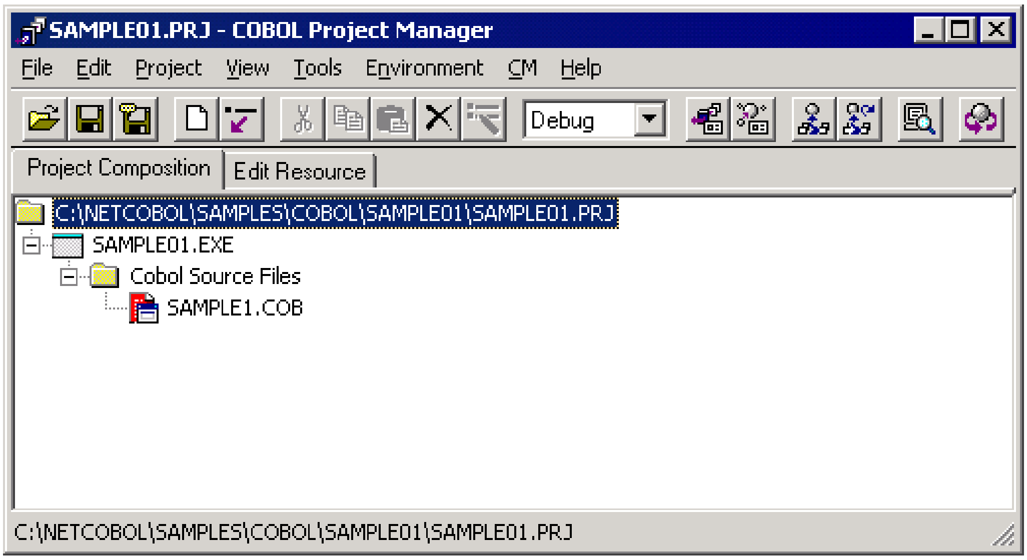
ACCEPT, DISPLAY, EXIT, IF, and PERFORM statements are used.

Building/Rebuilding the Program

Project manager's build function is used to create the executable program.

In the following screen snapshots, the sample program was installed to C:\NetCOBOL. Your installation folder may be different.

1. The project manager is started, project file "SAMPLE01.PRJ" is opened.

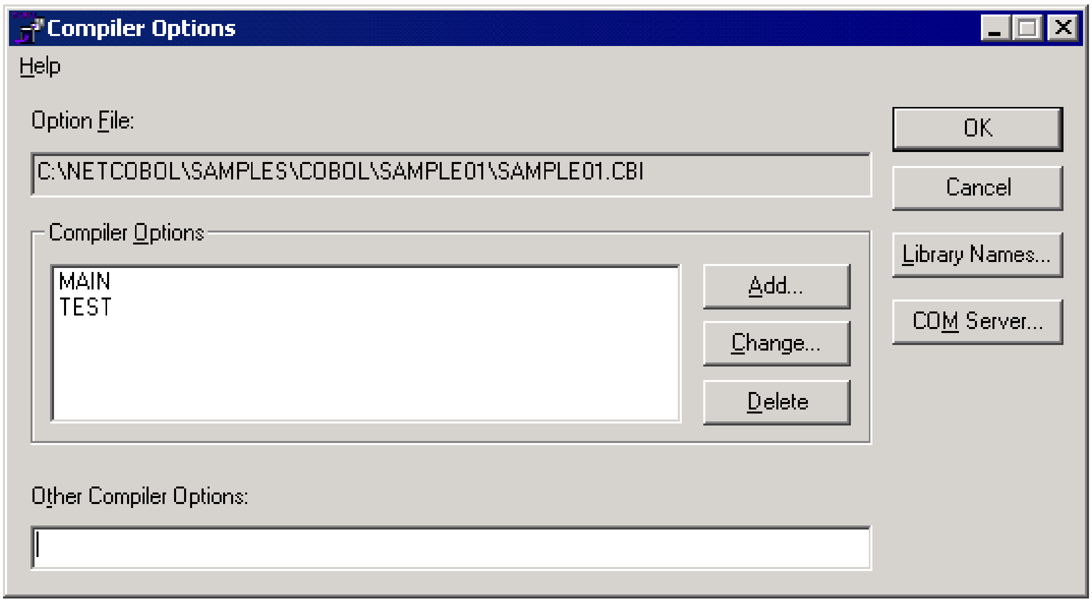

SAMPLE01.PRJ - COBOL Project Manager
File Edit Project View Tools Environment CM Help
Debug
Project Composition
Edit Resource

C:\NETCOBOL\SAMPLES\COBOL\SAMPLE01\SAMPLE01.PRJ
SAMPLE01.EXE
Cobol Source Files
SAMPLE1.COB

C:\NETCOBOL\SAMPLES\COBOL\SAMPLE01\SAMPLE01.PRJ

2. The project file is selected, and "Compiler options" is selected from "Project"- "Options" menu.

   The "Compiler options" dialog is displayed.



Option File: C:\NETCOBOL\SAMPLES\COBOL\SAMPLE01\SAMPLE01.CBI

Compiler Options:
MAIN
TEST

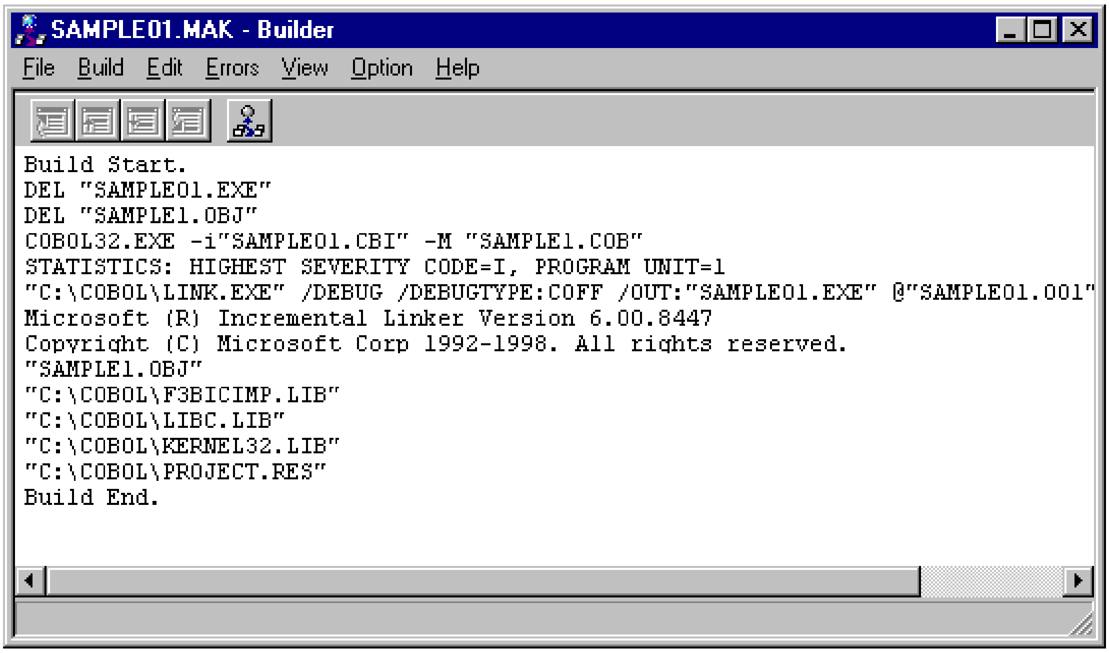
Other Compiler Options:

3. Confirm compiler option TEST is specified. After confirming the information, click the OK button.

   You are now returned to the Project Manager window.

4. Select "Build" from Project Manager's "Project" menu or click on the Build button in the Project Manager toolbar.

   After the build terminates, check that SAMPLE01.EXE is created.



```
Build Start.
DEL "SAMPLE01.EXE"
DEL "SAMPLE1.OBJ"
COBOL32.EXE -i"SAMPLE01.CBI" -M "SAMPLE1.COB"
STATISTICS: HIGHEST SEVERITY CODE=I, PROGRAM UNIT=1
"C:\COBOL\LINK.EXE" /DEBUG /DEBUGTYPE:COFF /OUT:"SAMPLE01.EXE" @"SAMPLE01.001"
Microsoft (R) Incremental Linker Version 6.00.8447
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.
"SAMPLE1.OBJ"
"C:\COBOL\F3BICIMP.LIB"
"C:\COBOL\LIBC.LIB"
"C:\COBOL\KERNEL32.LIB"
"C:\COBOL\PROJECT.RES"
Build End.
```

Note

If the sample has already been built you will just see the “Build Start” and “Build End” messages.

The MAIN compile option is needed to indicate the main program in a NetCOBOL application run-unit. In COBOL Project Manager you select Main - Window or Main - Console from the pop-up menu when the COBOL source program is selected. All sample programs have the appropriate program set as the main program.

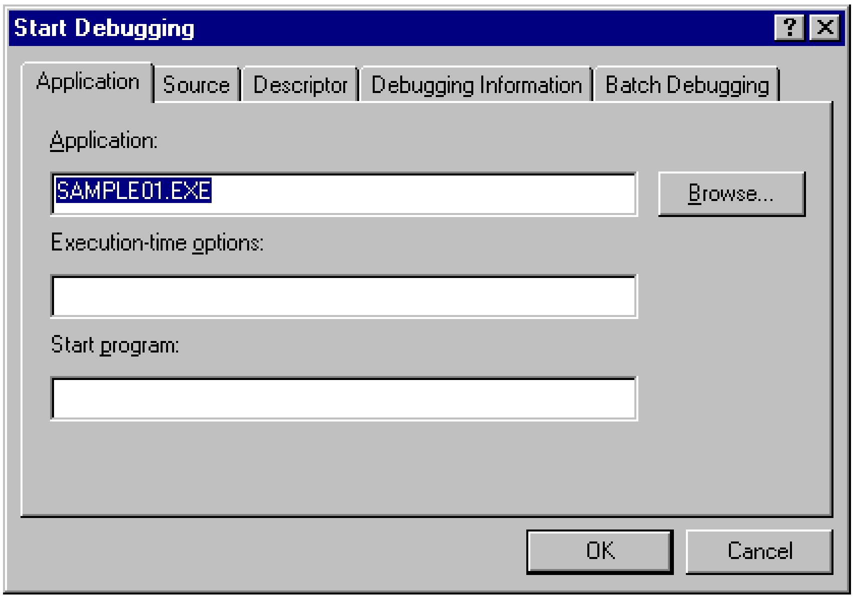
Debugging the Program

These instructions demonstrate starting the debugger, stepping through a program, changing a data item, setting and running to a breakpoint.

To start Sample 1 using the interactive debugger, do the following:

1. In COBOL Project Manager, from the Project menu select Debug.

   The debugger starts and displays the Start Debugging dialog:



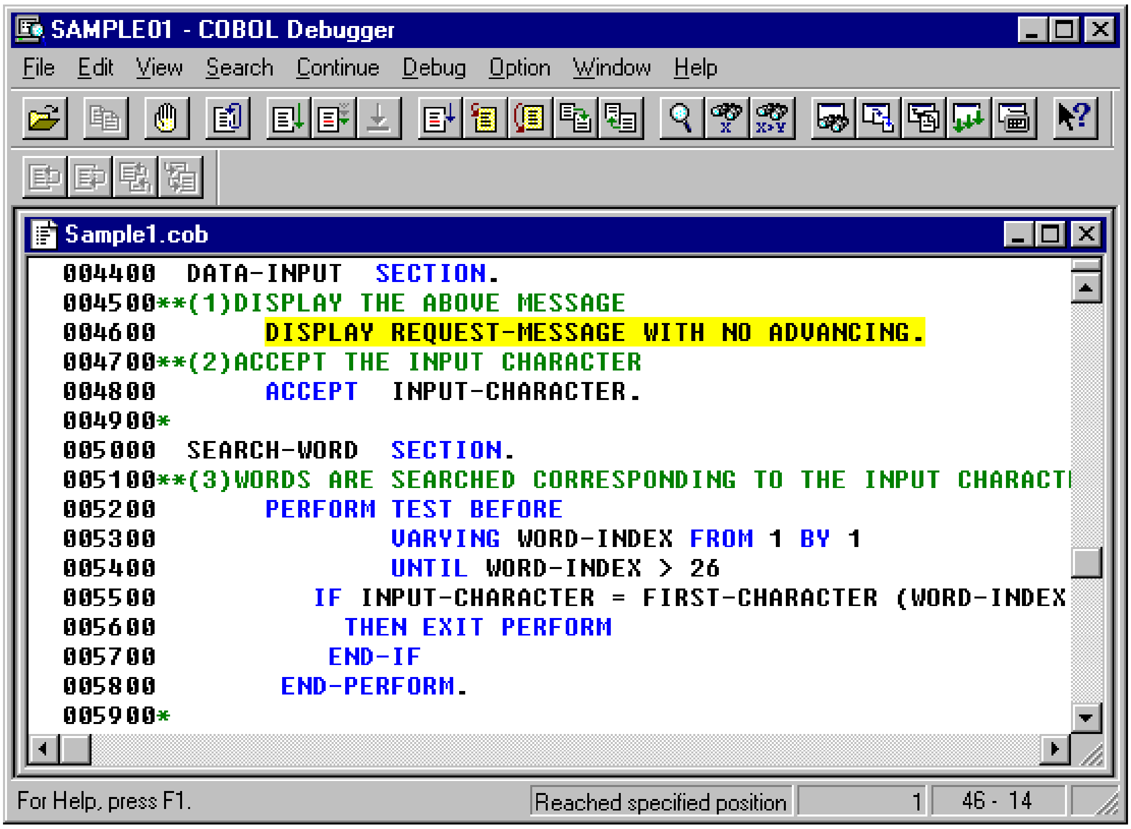
Application:

SAMPLE01.EXE

Execution-time options:

Start program:

2. Click on the OK button to enter the COBOL Debugger window.



```
004400  DATA-INPUT   SECTION.
004500**(1)DISPLAY THE ABOVE MESSAGE
004600       DISPLAY REQUEST-MESSAGE WITH NO ADVANCING.
004700**(2)ACCEPT THE INPUT CHARACTER
004800       ACCEPT  INPUT-CHARACTER.
004900*
005000  SEARCH-WORD  SECTION.
005100**(3)WORDS ARE SEARCHED CORRESPONDING TO THE INPUT CHARACT
005200       PERFORM TEST BEFORE
005300               VARYING WORD-INDEX FROM 1 BY 1
005400               UNTIL WORD-INDEX > 26
005500          IF INPUT-CHARACTER = FIRST-CHARACTER (WORD-INDEX
005600            THEN EXIT PERFORM
005700           END-IF
005800        END-PERFORM.
005900*
```

For Help, press F1.   Reached specified position   1   46 - 14

3. Step through the program.

   The program automatically positions at the first executable statement (Line 46).

   Select Step Into, from the Toolbar or press the F8 key to step (advance) to the next executable statement (Line 48). Step again to execute the Accept statement on Line 48.

Note

The Accept statement requires user input.

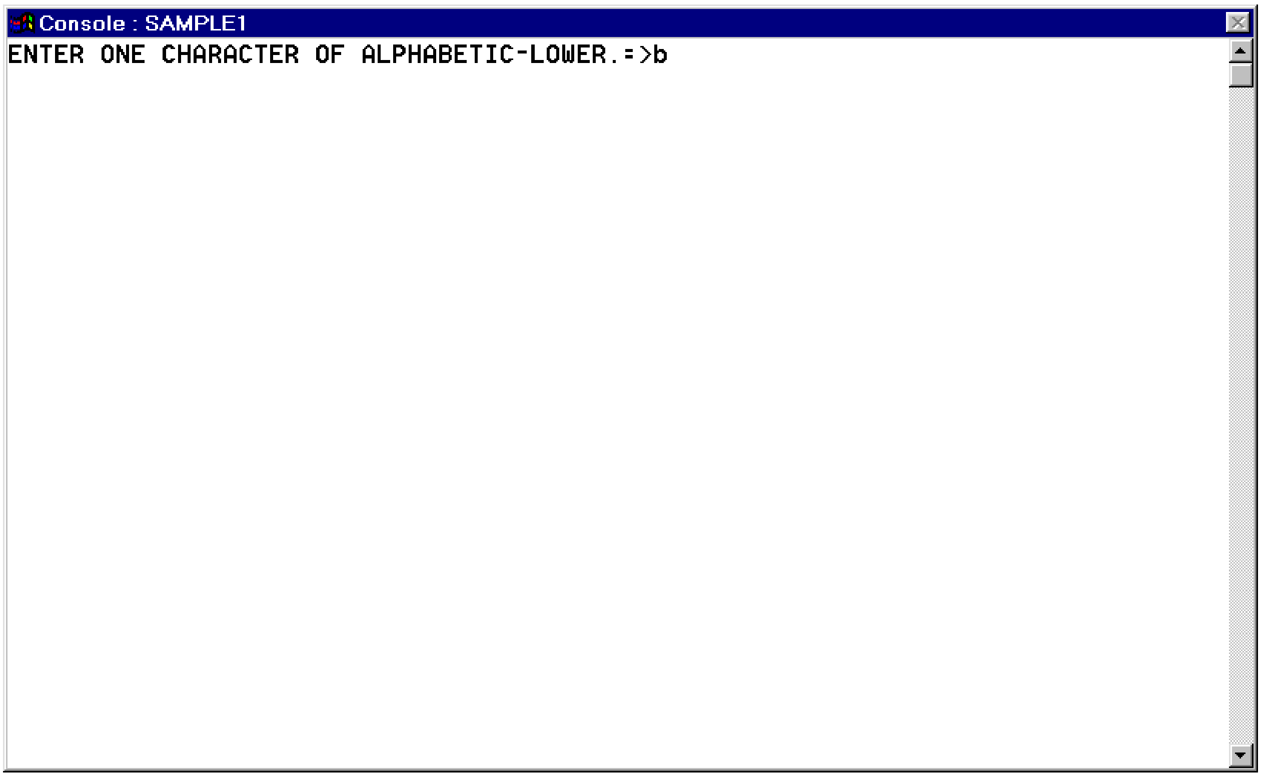
4. Input the data item (a letter of the alphabet).

   Make the console window active.

Note

The console window may be hidden behind the COBOL Debugger window.

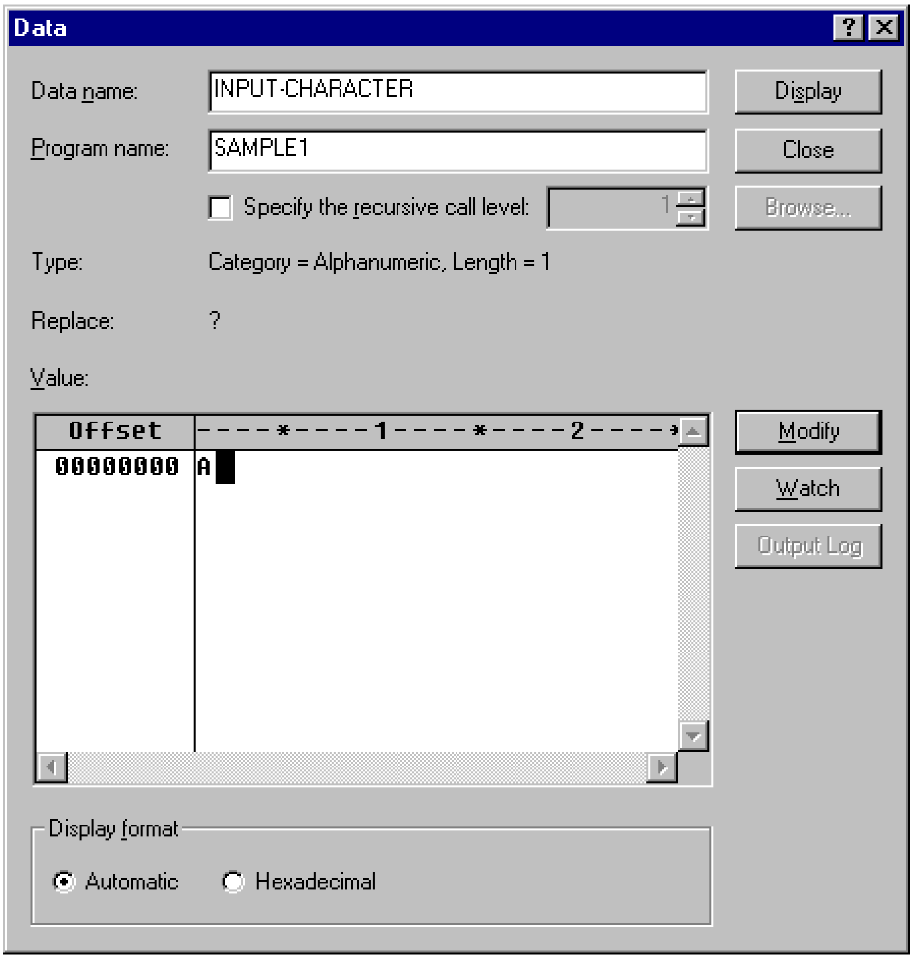
Type in any lowercase alphabetic character and then press the ENTER key. The debugger automatically returns to the next executable statement (Line 52).



```
ENTER ONE CHARACTER OF ALPHABETIC-LOWER.=>b
```

5. Change the data name (input-character) to “A”.

Move the cursor to Line 48, inside the data name INPUT-CHARACTER. Click on the magnifying glass in the toolbar or select Data from the Debug menu. Alternatively, you can click on the right mouse button to select Data in the popup menu. The Data dialog box is displayed.



Data name: INPUT-CHARACTER

Program name: SAMPLE1

□ Specify the recursive call level: 1

Type: Category = Alphanumeric, Length = 1

Replace: ?

Value:

```
Offset    ----*----1----*----2----
00000000  A
```

Display format

● Automatic ○ Hexadecimal

Change the value of the data in the data value area and click on the Modify button. Click on the Close button to exit the Data dialog box. (Answer “Yes” to save the changed data.)

6. Specify a breakpoint.

Move the cursor to Line 62, click on the right mouse button and select Set Breakpoint from the pop-up menu.

Note that the statement changes color when a breakpoint has been applied.

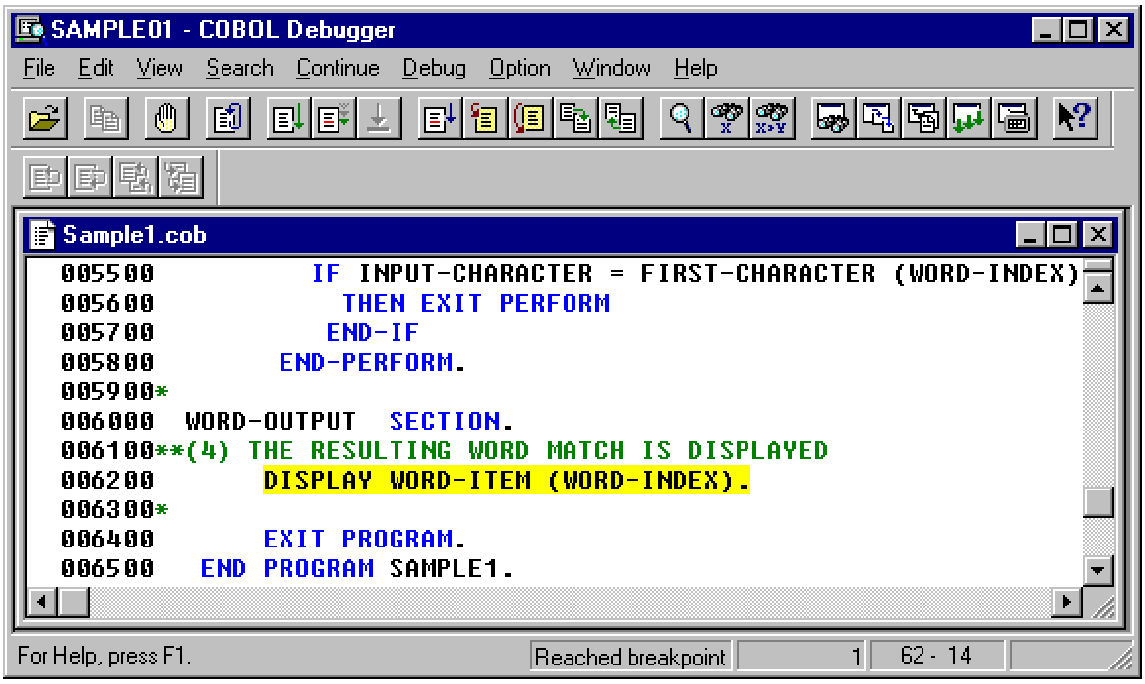
To delete a breakpoint, right click on the line containing the breakpoint and select Delete Breakpoint to Cursor from the pop-up menu. You can also use the Breakpoint dialog box that is displayed by selecting Breakpoint from the Debug menu. Highlight the breakpoint you want to delete, click on the Delete button, and then click on the Close button to exit the Breakpoint dialog box.

Note

You can delete more than one breakpoint at a time by highlighting multiple items in the Breakpoint dialog box and clicking on the Delete button.

1. Execute the program up to the breakpoint.

   Select Go in the Continue menu to execute the program to the breakpoint. Alternatively, you can press the F5 key. The program stops at Line 62.


SAMPLE01 - COBOL Debugger
File Edit View Search Continue Debug Option Window Help
Sample1.cob
For Help, press F1.
Reached breakpoint
1
62 - 14

```
005500              IF INPUT-CHARACTER = FIRST-CHARACTER (WORD-INDEX)
005600                THEN EXIT PERFORM
005700               END-IF
005800            END-PERFORM.
005900*
006000  WORD-OUTPUT  SECTION.
006100**(4) THE RESULTING WORD MATCH IS DISPLAYED
006200          DISPLAY WORD-ITEM (WORD-INDEX).
006300*
006400          EXIT PROGRAM.
006500      END PROGRAM SAMPLE1.
```

   Select Go or press the F5 key again to complete the debugging session.

2. To exit the Debugger, select Exit from the File menu.

A.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, 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.

Files Included in Sample 2

- SAMPLE2.COB (COBOL source program)
- SAMPLE02. PRJ (COBOL project file)
- SAMPLE02. CBI (COBOL compilation option file)
- COBOL85.CBR (COBOL runtime initialization file)
- DATAFILE(Data file)

COBOL Statements Used

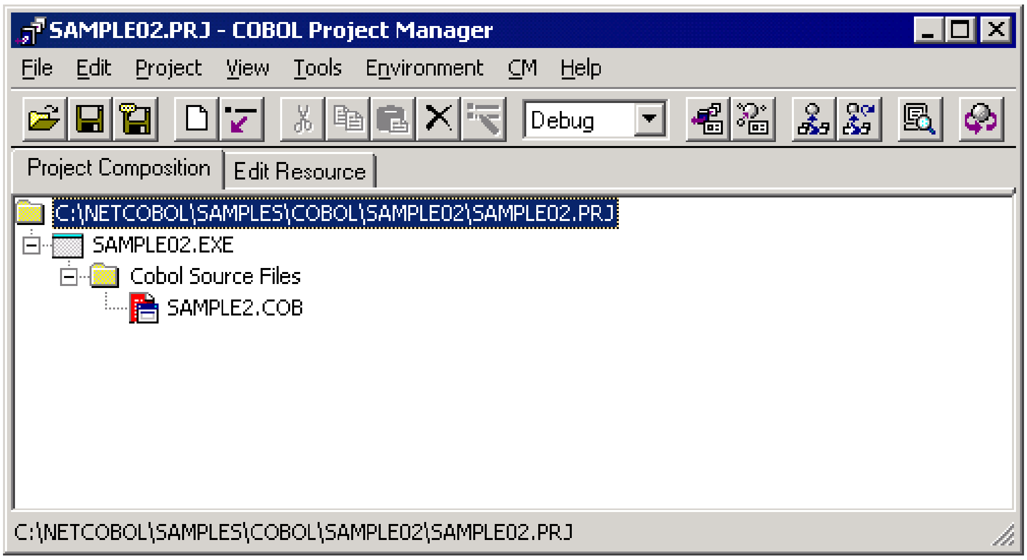
The CLOSE , EXIT , GO TO , MOVE , OPEN , READ , and WRITE statements are used.

Building/Rebuilding the Program

Project manager's build function is used to create the executable program.

In the following screen snapshots, the sample program was installed to C:\NetCOBOL. Your installation folder may be different.

1. The project manager is started, and project file "SAMPLE02.PRJ" is opened.


SAMPLE02.PRJ - COBOL Project Manager
File Edit Project View Tools Environment CM Help
Debug
Project Composition
Edit Resource
C:\NETCOBOL\SAMPLES\COBOL\SAMPLE02\SAMPLE02.PRJ
SAMPLE02.EXE
Cobol Source Files
SAMPLE2.COB
C:\NETCOBOL\SAMPLES\COBOL\SAMPLE02\SAMPLE02.PRJ

2. Select "Build" from Project Manager's "Project" menu or click on the Build button in the Project Manager toolbar.

   After the build terminates, check that SAMPLE02.EXE is created.

Setting Runtime Environment Information

1. Select "Runtime Environment Setup Tool" from the "Tools" menu of the Project Manager.

   The runtime environment setup tool is displayed.

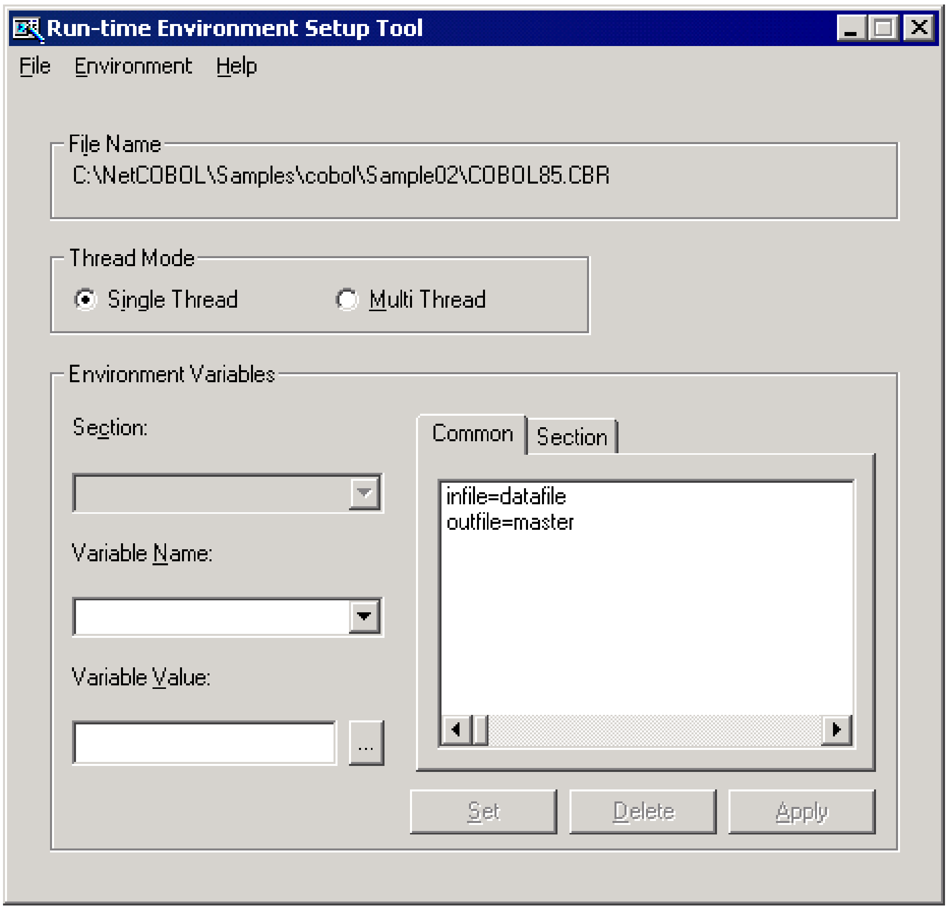
2. Select "Open" on the "File" menu and select COBOL85.CBR in the folder that contains the executable program (SAMPLE02.EXE).

3. Select the Common tab and enter data as shown below:

   - For the file-identifier INFILE, specify DATAFILE that is the file name of data file (line sequential file).

   - For the file-identifier OUTFILE, specify MASTER that is the file name of master file (indexed file).

4. If MASTER is specified for OUTFILE, input data as follow and push the "Set" button.



File Name: C:\NetCOBOL\Samples\cobol\Sample02\COBOL85.CBR

Thread Mode: ● Single Thread ○ Multi Thread

```
infile=datafile
outfile=master
```

5. Click the Apply button.

   The data is saved in the object initialization file.

6. Select "Exit" on the "File" menu to terminate the runtime environment setup tool.

Debugging the Program

To run this program under the control of the Debugger, select Debug from the Project menu. The Debugger starts and displays the “Start Debugging” dialog box.

Press the ENTER key here and you will be taken into the Debugger.

Executing the Program

Once the project is built, you can run Sample 2 by double-clicking the mouse on SAMPLE2.EXE in the COBOL Project Manager window.

Execution Result

No termination message is displayed.

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

Use the COBOL File Utility to confirm that 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 Chapter 7 of the NetCOBOL User’s Guide for details.

A.4 Sample 3: Screen Input-Output Using the Presentation File Function

Sample 3 is not available in the English version of NetCOBOL.

A.5 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)
- SAMPLE04. PRJ (COBOL project file)
- SAMPLE04. CBI (COBOL compilation option file)
- COBOL85.CBR (COBOL runtime initialization file)
- SAMPLE4.KDB(key definition file)

COBOL Statements Used

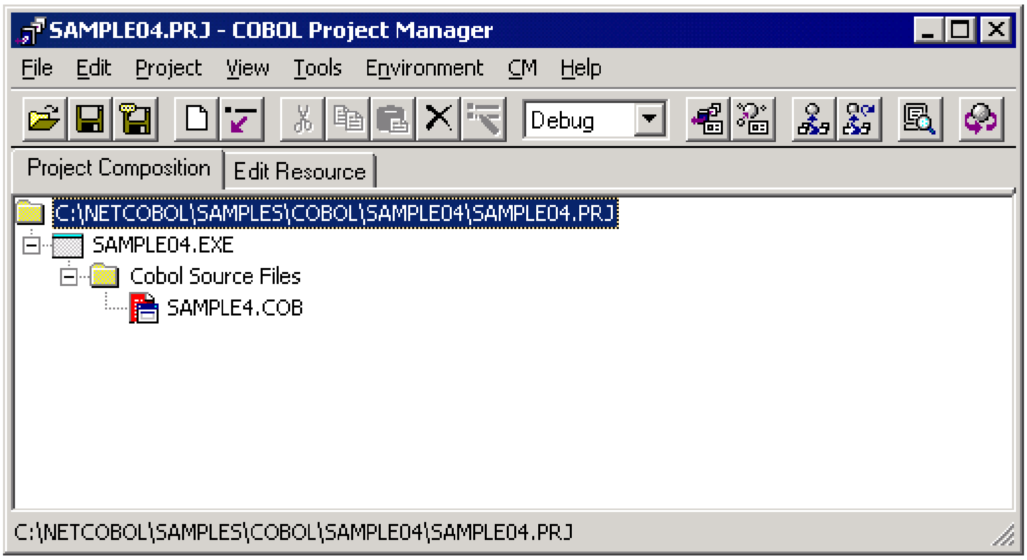
The ACCEPT, CLOSE, DISPLAY, EXIT, GO TO, IF, MOVE, OPEN, and WRITE statements are used.

Building/Rebuilding the Program

Project manager's build function is used to create the executable program.

In the following screen snapshots, the sample program was installed to C:\NetCOBOL. Your installation folder may be different.

1. The project manager is started, and project file "SAMPLE04.PRJ" is opened.



SAMPLE04.PRJ - COBOL Project Manager

C:\NETCOBOL\SAMPLES\COBOL\SAMPLE04\SAMPLE04.PRJ
SAMPLE04.EXE
Cobol Source Files
SAMPLE4.COB

C:\NETCOBOL\SAMPLES\COBOL\SAMPLE04\SAMPLE04.PRJ

2. Select "Build" from Project Manager's "Project" menu or click on the Build button in the Project Manager toolbar.

   After the build terminates, check that SAMPLE04.EXE is created.

Setting Runtime Environment Information

1. Select "Runtime Environment Setup Tool" from the "Tools" menu of the Project Manager.

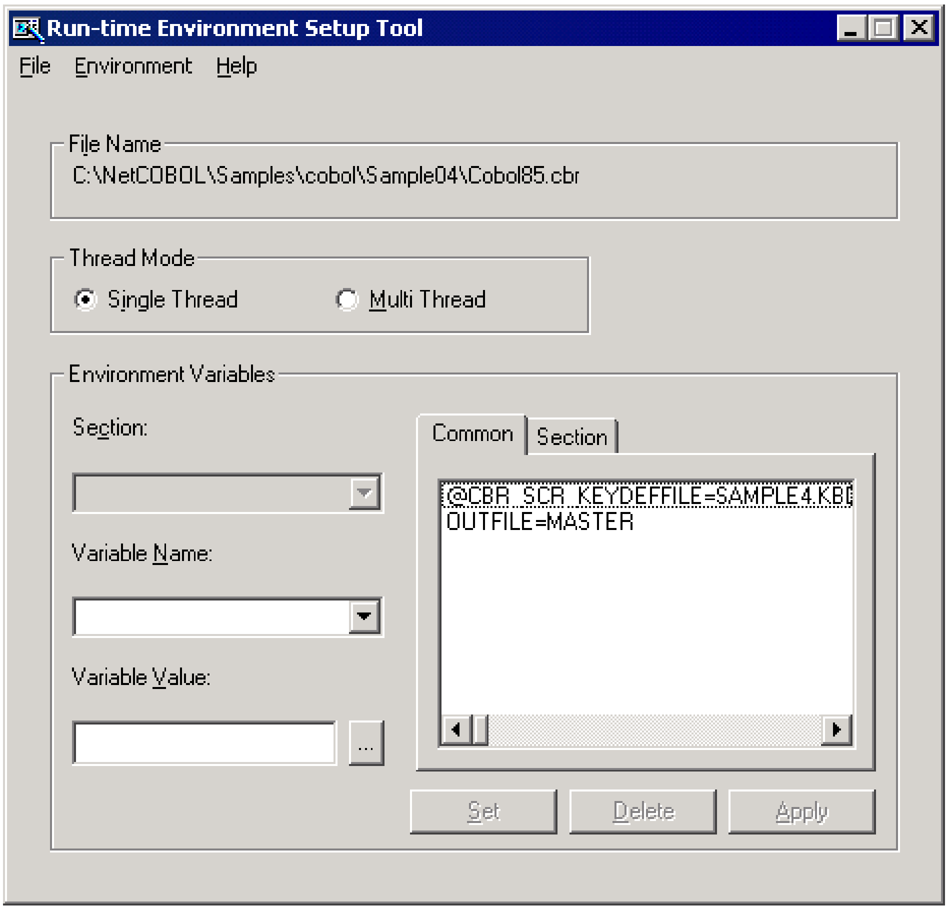
   The runtime environment setup tool is displayed.

2. Select "Open" on the "File" menu and select COBOL85.CBR in the folder that contains the executable program (SAMPLE04.EXE).

3. Select the Common tab and enter data as shown below:

   - For the file-identifier OUTFILE, specify MASTER that is the file name of master file (indexed file).

   - For the environment variable @CBR_SCR_KEYDEFFILE, specify SAMPLE4.KDB that is the file name of key definition file defined only for the input F2 key to become effective.



File Name: C:\NetCOBOL\Samples\cobol\Sample04\Cobol85.cbr

Thread Mode: ● Single Thread ○ Multi Thread

@CBR_SCR_KEYDEFFILE=SAMPLE4.KBD
OUTFILE=MASTER

4. Click the Apply button.

   The data is saved in the object initialization file.

5. Select "Exit" on the "File" menu to terminate the runtime environment setup tool.

Debugging the Program

To run this program under the control of the Debugger, select Debug from the Project menu. The Debugger starts and displays the “Start Debugging” dialog box.

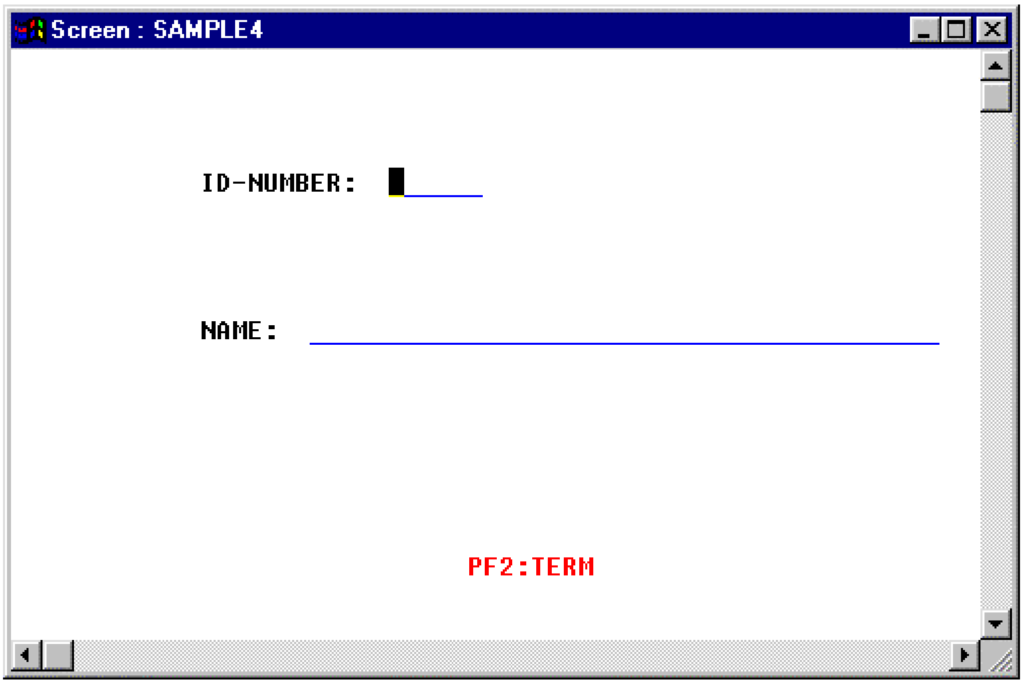
Press the ENTER key here and you will be taken into the Debugger.

Executing the Program

Execute Sample 4 by double-clicking the mouse on SAMPLE04.EXE in the COBOL

Project Manager window.

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


Screen : SAMPLE4

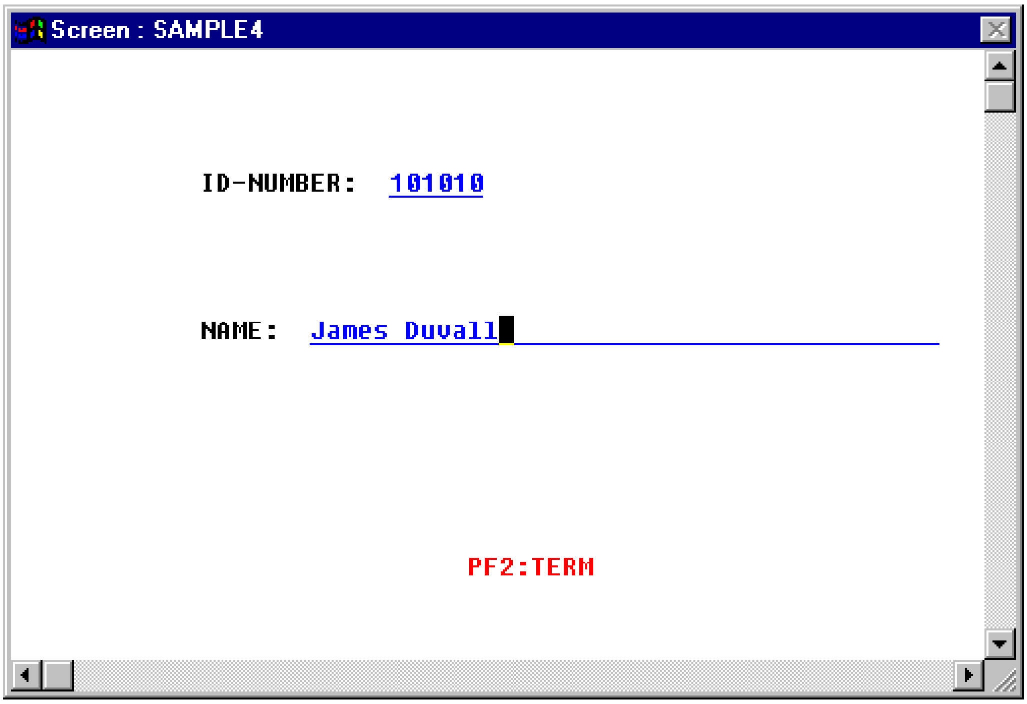
ID-NUMBER:

NAME:

PF2:TERM

Input the data:

Enter an employee's number (6 digit number) and name (up to 40 alphanumeric characters).


Screen : SAMPLE4

ID-NUMBER: 101010

NAME: James Duvall

PF2:TERM

To register data, press the ENTER key. The input data is written to the master file, and the screen is cleared for input of subsequent data.

To quit processing, press the F2 key.

Execution Result

Because the employee number becomes a main record key and the employee name becomes a record sub key, you should use Windows Explorer to confirm that the index file (MASTER) was created in the SAMPLE04 directory.

Note

You can edit the MASTER file using the Edit command in the COBOL File Utility (available from the Tools menu in the COBOL Project Manager window).

A.6 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.

It 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), then passes the work file to a subprogram which 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

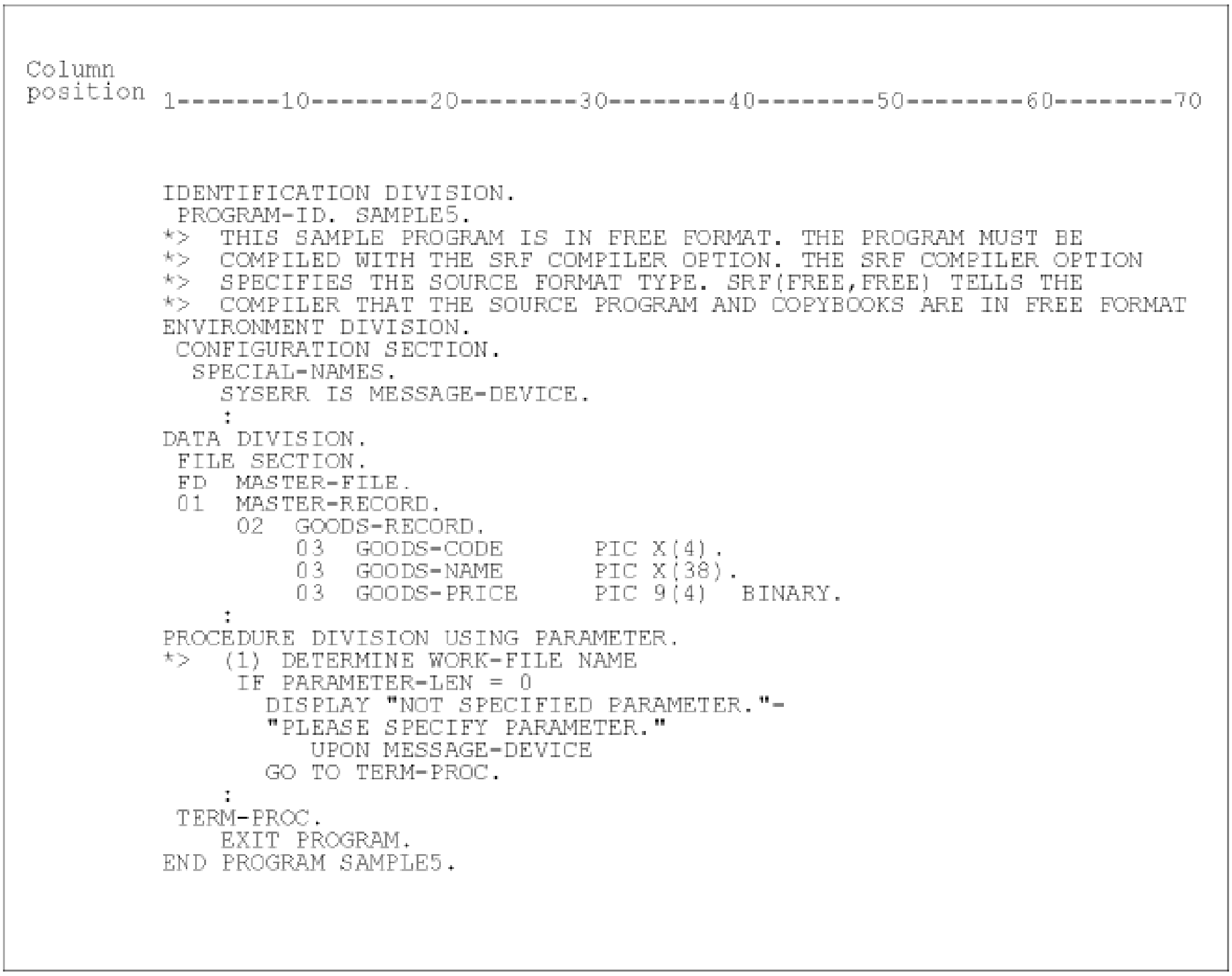
- SAMPLE5.COB (COBOL source program)
- PRINTPRC.COB (COBOL source program)
- S_REC.CBL (COBOL library file)
- SAMPLE05. PRJ (COBOL project file)
- SAMPLE04. CBI (COBOL compilation option file)
- COBOL85.CBR (COBOL runtime initialization file)

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

         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.
```

Note

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

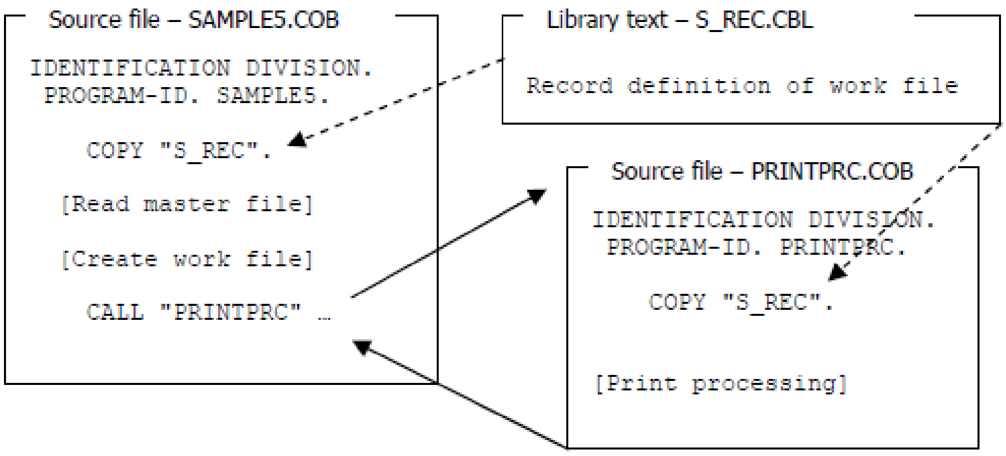
In free format, COBOL statements can be written in any character position on the line. Lines beginning with “*>“ are treated as comments.

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

Following figure shows the relationship between sources files used in Sample05.



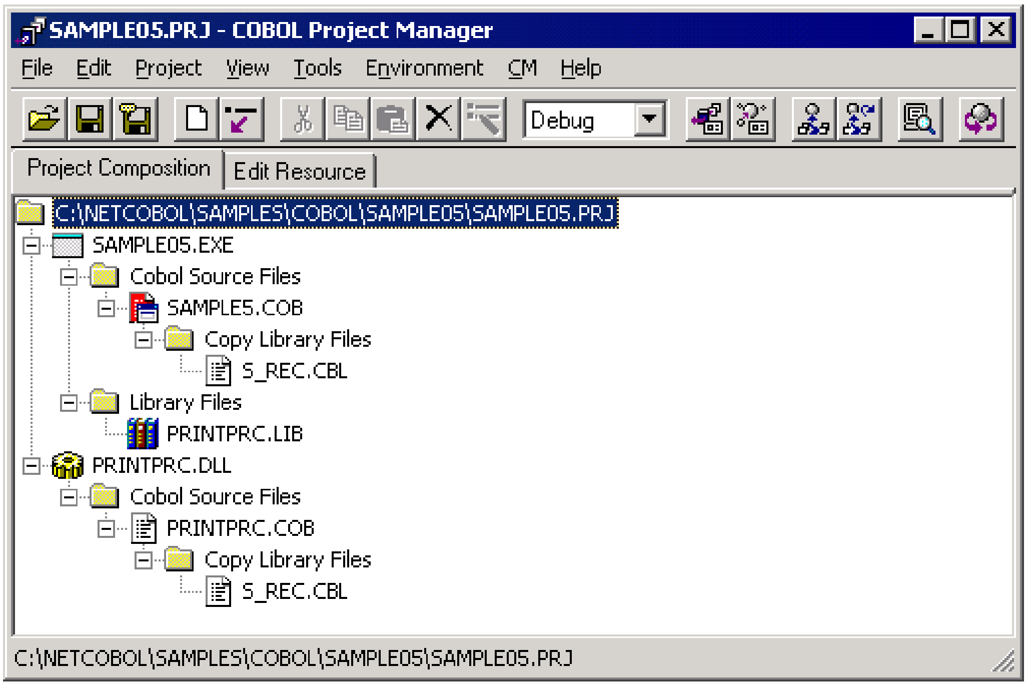
Prerequisite to Executing the Program

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

Building/Rebuilding the Program

Project manager's build function is used to create the executable program. In the following screen snapshots, the sample program was installed to C:\NetCOBOL. Your installation folder may be different.

1. The project manager is started, project file "SAMPLE05.PRJ" is opened.



C:\NETCOBOL\SAMPLES\COBOL\SAMPLE05\SAMPLE05.PRJ
- SAMPLE05.EXE
  - Cobol Source Files
    - SAMPLE5.COB
      - Copy Library Files
        - S_REC.CBL
  - Library Files
    - PRINTPRC.LIB
- PRINTPRC.DLL
  - Cobol Source Files
    - PRINTPRC.COB
      - Copy Library Files
        - S_REC.CBL

C:\NETCOBOL\SAMPLES\COBOL\SAMPLE05\SAMPLE05.PRJ

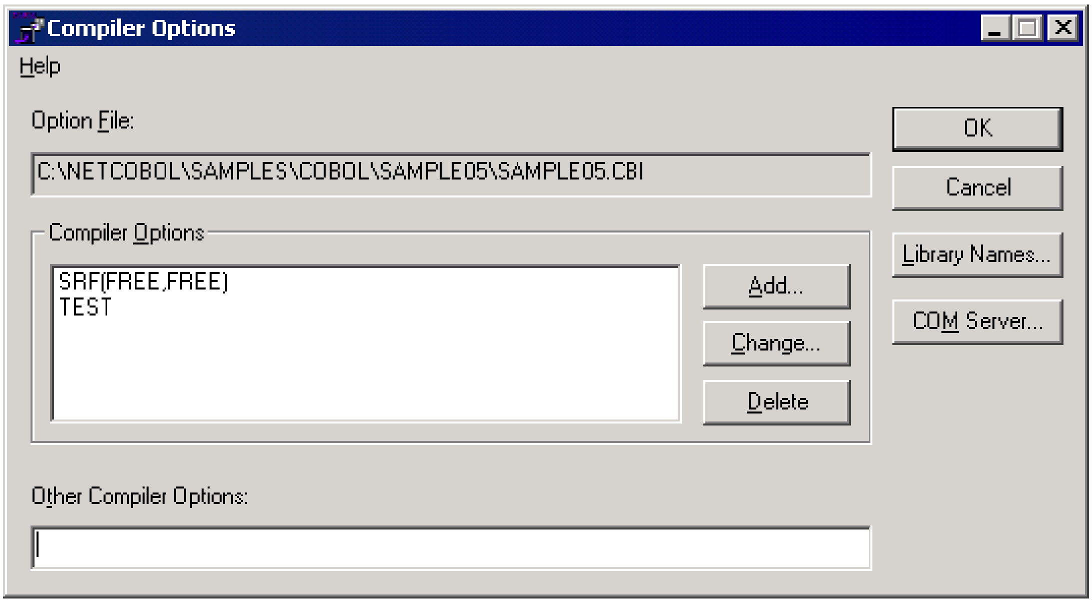
2. This project file contains the following two TARGET.

   - SAMPLE05.EXE (executable file name of main program)

   - PRINTPRC.DLL (DLL file name of subprogram)

3. The project file is selected, and "Compiler options" is selected from "Project"-"Options" menu.

   The "Compiler options" dialog is displayed.



Option File: C:\NETCOBOL\SAMPLES\COBOL\SAMPLE05\SAMPLE05.CBI

Compiler Options:
SRF(FREE,FREE)
TEST

Other Compiler Options:

4. Confirm compiler option SRF(FREE,FREE) is specified. After confirming the information, click the OK button.

   You are now returned to the Project Manager window.

5. Select "Build" from Project Manager's "Project" menu or click on the Build button in the Project Manager toolbar.

   After the build terminates, check that SAMPLE05.EXE and PRINTPRC.DLL are created.

Setting Runtime Environment Information

1. Select "Runtime Environment Setup Tool" from the "Tools" menu of the Project Manager.

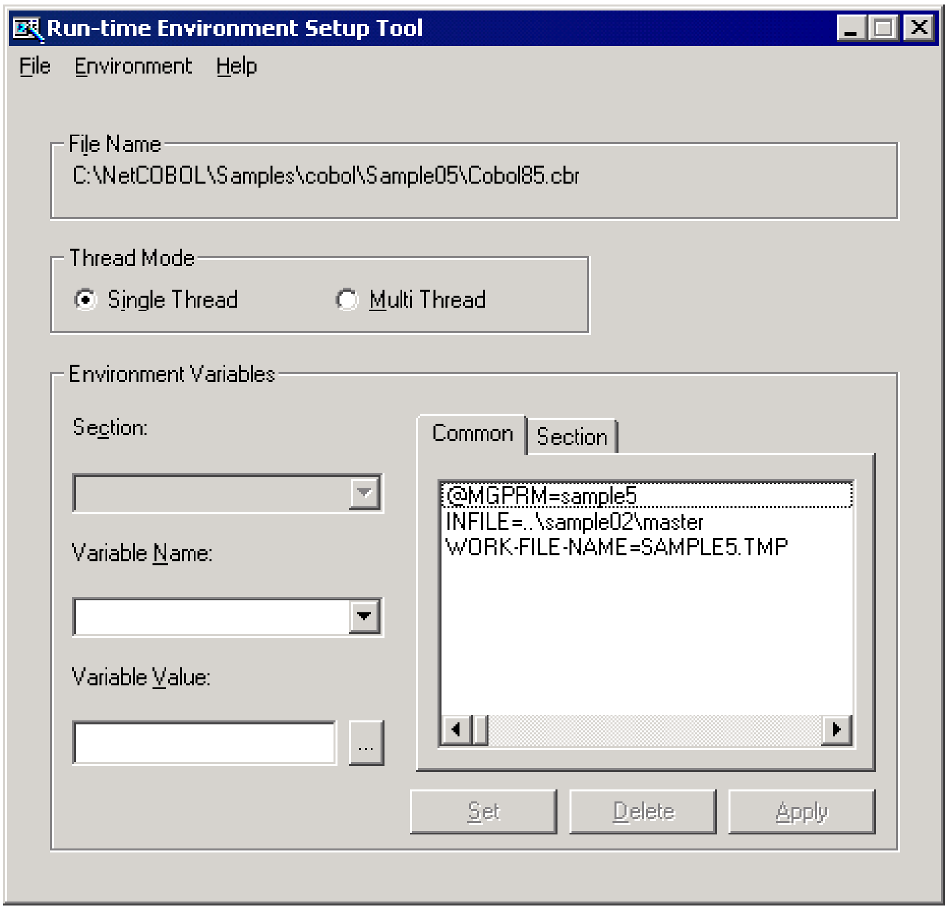
   The runtime environment setup tool is displayed.

2. Select "Open" on the "File" menu and select COBOL85.CBR in the folder that contains the executable program (SAMPLE05.EXE).

3. Select 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.

- 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.



File Name
C:\NetCOBOL\Samples\cobol\Sample05\Cobol85.cbr

Thread Mode
● Single Thread ○ Multi Thread

```
@MGPRM=sample5
INFILE=..\sample02\master
WORK-FILE-NAME=SAMPLE5.TMP
```

4. Click the Apply button.

   The data is saved in the object initialization file.

5. Select "Exit" on the "File" menu to terminate the runtime environment setup tool.

Debugging the Program

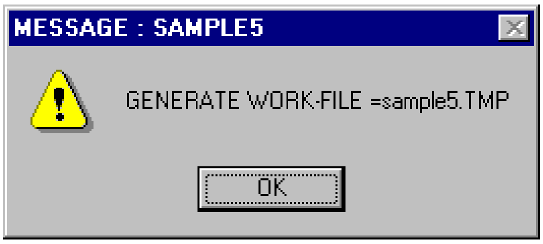
To run this program under the control of the Debugger, select Debug from the Project menu. The Debugger starts and displays the “Start Debugging” dialog box.

Press the ENTER key here and you will be taken into the Debugger.

Executing the Program

To execute the sample program, click on the Execute button in the Project Manager window.

The message “GENERATE WORK-FILE=sample5.TMP” is displayed.


MESSAGE : SAMPLE5

GENERATE WORK-FILE =sample5.TMP

Confirm the contents, then click on the OK button to close the message box.

Execution Result

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

A.7 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 the "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)
- SAMPLE06. PRJ (COBOL project file)
- SAMPLE06. CBI (COBOL compilation option file)

COBOL Statements Used

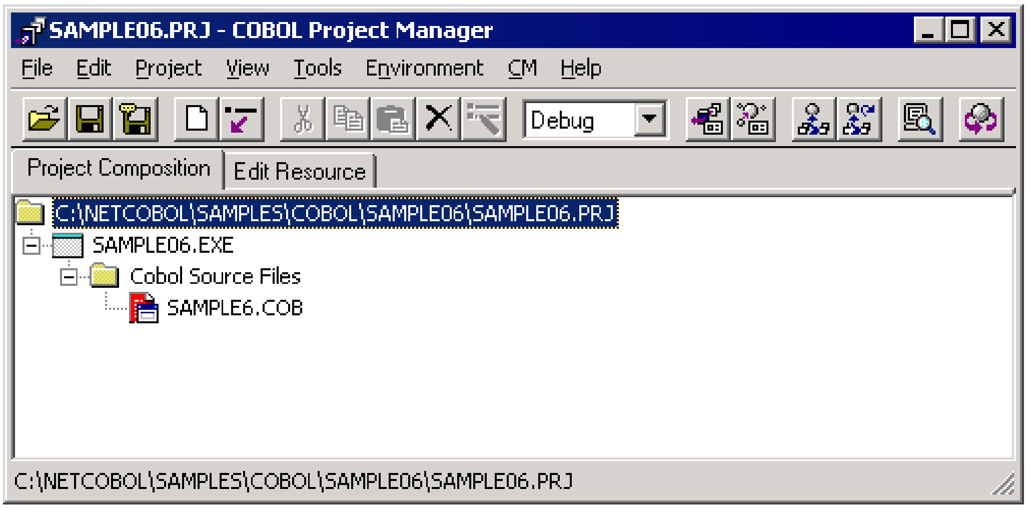
The ACCEPT, CALL, COMPUTE, COPY, DISPLAY, DIVIDE, EXIT, GO TO, IF, MOVE, and PERFORM statements are used.

Building/Rebuilding the Program

Project manager's build function is used to create the executable program.

In the following screen snapshots, the sample program was installed to C:\NetCOBOL. Your installation folder may be different.

1. The project manager is started, and project file "SAMPLE06.PRJ" is opened.



SAMPLE06.PRJ - COBOL Project Manager

C:\NETCOBOL\SAMPLES\COBOL\SAMPLE06\SAMPLE06.PRJ
SAMPLE06.EXE
Cobol Source Files
SAMPLE6.COB

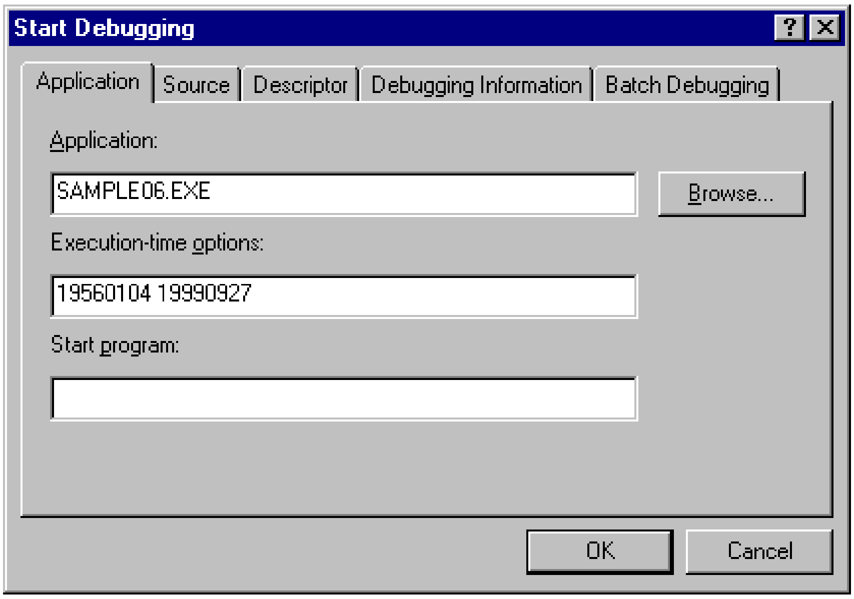
C:\NETCOBOL\SAMPLES\COBOL\SAMPLE06\SAMPLE06.PRJ

2. Select "Build" from Project Manager's "Project" menu or click on the Build button in the Project Manager toolbar.

   After the build terminates, check that SAMPLE06.EXE is created.

Debugging the Program

To run this program under the control of the Debugger, select Debug from the Project menu. The COBOL Debugger starts and displays the Start Debugging dialog box. The application name is already entered for you, you need to enter the execution-time options (command line arguments) the program expects - a start and end date. For example:

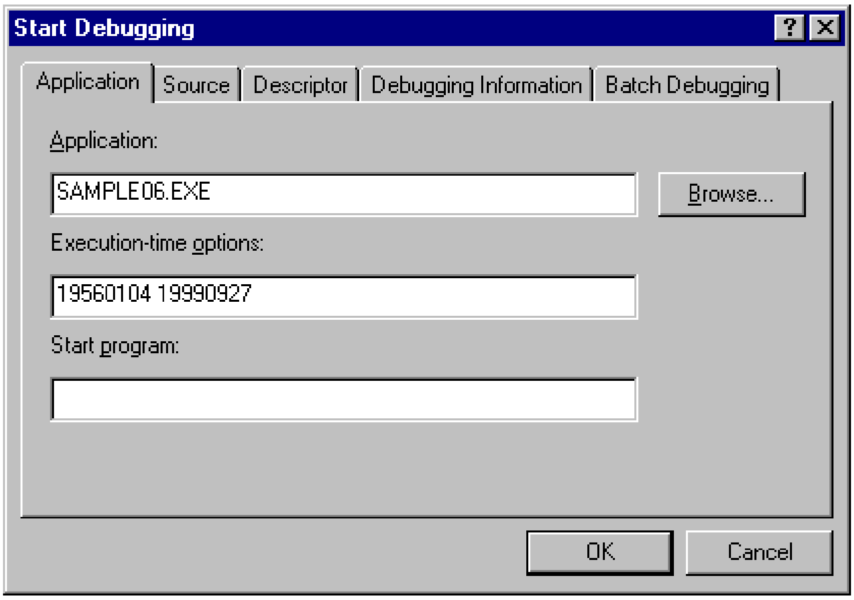


Application: SAMPLE06.EXE

Execution-time options: 19560104 19990927

Start program:

After you type in the command-line parameters for the program in the “Executiontime options” field, press the ENTER key. The Debugger starts as shown in the following figure.



Application: SAMPLE06.EXE

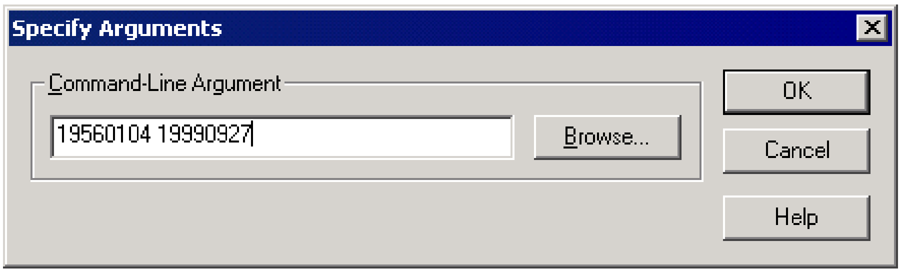
Execution-time options: 19560104 19990927

Start program:

Executing the Program

To execute the program from COBOL Project Manager, you need to select “Execute with Arguments” from the Project menu.

Enter the start date and end date arguments in the “Specify Arguments” dialog.

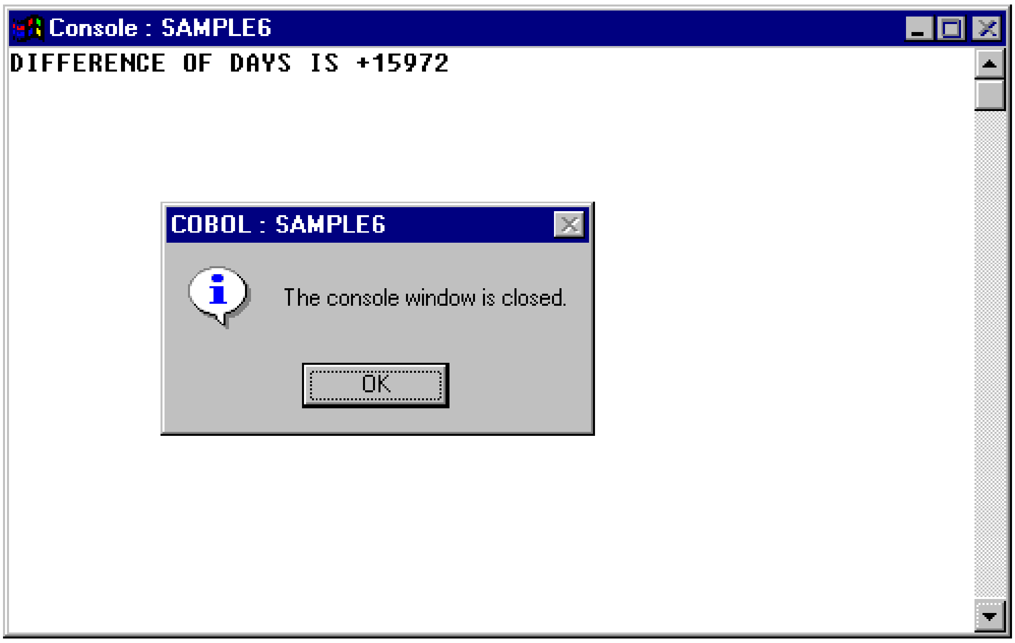


19560104 19990927

Refer to Chapter 1, “A Quick Tour” for details on using the COBOL Project Manager.

Execution Result

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



```
DIFFERENCE OF DAYS IS +15972
```

The console window is closed.

A.8 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 the “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), storing 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:

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)

- SAMPLE07. PRJ (COBOL project file)
- SAMPLE07. CBI (COBOL compilation option file)
- COBOL85.CBR (COBOL run-time initialization file)

COBOL Statements Used

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

Prerequisite to Executing the Program

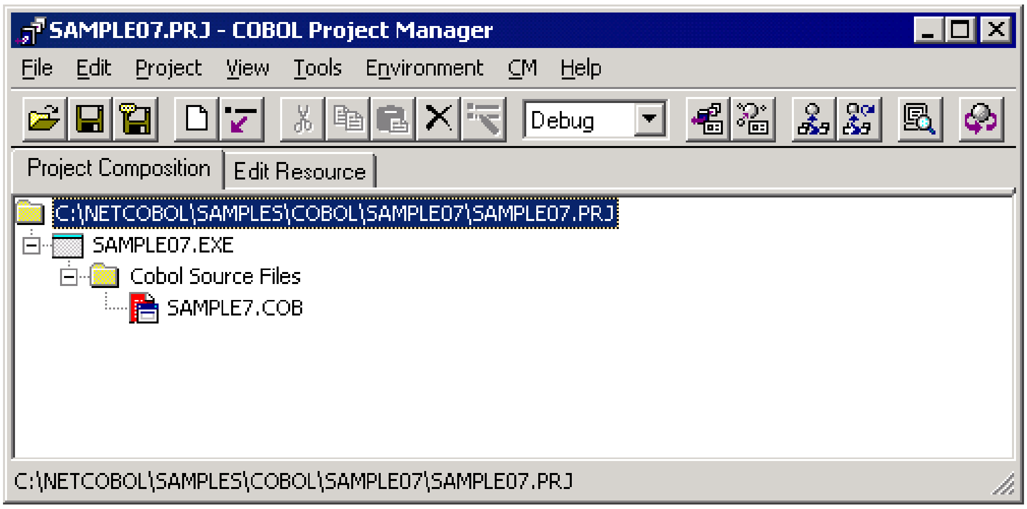
The master file created in Sample 2 is used. Therefore, execute the program in Sample 2 before executing Sample 7.

Building/Rebuilding the Program

Project manager's build function is used to create the executable program.

In the following screen snapshots, the sample program was installed to C:\NetCOBOL. Your installation folder may be different.

1. The project manager is started, and project file "SAMPLE07.PRJ" is opened.



SAMPLE07.PRJ - COBOL Project Manager

C:\NETCOBOL\SAMPLES\COBOL\SAMPLE07\SAMPLE07.PRJ
- SAMPLE07.EXE
  - Cobol Source Files
    - SAMPLE7.COB

C:\NETCOBOL\SAMPLES\COBOL\SAMPLE07\SAMPLE07.PRJ

2. Select "Build" from Project Manager's "Project" menu or click on the Build button in the Project Manager toolbar.

   After the build terminates, check that SAMPLE07.EXE is created.

Setting Runtime Environment Information

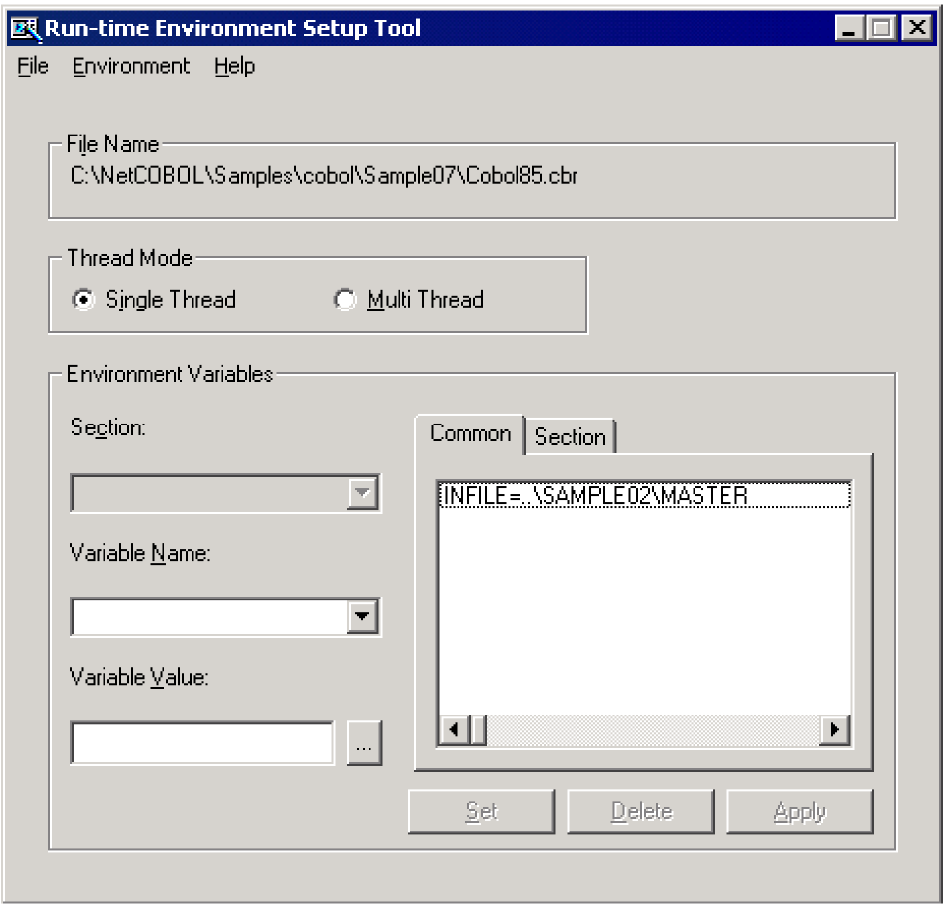
1. Select "Runtime Environment Setup Tool" from the "Tools" menu of the Project Manager.

   The run-time environment setup tool is displayed.

2. Select "Open" on the "File" menu and select COBOL85.CBR in the folder that contains the executable program (SAMPLE07.EXE).

3. Select 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.



File Name
C:\NetCOBOL\Samples\cobol\Sample07\Cobol85.cbr

Thread Mode
● Single Thread ○ Multi Thread

INFILE=..\SAMPLE02\MASTER

4. Click the Apply button.

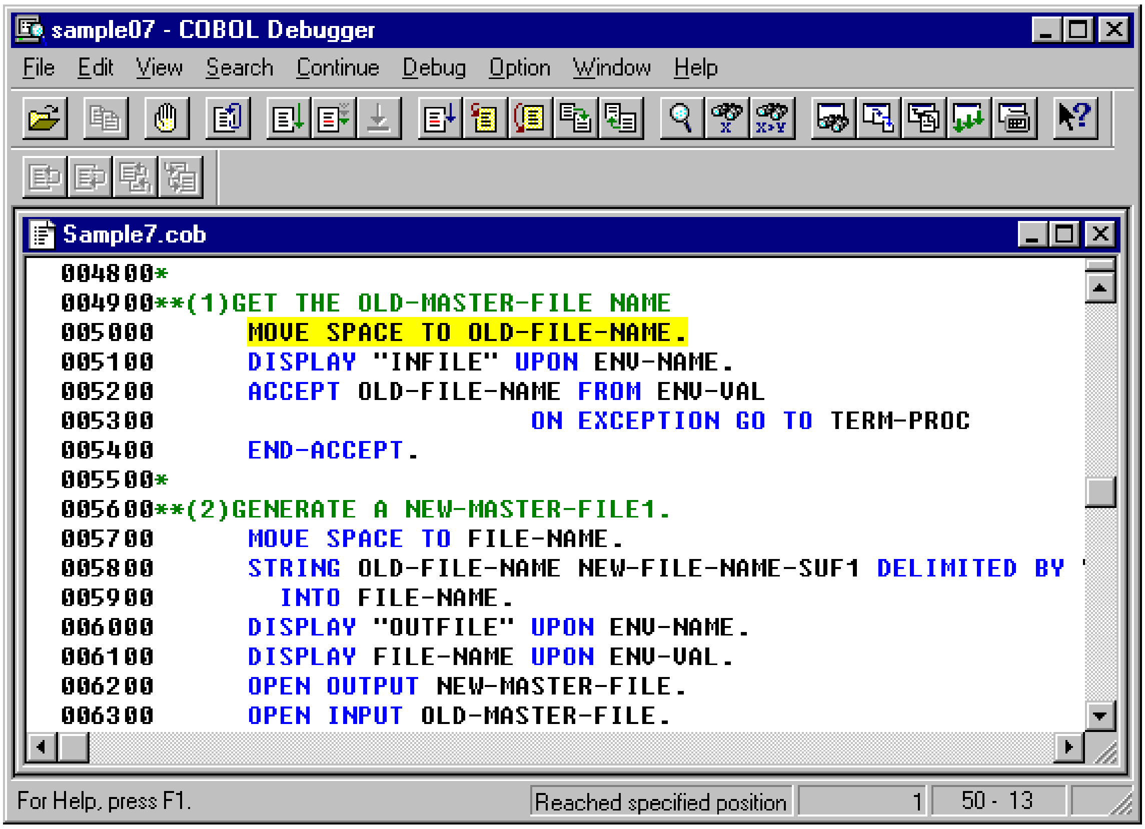
   The data is saved in the object initialization file.

5. Select "Exit" on the "File" menu to terminate the run-time environment setup tool.

Debugging the Program

To run this program under the control of the Debugger, select Debug from the Project menu. The Debugger starts and displays the Start Debugging dialog box.

Click the OK button or press the ENTER key. You will be taken into the Debugger as shown in the following figure.


sample07 - COBOL Debugger
File Edit View Search Continue Debug Option Window Help
Sample7.cob

```
004800*
004900**(1)GET THE OLD-MASTER-FILE NAME
005000      MOVE SPACE TO OLD-FILE-NAME.
005100      DISPLAY "INFILE" UPON ENV-NAME.
005200      ACCEPT OLD-FILE-NAME FROM ENV-VAL
005300                        ON EXCEPTION GO TO TERM-PROC
005400      END-ACCEPT.
005500*
005600**(2)GENERATE A NEW-MASTER-FILE1.
005700      MOVE SPACE TO FILE-NAME.
005800      STRING OLD-FILE-NAME NEW-FILE-NAME-SUF1 DELIMITED BY '
005900        INTO FILE-NAME.
006000      DISPLAY "OUTFILE" UPON ENV-NAME.
006100      DISPLAY FILE-NAME UPON ENV-VAL.
006200      OPEN OUTPUT NEW-MASTER-FILE.
006300      OPEN INPUT OLD-MASTER-FILE.
```

For Help, press F1. Reached specified position 1 50 - 13

Executing the Program

To execute the program from COBOL Project Manager, select Execute from the Project menu. There are no command line arguments.

Note

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.

A.9 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 the “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)

- SAMPLE08. PRJ (COBOL project file)
- SAMPLE08. CBI (COBOL compilation option file)

COBOL Statements Used

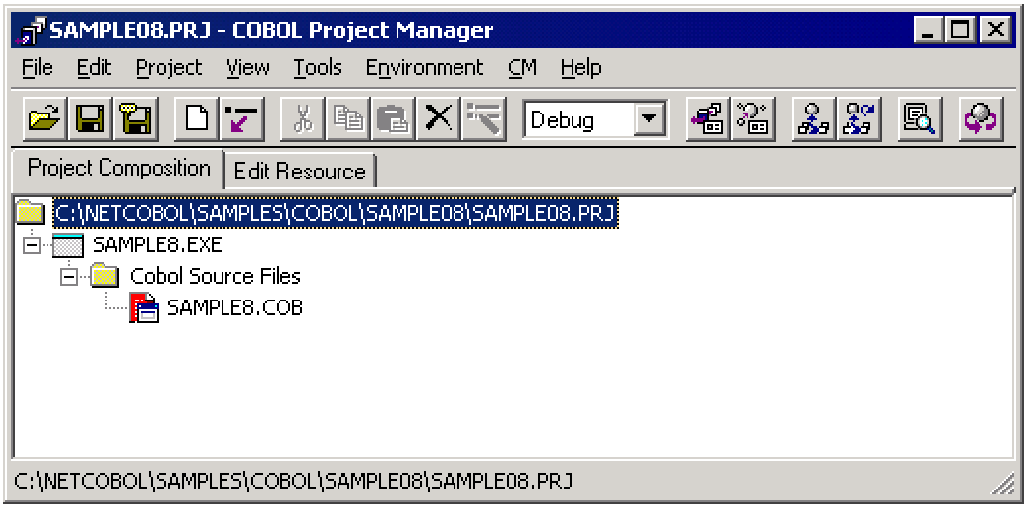
The ACCEPT, CLOSE, EXIT, GO TO, IF, OPEN, and WRITE statements are used.

Building/Rebuilding the Program

Project manager's build function is used to create the executable program.

In the following screen snapshots, the sample program was installed to C:\NetCOBOL. Your installation folder may be different.

1. The project manager is started, and project file "SAMPLE08.PRJ" is opened.



C:\NETCOBOL\SAMPLES\COBOL\SAMPLE08\SAMPLE08.PRJ
SAMPLE8.EXE
Cobol Source Files
SAMPLE8.COB

C:\NETCOBOL\SAMPLES\COBOL\SAMPLE08\SAMPLE08.PRJ

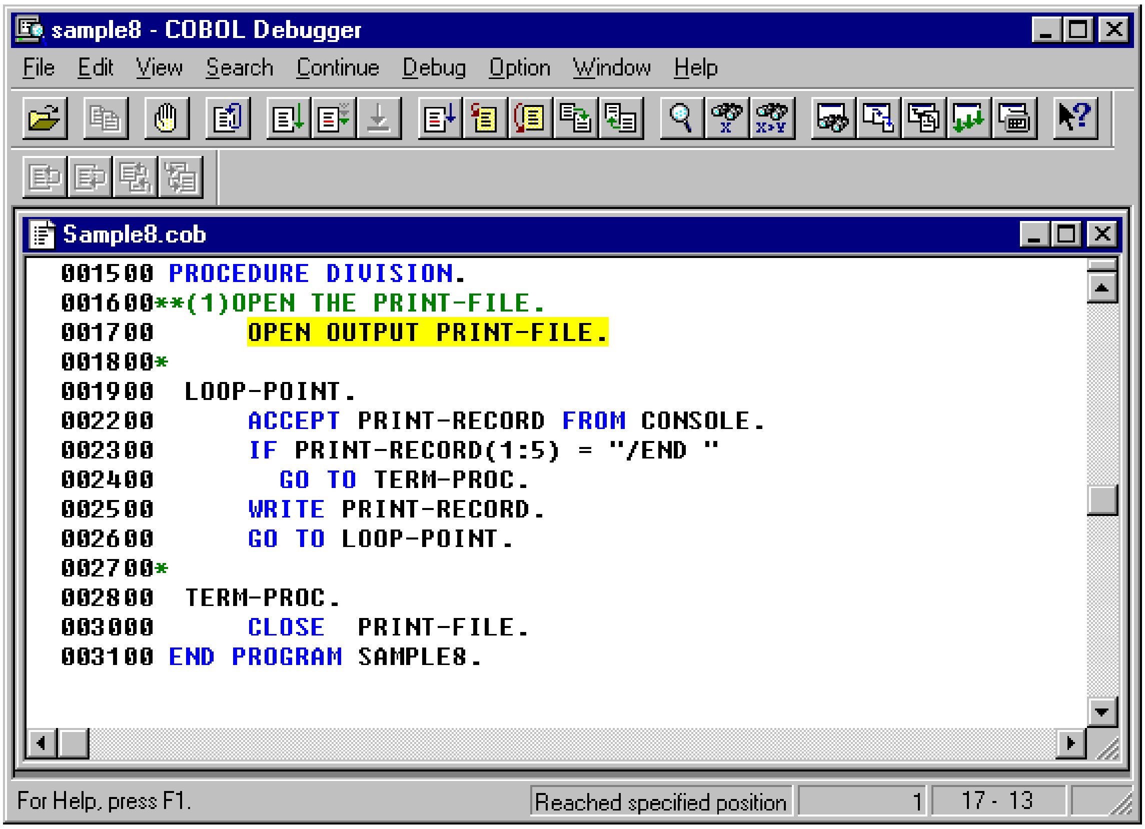
2. Select "Build" from Project Manager's "Project" menu or click on the Build button in the Project Manager toolbar.

   After the build terminates, check that SAMPLE08.EXE is created.

Debugging the Program

To run this program under the control of the Debugger, select Debug from the Project menu. The Debugger starts and displays the “Start Debugging” dialog box.

Press the ENTER key here and you will be taken into the Debugger, as shown in the following figure.



```
001500 PROCEDURE DIVISION.
001600**(1)OPEN THE PRINT-FILE.
001700     OPEN OUTPUT PRINT-FILE.
001800*
001900  LOOP-POINT.
002200     ACCEPT PRINT-RECORD FROM CONSOLE.
002300     IF PRINT-RECORD(1:5) = "/END "
002400       GO TO TERM-PROC.
002500     WRITE PRINT-RECORD.
002600     GO TO LOOP-POINT.
002700*
002800  TERM-PROC.
003000     CLOSE  PRINT-FILE.
003100 END PROGRAM SAMPLE8.
```

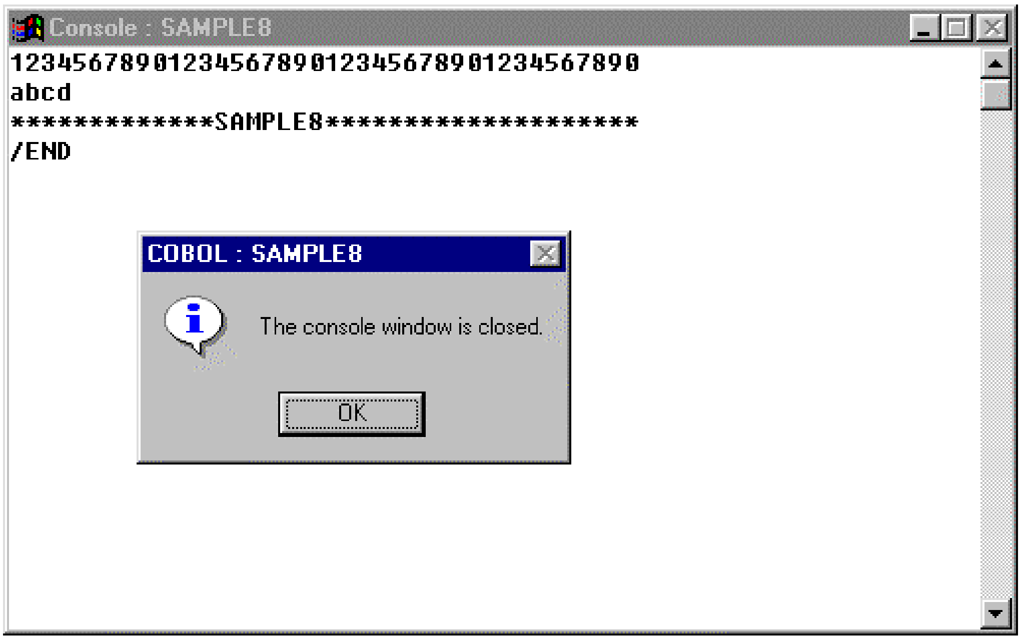
For Help, press F1. Reached specified position 1 17 - 13

Executing the Program

To execute in COBOL Project Manager, select Execute from the Project menu. There are no command line arguments.

A console window is then displayed.

In the console window, enter the data to be printed . Data of up to 40 characters can be entered at a time.



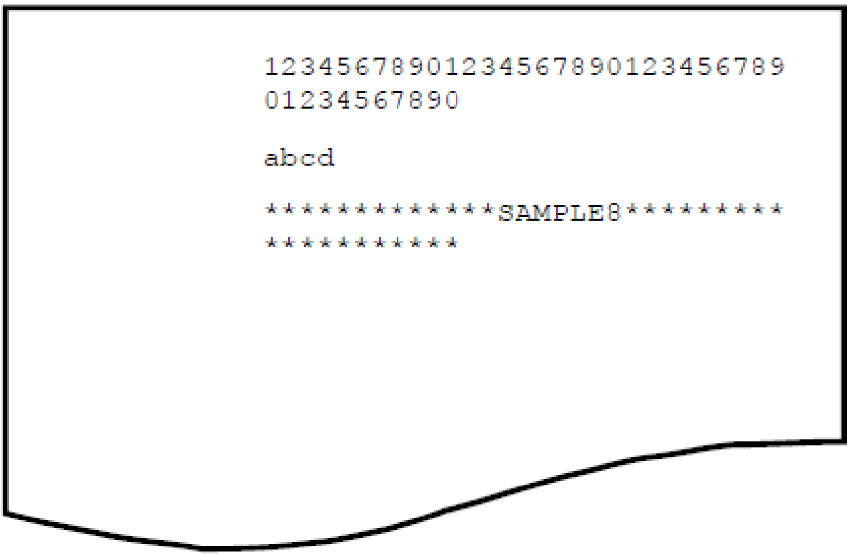
```
1234567890123456789012345678901234567890
abcd
*************SAMPLE8*********************
/END
```

The console window is closed.

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

Execution Result

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



```
12345678901234567890123456789
01234567890

abcd

*************SAMPLE8*********
***********
```

A.10 Sample 9: Using a Print File (Advanced usage)

Sample 9 demonstrates:

- 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; and
- Using the PRINTING POSITION clause to control the layout (line / column).

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

Overview

Following 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-cntl” 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 run-time 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.

Following table 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 the features being demonstrated.

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 ··· + CHARACTER TYPE x	
	Document name displayed by Windows	I-Control: DOCUMENT-NAME (=DOC1)	@CBR_DocumentName_DOC1=<document name string>

Task Description	Detailed features	Controlled by I-Control field / COBOL clause	Related Environment Variable(s)
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) COBOL run time system automatically calculates the best character pitch fitted to the character size	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 ⋯ + CHARACTER TYPE x	
	Legal size paper	I-Control: PAPER-SIZE (=”XXX”)	@PRN_FormName_XXX =Legal 8 1/2 x 14 in

Task Description	Detailed features	Controlled by I-Control field / COBOL clause	Related Environment Variable(s)
(character pitch specification is omitted).		FCB-NAME (=”LPI6”)	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. Here, the COBOL run time 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)
	Courier New font	PRINTING MODE x ⋯ WITH FONT {MINCHOU} {MINCHOU-HANKAKU} ⋯	@PrinterFontName= (Courier New, ⋯)

Task Description	Detailed features	Controlled by I-Control field / COBOL clause	Related Environment Variable(s)
		+ CHARACTER TYPE x	
	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 emsize, expanded ensize, 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 ···	
	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:	

Task Description	Detailed features	Controlled by I-Control field / COBOL clause	Related Environment Variable(s)
		··· 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)
- COBOL85.CBR (Environment variable initialization file)
- KOL5A4L6.OVD (Form overlay pattern)
- KOL5A4L8.OVD (Form overlay pattern)
- KOL5B4OV.OVD (Form overlay pattern)
- KOL5A4L6.PDM (Form descriptor)
- KOL5A4L8.PDM (Form descriptor)
- KOL5B4OV.PDM (Form descriptor)

COBOL Statements Used

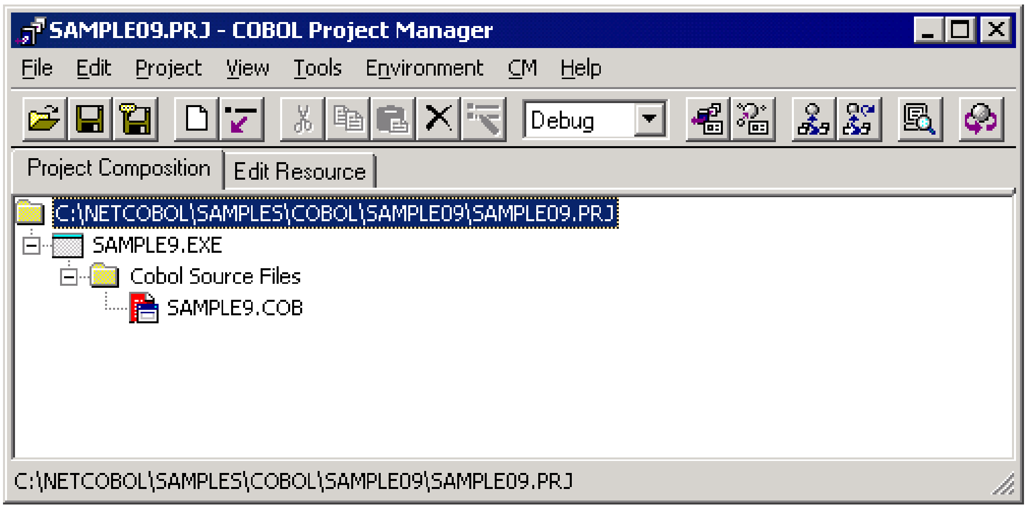
The ADD, CLOSE, DISPLAY, IF, MOVE, OPEN, PERFORM, STOP, and WRITE statements are used.

Building/Rebuilding the Program

Project manager's build function is used to create the executable program.

In the following screen snapshots, the sample program was installed to C:\NetCOBOL. Your installation folder may be different.

1. The project manager is started, and project file "SAMPLE09.PRJ" is opened.



C:\NETCOBOL\SAMPLES\COBOL\SAMPLE09\SAMPLE09.PRJ
SAMPLE9.EXE
Cobol Source Files
SAMPLE9.COB

C:\NETCOBOL\SAMPLES\COBOL\SAMPLE09\SAMPLE09.PRJ

2. Select "Build" from Project Manager's "Project" menu or click on the Build button in the Project Manager toolbar.

   After the build terminates, check that SAMPLE09.EXE is created.

Checking Execution Environment Information

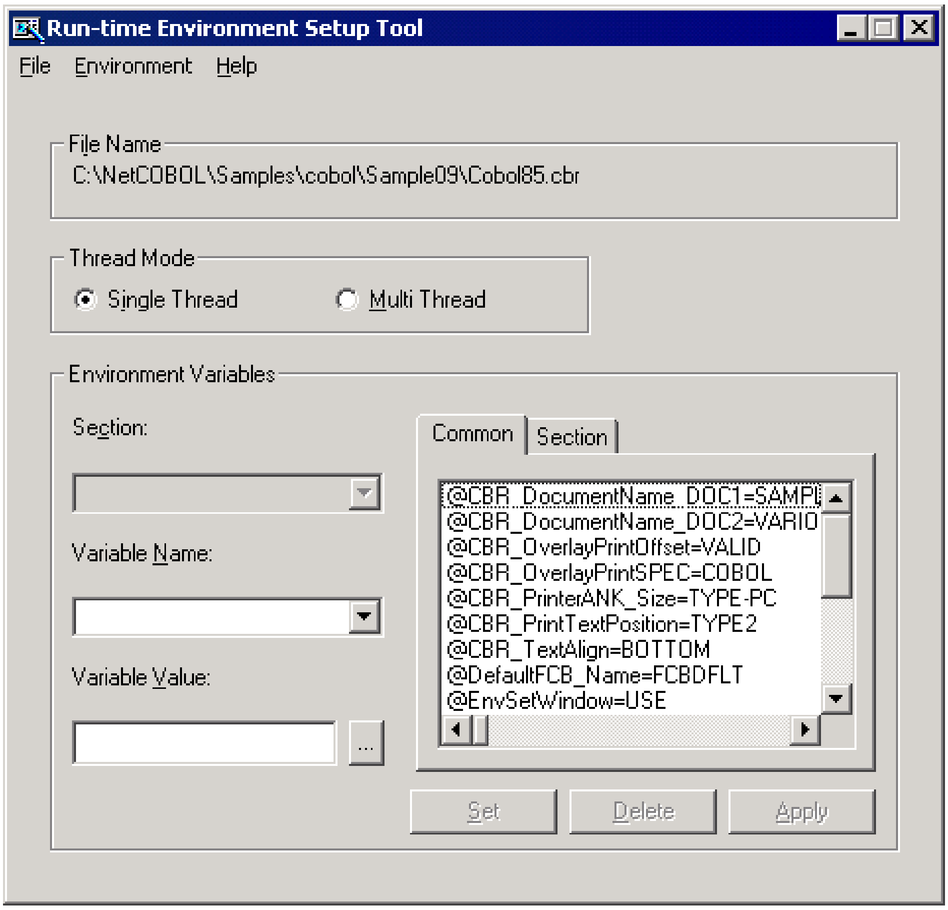
Sample 9 uses a number of run-time environment variables that play an important part in achieving the demonstrated features.

To check these variables:

1. From the COBOL Project Manager Tools menu, select “Run-time Environment Setup Tool”.

   The Run-time Environment Setup Tool Window opens.

2. From the File menu select Open, and select COBOL85.CBR in the sample09 folder. The window should look like below figure.



File Name
C:\NetCOBOL\Samples\cobol\Sample09\Cobol85.cbr

Thread Mode
● Single Thread ○ Multi Thread

Environment Variables

Section:

Variable Name:

Variable Value:

@CBR_DocumentName_DOC1=SAMPL
@CBR_DocumentName_DOC2=VARIO
@CBR_OverlayPrintOffset=VALID
@CBR_OverlayPrintSPEC=COBOL
@CBR_PrinterANK_Size=TYPE-PC
@CBR_PrintTextPosition=TYPE2
@CBR_TextAlign=BOTTOM
@DefaultFCB_Name=FCBDFLT
@EnvSetWindow=USE

3. 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:

   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.

4. When you have finished reviewing the environment variables, select Exit from the File menu.

Debugging the Program

To run this program under the control of the Debugger, select Debug from the Project menu. The Debugger starts and displays the “Start Debugging” dialog box.

Press the ENTER key here and you will be taken into the Debugger.

Executing the Program

To execute in COBOL Project Manager, select Execute from the Project menu. There are no command line arguments.

Execution Results

The sample pages described in table A.3 above are printed to the default printer.

A.11 Sample 10: Using a Print File with FORMAT clause

Sample 10 is not available in the English version of NetCOBOL.

A.12 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.

However, a database file is included with this sample to enable you to use a relational database off-line. This database file-called STOCK.MDB-- includes the Microsoft Access runtime support. It can therefore be used as long as Open Database Connectivity (ODBC) is installed and configured properly.

For more about using ODBC drivers, refer to the “Database (SQL)” chapter in the “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)
- SAMPLE11.PRJ (COBOL project file)
- SAMPLE11.CBI (COBOL compiler option file)
- COBOL85.CBR (COBOL run-time initialization file)
- SAMPLE11.INF (ODBC information file)
- STOCK.MDB (Sample database for Microsoft Access)

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

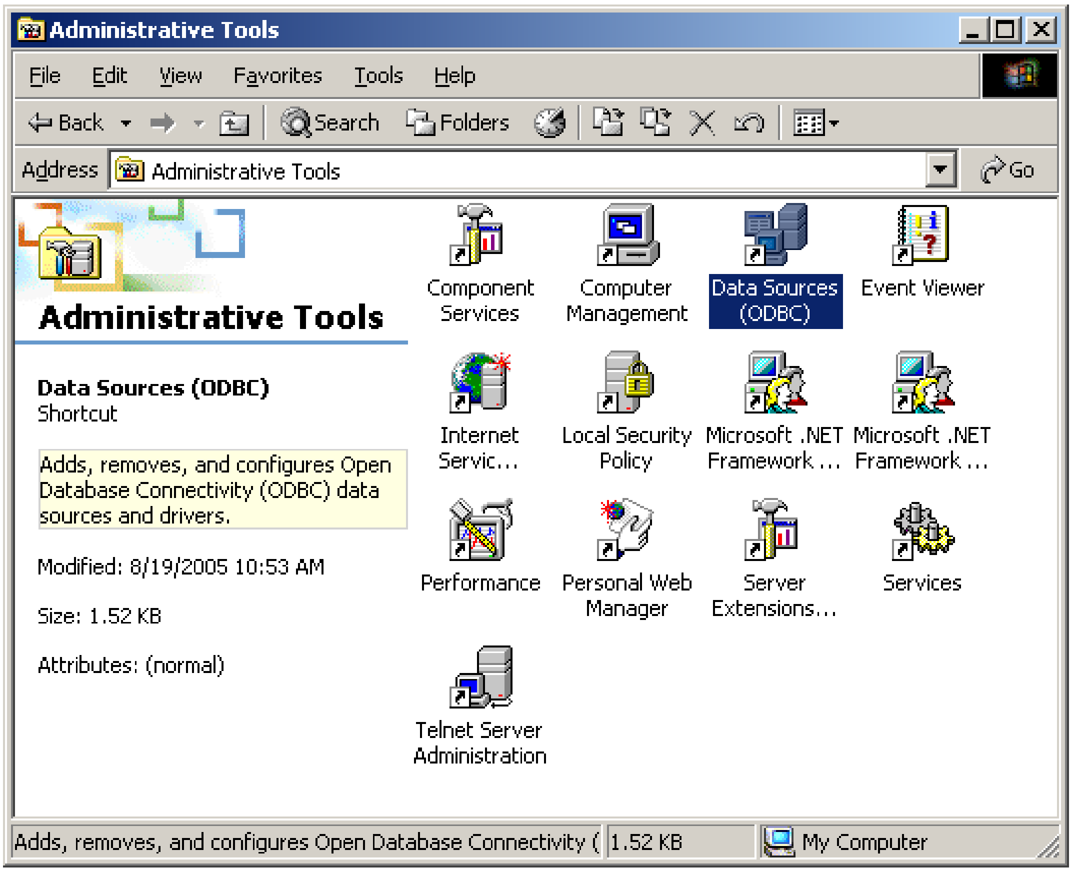
ODBC is a defined interface that attempts to provide a highly generic API into any database system that provides compliant drivers. Just about every database system available today provides ODBC drivers for a variety of platforms.

In order to execute this sample, you must have already installed 32-bit ODBC support on your Windows 2000, XP, Windows Server 2003, or Windows Server 2008, along with the Microsoft Access Driver.

Verifying Whether ODBC is installed

In order to verify whether your system has the proper support installed, look in your Administrative Tools for the icon of ODBC data source administrator.

An example is given below.



Data Sources (ODBC)
Shortcut

Adds, removes, and configures Open Database Connectivity (ODBC) data sources and drivers.

Modified: 8/19/2005 10:53 AM

Size: 1.52 KB

Attributes: (normal)

Adds, removes, and configures Open Database Connectivity (| 1.52 KB | My Computer

Double-click on this icon.

Select the Drivers (or ODBC Drivers) tab and verify that the Microsoft Access Driver (*.mdb) is installed. The driver is shown highlighted in the following figure.

Note

Depending on the version of Microsoft Access, the ODBC drivers may not be installed by default - you must manually select these options when you install Microsoft Access (either from the Microsoft Access installation CD or the Office Professional CD).

You will also require a sample Microsoft Access run-time enabled database called "STOCK.MDB". This file can be found in the SAMPLE11 folder.

Once you have verified installation of 32 Bit ODBC, Microsoft Access and the associated drivers, you are ready to proceed with executing the sample.

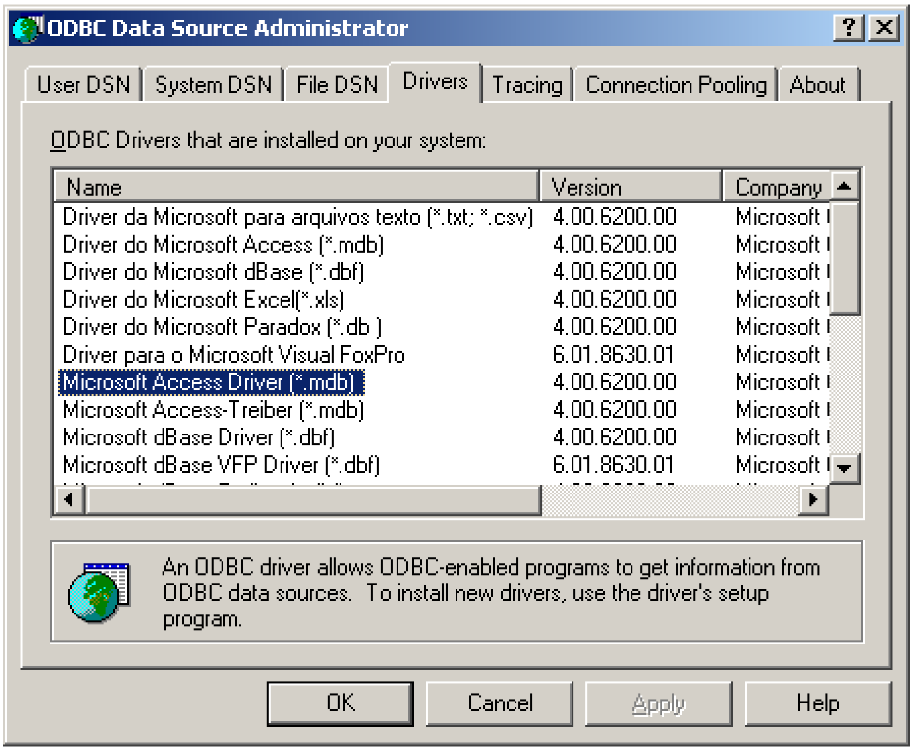
Defining an ODBC Source for Your Application

The first development task that needs to be accomplished is to define an ODBC Source for the application.

An ODBC Source can be thought of as simply specifying to the ODBC manager a symbolic name that you wish to use in your application to describe an ODBC connection to a specific ODBC database driver and database.

The following figures may or may not match what you see on your particular operating system, but you should be able to perform the following tasks by using the instructions provided.

Bring up the Administrative Tools and double click on the Data Sources (ODBC) icon. The main ODBC Data Source Administrator window is displayed.



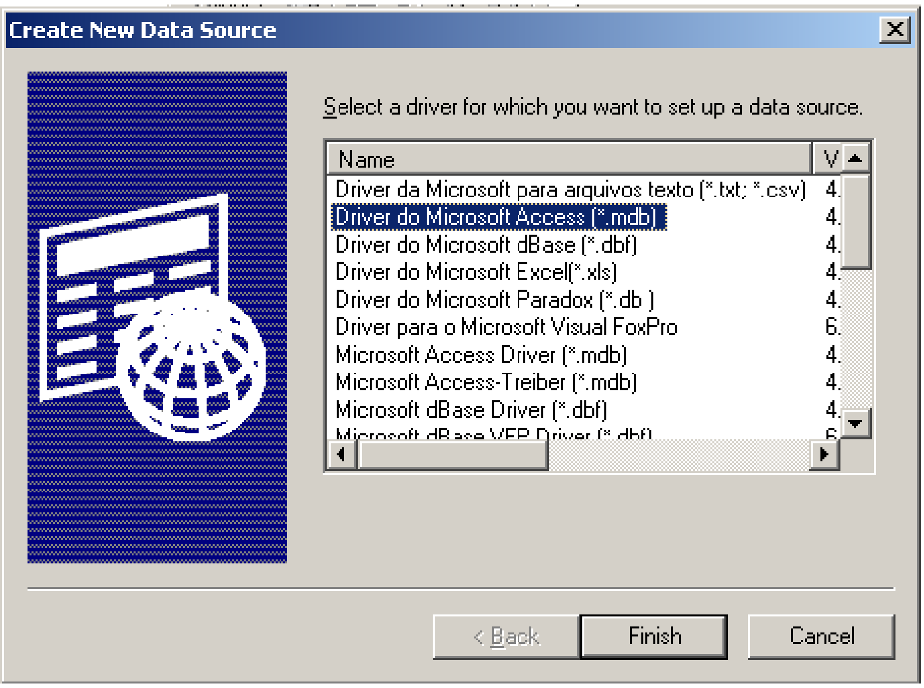
ODBC Drivers that are installed on your system:

Name	Version	Company
Driver da Microsoft para arquivos texto (*.txt; *.csv)	4.00.6200.00	Microsoft I
Driver do Microsoft Access (*.mdb)	4.00.6200.00	Microsoft I
Driver do Microsoft dBase (*.dbf)	4.00.6200.00	Microsoft I
Driver do Microsoft Excel(*.xls)	4.00.6200.00	Microsoft I
Driver do Microsoft Paradox (*.db)	4.00.6200.00	Microsoft I
Driver para o Microsoft Visual FoxPro	6.01.8630.01	Microsoft I
Microsoft Access Driver (*.mdb)	4.00.6200.00	Microsoft I
Microsoft Access-Treiber (*.mdb)	4.00.6200.00	Microsoft I
Microsoft dBase Driver (*.dbf)	4.00.6200.00	Microsoft I
Microsoft dBase VFP Driver (*.dbf)	6.01.8630.01	Microsoft I

An ODBC driver allows ODBC-enabled programs to get information from ODBC data sources. To install new drivers, use the driver's setup program.

You will add a new User DSN. Select the User DSN tab, and click on the Add button.

ODBC Data Source Administrator displays the “Create New Data Source” dialog box.

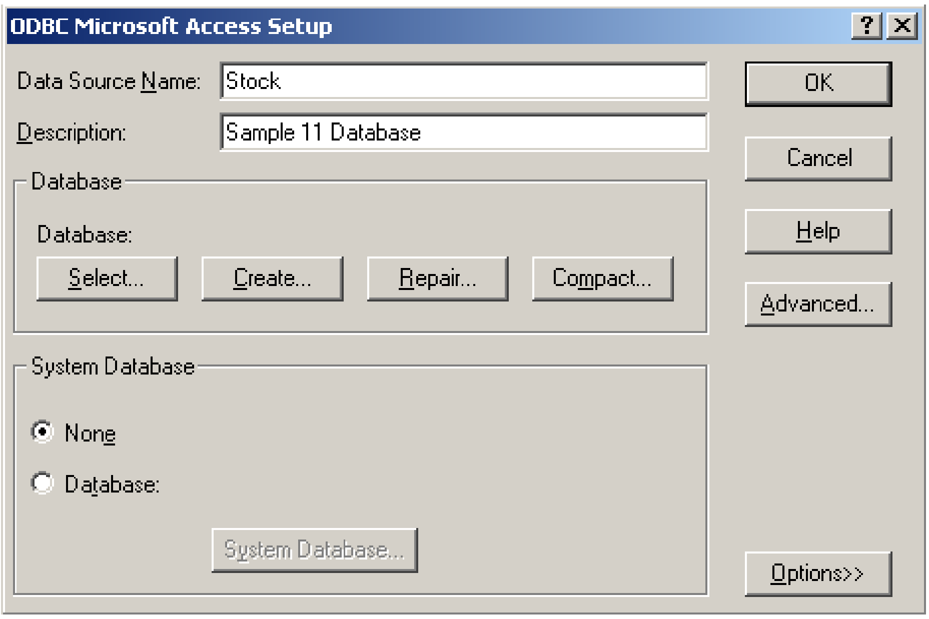


Select a driver for which you want to set up a data source.

Name	V
Driver da Microsoft para arquivos texto (*.txt; *.csv)	4.
Driver do Microsoft Access (*.mdb)	4.
Driver do Microsoft dBase (*.dbf)	4.
Driver do Microsoft Excel(*.xls)	4.
Driver do Microsoft Paradox (*.db)	4.
Driver para o Microsoft Visual FoxPro	6.
Microsoft Access Driver (*.mdb)	4.
Microsoft Access-Treiber (*.mdb)	4.
Microsoft dBase Driver (*.dbf)	4.
Microsoft dBase VFP Driver (*.dbf)	6.

Select the Microsoft Access Driver (*.mdb) and click on the Finish button.

The ODBC Microsoft Access Setup dialog box is displayed.



ODBC Microsoft Access Setup

Data Source Name: Stock

Description: Sample 11 Database

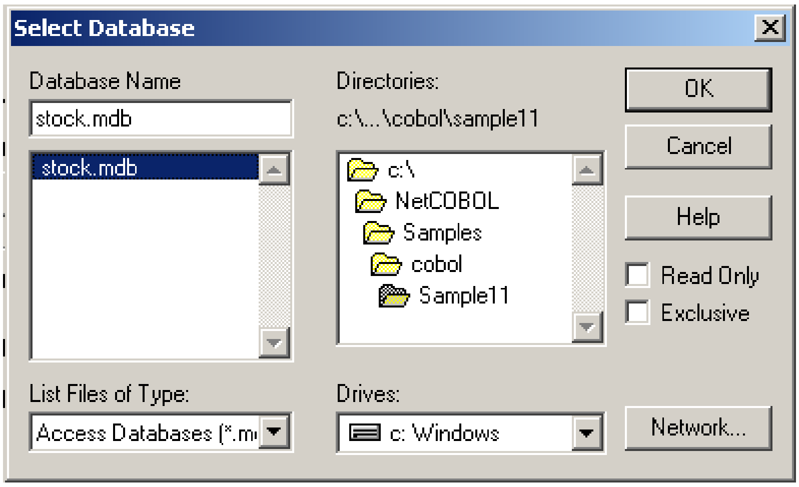
System Database

● None

○ Database:

Enter Stock in the Data Source Name field and Sample 11 Database in the Description field as shown above.

You also need to select a file (database) to bind this driver to. Click on the Select button in the ODBC Microsoft Access Setup dialog box. The Select Database dialog box is displayed. Navigate through the directories and find the "Stock.mdb" database file:



Select Database

Database Name: stock.mdb

Directories: c:\...\cobol\sample11

List Files of Type: Access Databases (*.m

Drives: c: Windows

□ Read Only

□ Exclusive

Click on the OK button. This will take you back to the ODBC Microsoft Access Setup dialog box.

Click on the OK button. This will take you back to the main ODBC Source Administrator window.

Note that "Stock" has been added to the list in User DSN. Although you should not set any additional options for this exercise, you may wish to highlight “Stock” and click on the Configure button.

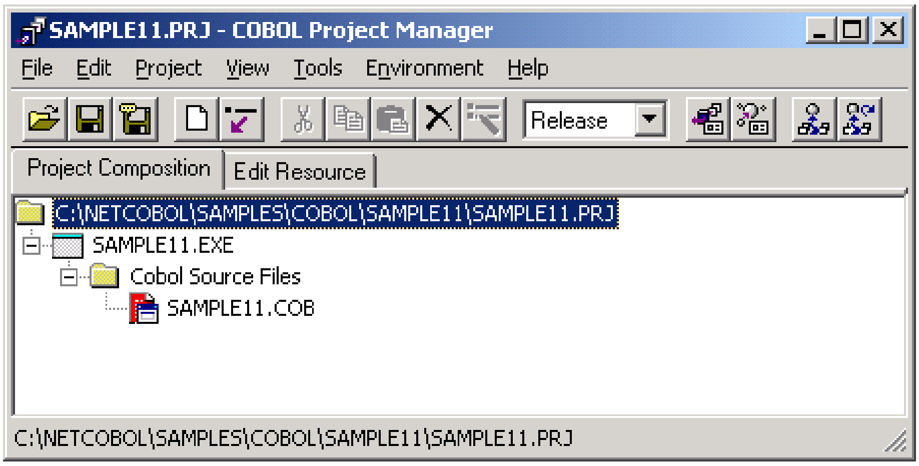
This will give you an idea of some of the additional options that are available when developing these types of applications. Once you are back to the main ODBC Source Administrator window, click on the OK button. This will exit the ODBC configuration facility. You have now defined the new ODBC Source (Stock) you are going to use with COBOL.

Building/Rebuilding the Program

Project manager's build function is used to create the executable program.

In the following screen snapshots, the sample program was installed to C:\NetCOBOL. Your installation folder may be different.

1. The project manager is started, and project file "SAMPLE11.PRJ" is opened.


SAMPLE11.PRJ - COBOL Project Manager
File Edit Project View Tools Environment Help
Release
Project Composition
Edit Resource

C:\NETCOBOL\SAMPLES\COBOL\SAMPLE11\SAMPLE11.PRJ
SAMPLE11.EXE
Cobol Source Files
SAMPLE11.COB

C:\NETCOBOL\SAMPLES\COBOL\SAMPLE11\SAMPLE11.PRJ

2. Select "Build" from Project Manager's "Project" menu or click on the Build button in the Project Manager toolbar.

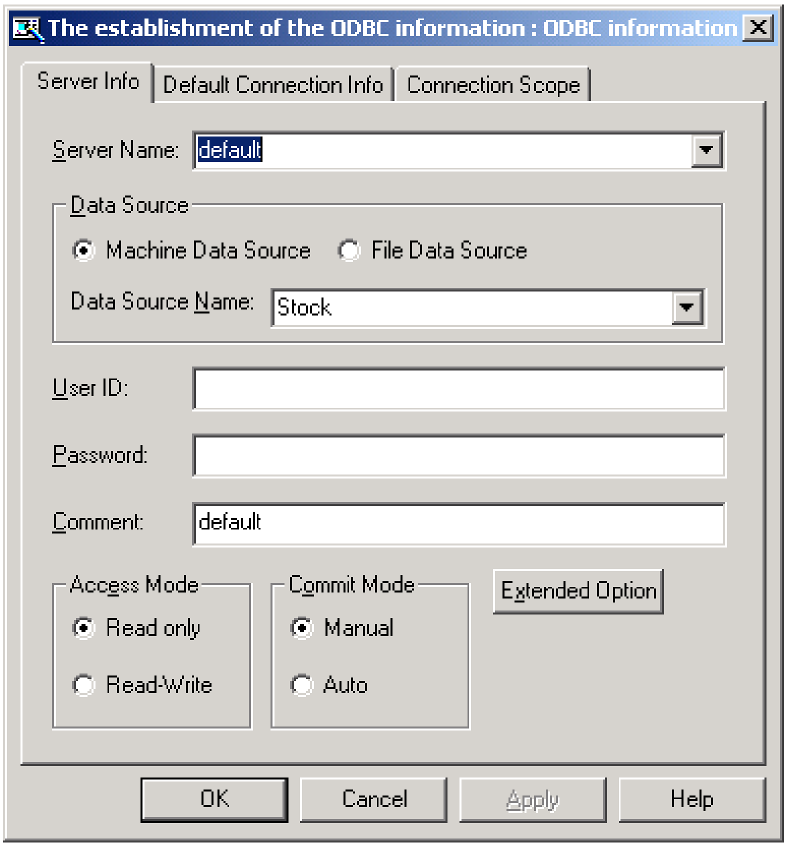
   After the build terminates, check that SAMPLE11.EXE is created.

Setting ODBC information

The ODBC information file is necessary for the execution of this sample program. When connecting it with the database by using the ODBC driver, necessary information is included as for the ODBC information file.

The ODBC information file is made by using ODBC information Setup Tool (SQLODBCS.EXE). Refer to the “Database (SQL)” chapter in the “NetCOBOL User’s Guide” for more detailed information.

The following figures show the place where information necessary to connect it with an attached sample database by this sample program was set.



Server Name: default

Data Source
● Machine Data Source ○ File Data Source
Data Source Name: Stock

User ID:

Password:

Comment: default

Access Mode
● Read only
○ Read-Write

Commit Mode
● Manual
○ Auto

Note

Information shown up is included in SAMPLE11.INF. When data bases other than Microsoft Access are used, setting other items might become necessary.

Setting Run-time Environment Information

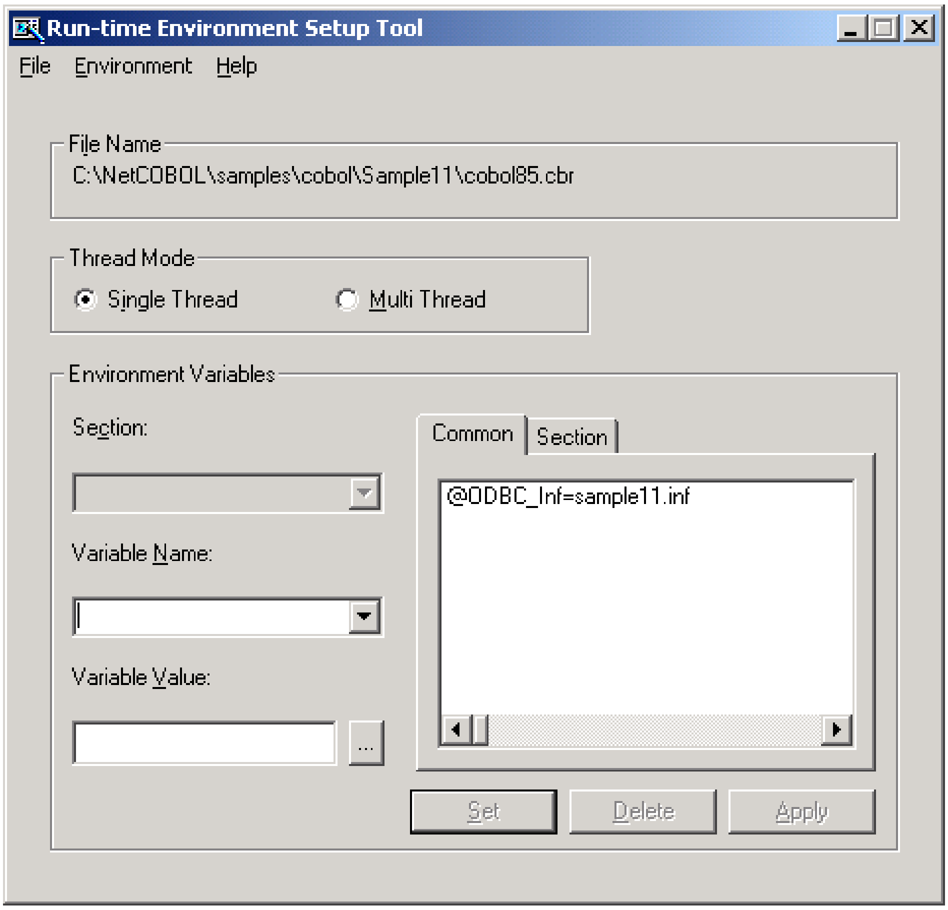
1. Select "Run-time Environment Setup Tool" from the "Tools" menu of the Project Manager.

   The run-time environment setup tool is displayed.

2. Select "Open" on the "File" menu and select COBOL85.CBR in the folder that contains the executable program (SAMPLE11.EXE).

3. Select the Common tab and enter data as shown below:

   - For the environment variable @ODBC_Inf , specify SAMPLE11.INF.



Run-time Environment Setup Tool

File Name
C:\NetCOBOL\samples\cobol\Sample11\cobol85.cbr

Thread Mode
● Single Thread ○ Multi Thread

Environment Variables

Common

@ODBC_Inf=sample11.inf

4. Click the Apply button.

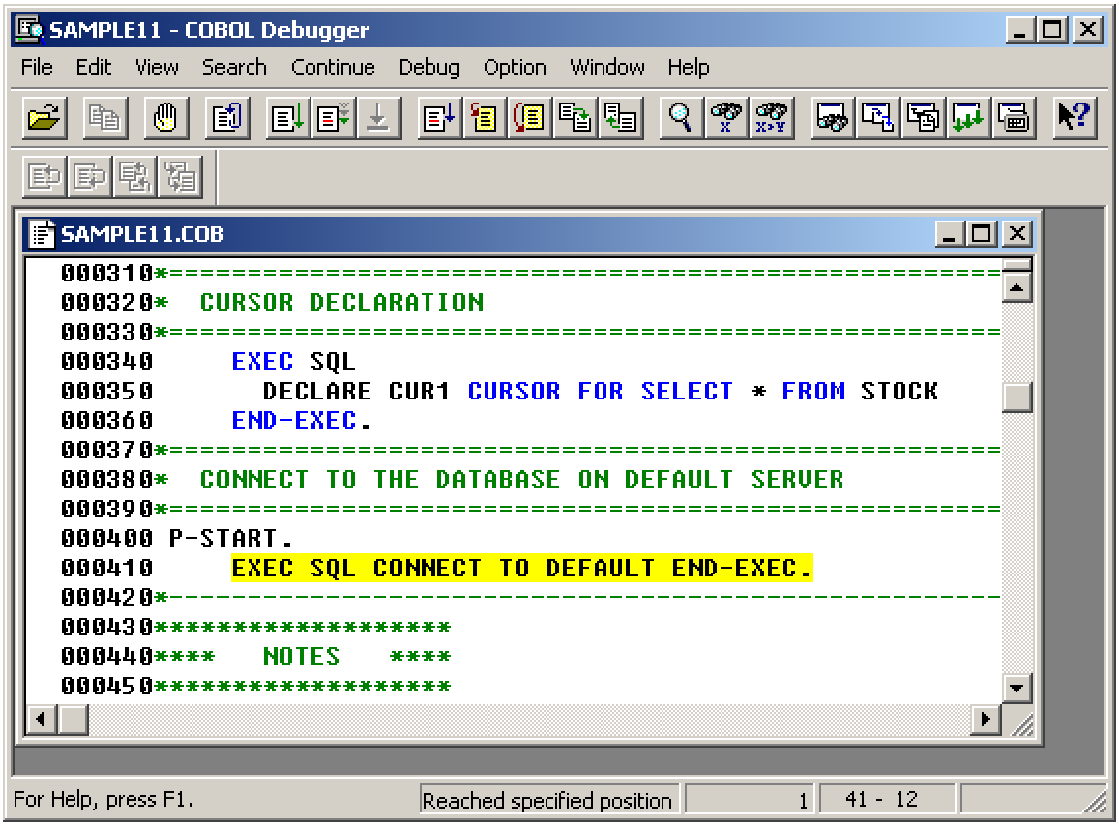
   The data is saved in the object initialization file.

5. Select "Exit" on the "File" menu to terminate the run-time environment setup tool.

Debugging the Program

To run the program under the control of the Debugger, select Debug from the Project menu. The Debugger starts and displays the Start Debugging dialog box. Press the ENTER key.

When the source is read into the COBOL Debugger, it will look like the following figure.



SAMPLE11 - COBOL Debugger
File Edit View Search Continue Debug Option Window Help
SAMPLE11.COB

```
000310*==========================================================
000320*  CURSOR DECLARATION
000330*==========================================================
000340     EXEC SQL
000350       DECLARE CUR1 CURSOR FOR SELECT * FROM STOCK
000360     END-EXEC.
000370*==========================================================
000380*  CONNECT TO THE DATABASE ON DEFAULT SERVER
000390*==========================================================
000400 P-START.
000410     EXEC SQL CONNECT TO DEFAULT END-EXEC.
000420*----------------------------------------------------------
000430*******************
000440****    NOTES    ****
000450*******************
```

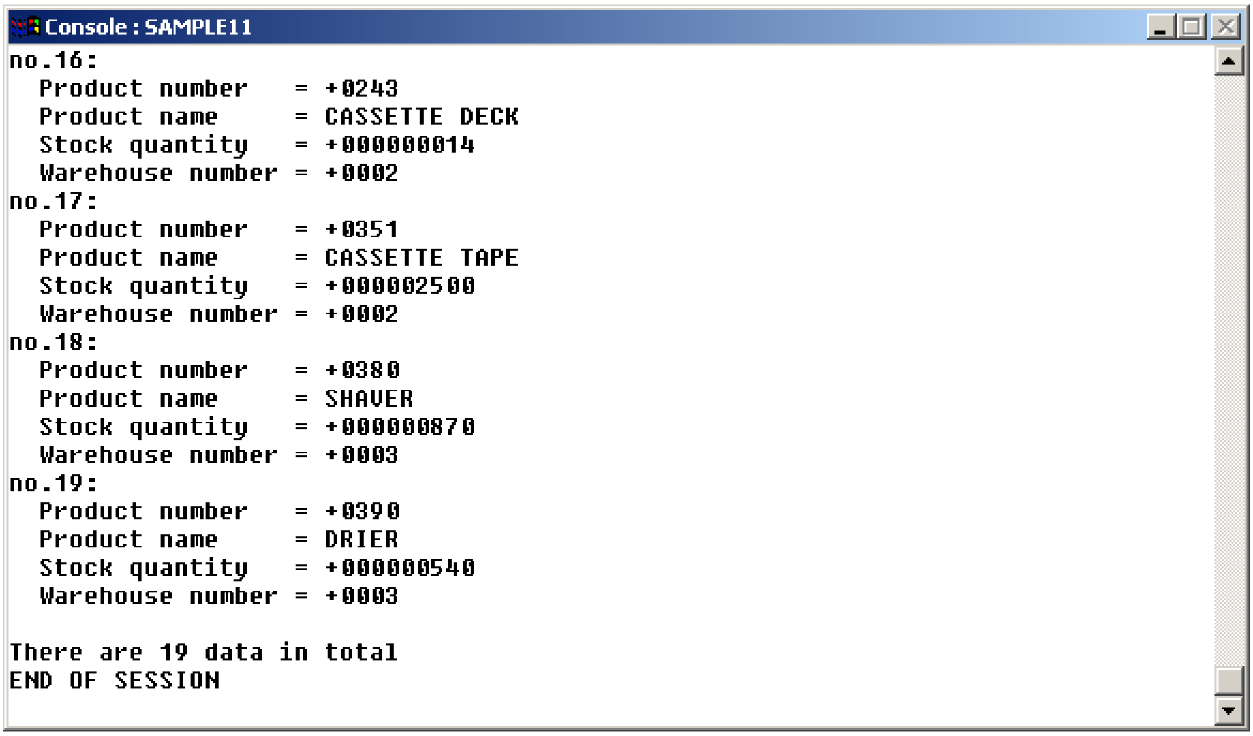
For Help, press F1. Reached specified position 1 41 - 12

Executing the Program

To execute this program, you can double-click on SAMPLE11.EXE in the Project Tree or select Execute from the Project menu, with the project or .EXE file selected.

Execution Result

The data extracted from the table is displayed, as shown in the following figure.



```
no.16:
  Product number   = +0243
  Product name     = CASSETTE DECK
  Stock quantity   = +000000014
  Warehouse number = +0002
no.17:
  Product number   = +0351
  Product name     = CASSETTE TAPE
  Stock quantity   = +000002500
  Warehouse number = +0002
no.18:
  Product number   = +0380
  Product name     = SHAVER
  Stock quantity   = +000000870
  Warehouse number = +0003
no.19:
  Product number   = +0390
  Product name     = DRIER
  Stock quantity   = +000000540
  Warehouse number = +0003

There are 19 data in total
END OF SESSION
```

A.13 Sample 12: Remote Database Access (Multiple row processing)

A sample program (sample 12) provided with this product is explained below.

Sample 12 shows an example where two or more lines in a database are operated with one SQL statement as an example of advanced usage of the database (SQL) function.

A database is present in a server and is accessed from a client.

A database is accessed via the ODBC driver. For details of database access using the ODBC driver, refer to Chapter 19 in the “NetCOBOL User’s Guide”.

To use this program, the following products are necessary.

Client:

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

Server:

- Database

NOTE:

- The function to use with this sample depends on your database. With Microsoft® Access, we could not execute this sample correctly.
- Products necessary for database access using ODBC

Overview

Sample 12 uses the STOCK table of the sample database. Refer to “Sample Database” in the “NetCOBOL User’s Guide” for details of the sample database.

- 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)
- SAMPLE12.PRJ (Project file)
- SAMPLE12.CBI (COBOL compiler option file)
- COBOL85.CBR (COBOL run-time initialization file)
- SAMPLE12.INF (ODBC information file)

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 following 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

GNO	Goods	QOH	WHNO
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

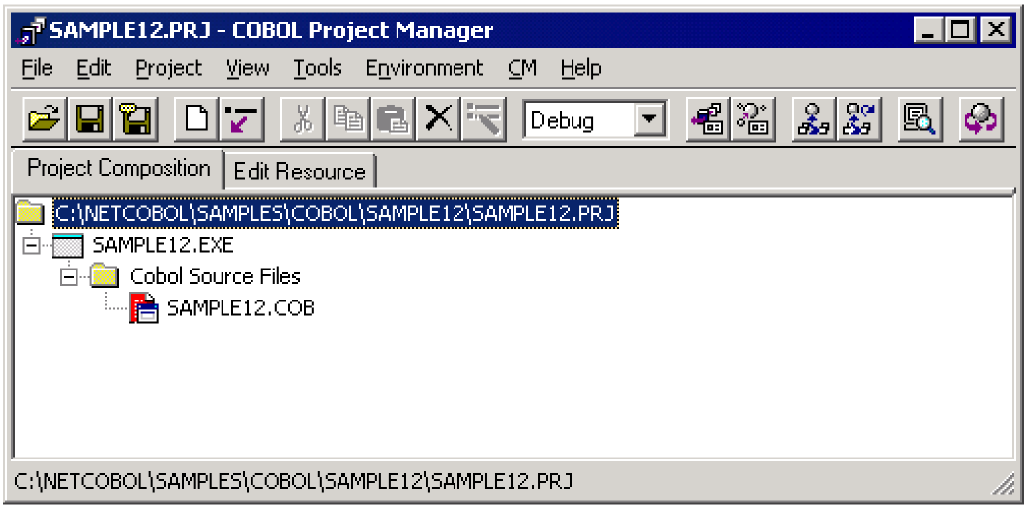
Building and rebuilding the program

The project manager's build function is used to compile and link this program.

Folder C:\NetCOBOL is assumed below as the location where NetCOBOL is installed.

Change the folder name C:\NetCOBOL to the name of the folder where NetCOBOL is installed.

1. Start the Project Manager, and project file SAMPLE12.PRJ is open.


SAMPLE12.PRJ - COBOL Project Manager
File Edit Project View Tools Environment CM Help
Debug
Project Composition
Edit Resource

C:\NETCOBOL\SAMPLES\COBOL\SAMPLE12\SAMPLE12.PRJ
SAMPLE12.EXE
Cobol Source Files
SAMPLE12.COB

C:\NETCOBOL\SAMPLES\COBOL\SAMPLE12\SAMPLE12.PRJ

2. Select Build from the Project menu.

   After build termination, check that SAMPLE12.EXE is created.

Setting ODBC information

The ODBC information file is necessary for the execution of this sample program. When connecting it with the database by using the ODBC driver, necessary information is included as for the ODBC information file.

The ODBC information file is made by using ODBC information Setup Tool (SQLODBCS.EXE). Refer to the “Database (SQL)” chapter in the “NetCOBOL User’s Guide” for more detailed information.

Setting Run-time Environment Information

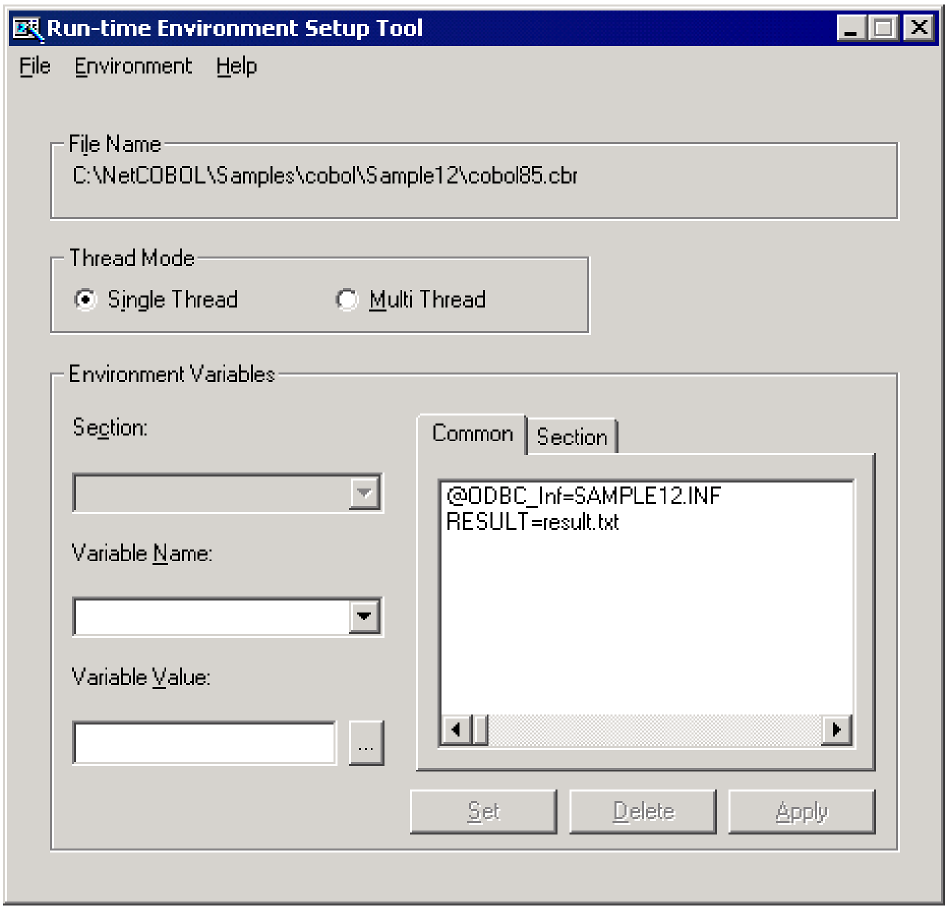
1. Select “Run-time Environment Setup Tool” from the Tools menu.

   The Run-time Environment Setup Tool is displayed.

2. Select "Open" on the File menu and create an object initialization file (COBOL85.CBR) in the folder that contains the executable program (SAMPLE12.EXE).

3. Select a common tab and set data as shown below:

   - Specify an ODBC information file name in environment variable information @ODBC_Inf (ODBC information file specification).
   - Specify a file to save the DISPLAY statement output result in environment variable RESULT.



File Name: C:\NetCOBOL\Samples\cobol\Sample12\cobol85.cbr

Thread Mode: ● Single Thread ○ Multi Thread

```
@ODBC_Inf=SAMPLE12.INF
RESULT=result.txt
```

4. Click the Apply button.

   The set data is saved in the object initialization file.

5. Select "Exit" on the File menu to terminate the Run-time environment setup tool.

Debugging the Program

To run the program under the control of the Debugger, select Debug from the Project menu. The Debugger starts and displays the Start Debugging dialog box.

Press the ENTER key.

Executing the program

Select "Run" on the Project Manager "Project" menu.

The following is displayed in the COBOL console window.

```
SUCCESSFUL CONNECTION WITH DATABASE.
RECEIVE THE PRODUCT NUMBER WHOSE PRODUCT NAME IS ‘TELEVISION’
SET STOCKS OF THE FOLLOWING PRODUCTS DECREASING 10
TELEVISION -> + 0110
TELEVISION -> + 0111
TELEVISION -> + 0212
DELETE THE ROW WHICH HAS PRODUCT NAME IS ‘RADIO’, ‘SHAVER’ OR ‘DRIER’.
PROGRAM END
```

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

```
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.
```

A.14 Sample 13: Calling COBOL from Visual Basic

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

This sample program requires Visual Basic 5.0 or later.

Overview

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

The Visual Basic form shows a simple equation in which the user enters two numbers (either side of a multiply “*” sign) and presses the “=” sign, which is a 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.

In the Visual Basic application’s termination code, it calls the JMPCINT3 subroutine to close the COBOL run-time environment.

Files Included in Sample 13

- SAMPLE13.EXE (Visual Basic executable file)
- SAMPLE13.VBP (Visual Basic project file)
- SAMPLE13.FRM (Visual Basic form module)

- SAMPLE13.COB (COBOL source program)
- SAMPLE13.PRJ (COBOL project file)
- SAMPLE13.CBI (COBOL compiler option file)
- SAMPLE13.LNI (Linkage option file)
- COBOL85.CBR (COBOL run-time initialization file)

Subroutines used in Sample 13

These subroutines are used by in the Visual Basic part of SAMPLE 13 to initialize and terminate the COBOL run-time system.

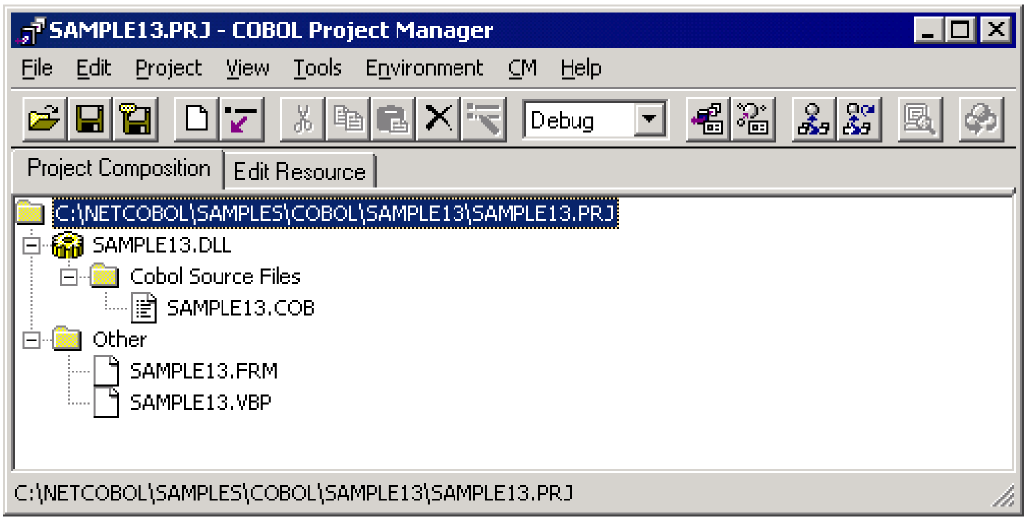
- JMPCINT2
- JMPCINT3

Building/Rebuilding the Program

Project manager's build function is used to create the executable program.

In the following screen snapshots, the sample program was installed to C:\NetCOBOL. Your installation folder may be different.

1. The project manager is started, and project file "SAMPLE13.PRJ" is opened.

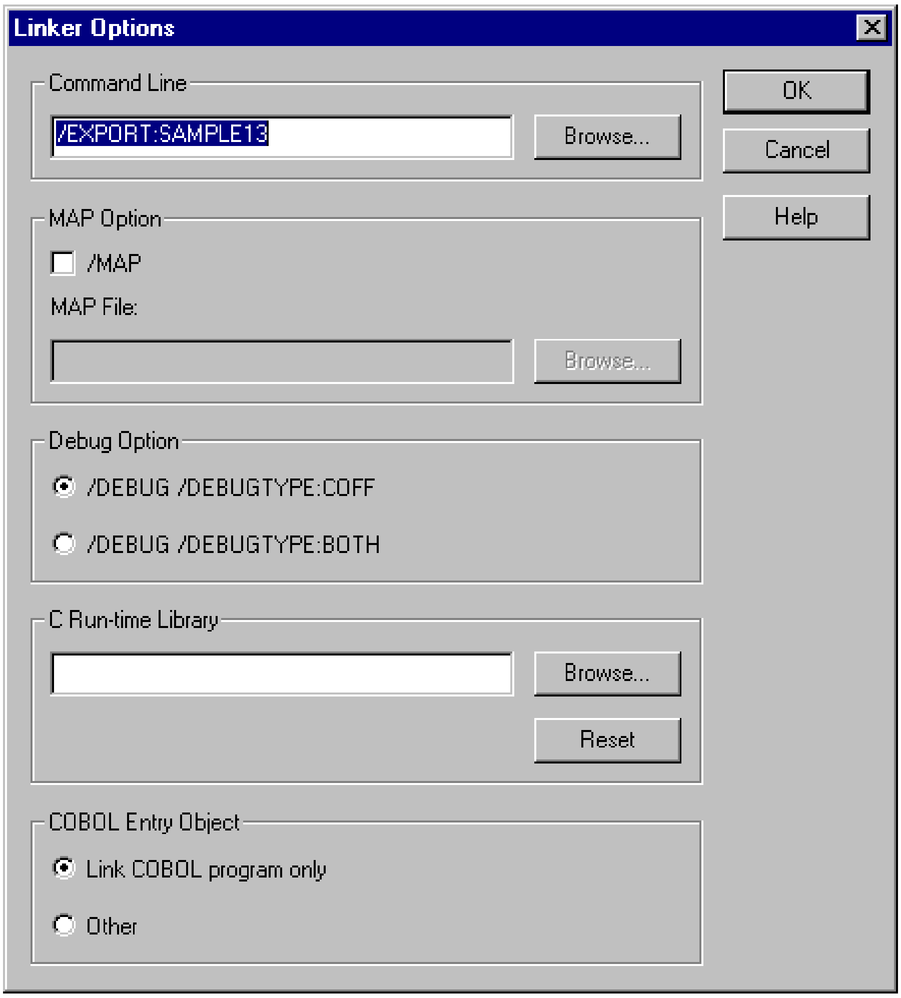


C:\NETCOBOL\SAMPLES\COBOL\SAMPLE13\SAMPLE13.PRJ
SAMPLE13.DLL
Cobol Source Files
SAMPLE13.COB
Other
SAMPLE13.FRM
SAMPLE13.VBP

C:\NETCOBOL\SAMPLES\COBOL\SAMPLE13\SAMPLE13.PRJ

2. The project file is selected, and "Linker options" is selected from "Project"-"Options" menu.

   The "Linker options" dialog is displayed.



Command Line: /EXPORT:SAMPLE13

MAP Option: ☐ /MAP

Debug Option:
● /DEBUG /DEBUGTYPE:COFF
○ /DEBUG /DEBUGTYPE:BOTH

COBOL Entry Object:
● Link COBOL program only
○ Other

3. "/EXPORT:SAMPLE13" is specified for the "Command line" editing box. After confirms it, the OK button is clicked.

   You are now returned to the Project Manager window.

4. Select "Build" from Project Manager's "Project" menu or click on the Build button in the Project Manager toolbar.

   After the build terminates, check that SAMPLE13.dll is created.

5. When you rebuild executable file of Visual Basic, SAMPLE13.VBP (Visual Basic project file) included in the "Other" folder of the project is selected and "Edit" is selected from the project menu.

   Visual Basic starts; "Make of SAMPLE13.EXE" is selected from the file menu.

Debugging the Program

Applications containing a mixture of COBOL and Visual Basic code can be debugged under the control of both the COBOL and Visual Basic debuggers. To accomplish this, follow these steps:

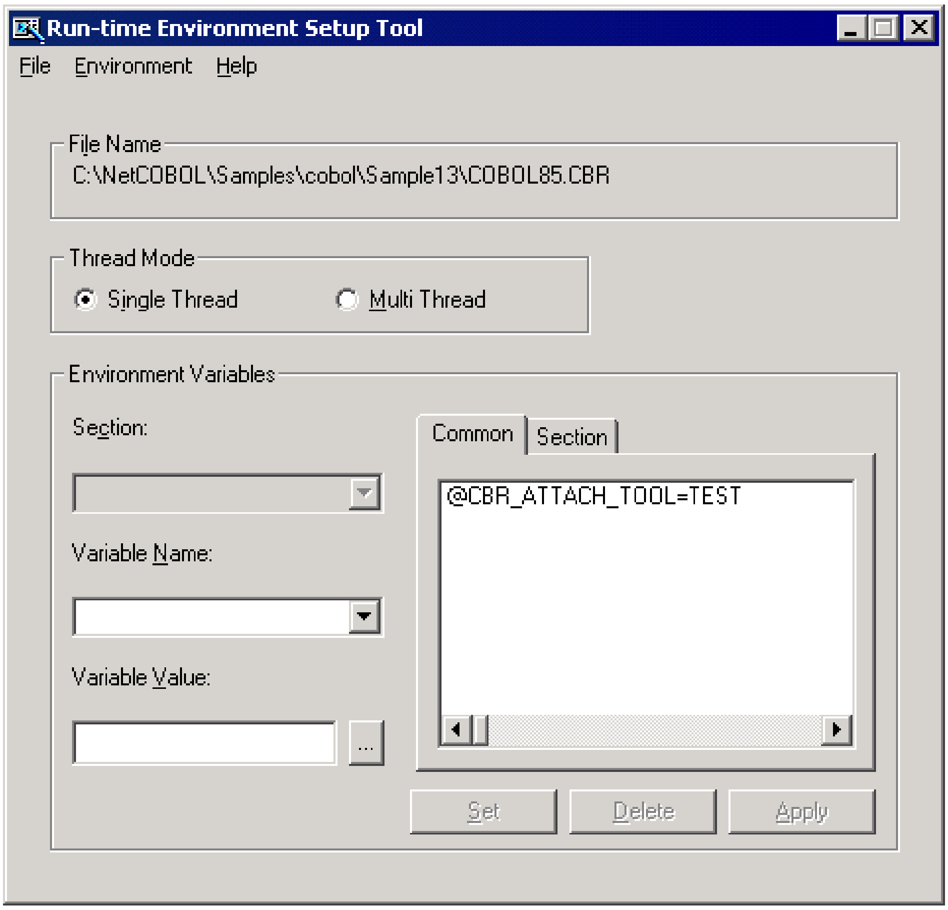
1. Select "Run-time Environment Setup Tool" from the "Tools" menu of the Project Manager.

   The run-time environment setup tool is displayed.

2. Select "Open" on the "File" menu and select COBOL85.CBR in the folder that contains the executable program (SAMPLE13.EXE).

3. Select the Common tab and enter data as shown below:

   - For the environment variable @CBR_ATTACH_TOOL, specify TEST.



Run-time Environment Setup Tool

File Name: C:\NetCOBOL\Samples\cobol\Sample13\COBOL85.CBR

Thread Mode: ● Single Thread ○ Multi Thread

Environment Variables

```
@CBR_ATTACH_TOOL=TEST
```

4. Click the Apply button.

   The data is saved in the object initialization file.

5. Select "Exit" on the "File" menu to terminate the run-time environment setup tool.
6. Start Visual Basic and load the SAMPLE13.VBP project.
7. Start debugging the Visual Basic code when you execute the call to SAMPLE13 (the COBOL program) the COBOL debugger is invoked so that you can continue debugging in the COBOL part of the application.
8. When you exit from the COBOL application control returns to the Visual Basic code.

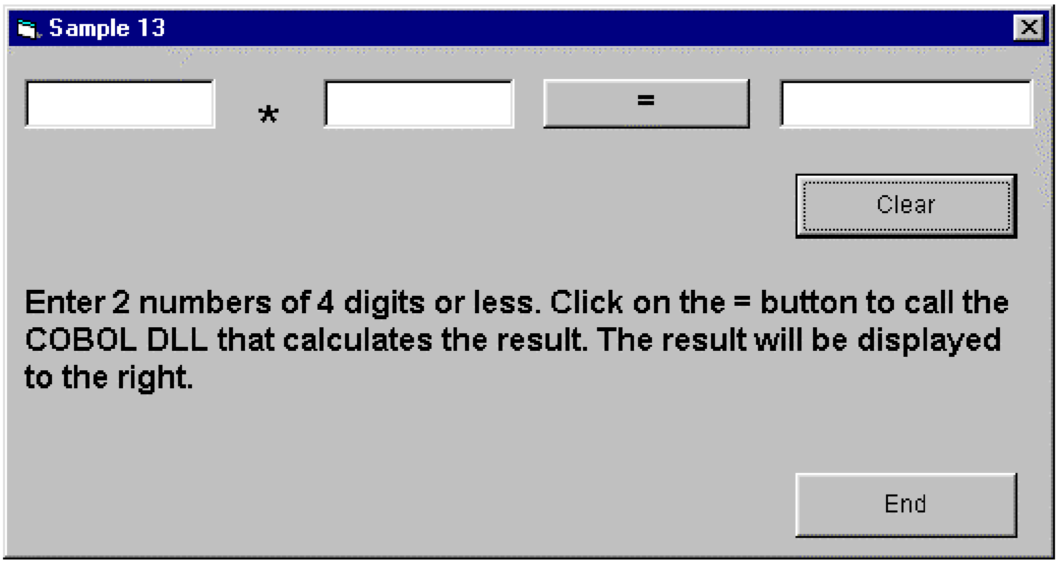
The section “H. Debugging with Visual Basic” in Chapter 1’s Quick Tour steps through this example in some detail.

For more information on this topic, see Chapter 3, “Working with Visual Basic and COBOL” and the “NetCOBOL Debugging Guide”.

Executing the Program

To execute the program:

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



Enter 2 numbers of 4 digits or less. Click on the = button to call the COBOL DLL that calculates the result. The result will be displayed to the right.

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.

Visual Basic calls COBOL to perform the calculation and format the answer. Visual Basic then displays the answer to the right of the “=” button.

A.15 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.

Operating this program requires Visual Basic version 5.0 or later.

Overview

The 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

- SAMPLE14.EXE (Visual Basic executable file)
- SAMPLE14.VBP (Visual Basic project file)
- SAMPLE14.BAS (Visual Basic standard module)

- SAMPLE14.FRM (Visual Basic form module)

  The main window for the “ATM terminal” processing.

- SINKI.FRM (Visual Basic form module)

  Dialog for opening a new account.

- SINKCHK.FRM (Visual Basic form module)

  Displays the assigned account number for a new account.

- SELE.FRM (Visual Basic form module)

  Account-handling dialog - shows account number, name and balance, and offers the withdrawal and deposit functions.

- NYUKIN.FRM (Visual Basic form module)

  Dialog for performing a deposit.

- SYUKIN.FRM (Visual Basic form module)

  Dialog for performing a withdrawal.

- ERROR_H.FRM (Visual Basic form module)

  Error message box.

- 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.

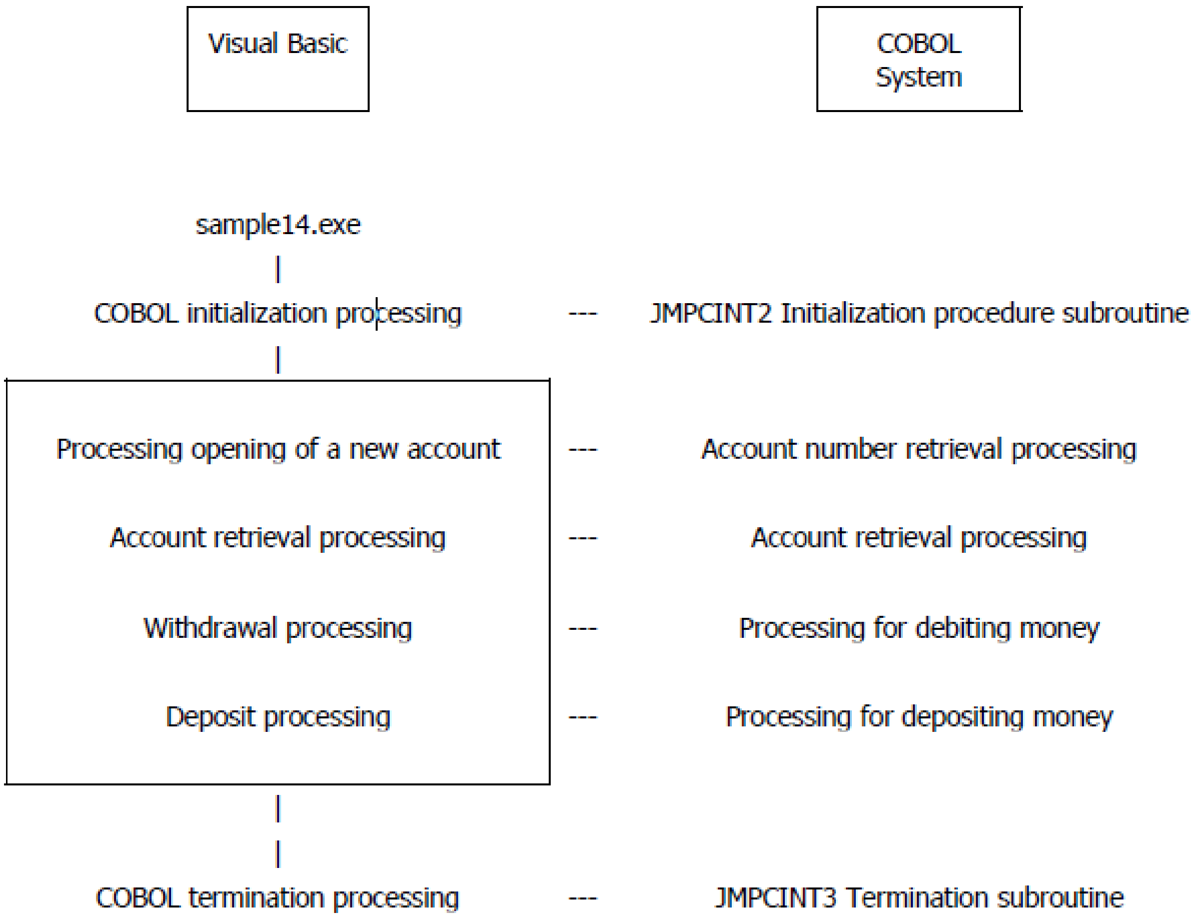
Processing Overview

The Visual Basic application starts, and subroutine JMPCINT2 (which initializes the COBOL run-time 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 run-time).

Following figure shows the structure of the application:


Visual Basic
COBOL System
sample14.exe
COBOL initialization processing

JMPCINT2 Initialization procedure subroutine
Processing opening of a new account

Account number retrieval processing
Account retrieval processing

Account retrieval processing
Withdrawal processing

Processing for debiting money
Deposit processing

Processing for depositing money
COBOL termination processing

JMPCINT3 Termination subroutine

COBOL Statements used in Sample 14

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

COBOL Run-time System Subroutines

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

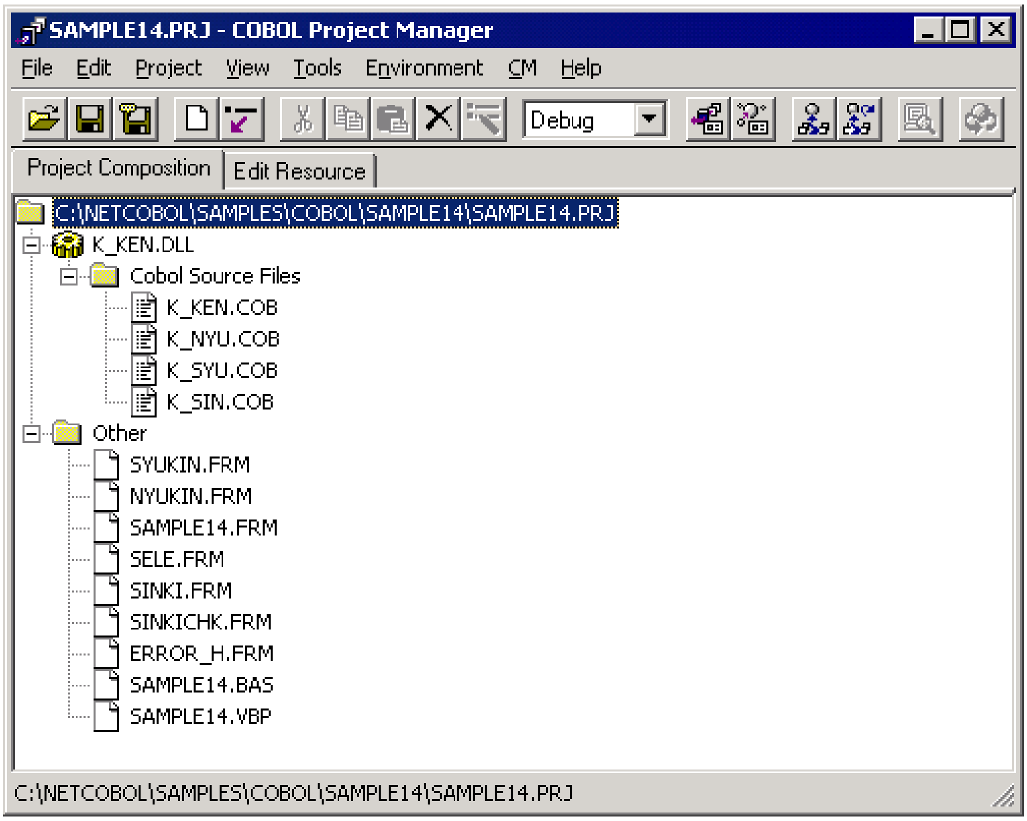
- JMPCINT2
- JMPCINT3

Building/Rebuilding the Program

Project manager's build function is used to create the executable program.

In the following screen snapshots, the sample program was installed to C:\NetCOBOL. Your installation folder may be different.

1. The project manager is started, and project file "SAMPLE14.PRJ" is opened.



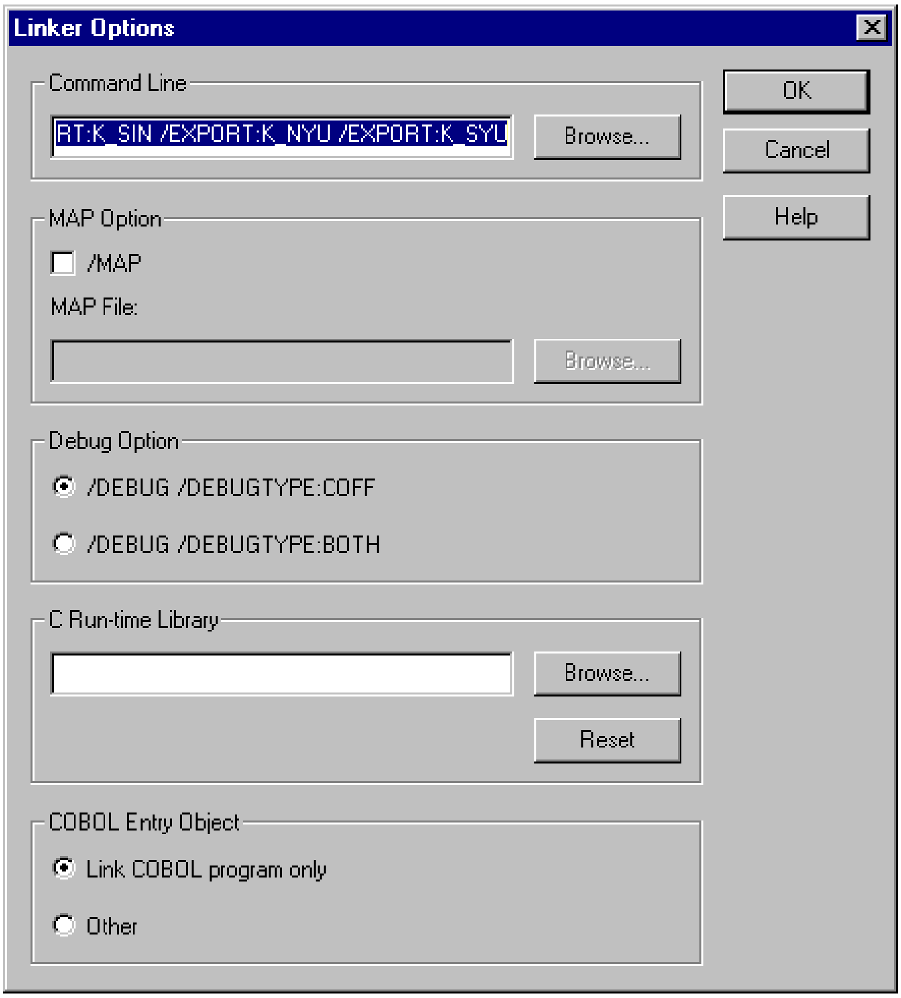
SAMPLE14.PRJ - COBOL Project Manager

```
C:\NETCOBOL\SAMPLES\COBOL\SAMPLE14\SAMPLE14.PRJ
  K_KEN.DLL
    Cobol Source Files
      K_KEN.COB
      K_NYU.COB
      K_SYU.COB
      K_SIN.COB
  Other
    SYUKIN.FRM
    NYUKIN.FRM
    SAMPLE14.FRM
    SELE.FRM
    SINKI.FRM
    SINKICHK.FRM
    ERROR_H.FRM
    SAMPLE14.BAS
    SAMPLE14.VBP
```

C:\NETCOBOL\SAMPLES\COBOL\SAMPLE14\SAMPLE14.PRJ

2. The project file is selected, and "Linker options" is selected from "Project"-"Options" menu.

   The "Linker options" dialog is displayed.



Command Line

RT:K_SIN /EXPORT:K_NYU /EXPORT:K_SYU

MAP Option

□ /MAP

MAP File:

Debug Option

● /DEBUG /DEBUGTYPE:COFF

○ /DEBUG /DEBUGTYPE:BOTH

C Run-time Library

COBOL Entry Object

● Link COBOL program only

○ Other

3. "/EXPORT:K_KEN /EXPORT:K_SIN /EXPORT:K_NYU /EXPORT:K_SYU" is specified for the "Command line" editing box. After confirms it, the OK button is clicked.

   You are now returned to the Project Manager window.

4. Select "Build" from Project Manager's "Project" menu or click on the Build button in the Project Manager toolbar.

   After the build terminates, check that K_KEN .DLL is created.

5. When you rebuild executable file of Visual Basic, SAMPLE14.VBP (Visual Basic project file) included in the "Other" folder of the project is selected and "Edit" is selected from the project menu.

   Visual Basic starts; "Make of SAMPLE14.EXE" is selected from the file menu.

Debugging the Program

Applications containing a mixture of COBOL and Visual Basic code can be debugged under the control of both the COBOL and Visual Basic debuggers. To accomplish this, follow these steps:

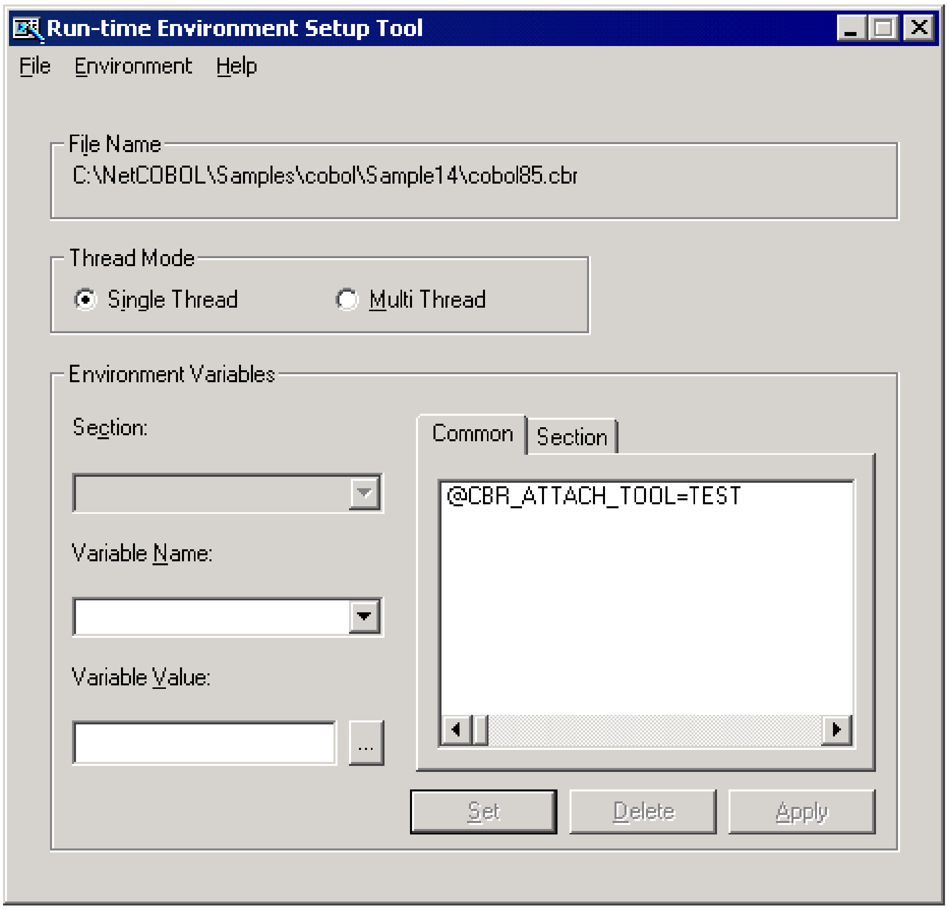
1. Select "Run-time Environment Setup Tool" from the "Tools" menu of the Project Manager.

   The run-time environment setup tool is displayed.

2. Select "Open" on the "File" menu and select COBOL85.CBR in the folder that contains the executable program (SAMPLE14.EXE).

3. Select the Common tab and enter data as shown below:

   - For the environment variable @CBR_ATTACH_TOOL, specify TEST.



File Name
C:\NetCOBOL\Samples\cobol\Sample14\cobol85.cbr

Thread Mode
● Single Thread ○ Multi Thread

@CBR_ATTACH_TOOL=TEST

4. Click the Apply button.

   The data is saved in the object initialization file.

5. Select "Exit" on the "File" menu to terminate the run-time environment setup tool.
6. Start Visual Basic and load the SAMPLE14.VBP project.
7. Start debugging the Visual Basic code when you execute the call to SAMPLE14 (the COBOL program) the COBOL debugger is invoked so that you can continue debugging in the COBOL part of the application.
8. When you exit from the COBOL application control returns to the Visual Basic code.

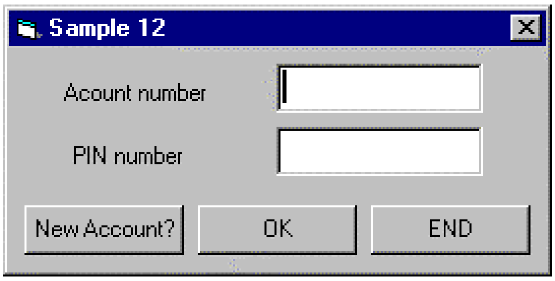
The section “H. Debugging with Visual Basic” in Chapter 1’s Quick Tour steps through this example in some detail.

For more information on this topic, see Chapter 3, “Working with Visual Basic and COBOL” and the “NetCOBOL Debugging Guide”.

Executing the Program

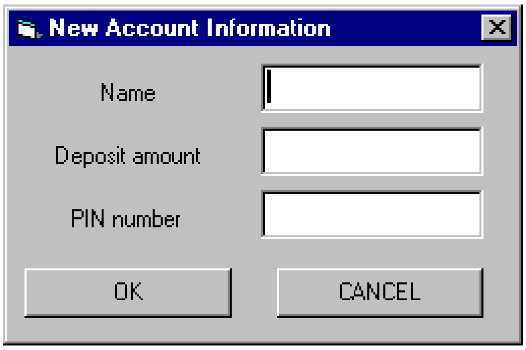
To execute the program:

1. The first application dialog is displayed:



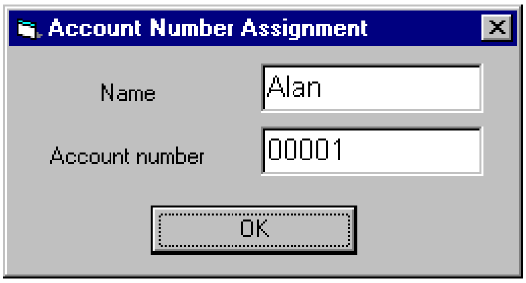
2. To work with the application click on the “New Account?” button.

   The New Account Information dialog is displayed:



3. Open a new account. Type in a name (such as Alan), an amount (such as 100000) and a PIN number (such as 1234). Click the OK button.

   A dialog displaying the assigned account number is displayed:



Name: Alan

Account number: 00001

4. Note the account number and the PIN number you used - you will need to enter them in subsequent transactions.
5. Click on the OK button to return to the main Sample 14 dialog box.
6. You can now enter the account and PIN numbers and use the Withdrawal and Deposit functions.

A.16 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 the employee objects have the same form, they have different data (employee numbers, names, departments and sections, and address information). Address information also belongs to an object, containing postal codes and addresses.

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

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. The user must, however, understand the two methods of "Data-Set" and "Data-Display."

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)
- SAMPLE15.PRJ (Project file)
- SAMPLE15.CBI (Compiler option file)
- SAMPLE15.TXT (Program guide)

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

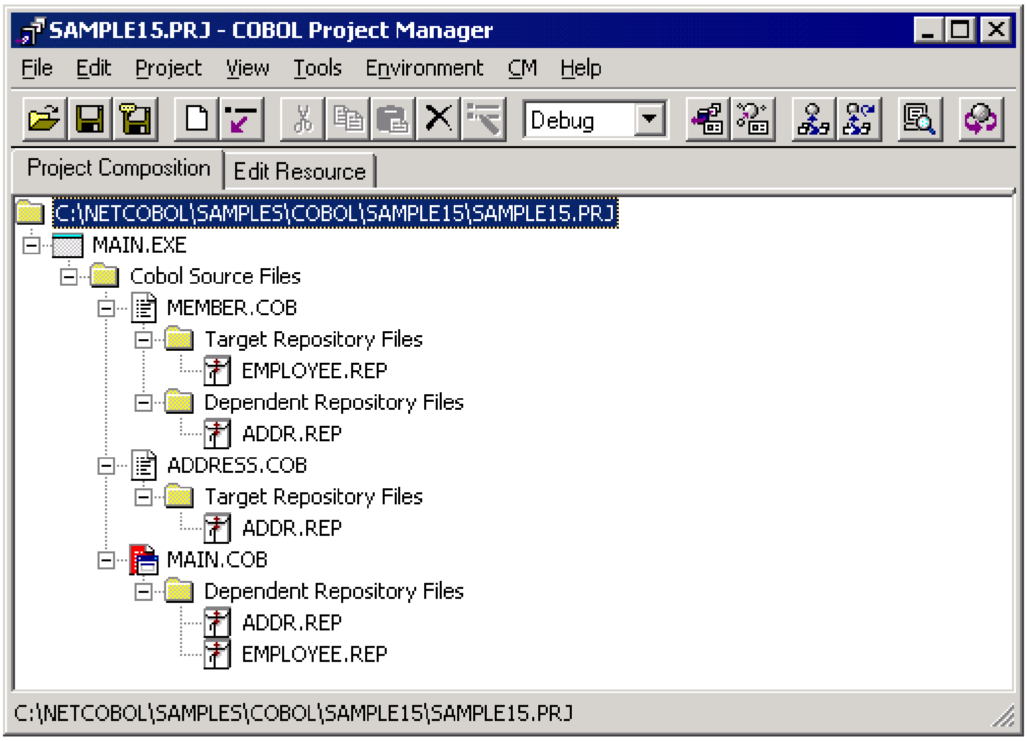
- INVOKE and SET statements
- REPOSITORY paragraph
- Class, object and method definitions

Building/Rebuilding the Program

Project manager's build function is used to create the executable program.

In the following screen snapshots, the sample program was installed to C:\NetCOBOL. Your installation folder may be different.

1. The project manager is started, and project file "SAMPLE15.PRJ" is opened.



```
C:\NETCOBOL\SAMPLES\COBOL\SAMPLE15\SAMPLE15.PRJ
  MAIN.EXE
    Cobol Source Files
      MEMBER.COB
        Target Repository Files
          EMPLOYEE.REP
        Dependent Repository Files
          ADDR.REP
      ADDRESS.COB
        Target Repository Files
          ADDR.REP
      MAIN.COB
        Dependent Repository Files
          ADDR.REP
          EMPLOYEE.REP
C:\NETCOBOL\SAMPLES\COBOL\SAMPLE15\SAMPLE15.PRJ
```

2. Select "Build" from Project Manager's "Project" menu or click on the Build button in the Project Manager toolbar.

   After the build terminates, check that MAIN.EXE is created.

Debugging the Program

You can execute this program under the control of the COBOL Debugger by selecting Debug from the Project menu.

Executing the Program

To execute the linked program, select Execute from the Project menu.

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

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 Y or N to terminate or continue.

A.17 Sample 16: Collection Class (Class Library)

Sample 16 demonstrates using a collection class for creating a class library.

This sample can be used as an example of class library creation, and 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 rolled out by modifying and changing this sample.

Overview

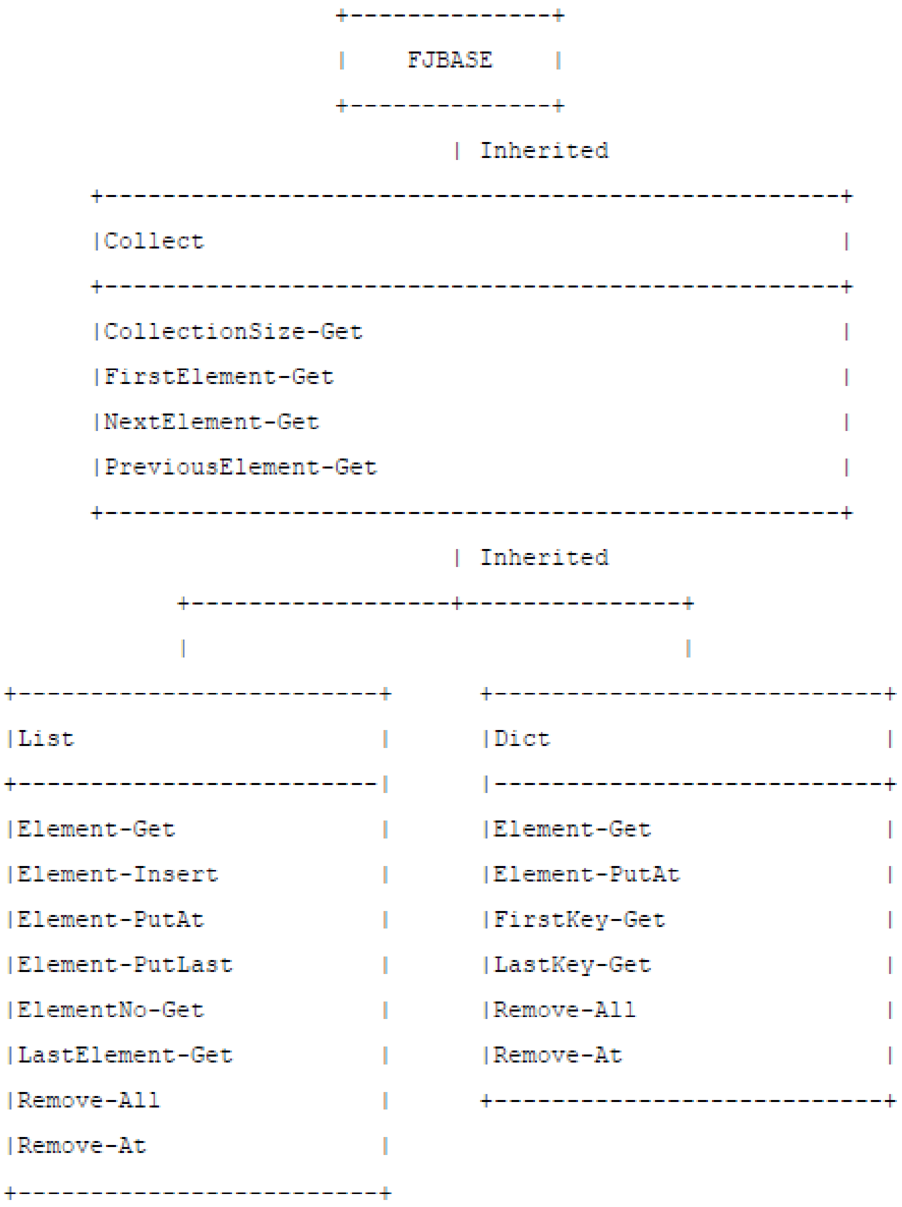
A collection class is the generic name of a class that handles a set of objects-it is used to collectively handle and store many objects.

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.



```
                                  +---------------+
                                  |     FJBASE    |
                                  +---------------+
                                          | Inherited
                 +-----------------------------------------------------+
                 |Collect                                              |
                 +-----------------------------------------------------+
                 |CollectionSize-Get                                   |
                 |FirstElement-Get                                     |
                 |NextElement-Get                                      |
                 |PreviousElement-Get                                  |
                 +-----------------------------------------------------+
                                          | Inherited
                       +-------------------+----------------+
                       |                                    |
           +--------------------------+      +-----------------------------+
           |List                      |      |Dict                         |
           +--------------------------|      |-----------------------------+
           |Element-Get               |      |Element-Get                  |
           |Element-Insert            |      |Element-PutAt                |
           |Element-PutAt             |      |FirstKey-Get                 |
           |Element-PutLast           |      |LastKey-Get                  |
           |ElementNo-Get             |      |Remove-All                   |
           |LastElement-Get           |      |Remove-At                    |
           |Remove-All                |      +-----------------------------+
           |Remove-At                 |
           +--------------------------+
```

Note

In addition to the above classes, Sample 15 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 investigates the number of elements in a set.

Parameter

None

Return value

PIC 9(8) BINARY

Returns the element immediately preceding the one currently pointed to. If a previous element does not exist, 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.
- Duplicate elements can be contained.

Since this class inherits from the Collect class, all 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)
- SAMPLE16.PRJ (Project file)
- SAMPLE16.CBI (Compiler option file)
- SAMPLE16.TXT (Program guide)

COBOL Functions Used in Sample 16

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

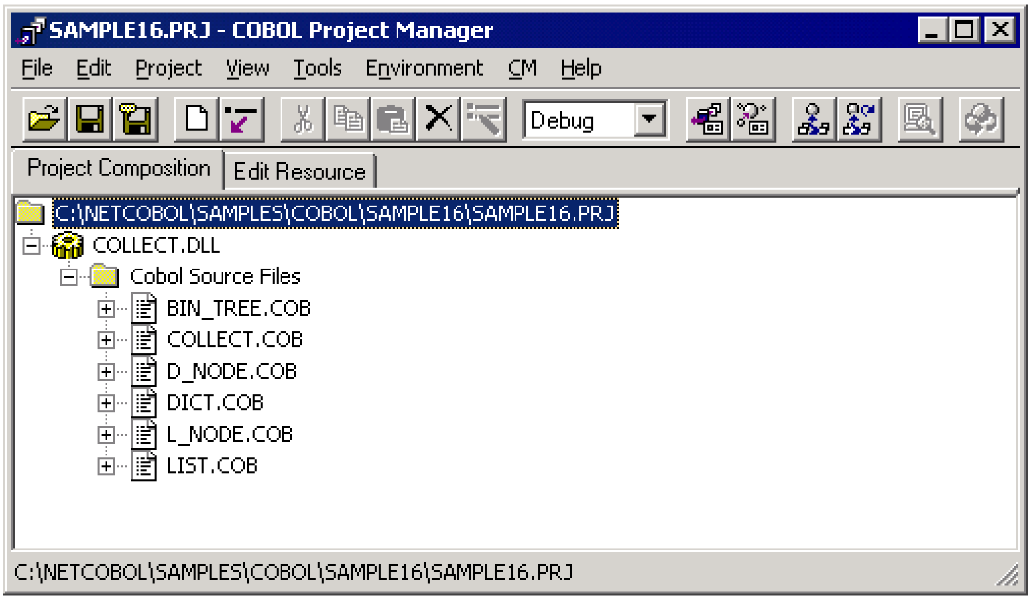
Object-Oriented Syntax used in Sample 16

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

Building the Class Library

The Build function supported by the COBOL Project Manager is used to create the class library.

1. Start the Project Manager, and project file SAMPLE16.PRJ is opened.



SAMPLE16.PRJ - COBOL Project Manager
File Edit Project View Tools Environment CM Help
Debug
Project Composition
Edit Resource

C:\NETCOBOL\SAMPLES\COBOL\SAMPLE16\SAMPLE16.PRJ
COLLECT.DLL
Cobol Source Files
BIN_TREE.COB
COLLECT.COB
D_NODE.COB
DICT.COB
L_NODE.COB
LIST.COB

C:\NETCOBOL\SAMPLES\COBOL\SAMPLE16\SAMPLE16.PRJ

2. Select Build from the Project menu.

When building 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. See Sample 18, ”Advanced Object-oriented Programming.”

For Executing

- COLLECT.DLL (DLL file)

A.18 Sample 17: Object-Oriented Cobol (Aggregation, Singleton, and Iteration)

Sample 17 demonstrates the general object-oriented design concepts of aggregation, singleton, and iteration.

The program use the Dict and List classes created in "Sample 16 Collection Class (Class Library)".

It also requires Microsoft Excel (abbreviated to “Excel”).

Overview

The purpose of Sample 17 is to illustrate implementation of the object-oriented concepts of aggregation, singleton and iteration (explained below). It uses a conference room booking system to demonstrate these features. The user interface is very basic as the focus is to demonstrate the OO concepts.

The conference room booking system provides functions for:

- Reserving a conference room.
- Canceling a conference booking.
- Reviewing conference room bookings.
- Listing, adding and deleting conference rooms.
- Updating conference room details.

The OO concepts are demonstrated as described below.

Aggregation

Aggregation is a relation of "whole to part". The life of the downstream class instance in the aggregate relation is controlled completely by the life of the upstream class instance. In this sample, the following classes are in the aggregation relation:

- Reservation-status-Date class (RSVSTATE-DATE-CLASS) and Reservation-status- Conference-Room class (RSVSTATE-CFNROOM-CLASS).
- Reservation-status-Conference-Room class (RSVSTATE-CFNROOM-CLASS) and Reservation-status-Time-Frame class (RSVSTATE-TIME-CLASS).
- Conference-Room-Information-Control calls (CFRINFO-CNT-CLASS) and Conference-Room-Information class (CFRINFO-CLASS).

Singleton

Singleton refers to a mechanism that guarantees that only one object instance can exist for a class.

This is implemented in the SINGLE-TON class in this sample. The SINGLE-TON class is inherited by the following classes so that they can only create a single object instance.

- Reservation control class
  - RSVCNT-CLASS
- Conference room information control class
  - CFRINFO-CNT-CLASS.

- Error processing class
    - ERROR-CLASS

Iteration

Iteration provides a means of accessing elements sequentially without having to be aware of the internal structure of the aggregation object. In this sample, iteration is implemented by the LIST-ITERATER class.

Two or more LIST-ITERATER class objects can be created for a single list object.

In this sample, iteration is used in the following functions:

- Display of reservation information

  Method: RSV-SITUATION-LIST-DISP,

  Class: RSVCNT-CLASS

- Deletion of reservation state objects (date, conference room and time frame)

  Method: RSV-SITUATION-OBJ-DEL

  Class: RSVCNT-CLASS

- Retrieval of conference room information and reservation information

  Method: RSV-DATA-RETRIEVAL

  Class: RSVINFO-CLASS

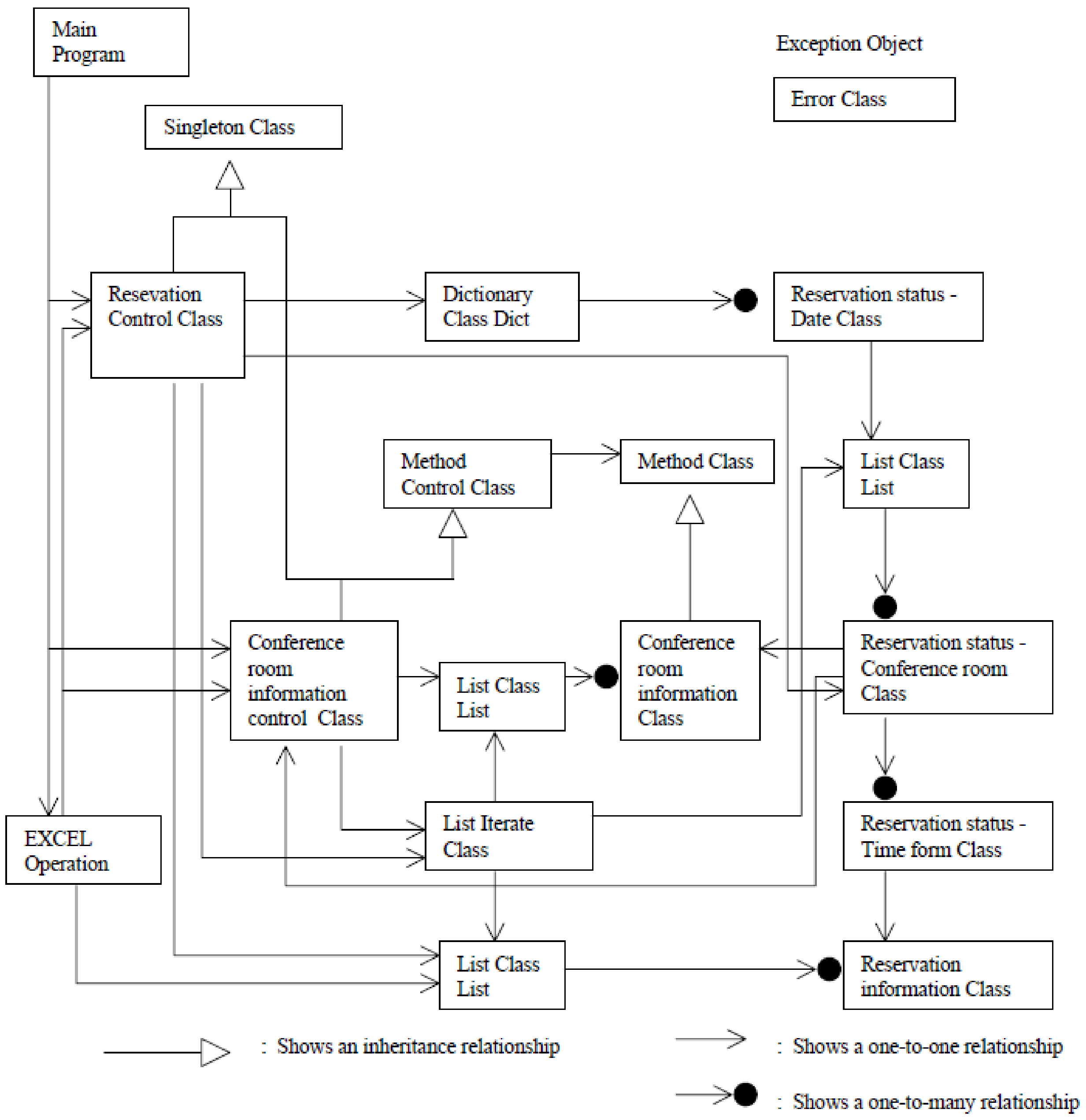
- Storing the information in an Excel file.

  Entry: PRESERVATION-DATA

  Program: EXCEL-EDIT

Design of Sample 17

Following figure shows the relationships between the classes used in Sample 17.


Main Program
Exception Object
Error Class
Singleton Class
Resevation Control Class
Dictionary Class Dict
Reservation status - Date Class
Method Control Class
Method Class
List Class List
Conference room information control Class
List Class List
Conference room information Class
Reservation status - Conference room Class
EXCEL Operation
List Iterate Class
Reservation status - Time form Class
List Class List
Reservation information Class
: Shows an inheritance relationship
: Shows a one-to-one relationship
: Shows a one-to-many relationship

Programs and Files in Sample 17

- MAIN.COB (COBOL source program)
- EXCELEDT.COB (COBOL source program)
- RSVCTRL.COB (COBOL source program)
- ROOMCTRL.COB (COBOL source program)
- DATESTA.COB (COBOL source program)
- ROOMSTA.COB (COBOL source program)
- TIMESTA.COB (COBOL source program)
- RESERVE.COB (COBOL source program)
- SPECCTRL.COB (COBOL source program)
- SPEC.COB (COBOL source program)

- SINGLETN.COB (COBOL source program)
- LISTITER.COB (COBOL source program)
- ERRORPUT.COB (COBOL source program)
- RSVINFO.CBL (library file)
- ROOMINFO.CBL (library file)
- R_CONST.CBL (library file)
- SPECINFO.CBL (library file)
- ROOMLIST.XLS (Excel file)
- RSVLIST.XLS (Excel file)
- DICT.REP (Repository file)
- LIST.REP (Repository file)
- SAMPLE17.PRJ (project file)
- COLLECT.DLL (DLL file)
- COLLECT.LIB (import library)

COBOL Functions used in Sample 17

- Object oriented programming functions
- Definition of class (encapsulation)
- Inheritance
- Multiple inheritance
- Creation of objects
- Method calls
- Exception handling
- OLE connection
- Project manager function

Object-Oriented Syntax used in Sample 17

- INVOKE and SET statements
- Object properties
- In-line method calls
- REPOSITORY paragraphs
- Class , object and method definitions
- Type definitions

Before Executing the Application

Check the definitions of the Excel file names in the file:

R_CONST.CBL

used by most of the programs. (To edit this file expand the project tree down to a list of Copy Library Files for one of the programs, e.g. MAIN, and double click on the R_CONST.CBL file name.)

The names are defined at the head of the file as the constants: CNFROOM-FILENAME and RSV-FILE-NAME.

You may need to change the settings of these constants if you did not install to the default folder structure.

Sample17 frequently writes to the Excel files so if you want to preserve the original or other version of these files, please take a backup before running Sample17.

Attention

The code is written to discard out-of-date data. When objects are restored from the Excel file, any objects with dates earlier than the current date are discarded.

Likewise, when objects are preserved, any objects that are out-of-date are not written to the Excel file.

Building the Application

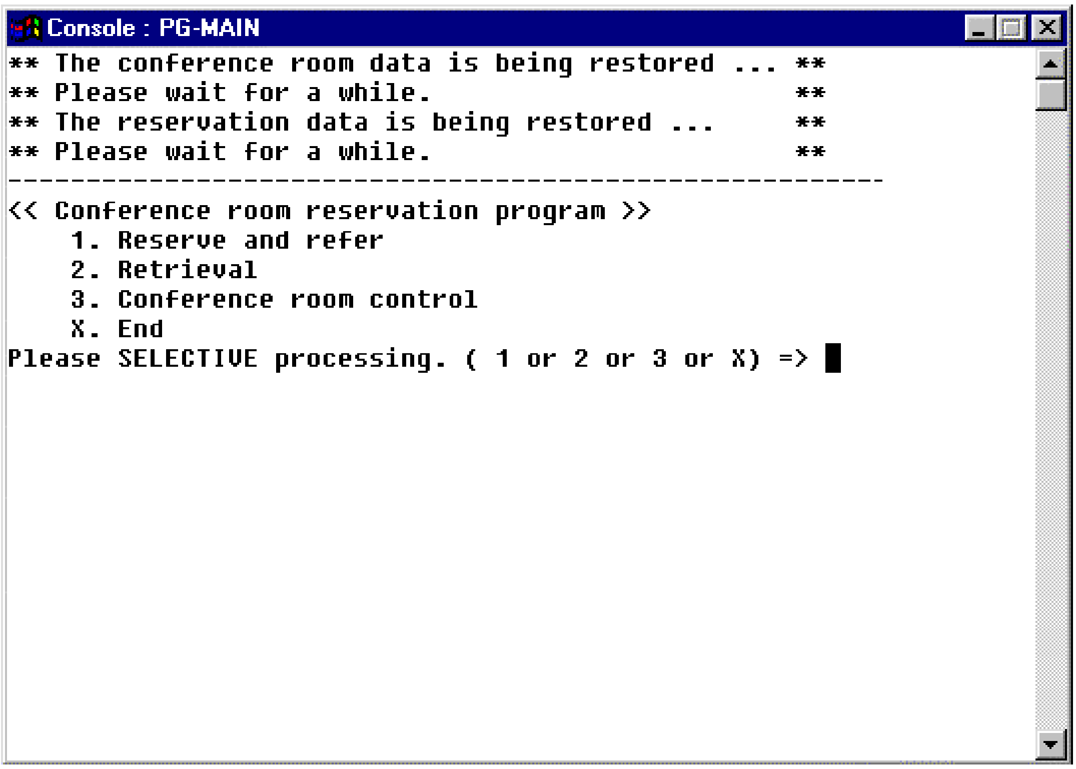
Project Manager's build function is used to make the executable program.

1. Start the Project Manager.
2. Open project file SAMPLE17.PRJ.
3. Select Build from the Project menu.

Executing the Application

To execute the program select Execute from the Project menu. (No run-time environment variables need to be set.)

When the program starts, it reloads the information from Excel, then displays a list of options:



```
** The conference room data is being restored ... **
** Please wait for a while.                        **
** The reservation data is being restored ...      **
** Please wait for a while.                        **
--------------------------------------------------------
<< Conference room reservation program >>
    1. Reserve and refer
    2. Retrieval
    3. Conference room control
    X. End
Please SELECTIVE processing. ( 1 or 2 or 3 or X) =>
```

Guidelines for Operating the Application

The application has a simple, menu-driven interface, controlled by the MAIN.COB program.

The main menu lets you select functions by entering a number from 1 to 3 or X to exit. The functions are:

1. Reserve and refer

   this provides the ability to reserve conference rooms, see bookings of conference rooms (on a given date), and cancel bookings.

2. Retrieval

   The retrieval function returns all conference room bookings for a given name.

3. Conference room control

   The conference room control function lets you update, add and delete conference room details.

Once you have entered the primary function you are prompted to enter data an item at a time. Sometimes the interface introduces a set of information to be entered, such as "Enter the conference room name, and time at which it should be booked" and the prompts for the first piece of information (the conference room name). Just enter the first piece of information and you will be prompted for the other pieces in the set.

You'll be asked for the following information for the different functions:

Reserve and refer

An 8-digit date (equal to or later than today's date)

Conference room name (4 characters)

Time (AM or PM)

If there is no booking for that room, date and time, you are prompted for:

Name (10 characters)

Extension (9 digits)

Department (10 characters)

and are returned a reservation number (required if you later want to cancel the reservation).

If there is a booking for the room, the booking details are displayed and you have the option to cancel the room. If you choose to cancel the booking you are prompted for the 4 digit reservation number.

Retrieval

You are prompted to enter:

Subscriber name (10 digits) - the name entered when booking the room.

Information corresponding to the name is displayed.

NOTE: The program is configured so that it only displays five reservations. To increase this change the following item in the R_CONST.CBL file.

RSV-MAX IS 5

Conference room control

You are prompted to enter:

Conference room name (4 characters)

If the conference room exists, details for that conference room are listed. If it doesn't exist you are prompted to enter details to register the conference room. The conference room details are:

Capacity (2 digits)

Extension (9 digits)

Type (N for general, T for video)

When changes are completed the appropriate updates are made to the Excel file to reflect the changes.

End

The program saves reservation information objects in the Excel file and terminates.

A.19 Sample 18: Advanced Object-Oriented Programming

Sample 18 uses all object-oriented specific functions including encapsulation, inheritance and polymorphism.

This program, which deals with two or more employee objects, uses the Dict class created in Sample 16, “Collection Class (Class Library).”

Overview

The application shows how an OO system can be set up to manage employee information. It implements, in a basic manner, the functions of registering, deleting, and modifying employee details, calculating salaries and printing a directory of employee information.

The employees are classified into staff members and managers. Different data is maintained and the pay calculation method differs between staff members and managers. The elements of the application are:

- An employee class that defines the attributes common to all employees (AllMember-class)
- A staff member class that defines the attributes particular to staff employees (Member-Class)
- A manager class that defines the attributes particular to managers (Manager-Class)

The staff member and manager classes inherit the employee class.

The addresses portion of the employee information is managed by an independent class (Address-Class).

Address objects are referenced from employee objects (and their subclasses).

The employee object defines the employee data to be managed by the supervisor.

Sample 18 use the dictionary class (Dict) created in Sample 16, “Collection Class (Class Library)”. It registers an employee object with the dictionary using the employee number as the key. The dictionary class allows easy repeat processing of two or more employee objects, and easy retrieval of an employee object.

Programs and Files in Sample 18

- MAIN.COB (COBOL source program)
- ALLMEM.COB (COBOL source program)
- MEMBER.COB (COBOL source program)
- MANAGER.COB (COBOL source program)
- ADDRESS.COB (COBOL source program)
- BONU_MAN.COB (COBOL source program)
- BONU_MEM.COB (COBOL source program)
- SALA_MAN.COB (COBOL source program)
- SALA_MEM.COB (COBOL source program)
- DICT.REP (Repository file)
- SAMPLE18.PRJ (Project file)
- SAMPLE18.CBI (Compiler option file)
- COBOL85.CBR (Environment variable initialization file)
- COLLECT.DLL (DLL file)
- COLLECT.LIB (Import library)

COBOL Functions used in Sample 18

- Object-oriented programming functions
  - Class definition (Encapsulation)
  - Inheritance

    - Object generation
    - Method calling
    - Polymorphism
- Screen operation function
- Project management

Object-Oriented Syntax used in Sample 18

- INVOKE and SET statements
- Object properties
- In-line method calls
- REPOSITORY paragraphs
- Class, object and method definitions

Building the Application

Project Manager's build function is used to make the executable program.

1. Start the Project Manager.
2. Open project file SAMPLE18.PRJ.
3. Select Build from the Project menu.

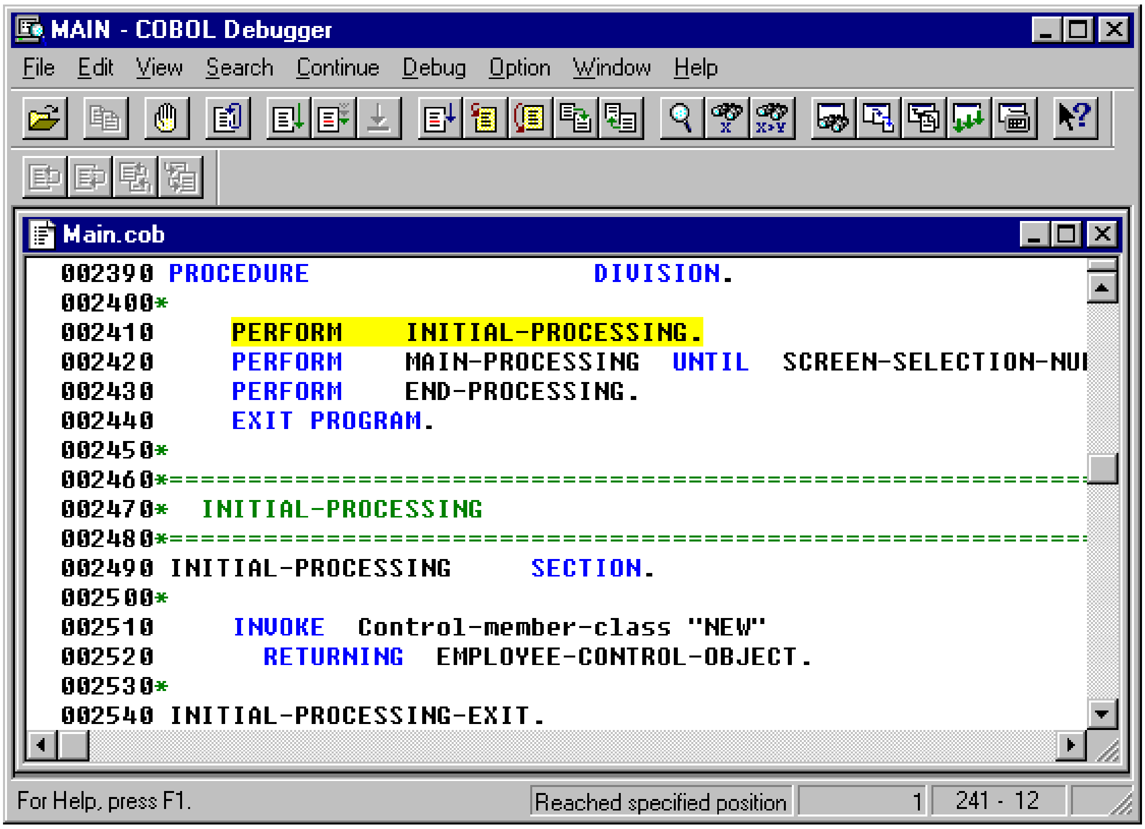
Debugging the Application

The project file is supplied with the debugging option set. You can execute the program under the control of the COBOL Debugger by:

1. Selecting Debug from the Project menu.

2. Clicking on the OK button in the Start Debugging dialog box.

   The COBOL Debugger loads the code as shown in the following figure.



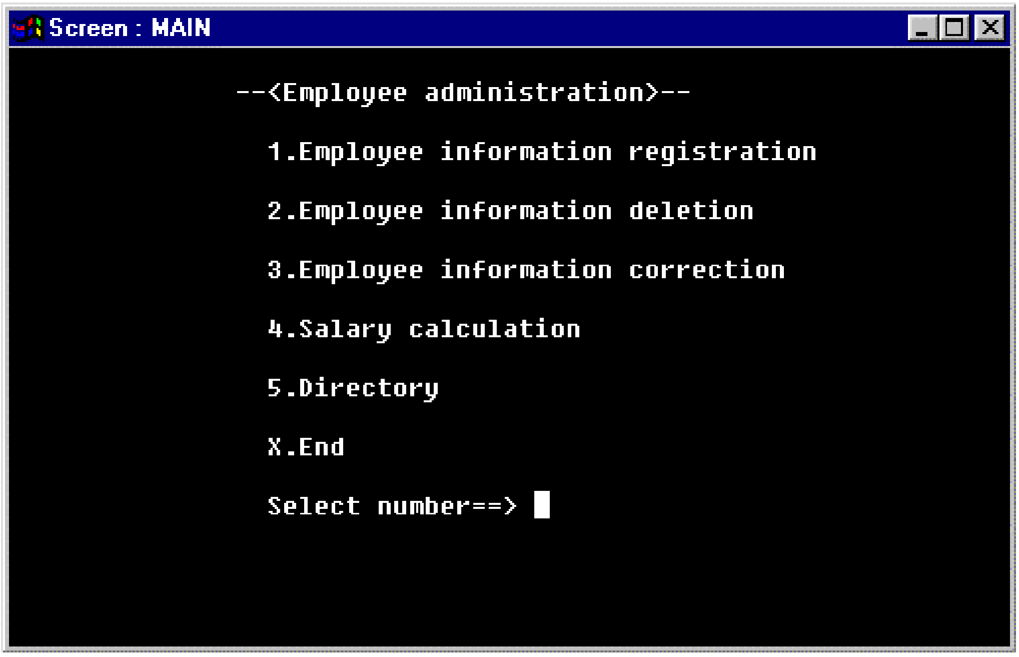
```
002390 PROCEDURE                    DIVISION.
002400*
002410      PERFORM     INITIAL-PROCESSING.
002420      PERFORM     MAIN-PROCESSING  UNTIL   SCREEN-SELECTION-NUI
002430      PERFORM     END-PROCESSING.
002440      EXIT PROGRAM.
002450*
002460*==================================================================
002470*   INITIAL-PROCESSING
002480*==================================================================
002490 INITIAL-PROCESSING      SECTION.
002500*
002510      INVOKE   Control-member-class "NEW"
002520        RETURNING   EMPLOYEE-CONTROL-OBJECT.
002530*
002540 INITIAL-PROCESSING-EXIT.
```

For Help, press F1. | Reached specified position | 1 | 241 - 12

Executing the Application

To run the application, select Execute from the Project menu.

The main application window is displayed:



```
              --<Employee administration>--

                1.Employee information registration

                2.Employee information deletion

                3.Employee information correction

                4.Salary calculation

                5.Directory

                X.End

                Select number==> █
```

To operate the program, enter the number of the desired process into “Select number” and press ENTER.

Note that this sample does not implement object persistence so data is lost when the application terminates. See Sample 19 for the same application with persistence added.

Registering Employee Information

Enter the employee information including:

- Employee number (four-digit numeric character string);
- Name (Character string of eight or fewer characters);
- Address consisting of a (Japanese style) postal code (7 digits) and a character string of 20 or fewer characters
- Date the employee joined the company (in the format of YY.MM.DD - for this program there is no date processing so the format is not significant)
- Status code (1 for a manager, and 2 for a staff member); and
- Basic pay (numeric character string of eight or fewer digits)

For a manager, type in the special allowances (numeric character string of six or fewer digits). For a general staff member, type in the overtime (numeric character string of five or fewer digits).

Press the F3 key to register the data.

To terminate processing of employee registration information, press the F2 key.

Deleting Employee Information

To delete employee information, type in the employee number (four-digit numeric character string), and press the F3 key.

The employee data is displayed and can be corrected.

Calculating Employee Pay

When you select the “Salary calculation” function, the pay of all employees is calculated.

Entering an employee number and pressing the F3 key displays the pay information for that employee.

To terminate processing of the salary calculation function, press the F2 key.

Address list

An employee address list can be printed. Select Manager (1) or General employee (2), and then press the F3 key.

End

Terminates processing.

A.20 Sample 19: Object Persistence (Indexed File)

Sample 19 demonstrates object persistence based on the program created in Sample 18, “Advanced Object-Oriented Programming.” In Sample 18, all objects are created in memory and thus upon termination of the program, all objects are lost.

In an actual system, however, data must be preserved when the program terminates. For an object-oriented system, in which the data is held in objects, the objects must continue to exist whether or not the application is executing. These objects are known as persistent objects.

In general, a persistent object can be implemented by relating the object to databases or files. In this example, persistent objects are rolled out by relating each object to an index file record.

For further details on object perpetuation, refer to the section “Making Objects Persistent” in the “NetCOBOL User’s Guide”.

Overview

The function of this sample is identical to that of Sample 18 except that the employee and address objects are stored in an index file, and persistence functions are added.

Persistence is implemented by adding the following methods to the employee and address objects.

- Store-Method (Object method)

  Stores an object in the file.

- RetAt-Method (Factory method)

  Loads an object from the file using the employee number as the key.

- Remove-Method (Factory method)

  Deletes an object from the file using the employee number as the key.

Actually, the object saving and retrieving is implemented by using the employee master class and the address master class. From the object user (the MAIN program in this example), however, the process is viewed as if the employee object assumed all the tasks.

Programs and Files in Sample 19

- MAIN.COB (COBOL source program)
- ALLMEM.COB (COBOL source program)
- MEMBER.COB (COBOL source program)
- MANAGER.COB (COBOL source program)
- ADDRESS.COB (COBOL source program)
- SET.COB (COBOL source program)
- STORE.COB (COBOL source program)
- ALLMEM_M.COB (COBOL source program)
- ALLMEMMF (Data file)
- BONU_MAN.COB (COBOL source program)
- BONU_MEM.COB (COBOL source program)
- MEM_SET.COB (COBOL source program)
- MEM_STOR.COB (COBOL source program)
- MAN_SET.COB (COBOL source program)
- MAN_STOR.COB (COBOL source program)
- ADDR_M.COB (COBOL source program)
- ADDR_MF (Data file)
- SALA_MAN.COB (COBOL source program)
- SALA_MEM.COB (COBOL source program)
- SAMPLE19.PRJ (Project file)
- SAMPLE19.CBI (Compiler option file)
- COBOL85.CBR (Environment variable initialization file)

COBOL Functions used in Sample 19

- Object-oriented programming functions
- Indexed file functions

- Project management

Object-Oriented Syntax used in Sample 19

- INVOKE and SET statements
- Object properties
- In-line method calls
- REPOSITORY paragraphs
- Class, object, factory and method definitions

Building the Application

Project Manager's build function is used to make the executable program.

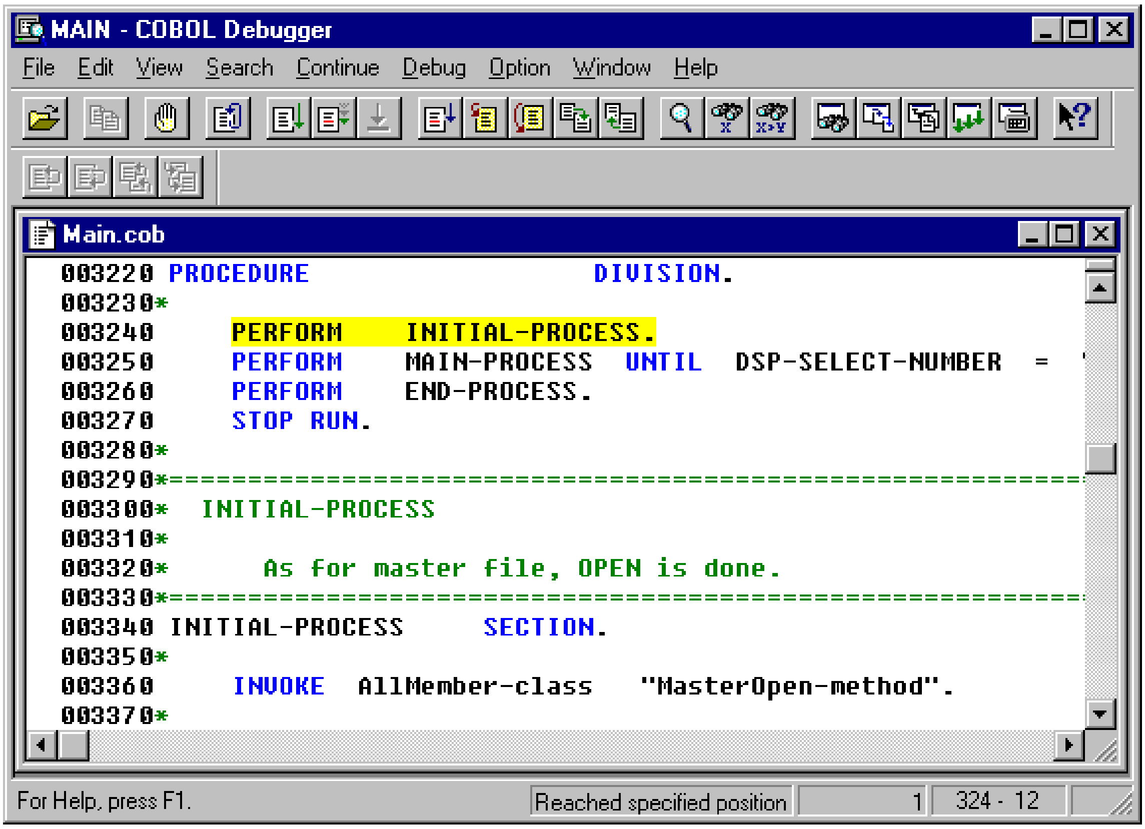
1. Start the Project Manager.
2. Open project file SAMPLE19.PRJ.
3. Select Build from the Project menu.

Debugging the Application

The project file is supplied with the debugging option set. You can execute the program under the control of the COBOL Debugger by:

1. Selecting Debug from the Project menu.
2. Clicking on the OK button in the Start Debugging dialog box.

   The COBOL Debugger loads the code as shown in the following figure.



```
003220 PROCEDURE                    DIVISION.
003230*
003240      PERFORM     INITIAL-PROCESS.
003250      PERFORM     MAIN-PROCESS  UNTIL   DSP-SELECT-NUMBER   =  '
003260      PERFORM     END-PROCESS.
003270      STOP RUN.
003280*
003290*=========================================================
003300*   INITIAL-PROCESS
003310*
003320*       As for master file, OPEN is done.
003330*=========================================================
003340 INITIAL-PROCESS      SECTION.
003350*
003360      INVOKE   AllMember-class    "MasterOpen-method".
003370*
```

For Help, press F1. | Reached specified position | 1 | 324 - 12

Executing the Application

To run the application, select Execute from the Project menu.

The main application window is displayed.

For details on running the program, see Sample 18, ”Advanced Object-oriented Programming”. The main difference is that data you enter is preserved from one execution of the application to the next to the next.

A.21 Sample 20: Object Persistence (Database)

Sample 20 demonstrates a method of making a permanent object and is based on the program created in "Sample 18 Object-Persistence". In sample 18, all objects were created in memory. Therefore, the objects disappear at program termination. However, some data has to remain in the system after the program ends. An object that retains data after the program ends is called a "Permanent Object".

In the previous program, such data was stored in a file or a database. To create a permanent object one must associate the permanent object with data in file or a database. In this example, the permanent object has been achieved by associating each object with a line in a data base table.

Refer to the "NetCOBOL User's Guide" for details on Object Persistence.

The database exists on the server, and is accessed from the client side.

The database is accessed via the ODBC driver. For information on setting up and using ODBC databases, please refer to the Chapter 19, "Database (SQL)" of the "NetCOBOL User's Guide".

The following products are necessary to execute this program.

- 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 management of employee information (addition, deletion, and modification), the salary calculations, and address are printed as was done in sample 18.

In addition to the functions above, in this example the employee object and the address object stored in the database, and a permanent object function is added.

These functions have been created in the employee and the address objects by adding the following methods.

- Store-Method (object method)

  : The object is stored in a database.

- RetAt-Method (object method)

  : The object is read from the database using the employee number as the key.

- RemoveAt-Method (object method)

  : The object is removed from the database using the employee number as the key.

- Update-Method (object method)

  : The object stored in the database is updated.

Available Programs

- MAIN.COB (COBOL source program)
- ALLMEM.COB (COBOL source program)
- MEMBER.COB (COBOL source program)

- MANAGER.COB (COBOL source program)
- ADDRESS.COB (COBOL source program)
- SET.COB (COBOL source program)
- ALLMEM_M.COB (COBOL source program)
- BONU_MAN.COB (COBOL source program)
- BONU_MEM.COB (COBOL source program)
- MEM_SET.COB (COBOL source program)
- MEM_STOR.COB (COBOL source program)
- MAN_SET.COB (COBOL source program)
- MAN_STOR.COB (COBOL source program)
- ADDR_M.COB (COBOL source program)
- SALA_MAN.COB (COBOL source program)
- SALA_MEM.COB (COBOL source program)
- SAMPLE20.PRJ (project file)
- SAMPLE20.CBI (compilation option file)
- SAMPLE20.MDB (Microsoft Access Database file)
- COBOL85.CBR (initialization file for execution)
- SAMPLE20.KBD (key definition file)
- SAMPLE20.INF (ODBC information file)

Applicable COBOL Functions

- Object oriented programming function
- Remote database access (ODBC) function
- Project manager function

Available Object-oriented Statements/Paragraphs/Definitions

INVOKE and SET statements

Object property

In-line invocation of a method

Repository paragraph

Class definition, object definition, factory definition, and method definition

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 proceed with this exercise, you must have already installed 32-bit ODBC support on your Windows 2000, XP, Windows Server 2003, or Windows Server 2008 system, along with the Microsoft Access Driver.

Verifying Whether ODBC is Installed

Please refer to "Sample11".

Defining an ODBC Source for Your Application

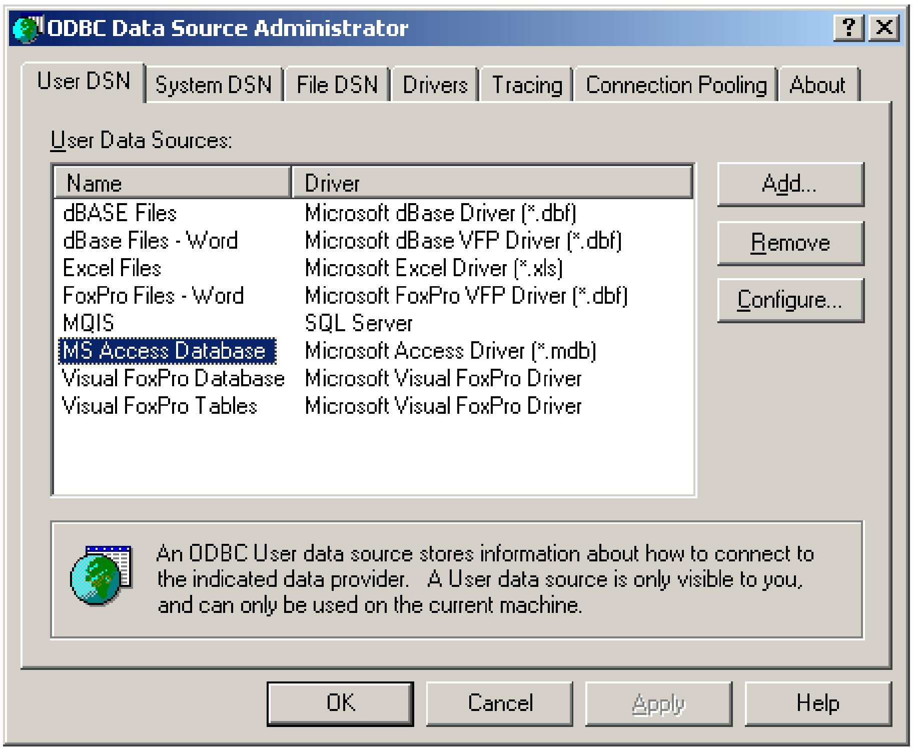
The first development task that needs to be accomplished is to define an ODBC Source for the application.

An ODBC Source can be thought of as simply specifying to the ODBC manager a symbolic name that you wish to use in your application to describe an ODBC connection to a specific ODBC database driver and database.

The following figures may or may not match what you see on your particular operating system, but you should be able to perform the following tasks by using the instructions provided.

Bring up the Administrative Tools and double click on the Data Sources (ODBC) icon.

The main ODBC Data Source Administrator window is displayed.



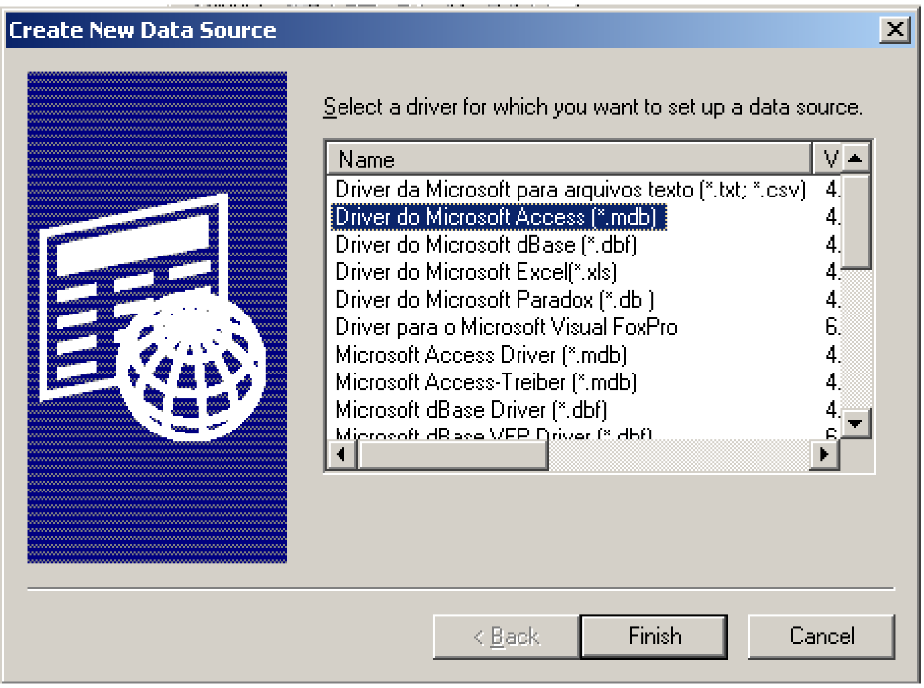
User Data Sources:

Name	Driver
dBASE Files	Microsoft dBase Driver (*.dbf)
dBase Files - Word	Microsoft dBase VFP Driver (*.dbf)
Excel Files	Microsoft Excel Driver (*.xls)
FoxPro Files - Word	Microsoft FoxPro VFP Driver (*.dbf)
MQIS	SQL Server
MS Access Database	Microsoft Access Driver (*.mdb)
Visual FoxPro Database	Microsoft Visual FoxPro Driver
Visual FoxPro Tables	Microsoft Visual FoxPro Driver

An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you, and can only be used on the current machine.

You will add a new User DSN. Select the User DSN tab, and click on the Add button.

ODBC Data Source Administrator displays the “Create New Data Source” dialog box.

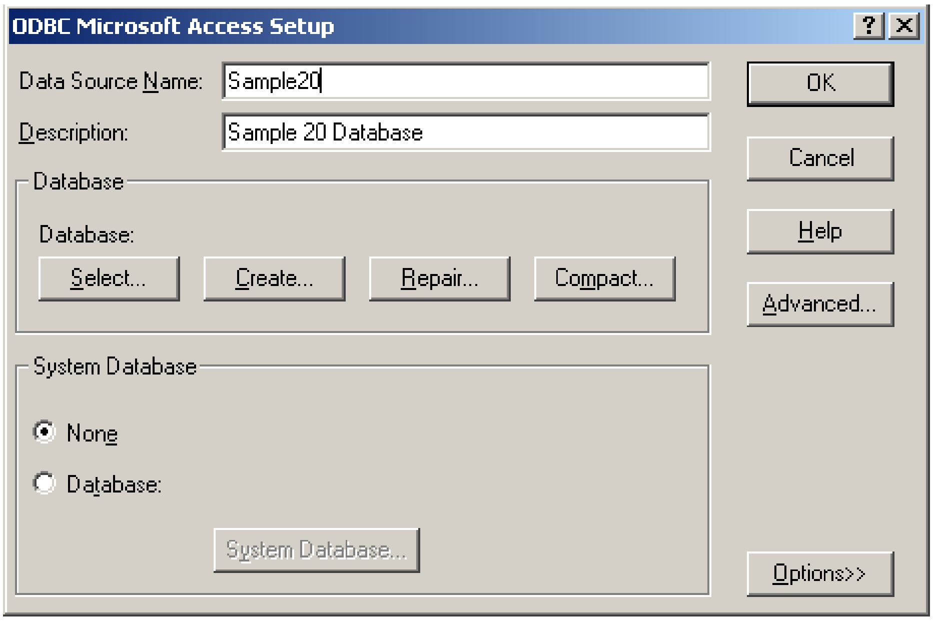


Select a driver for which you want to set up a data source.

Name	V
Driver da Microsoft para arquivos texto (*.txt; *.csv)	4.
Driver do Microsoft Access (*.mdb)	4.
Driver do Microsoft dBase (*.dbf)	4.
Driver do Microsoft Excel(*.xls)	4.
Driver do Microsoft Paradox (*.db)	4.
Driver para o Microsoft Visual FoxPro	6.
Microsoft Access Driver (*.mdb)	4.
Microsoft Access-Treiber (*.mdb)	4.
Microsoft dBase Driver (*.dbf)	4.
Microsoft dBase VFP Driver (*.dbf)	6.

Select the Microsoft Access Driver (*.mdb) and click on the Finish button.

The ODBC Microsoft Access Setup dialog box is displayed.



Data Source Name: Sample20

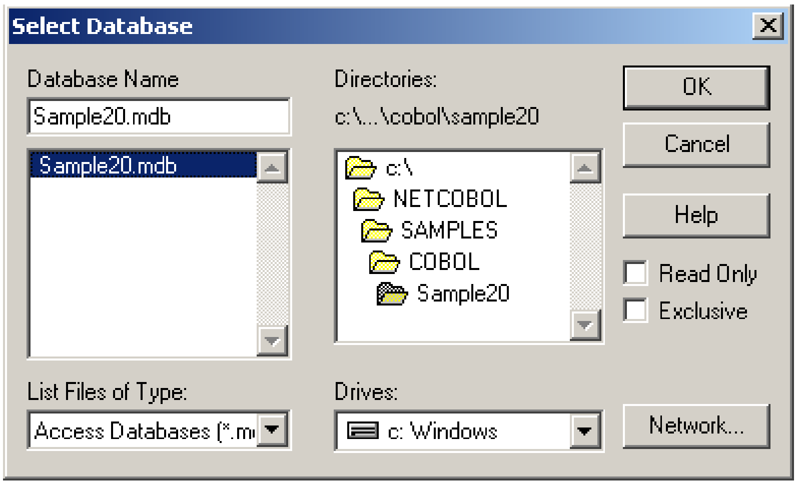
Description: Sample 20 Database

● None

○ Database:

Enter Sample20 in the Data Source Name field and Sample20 Database in the Description field as shown above.

You also need to select a file (database) to bind this driver to. Click on the Select button in the ODBC Microsoft Access Setup dialog box. The Select Database dialog box is displayed. Navigate through the directories and find the "Sample20.mdb" database file:



Database Name: Sample20.mdb

Sample20.mdb

Directories: c:\...\cobol\sample20

c:\
NETCOBOL
SAMPLES
COBOL
Sample20

List Files of Type: Access Databases (*.m

Drives: c: Windows

Click on the OK button. This will take you back to the ODBC Microsoft Access Setup dialog box.

Click on the OK button. This will take you back to the main ODBC Source Administrator window.

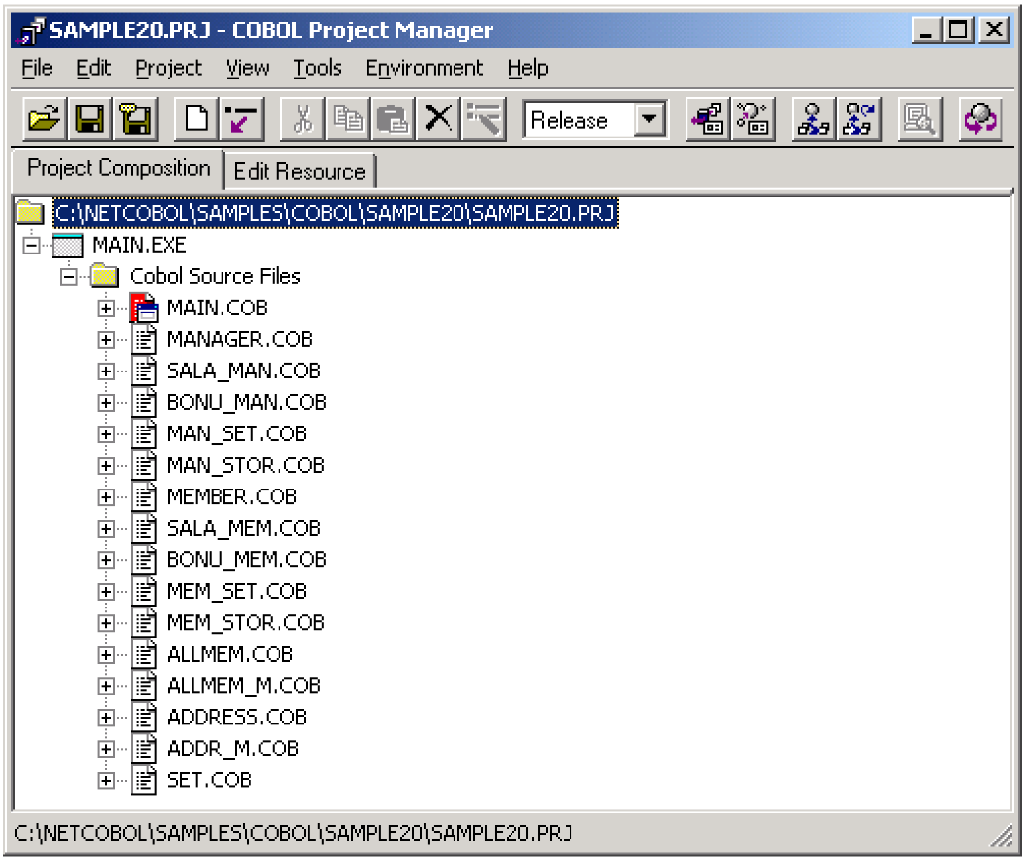
Note that "Sample20" has been added to the list in User DSN. Although you should not set any additional options for this exercise, you may wish to highlight “Sample20” and click on the Configure button.

This will give you an idea of some of the additional options that are available when developing these types of applications. Once you are back to the main ODBC Source Administrator window, click on the OK button. This will exit the ODBC configuration facility. You have now defined the new ODBC Source (Sample20) you are going to use with COBOL.

Building/Rebuilding the Program

Project manager's build function is used to create the executable program.

1. The project manager is started.



```
C:\NETCOBOL\SAMPLES\COBOL\SAMPLE20\SAMPLE20.PRJ
MAIN.EXE
    Cobol Source Files
        MAIN.COB
        MANAGER.COB
        SALA_MAN.COB
        BONU_MAN.COB
        MAN_SET.COB
        MAN_STOR.COB
        MEMBER.COB
        SALA_MEM.COB
        BONU_MEM.COB
        MEM_SET.COB
        MEM_STOR.COB
        ALLMEM.COB
        ALLMEM_M.COB
        ADDRESS.COB
        ADDR_M.COB
        SET.COB
```

C:\NETCOBOL\SAMPLES\COBOL\SAMPLE20\SAMPLE20.PRJ

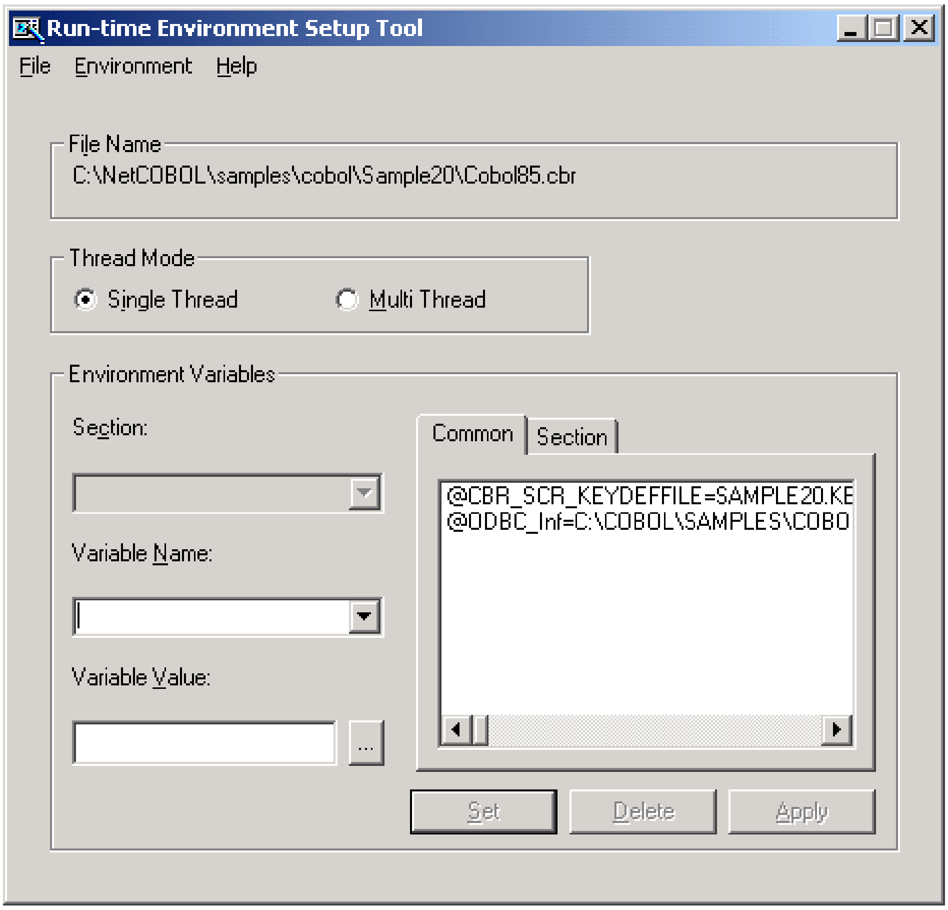
2. Project file "SAMPLE20.PRJ" is opened.

"Build" is selected from the "Project" menu.

Setup the Execution Environment Information

The initialization file for execution is updated with the Run-time Environment Setup Tool. The content is shown below.

1. The Run-time Environment Setup Tool is started.
2. The initialization file for execution is opened. (COBOL85.CBR)
3. A common tab is selected.
4. The ODBC information file name is specified for @ODBC_Inf. (SAMPLE20.INF)
5. The key definition file name is specified for @CBR_SCR_KEYDEFFILE. (SAMPLE20.KBD)



File Name
C:\NetCOBOL\samples\cobol\Sample20\Cobol85.cbr

● Single Thread ○ Multi Thread

@CBR_SCR_KEYDEFFILE=SAMPLE20.KE
@ODBC_Inf=C:\COBOL\SAMPLES\COBO

6. Click "Apply" button, and the content is saved in the execution initialization file.
7. Click "Exit" button in file menu, and the Run-time Environment Setup Tool ends.

Executing the Program

To execute the program, "Execute" is selected from the project menu.

Execution Procedure

Please refer to "Sample 18 Object Persistence" for execution procedures.

However, please note the following points.

- When the name and the address are input, National characters cannot be used.

  Please use upper and lower case letters or numeric characters.

- An error report is generated when an error occurs when accessing the database (such as connecting or getting a record).

A.22 Sample 21: Multi-Thread Programming

Sample 21 demonstrates how COBOL can support multithreading programming.

In sample 21, a resource (file data) is shared between threads, and the synchronous control between threads is shown by using the multi-thread programming functions of NetCOBOL.

Please refer to Chapter 22, "Multithread Programs" of the "NetCOBOL User's Guide" for details of the multi-thread programming functions of NetCOBOL.

Sample 21 is a Web application. Web applications typically can benefit from multithreading applications.

Please refer to the "NetCOBOL Web Guide" and the "NetCOBOL ISAPI Subroutines User's Guide" for details of the functions available for programming the Web.

To execute this sample, the following products are needed on the client side and the server side.

Client side

- World Wide Web browser
  - Microsoft(R) Internet Explorer 4.0 or more
  - Netscape Navigator(TM) 4.0 or more

Server side

- One of the following products:
  - Microsoft® Windows® 2000 Server operating system
  - Microsoft® Windows® 2000 Advanced Server operating system
  - Microsoft® Windows Server® 2003 Standard Edition
  - Microsoft® Windows Server® 2003 Enterprise Edition
  - Microsoft® Windows Server® 2008 Standard Edition
  - Microsoft® Windows Server® 2008 Enterprise Edition
- Microsoft® Internet Information Server 5.0 or later

Overview

The sample program consists of the following three parts.

- Initialization processing

  The resource (file data) between threads is acquired, and is initialized.
- Authorization processing

  The authorization process is achieved by accessing the resource (file data) between threads.
- Termination processing

  The resource (file data) between threads is closed.

This sample demonstrates how to share resource data with Web functions, and how to synchronize the threads.

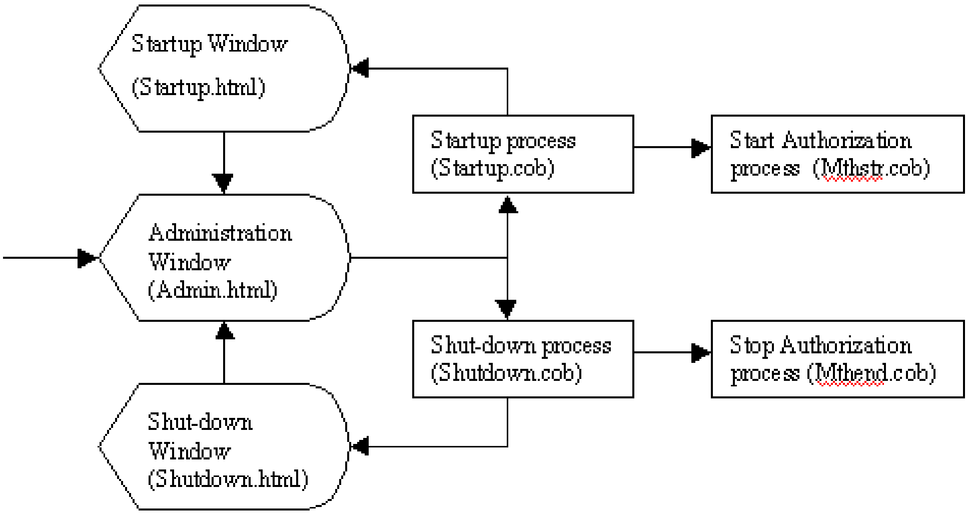
Available Programs

- Project files
  - ISAPIAPL.PRJ
  - MTHAPL.CBI

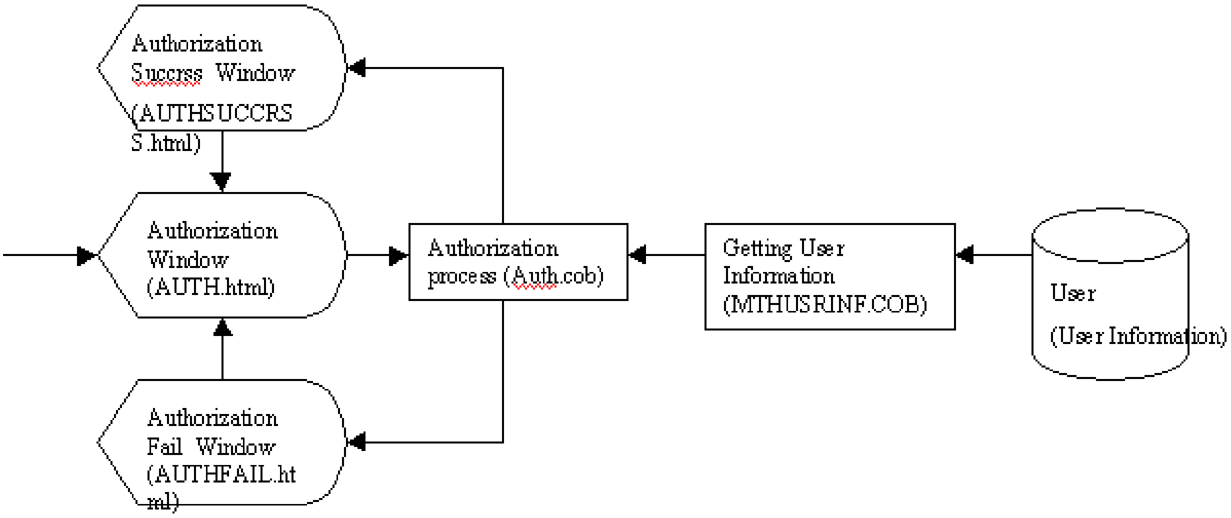
- Option files
  - ISAPIAPL.CBI
  - MTHAPL.CBI
- COBOL source files
  - AUTH.COB
  - ISAINIT.COB
  - ISATERM.COB
  - MTHEND.COB
  - MTHSTR.COB
  - MTHUSRINF.COB
  - SHUTDOWN.COB
  - STARTUP.COB
  - STUPINIT.COB
- Library text
  - User-Info.CBL
  - User-Lock.CBL
- Module definition files
  - AUTH.DEF
  - SHUTDOWN.DEF
  - STARTUP.DEF
- Data files
  - USERINFO
- Run-time Initialization file
  - COBOL85.CBR
- HTML files
  - ADMIN.HTML
  - AUTH.HTML
  - AUTHFAIL.HTML
  - AUTHSUCESS.HTML
  - NOTOPENED.HTML
  - OPENED.HTML
  - SHUTDOWN.HTML
  - STARTUP.HTML
  - SYSERROR.HTML
  - SYSTEMERROR.HTML

Process Flow

1. Business start and end



2. Authorization service



Applicable COBOL Functions

- Index file (reference)
- External data
- External file
- Data lock subroutine
- COBOL ISAPI subroutine

Applicable COBOL Statements

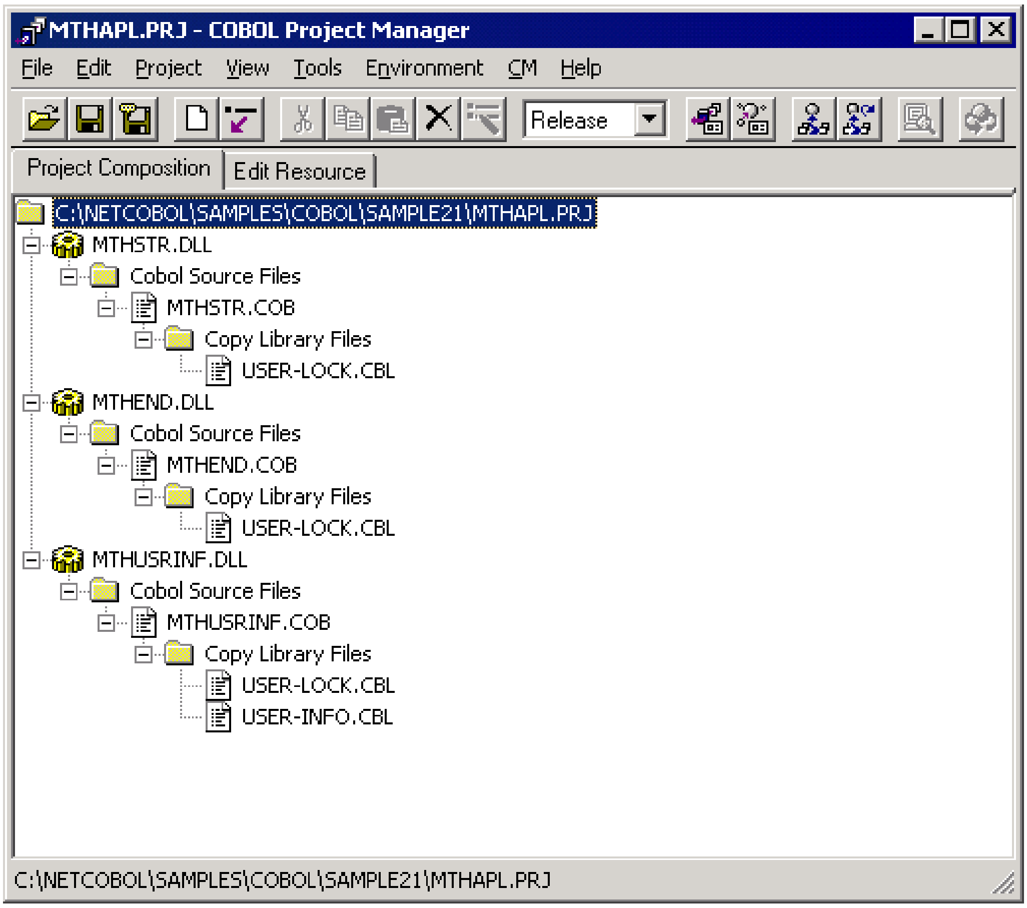
The CALL, CLOSE, EXIT, GO TO, IF, MOVE, OPEN, PERFORM, READ, and SET statements are used.

Building the Program

Project manager's Build function is used to create the executable program.

In the following screen snapshots, the sample program was installed to C:\NetCOBOL. Your installation folder may be different.

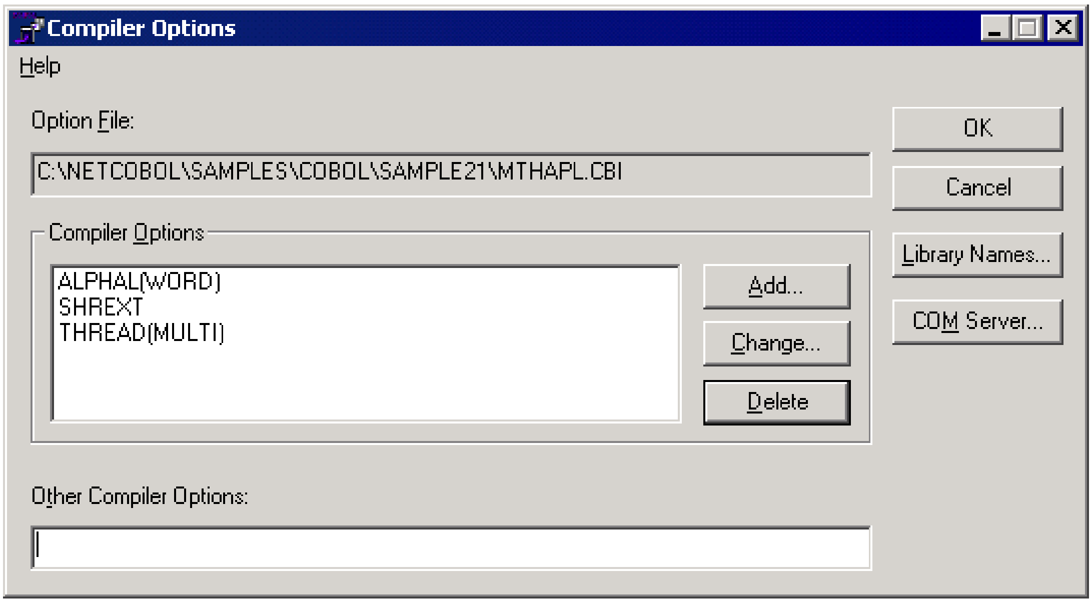
1. The project manager is started.
2. The project file "MTHAPL.PRJ" is opened.



```
C:\NETCOBOL\SAMPLES\COBOL\SAMPLE21\MTHAPL.PRJ
  MTHSTR.DLL
    Cobol Source Files
      MTHSTR.COB
        Copy Library Files
          USER-LOCK.CBL
  MTHEND.DLL
    Cobol Source Files
      MTHEND.COB
        Copy Library Files
          USER-LOCK.CBL
  MTHUSRINF.DLL
    Cobol Source Files
      MTHUSRINF.COB
        Copy Library Files
          USER-LOCK.CBL
          USER-INFO.CBL
C:\NETCOBOL\SAMPLES\COBOL\SAMPLE21\MTHAPL.PRJ
```

3. The project file is selected, and "Compiler options" is selected from "Project"-"Options" menu.

   The "Compiler options" dialog is displayed.



Option File:

C:\NETCOBOL\SAMPLES\COBOL\SAMPLE21\MTHAPL.CBI

Compiler Options

ALPHAL(WORD)
SHREXT
THREAD(MULTI)

Other Compiler Options:

4. Compiler option THREAD(MULTI), SHREXT and ALPHAL(WORD) are specified.

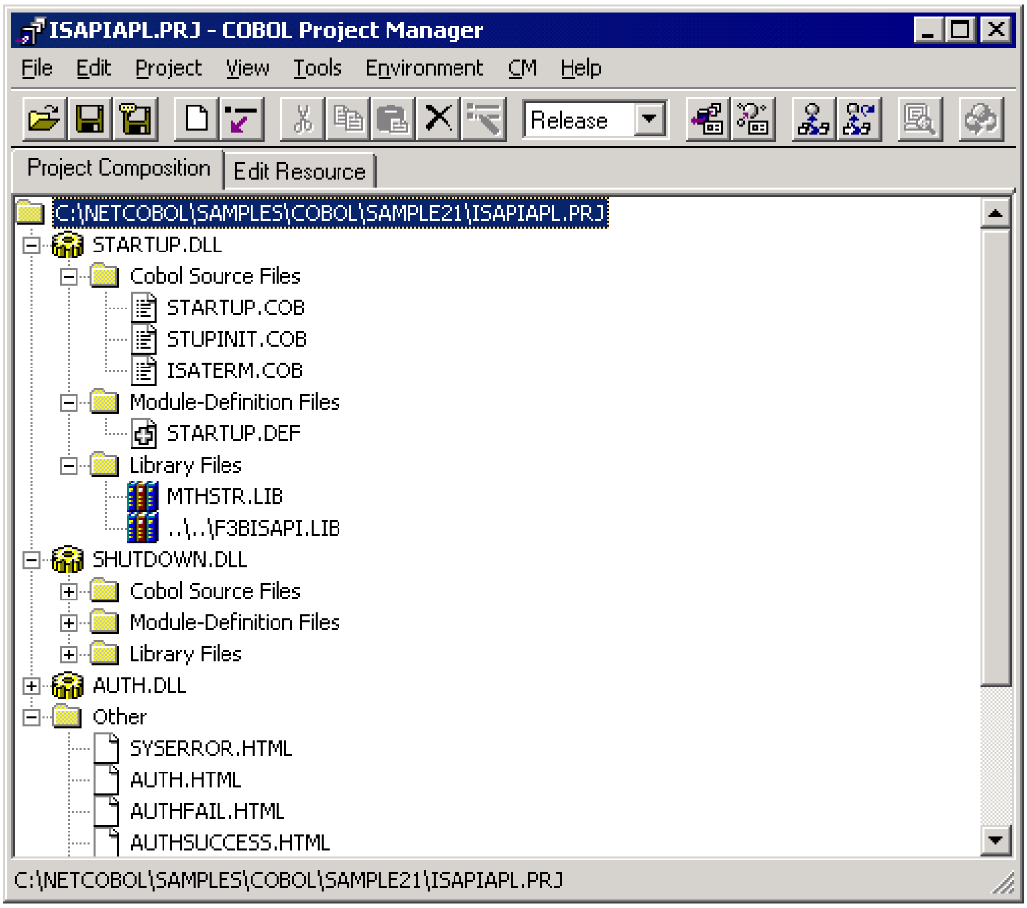
   After confirming the information, click the OK button.

   You are now returned to the Project Manager window.

5. "Build" is selected from Project Manager's "Project" menu.

   (Prior to executing the application, please confirm that all the DLL's in the application have been built correctly.)

6. The project file "ISAPIAPL.PRJ" is opened



ISAPIAPL.PRJ - COBOL Project Manager

```
C:\NETCOBOL\SAMPLES\COBOL\SAMPLE21\ISAPIAPL.PRJ
⊟ STARTUP.DLL
   ⊟ Cobol Source Files
      STARTUP.COB
      STUPINIT.COB
      ISATERM.COB
   ⊟ Module-Definition Files
      STARTUP.DEF
   ⊟ Library Files
      MTHSTR.LIB
      ..\..\F3BISAPI.LIB
⊟ SHUTDOWN.DLL
   ⊞ Cobol Source Files
   ⊞ Module-Definition Files
   ⊞ Library Files
⊞ AUTH.DLL
⊟ Other
   SYSERROR.HTML
   AUTH.HTML
   AUTHFAIL.HTML
   AUTHSUCCESS.HTML
```

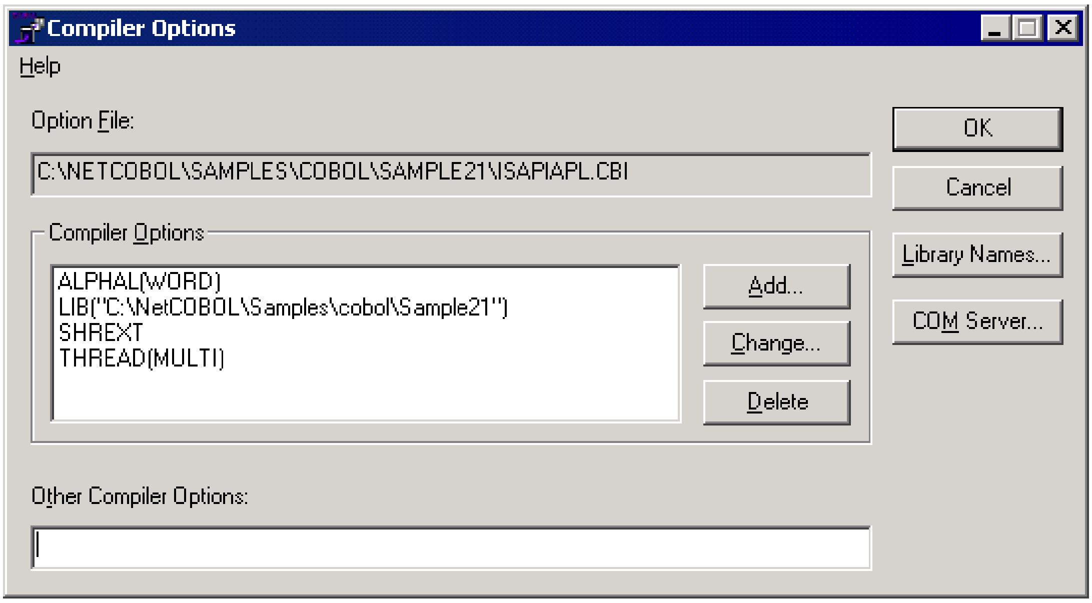
C:\NETCOBOL\SAMPLES\COBOL\SAMPLE21\ISAPIAPL.PRJ

7. The compiler option dialog is displayed as in step 3 above, showing that the compiler options THREAD(MULTI), SHREXT, and ALPHAL(WORD) are specified.

   Modify the folder name of the library file specified in the compiler option LIB.

   Click the OK button.

You are now returned to the Project Manager window.



Option File:

C:\NETCOBOL\SAMPLES\COBOL\SAMPLE21\ISAPIAPL.CBI

Compiler Options

```
ALPHAL(WORD)
LIB("C:\NetCOBOL\Samples\cobol\Sample21")
SHREXT
THREAD(MULTI)
```

Other Compiler Options:

8. "Build" is selected from Project Manager's "Project" menu.

Executing the Program

It is assumed that the domain-name and virtual directory name are registered in IIS (Internet Information Services) as "user" and "sample21" respectively.

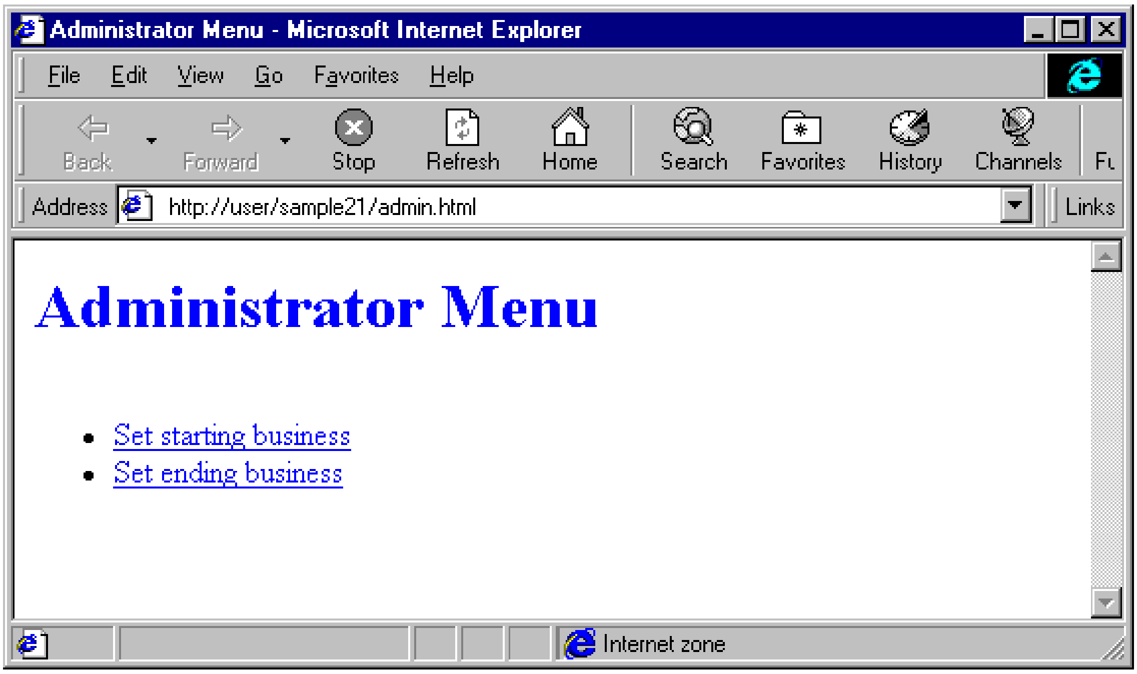
This example presumes that Microsoft Internet Explorer is being used.

1. The authentication service begins.

   The following information is set in URL.

```
Address | http://user/sample21/admin.html
```

   The administrator menu screen is displayed. Click on the “Set Starting Business” hyperlink.



Address http://user/sample21/admin.html

Administrator Menu

- Set starting business
- Set ending business

   When clicked, the authentication service begins. Please do so before starting the authentication service.

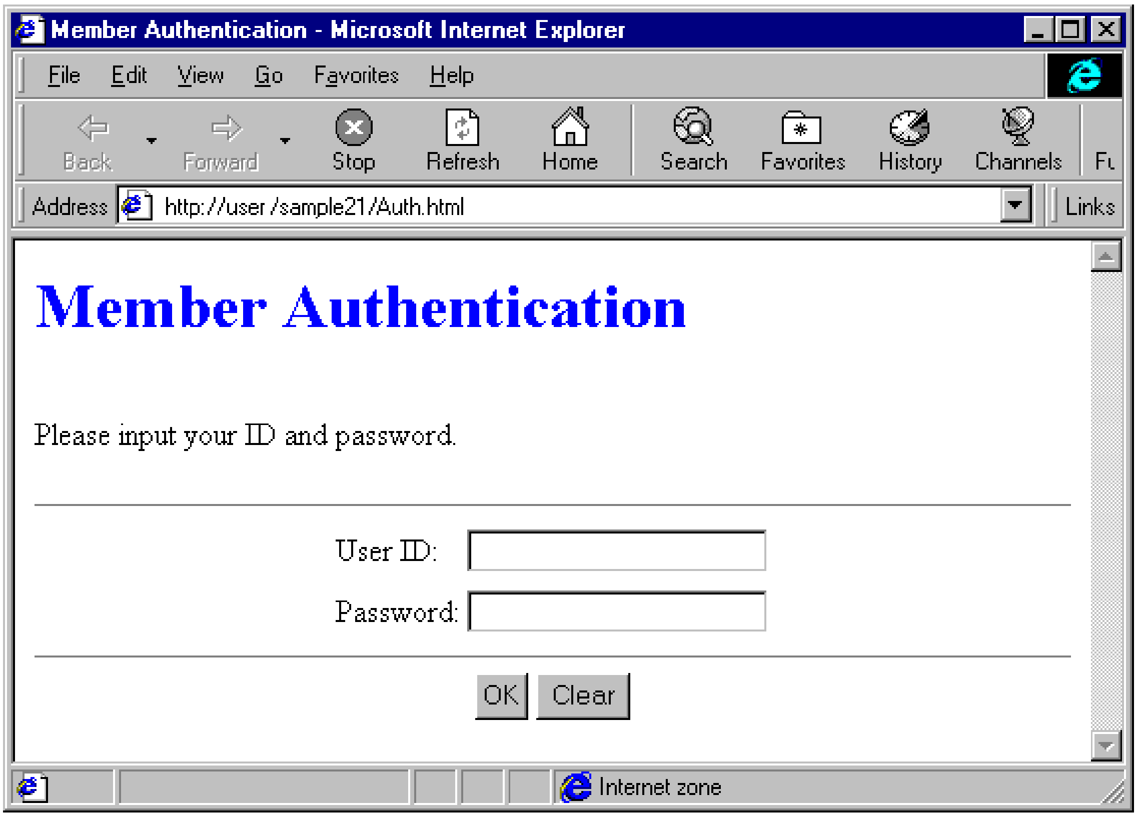
2. The authentication service is started.

   The following information is set in the URL and the "Execute" key is pushed.

```
Address | http://user/sample21/Auth.html
```

The authentication service screen is displayed. After the screen is displayed, input a User ID and password, then click the "OK" button.

Valid User ID's are USER0001 to USER0030. The password is the same as the User ID.



Member Authentication - Microsoft Internet Explorer

Address http://user /sample21/Auth.html

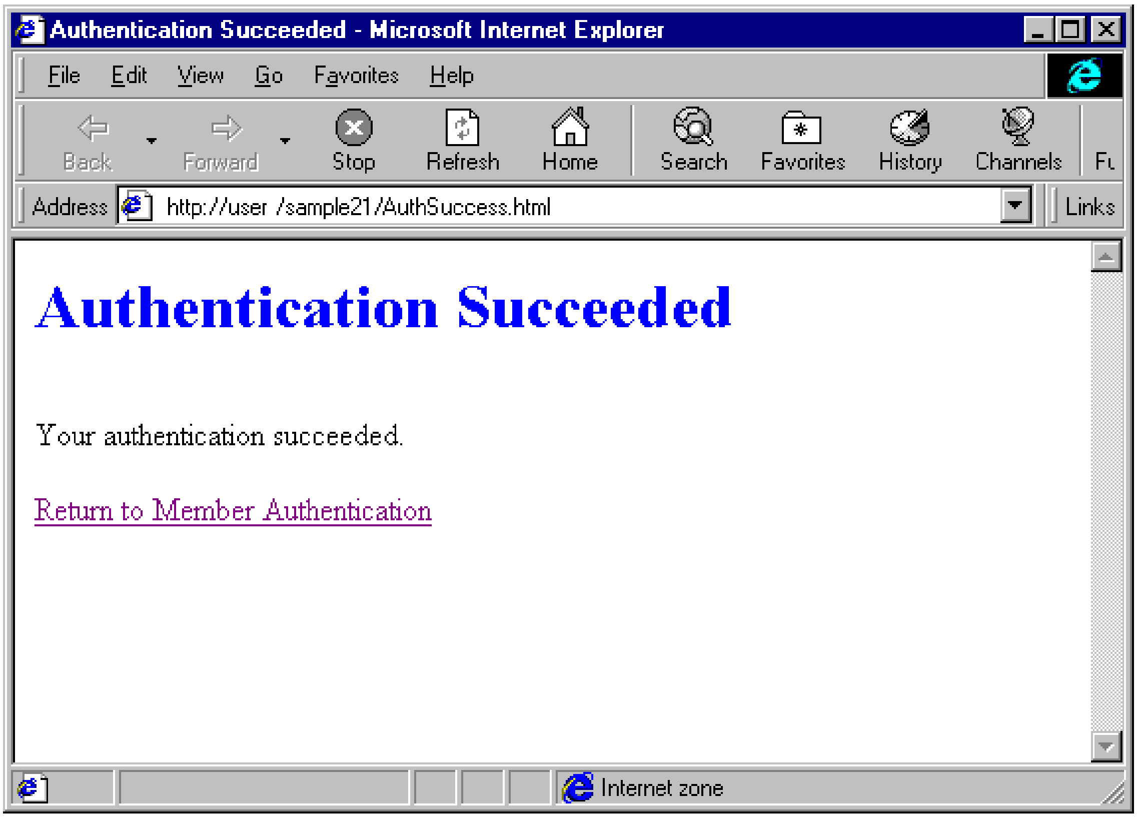
Member Authentication

Please input your ID and password.

User ID:

Password:

When the OK button is clicked, the authentication success screen is displayed.


Authentication Succeeded - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fu
Address
Links
Internet zone

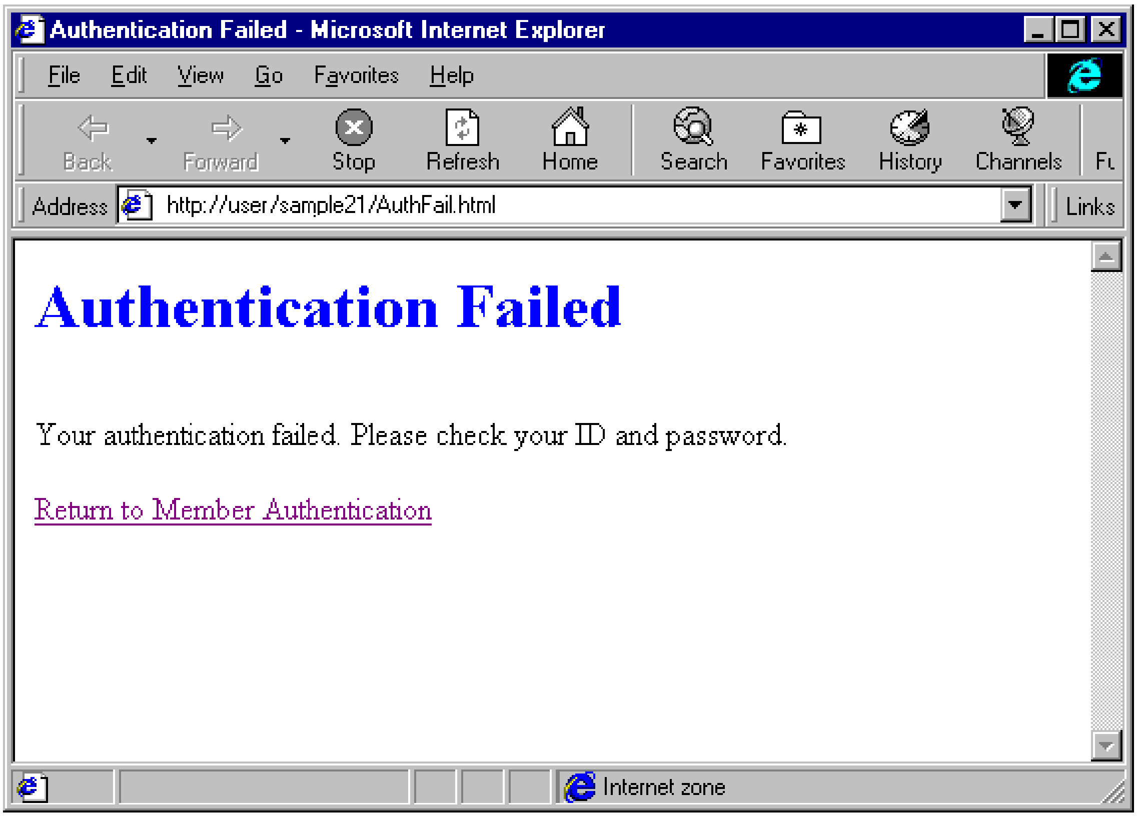
http://user /sample21/AuthSuccess.html

Authentication Succeeded

Your authentication succeeded.

Return to Member Authentication

If the User ID and/or password are invalid, an authentication failure screen is displayed.


Authentication Failed - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fu
Address
Links
Internet zone

http://user/sample21/AuthFail.html

Authentication Failed

Your authentication failed. Please check your ID and password.

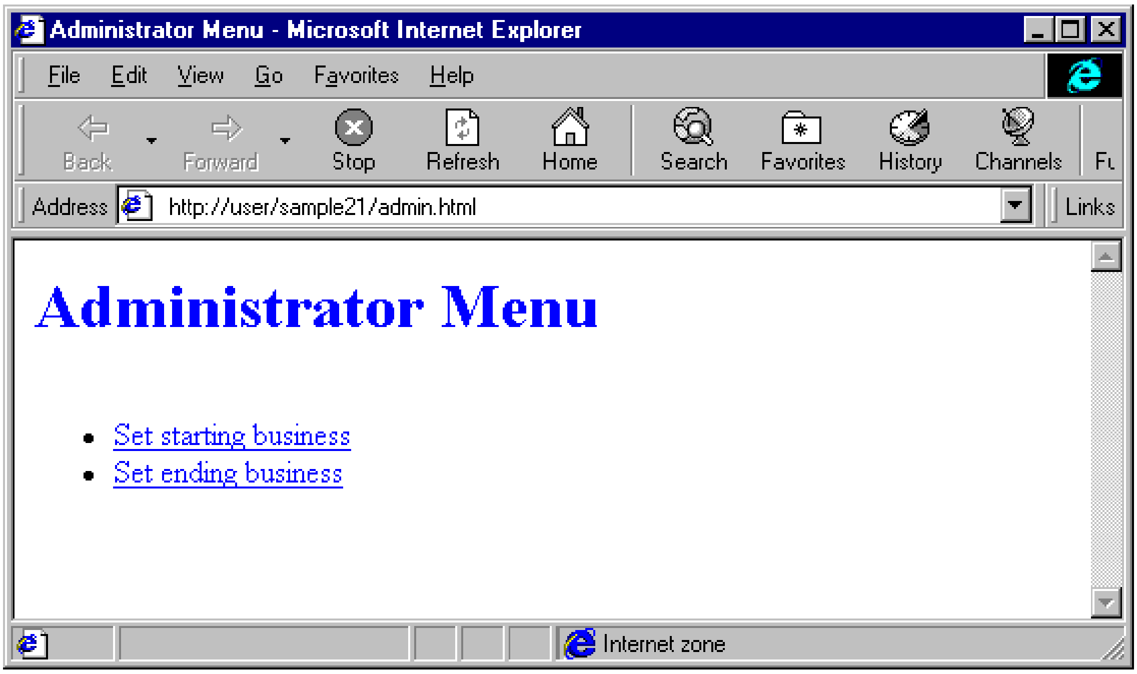
Return to Member Authentication

3. The authentication service is ended.

   Input following information in the URL and press the "Execute" key.

```
Address | http://user/sample21/admin.html
```

Because the administrator menu screen is displayed, "Set ending business" is clicked.



Administrator Menu - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fu
Address http://user/sample21/admin.html
Links
Administrator Menu
Set starting business
Set ending business
Internet zone

A.23 Sample 22: Multi-Thread Programming (Advanced usage)

In sample 22, the program in sample 21 is enhanced to demonstrate an online store.

The sample uses multi-thread programming functions as well as functions for programming the Web.

Please refer to "NetCOBOL Web Guide" and "NetCOBOL ISAPI Subroutines User's Guide" for details of the Web functions.

Please refer to Chapter 22 "Multithread Programs" of the "NetCOBOL User's Guide" for details of the multithreading programming functions.

This example also outputs an event log using the function “COB_REPORT_EVENT”.

This function is described in Appendix H of the “NetCOBOL User's Guide”.

To use this sample, the following products are needed on the client side and the server side.

Client side

- Microsoft® Internet Explorer 4.0 or more
- Netscape NavigatorTM 4.0 or more

Server side

- One of the following products:
  - Microsoft® Windows® 2000 Server operating system
  - Microsoft® Windows® 2000 Advanced Server operating system
  - Microsoft® Windows Server® 2003 Standard Edition
  - Microsoft® Windows Server® 2003 Enterprise Edition

    - Microsoft® Windows Server® 2008 Standard Edition
    - Microsoft® Windows Server® 2008 Enterprise Edition
- Microsoft® Internet Information Server 5.0 or later

Overview

The sample program consists of the following five parts.

- Begin processing

  The resource (file data) between threads is acquired and initialized.

- Authorization processing

  The authorization processing is achieved by referring to the resource (file data) between threads.

- Order confirmation processing

  The order confirmation processing is performed by referring to the resource (file data) between threads.

- Order issue processing

  The order issue processing is performed by referring to the resource (file data) between threads. Shared file update processing is done.

- Termination

  The resource (file data) between threads is closed.

Available Programs

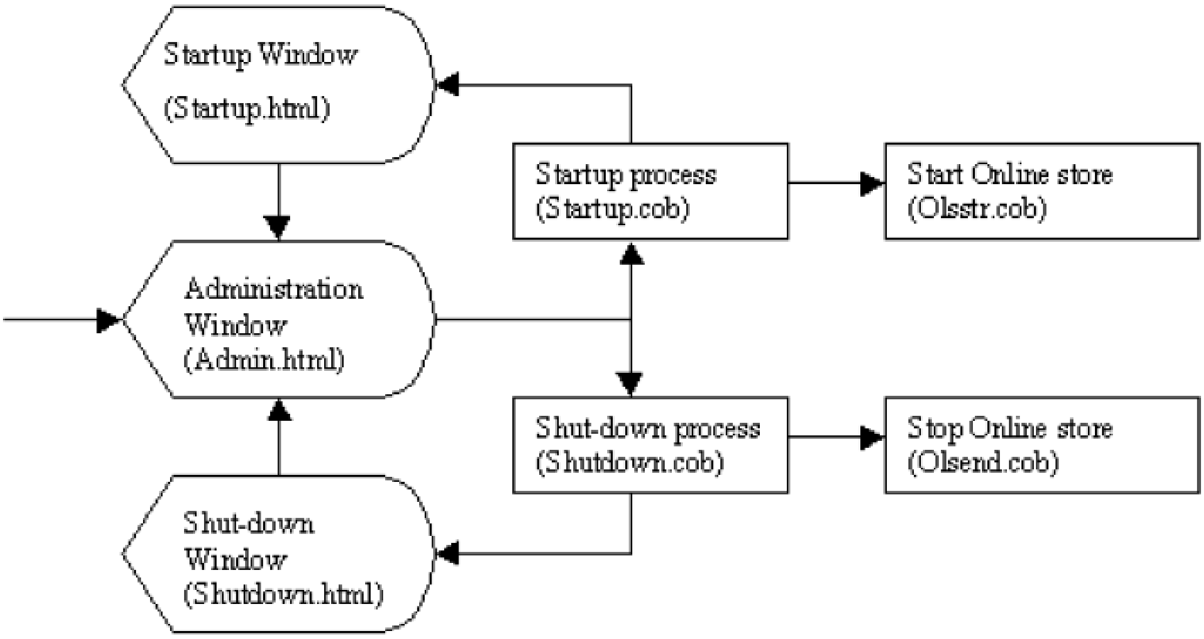
- Project files
    - ISAPIAPL.PRJ
    - OLSAPL.PRJ
- Option files
    - ISAPIAPL.CBI
    - OLSAPL.CBI
- COBOL source files
    - AUTH.COB
    - CONFIRM.COB
    - ENTRY.COB
    - ISAINIT.COB
    - ISATERM.COB
    - OLSEND.COB
    - OLSPRDGT.COB
    - OLSSTCGT.COB
    - OLSSTCODR.COB
    - OLSSTR.COB
    - OLSUSRINF.COB
    - SHUTDOWN.COB
    - STARTUP.COB
    - STUPINIT.COB

- Library text
    - Order-Info.CBL
    - Product-Info.CBL
    - Stock-Info.CBL
    - User-Info.CBL
    - User-Lock.CBL
    - User-Log.CBL
- Module definition files
    - AUTH.DEF
    - CONFIRM.DEF
    - ENTRY.DEF
    - SHUTDOWN.DEF
    - STARTUP.DEF
- Data files
    - PRODUCTINFO
    - STOCKINFO
    - USERINFO
- Run-time Initialization file
    - COBOL85.CBR
- HTML files
    - ADMIN.HTML
    - AUTH.HTML
    - AUTHFAIL.HTML
    - CATALOG.HTML
    - CONFIRMDETAILPARTS.HTML
    - CONFIRMHEAD.HTML
    - CONFIRMTAIL.HTML
    - ILLIGALACCESS.HTML
    - ILLIGALSYSTEM.HTML
    - NOTOPENED.HTML
    - OPENED.HTML
    - ORDERDETAILPARTS.HTML
    - ORDERRESULTHEAD.HTML
    - ORDERRESULTTAIL.HTML
    - SHOPPINGMENU.HTML
    - SHORTAGESTOCK.HTML
    - SHUTDOWN.HTML
    - STARTUP.HTML

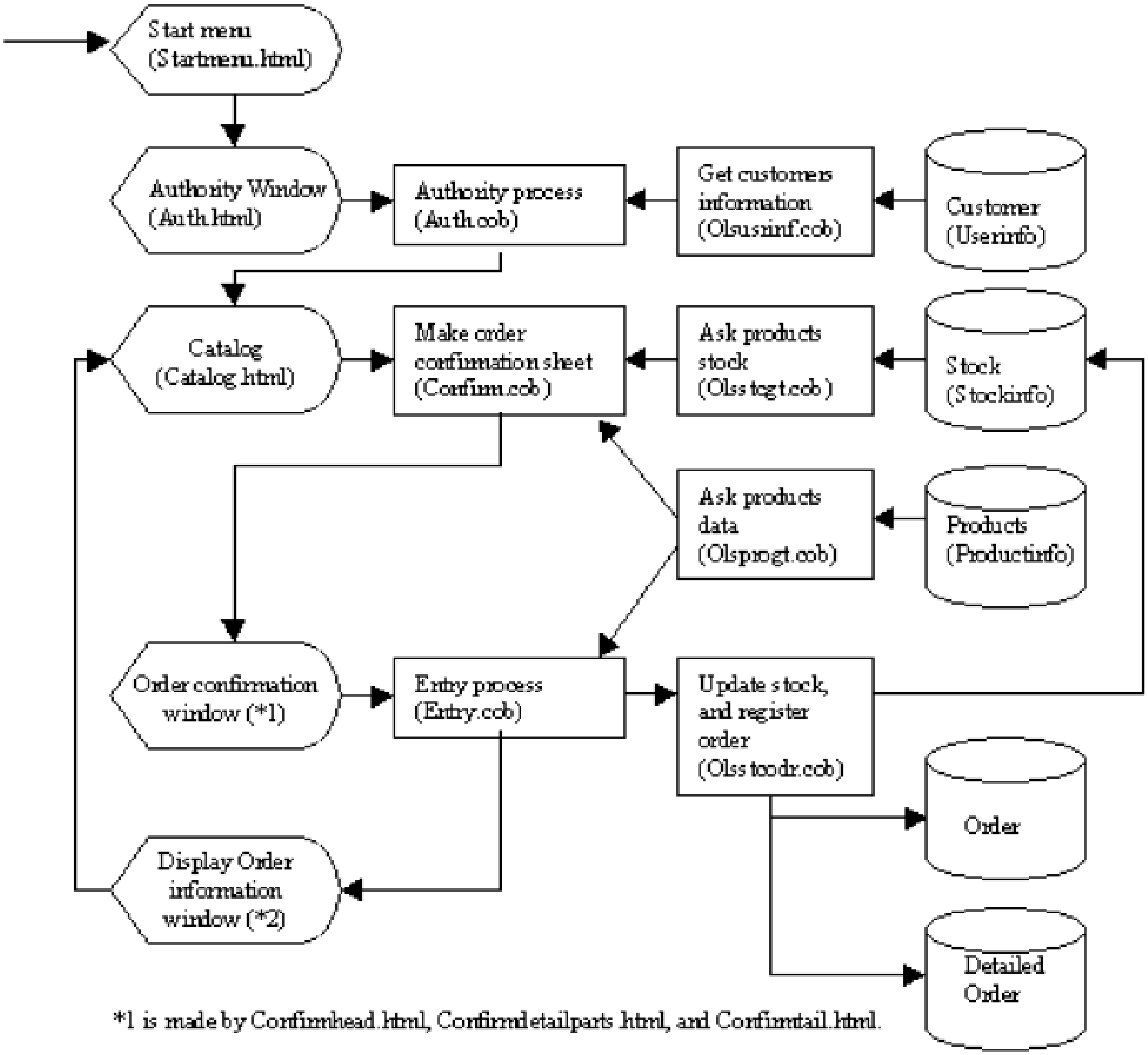
  - STARTMENU.HTML
  - SYSERROR.HTML
  - SYSTEMERROR.HTML
  - UNDERCONSTRUCTION.HTML
- GIF files
  - CATALOGTITLE.GIF
  - FJLOGO.GIF
  - TITLE.GIF
  - LifeBookE.GIF
- JPEG files
  - LifeBookB.JPG
  - LifeBookC.JPG
  - LifeBookL.JPG

Process Flow

1. Business start and end



2. Online store


Start menu (Startmenu.html)
Authority Window (Auth.html)
Authority process (Auth.cob)
Get customers information (Olsusrinf.cob)
Customer (Userinfo)
Catalog (Catalog.html)
Make order confirmation sheet (Confirm.cob)
Ask products stock (Olsstcgt.cob)
Stock (Stockinfo)
Ask products data (Olsprogt.cob)
Products (Productinfo)
Order confirmation window (*1)
Entry process (Entry.cob)
Update stock, and register order (Olsstcodr.cob)
Order
Display Order information window (*2)
Detailed Order

*1 is made by Confirmhead.html, Confirmdetailparts.html, and Confirmtail.html.

*2 is made by Orderesult.html, orderdetailparts.html, and confirmresulttail.html.

Applicable COBOL Functions

- Index file (creation, reference, update, and rewrite)
- External data
- External file
- Data lock subroutine
- External file event log (output of user definition information)
- COBOL ISAPI subroutine

Applicable COBOL Statements

The CALL, CLOSE, EXIT, GO TO, IF, MOVE, OPEN, PERFORM, READ REWRITE, SET, START, and WRITE statements are used.

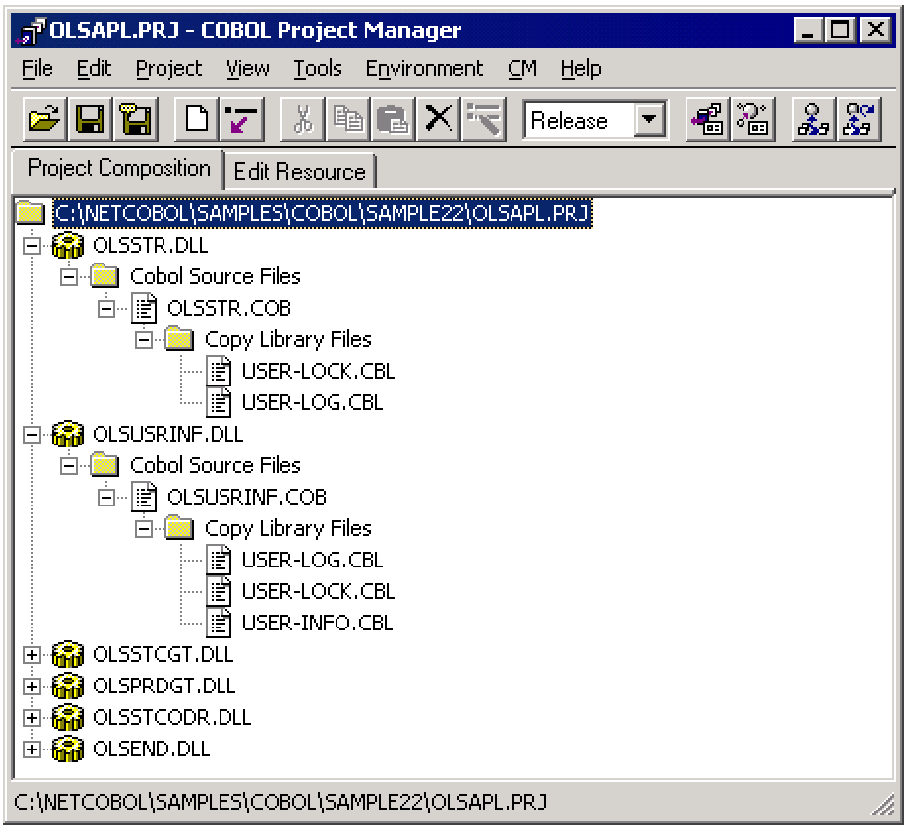
Building the Program

The Project Manager's Build function is used to create the executable program.

In the following screens, the user has installed the COBOL system in C:\NetCOBOL.

Your install location may be different.

1. The Project Manager is started.
2. The project file "OLSAPL.PRJ" is opened.



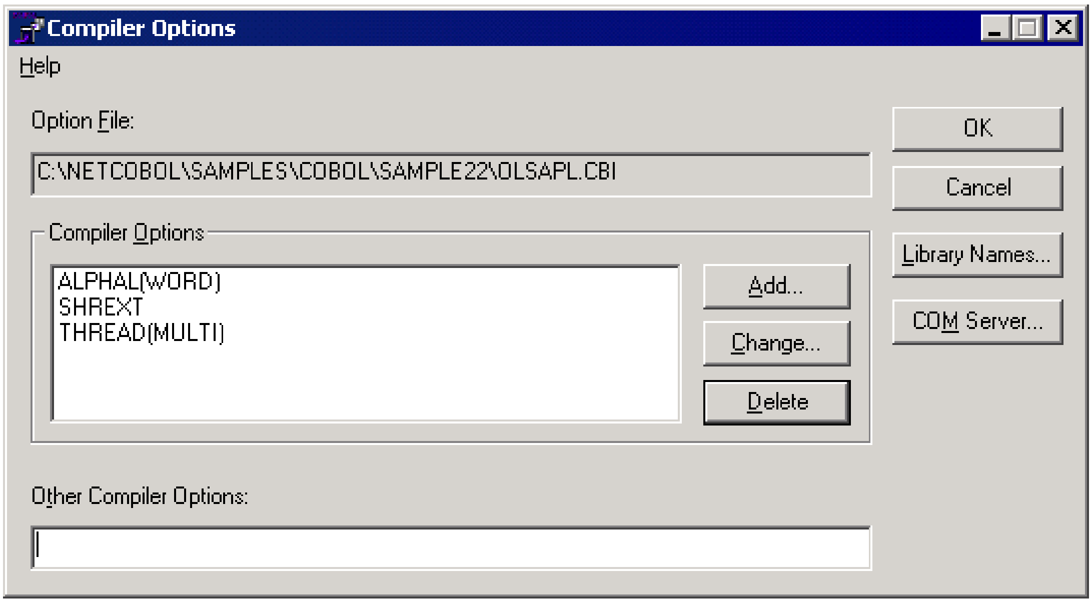
Release

```
C:\NETCOBOL\SAMPLES\COBOL\SAMPLE22\OLSAPL.PRJ
  OLSSTR.DLL
    Cobol Source Files
      OLSSTR.COB
        Copy Library Files
          USER-LOCK.CBL
          USER-LOG.CBL
  OLSUSRINF.DLL
    Cobol Source Files
      OLSUSRINF.COB
        Copy Library Files
          USER-LOG.CBL
          USER-LOCK.CBL
          USER-INFO.CBL
  OLSSTCGT.DLL
  OLSPRDGT.DLL
  OLSSTCODR.DLL
  OLSEND.DLL
```

C:\NETCOBOL\SAMPLES\COBOL\SAMPLE22\OLSAPL.PRJ

3. The project file is selected, and "Compiler options" is selected from the "Project"-"Options" menu.

   The "Compiler options" dialog is displayed.



Option File:

C:\NETCOBOL\SAMPLES\COBOL\SAMPLE22\OLSAPL.CBI

Compiler Options

```
ALPHAL(WORD)
SHREXT
THREAD(MULTI)
```

Other Compiler Options:

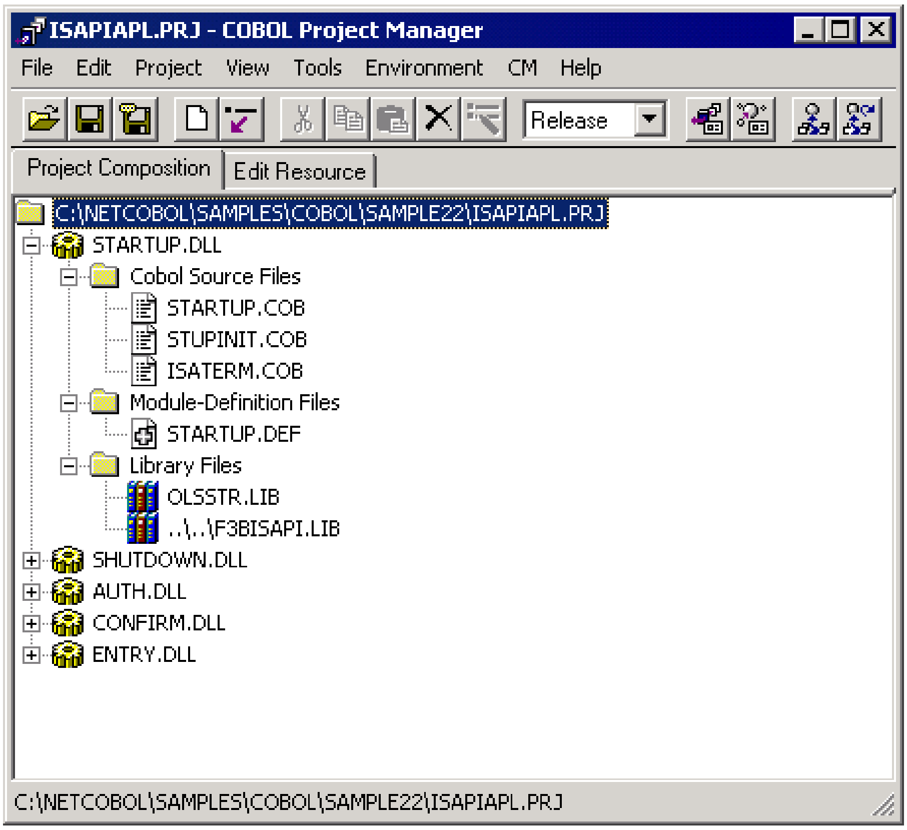
4. Compiler option THREAD(MULTI), SHREXT and ALPHAL(WORD) are specified.

   Click the OK button.

5. "Build" is selected from Project Manager's "Project" menu.

   Please confirm each DLL (dynamic link library) registered in the project is created.

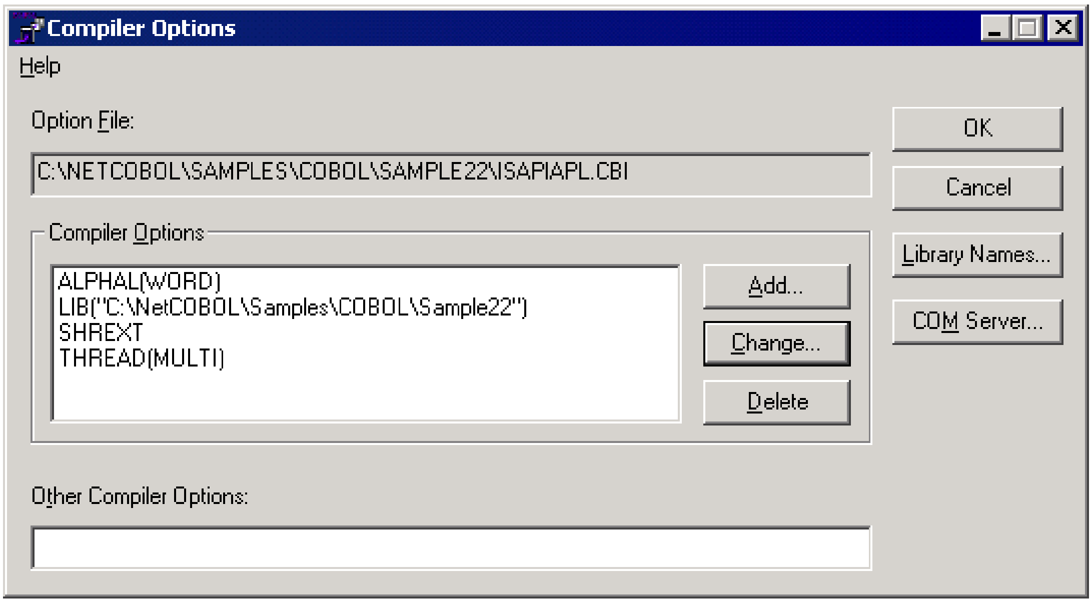
6. Next, The project file "ISAPIAPL.PRJ" is opened


ISAPIAPL.PRJ - COBOL Project Manager
File Edit Project View Tools Environment CM Help
Release
Project Composition
Edit Resource

```
C:\NETCOBOL\SAMPLES\COBOL\SAMPLE22\ISAPIAPL.PRJ
  STARTUP.DLL
    Cobol Source Files
      STARTUP.COB
      STUPINIT.COB
      ISATERM.COB
    Module-Definition Files
      STARTUP.DEF
    Library Files
      OLSSTR.LIB
      ..\..\F3BISAPI.LIB
  SHUTDOWN.DLL
  AUTH.DLL
  CONFIRM.DLL
  ENTRY.DLL
```

C:\NETCOBOL\SAMPLES\COBOL\SAMPLE22\ISAPIAPL.PRJ

7. The compiler option dialog is displayed as in step 3 above. The compiler option THREAD(MULTI), SHREXT, and ALPHAL(WORD) are specified. Modify the folder name of the library file specified with the compiler option LIB. After confirmation, the OK button is clicked.



Option File:
C:\NETCOBOL\SAMPLES\COBOL\SAMPLE22\ISAPIAPL.CBI

Compiler Options

```
ALPHAL(WORD)
LIB("C:\NetCOBOL\Samples\COBOL\Sample22")
SHREXT
THREAD(MULTI)
```

Other Compiler Options:

8. "Build" is selected from Project Manager's "Project" menu.

   Please confirm each DLL (dynamic link library) registered in the project is built.

Executing the Program

It is assumed that the domain-name and virtual directory name are registered in IIS (Internet Information Server) as "user" and "sample22" respectively.

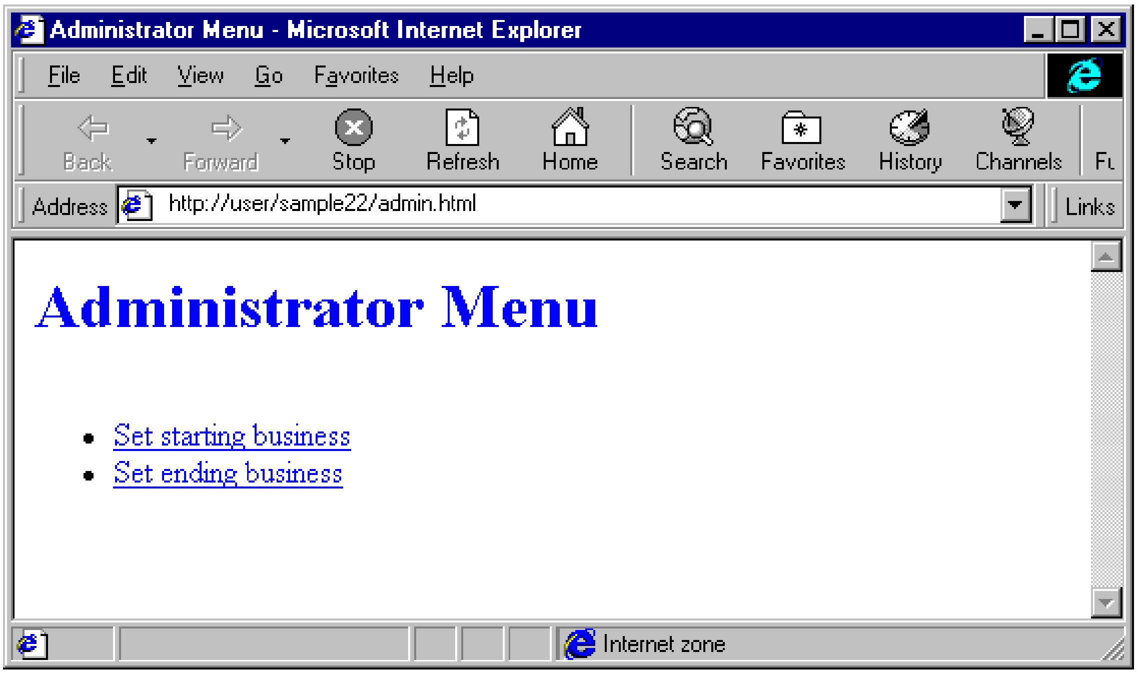
In this example uses Microsoft® Internet Explorer as the WWW browser.

1. Online store is started.

   The following information is set in theURL.

```
http://user/sample22/admin.html
```

   Then click "Set starting business".



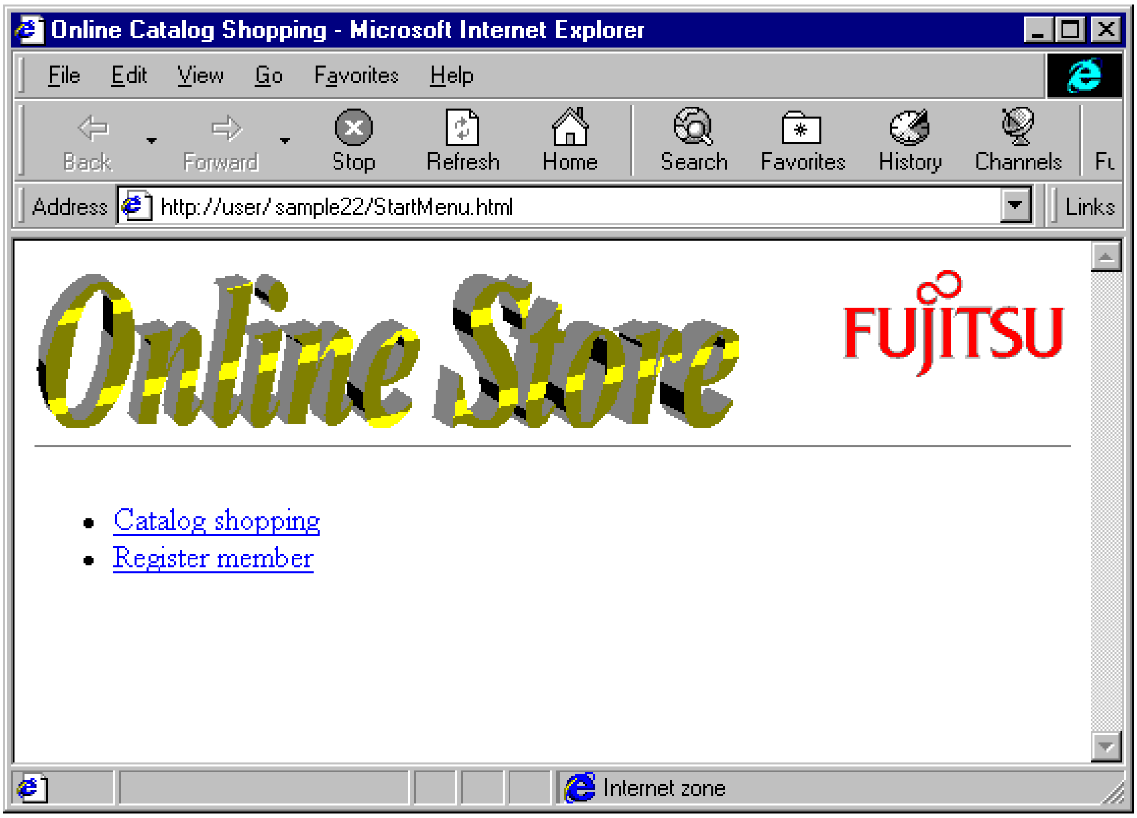
   When clicked, the online store is started. Please do the "Set starting business" before starting online shopping.

2. Online store is started.

   The following information is set in URL and the "Execute" key is pushed.

```
http://user/sample22/startmenu.html
```

Because the screen of an online store is displayed, the catalog shopping is clicked.



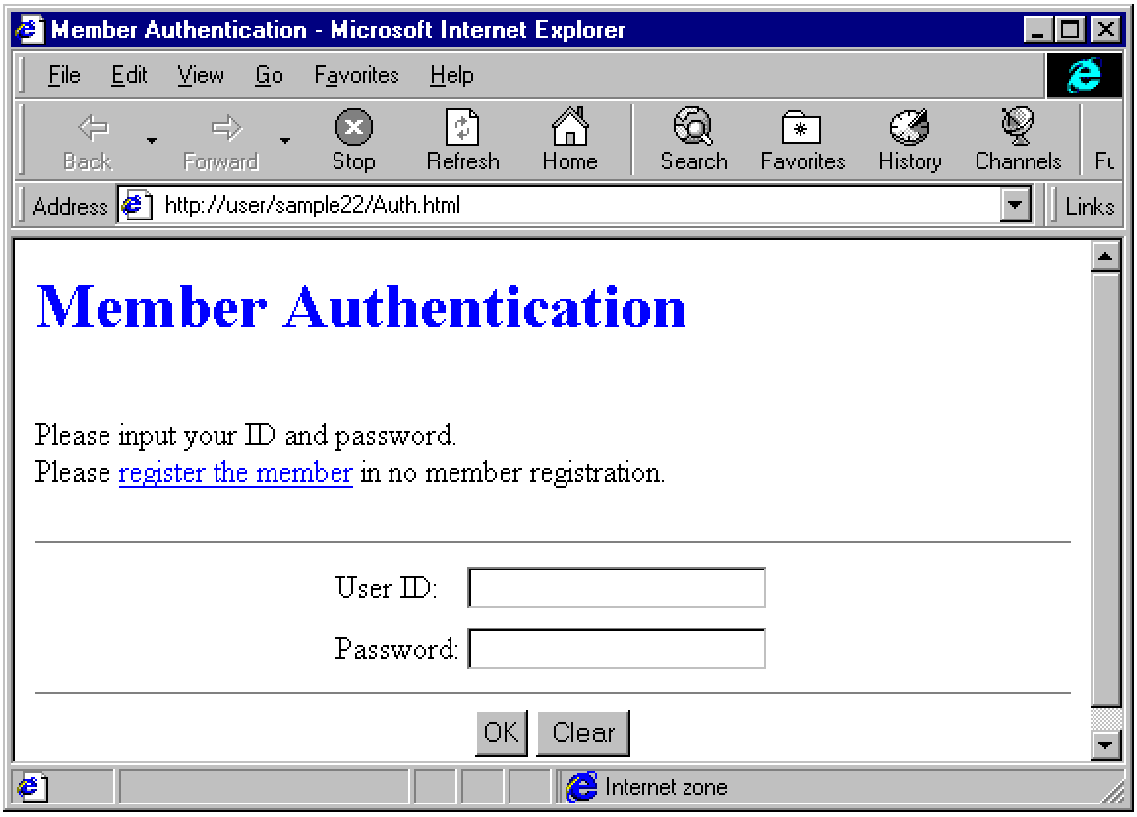
Address: http://user/ sample22/StartMenu.html

- Catalog shopping
- Register member

When Catalog Shopping is clicked, the member authentication screen is displayed.

After the screen is displayed, the User ID and password are input. Click the "OK" button.

Valid User ID's are USER0001 to USER0030. The password is the same as the user ID.


Member Authentication - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fu
Address
Links
Internet zone

http://user/sample22/Auth.html

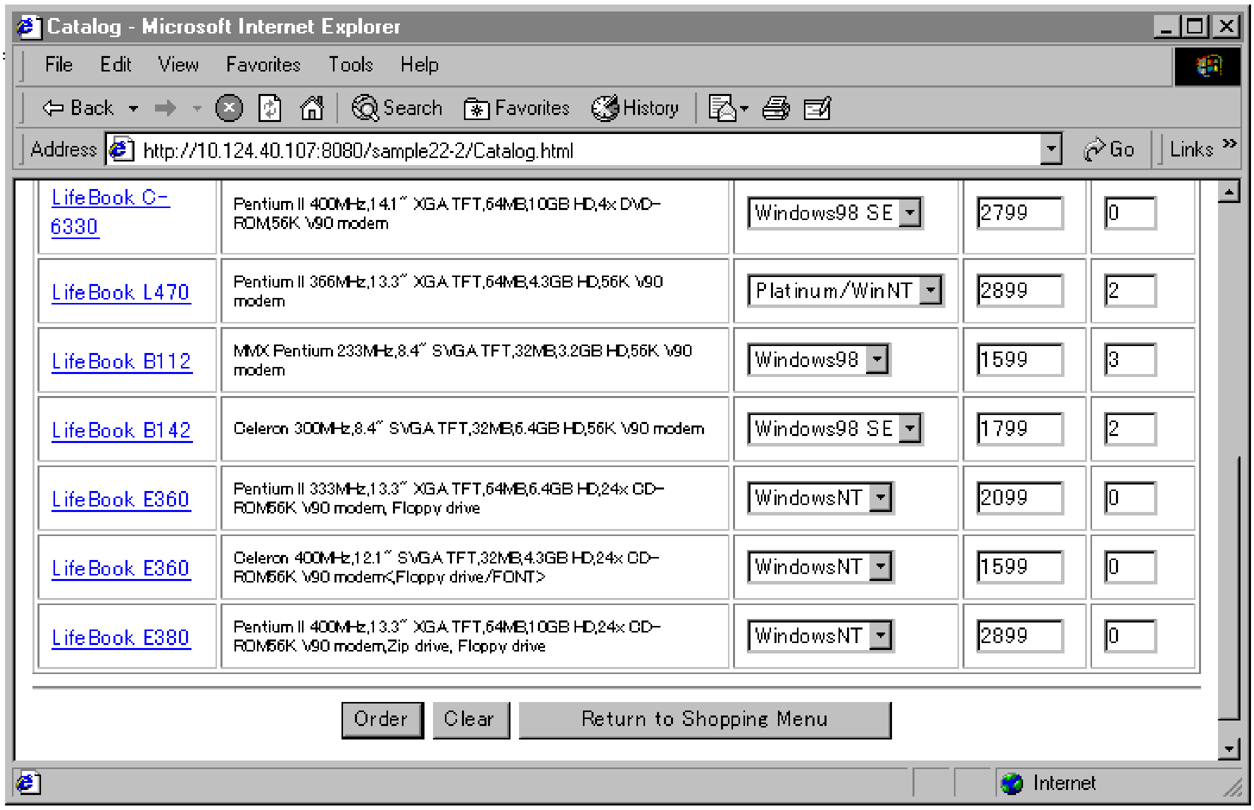
Member Authentication

Please input your ID and password.
Please register the member in no member registration.

User ID:

Password:

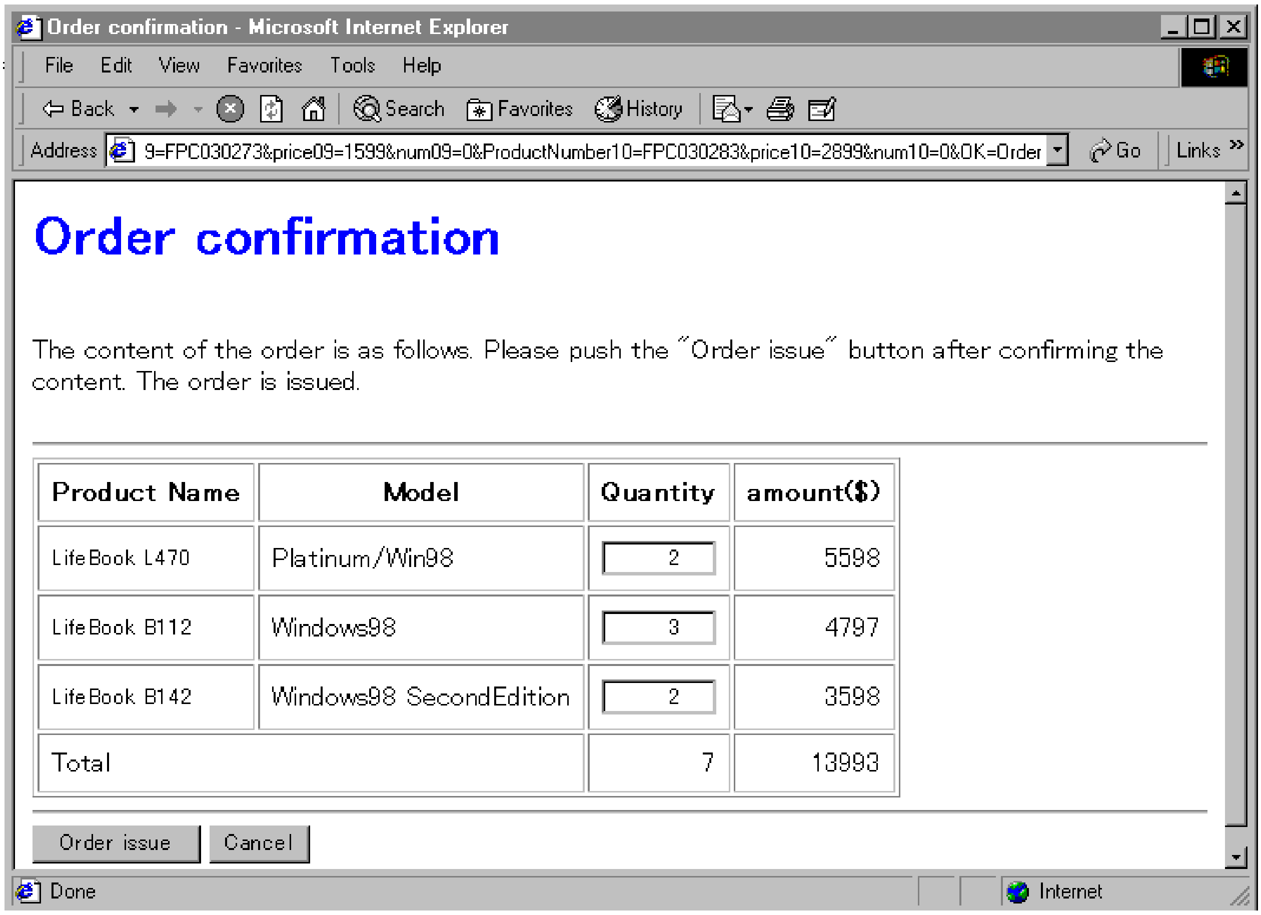
When the "OK" button is clicked, the catalog screen is displayed. The catalog screen shows you the various PC's and their quantities available for ordering. To place an order, select one or more products and enter an order quantity, then press the "Order" button.


Catalog - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address
Go
Links
Internet

http://10.124.40.107:8080/sample22-2/Catalog.html

LifeBook C-6330	Pentium II 400MHz,14.1" XGA TFT,64MB,10GB HD,4x DVD-ROM,56K V90 modem	Windows98 SE	2799	0
LifeBook L470	Pentium II 366MHz,13.3" XGA TFT,64MB,4.3GB HD,56K V90 modem	Platinum/WinNT	2899	2
LifeBook B112	MMX Pentium 233MHz,8.4" SVGA TFT,32MB,3.2GB HD,56K V90 modem	Windows98	1599	3
LifeBook B142	Celeron 300MHz,8.4" SVGA TFT,32MB,6.4GB HD,56K V90 modem	Windows98 SE	1799	2
LifeBook E360	Pentium II 333MHz,13.3" XGA TFT,64MB,6.4GB HD,24x CD-ROM,56K V90 modem, Floppy drive	WindowsNT	2099	0
LifeBook E360	Celeron 400MHz,12.1" SVGA TFT,32MB,4.3GB HD,24x CD-ROM,56K V90 modem<,Floppy drive/FONT>	WindowsNT	1599	0
LifeBook E380	Pentium II 400MHz,13.3" XGA TFT,64MB,10GB HD,24x CD-ROM,56K V90 modem,Zip drive, Floppy drive	WindowsNT	2899	0

When the "Order" button is clicked, the order confirmation screen is displayed. The content of the order is confirmed and the "Order issue" button is clicked.



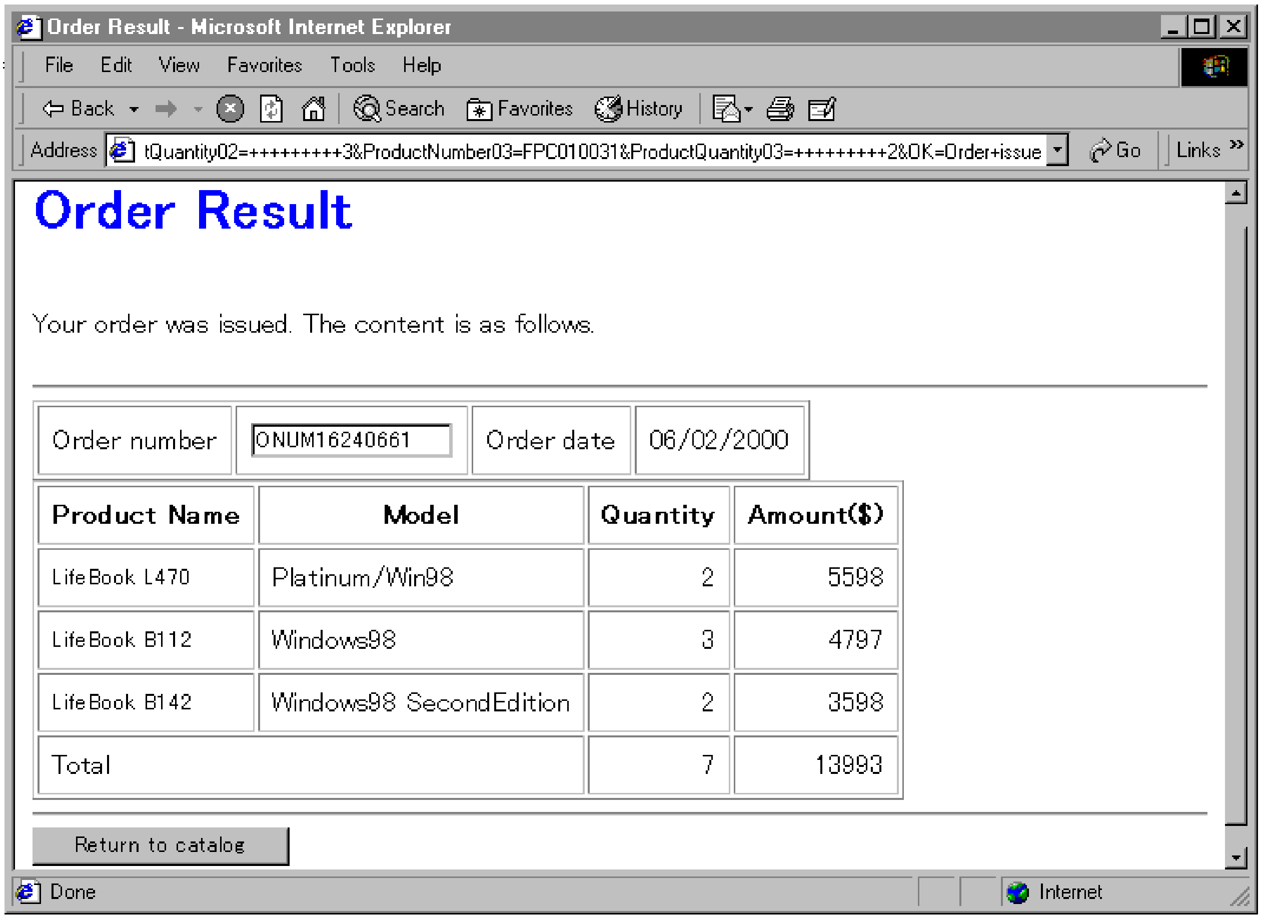
Order confirmation - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address 9=FPC030273&price09=1599&num09=0&ProductNumber10=FPC030283&price10=2899&num10=0&OK=Order
Go Links
Done
Internet

Order confirmation

The content of the order is as follows. Please push the "Order issue" button after confirming the content. The order is issued.

Product Name	Model	Quantity	amount($)
LifeBook L470	Platinum/Win98	2	5598
LifeBook B112	Windows98	3	4797
LifeBook B142	Windows98 SecondEdition	2	3598
Total		7	13993

When the "Order issue" button is clicked, the order result screen is displayed.



Order Result - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address tQuantity02=+++++++++3&ProductNumber03=FPC010031&ProductQuantity03=+++++++++2&OK=Order+issue
Go Links
Done
Internet

Order Result

Your order was issued. The content is as follows.

Order number	ONUM16240661	Order date	06/02/2000

Product Name	Model	Quantity	Amount($)
LifeBook L470	Platinum/Win98	2	5598
LifeBook B112	Windows98	3	4797
LifeBook B142	Windows98 SecondEdition	2	3598
Total		7	13993

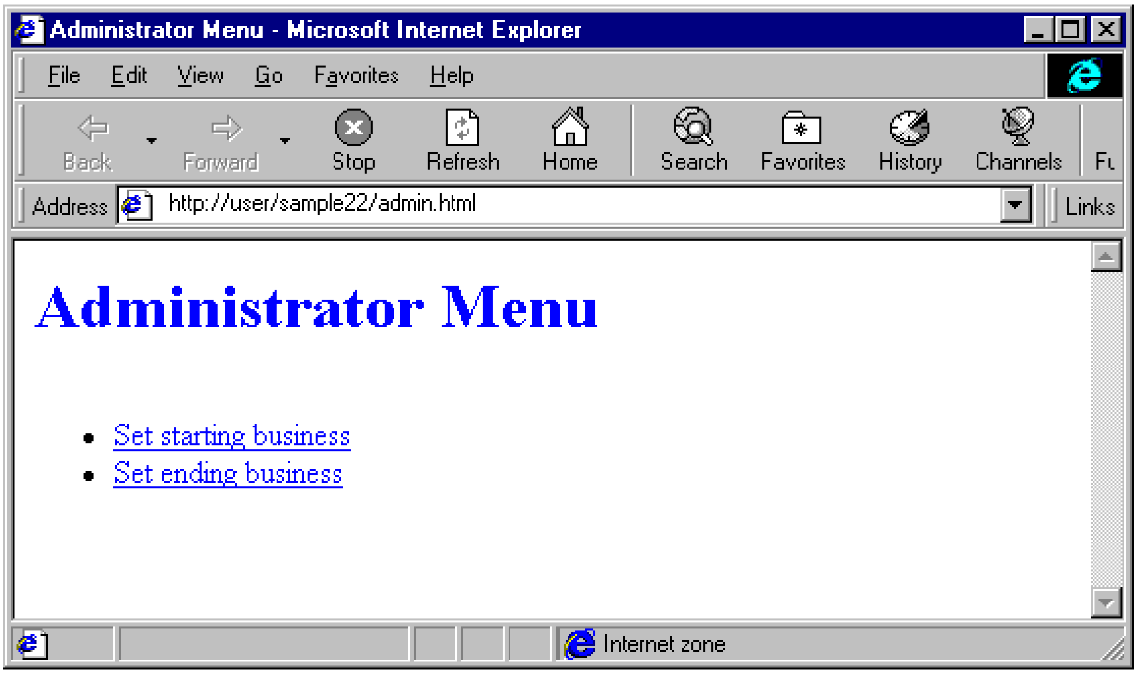
Return to catalog

3. Online shopping is terminated.

The following information is set in URL and the "Execute" key is pressed.

```
http://user/sample22/admin.html
```

Because the administrator menu screen is displayed, the "Set ending business" is clicked.



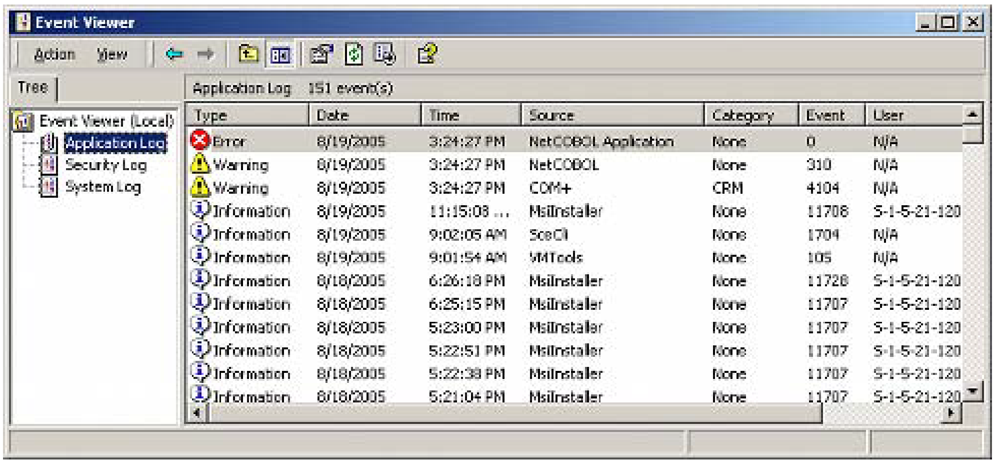
Administrator Menu

- Set starting business
- Set ending business

Creating an Event Log

In this exercise, detailed information of errors detected by the program is output to an event log by using the event log output subroutine. Note that the Event Viewer discussed below is only available on Windows Vista, Windows 2000, Windows Server 2003, Windows Server 2008 and Windows XP.

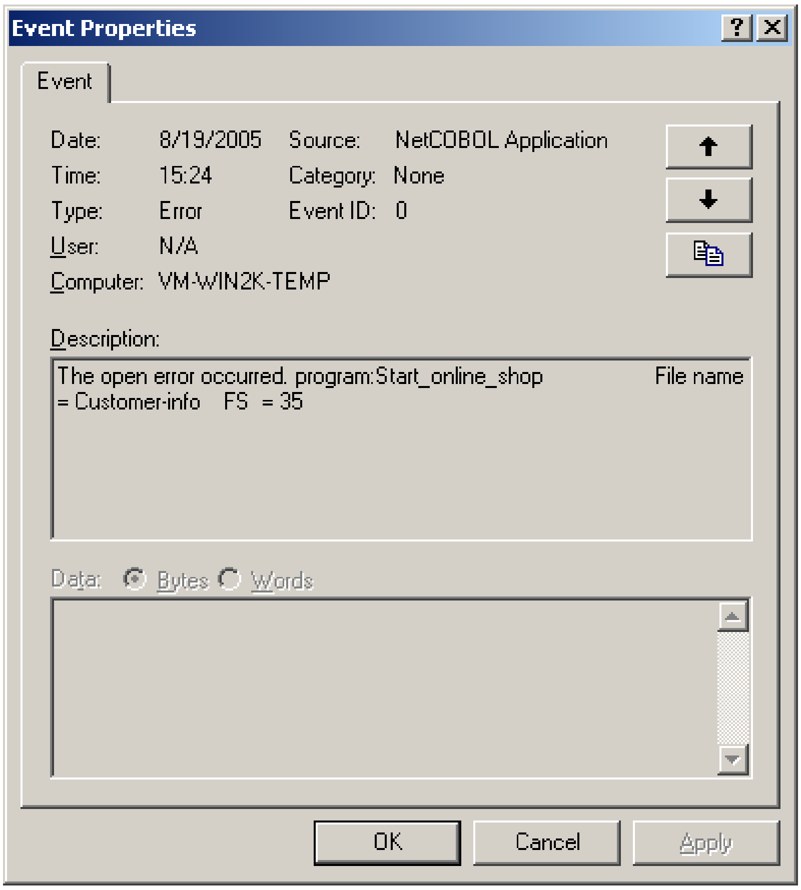
1. The log of the application is selected by starting the Event Viewer from the Administrative Tools.



Application Log 151 event(s)

Type	Date	Time	Source	Category	Event	User
Error	8/19/2005	3:24:27 PM	NetCOBOL Application	None	0	N/A
Warning	8/19/2005	3:24:27 PM	NetCOBOL	None	310	N/A
Warning	8/19/2005	3:24:27 PM	COM+	CRM	4104	N/A
Information	8/19/2005	11:15:08 ...	MsiInstaller	None	11708	S-1-5-21-120
Information	8/19/2005	9:02:05 AM	SceCli	None	1704	N/A
Information	8/19/2005	9:01:54 AM	VMTools	None	105	N/A
Information	8/18/2005	6:26:18 PM	MsiInstaller	None	11728	S-1-5-21-120
Information	8/18/2005	6:25:15 PM	MsiInstaller	None	11707	S-1-5-21-120
Information	8/18/2005	5:23:00 PM	MsiInstaller	None	11707	S-1-5-21-120
Information	8/18/2005	5:22:51 PM	MsiInstaller	None	11707	S-1-5-21-120
Information	8/18/2005	5:22:38 PM	MsiInstaller	None	11707	S-1-5-21-120
Information	8/18/2005	5:21:04 PM	MsiInstaller	None	11707	S-1-5-21-120

2. Detailed information is displayed when the source selects the log of "NetCOBOL Application", and double-clicks it.



Date: 8/19/2005 Source: NetCOBOL Application
Time: 15:24 Category: None
Type: Error Event ID: 0
User: N/A
Computer: VM-WIN2K-TEMP

Description:
The open error occurred. program:Start_online_shop File name = Customer-info FS = 35

Data: ● Bytes ○ Words

A.24 Sample 23: COM Program to Control Excel (Late Binding)

Sample 23 is a program that uses the COBOL COM client function to operate Excel.

The COBOL COM client function creates a COM object that uses functions which the COM server supports in the same way as a COBOL method.

Excel is an independent application and also functions as a COM server. Therefore, an application that operates Excel can be written with COBOL.

To run this program, the following products are necessary:

- Microsoft(R) Excel

"Excel" is used here as a generic name for these products.

For details on the COBOL COM function, refer to Chapter 24 in the "NetCOBOL User's Guide".

Overview

The sample program performs the operations listed below for Excel from a COBOL application.

- Excel activation and termination
- Excel sheet open
- Work sheet selection and data insertion
- Graph creation
- Printing of a selected sheet

The COBOL COM client function supports the early binding and late binding methods to recognize COM server methods and interfaces.

Late binding is used in this sample.

When late binding is used, usable COBOL descriptions are limited and execution performance is slower than early binding, but late binding is freer from the effects of COM server modifications than early binding. When using late binding this program can operate Excel97 and Excel2000, which are different COM servers.

For details of early binding, see Sample 24, "COM Program to Control Excel (Early Binding)."

Available Programs

- SAMPLE23.PRJ (Project file)
- SAMPLE23.COB (COBOL source file)
- GRAPHDATA.XLS (Excel file for test)

Available COBOL Functions

- COM client function

Applicable COBOL Statements

The DISPLAY, IF, INVOKE, PERFORM, and SET statements are used.

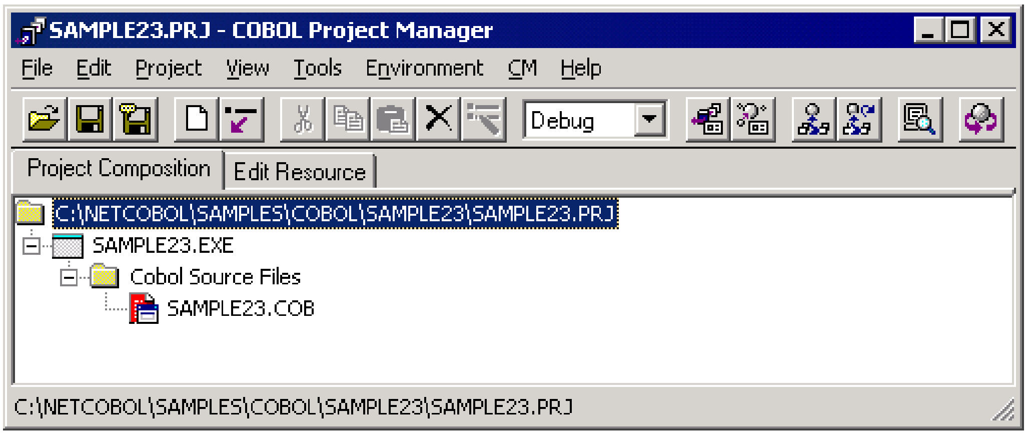
Building/Rebuilding

The Project Manager's Build function is used to compile and link this program.

Folder C:\NetCOBOL is assumed below as the location where NetCOBOL is installed.

Change the folder name C:\NetCOBOL to the name of the folder where NetCOBOL is installed.

1. Start the Project Manager.



SAMPLE23.PRJ - COBOL Project Manager

C:\NETCOBOL\SAMPLES\COBOL\SAMPLE23\SAMPLE23.PRJ
SAMPLE23.EXE
Cobol Source Files
SAMPLE23.COB

C:\NETCOBOL\SAMPLES\COBOL\SAMPLE23\SAMPLE23.PRJ

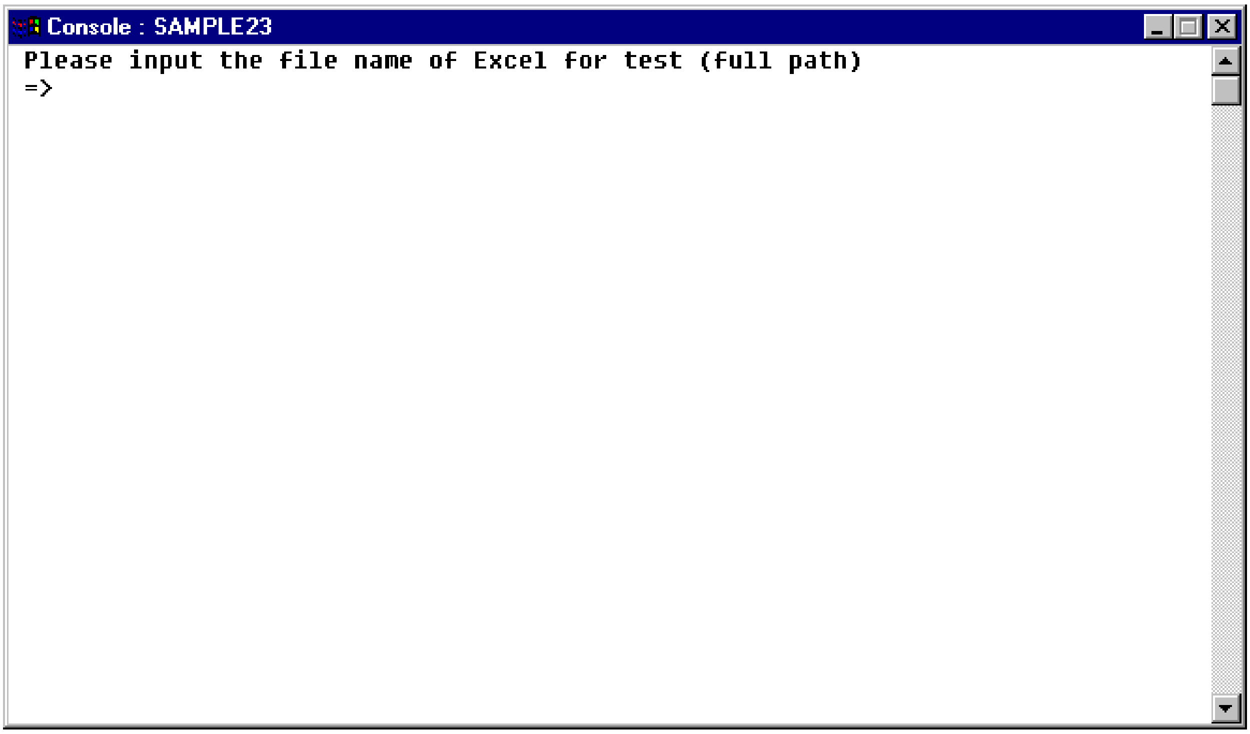
2. Open the project file SAMPLE23.PRJ.
3. Select "Build" on the "Project" menu.

   After the build terminates, check that SAMPLE23.EXE is created.

Executing the Program

1. Select "Execute" on the "Project" menu.

   Excel is started. The following message is also displayed on the COBOL console and the system waits for input.

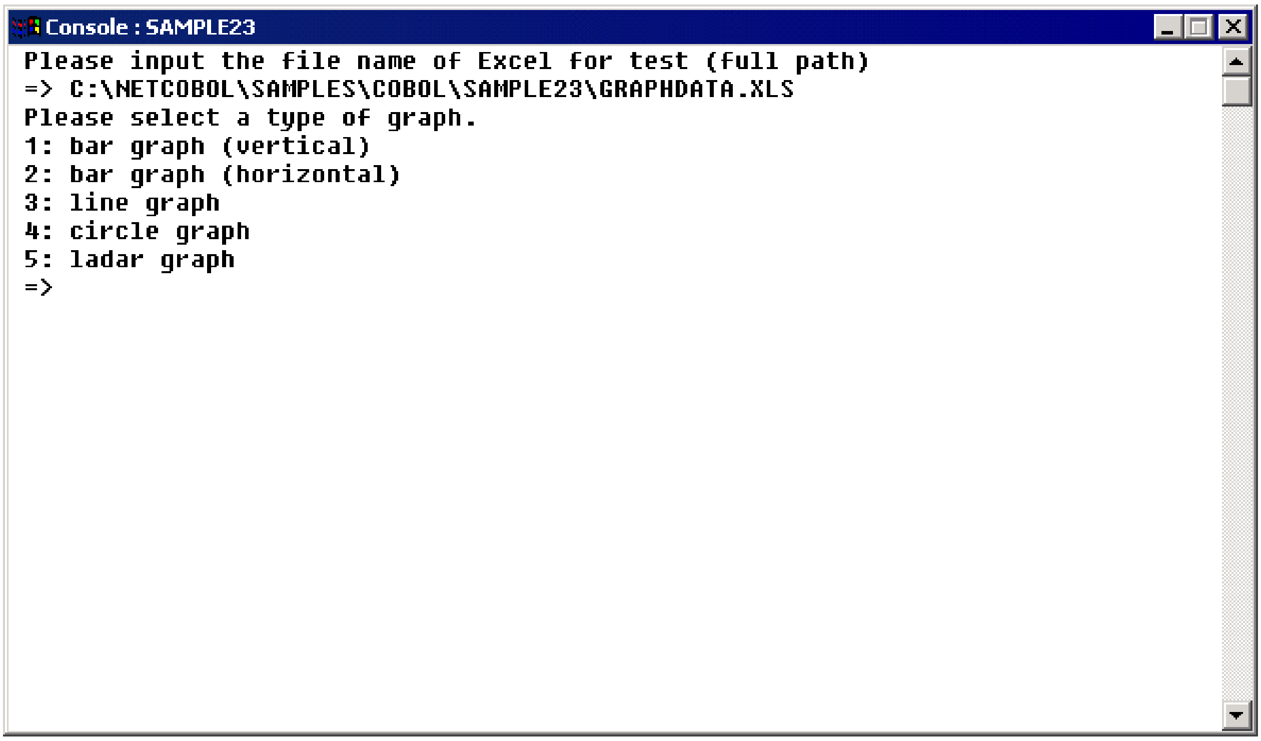


```
Please input the file name of Excel for test (full path)
=>
```

2. Use the full path-name to specify an Excel file for the test.

   The specified Excel file is opened and the sample program controls Excel.

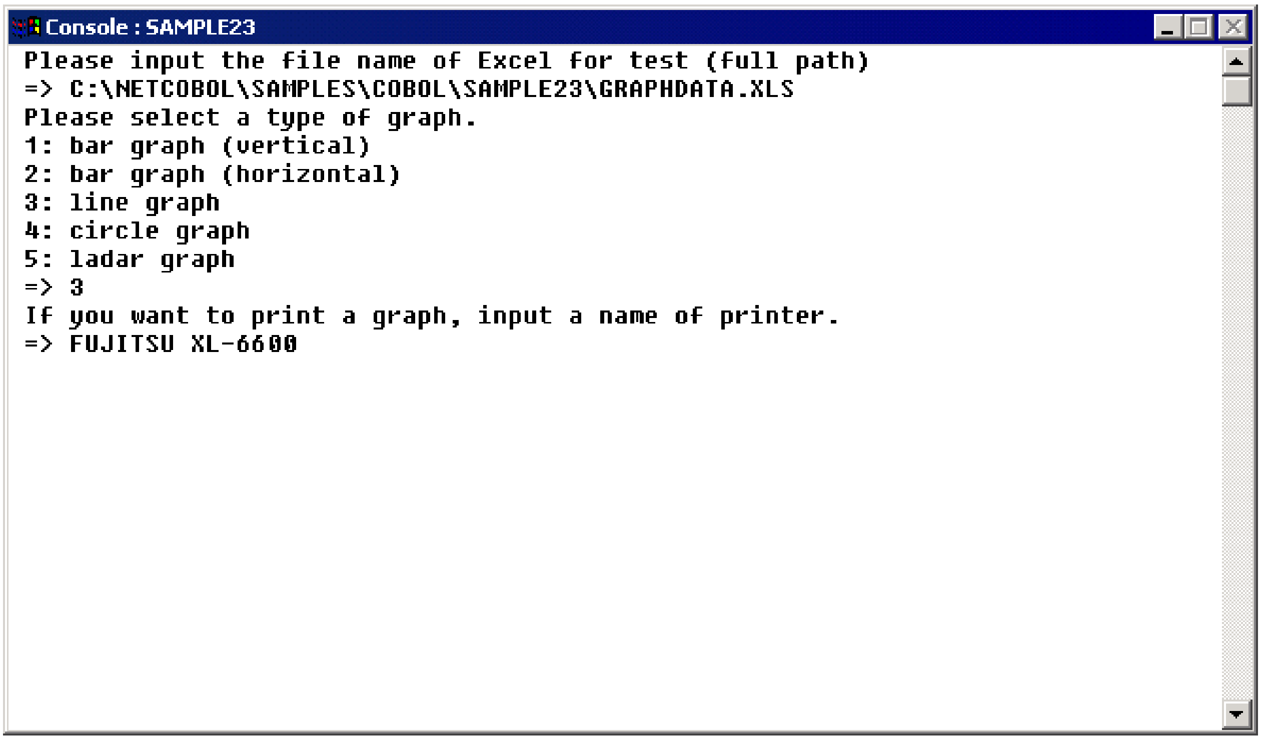
   The following message is displayed on the COBOL console and the system waits for input again.



```
Please input the file name of Excel for test (full path)
=> C:\NETCOBOL\SAMPLES\COBOL\SAMPLE23\GRAPHDATA.XLS
Please select a type of graph.
1: bar graph (vertical)
2: bar graph (horizontal)
3: line graph
4: circle graph
5: ladar graph
=>
```

3. Input the type of graph to be created.

   The specified graph is plotted. The following message is displayed on the COBOL console and the system waits for input again.



```
Please input the file name of Excel for test (full path)
=> C:\NETCOBOL\SAMPLES\COBOL\SAMPLE23\GRAPHDATA.XLS
Please select a type of graph.
1: bar graph (vertical)
2: bar graph (horizontal)
3: line graph
4: circle graph
5: ladar graph
=> 3
If you want to print a graph, input a name of printer.
=> FUJITSU XL-6600
```

4. If the graph is to be printed, specify a printer name. If it is not to be printed, press the ENTER key without inputting data.

   Excel is terminated and the program is terminated. The graph is printed before terminating Excel when a printer name is specified.

A.25 Sample 24: COM Program to Control Excel (Early Binding)

Sample 24 is a program that uses the COBOL COM client early binding function to operate Excel.

The COBOL COM client function creates a COM object to use functions which the COM server supports in the same way as the COBOL method.

Excel is an independent application and also functions as a COM server. So, an application that operates Excel can be written with COBOL.

To execute this program, the following products are necessary:

- Microsoft(R) Excel

"Excel" is used here as a generic name for these products.

For details on the COBOL COM function, refer to Chapter 24 in the "NetCOBOL User's Guide".

Overview

The sample program performs the operations listed below for Excel from a COBOL application.

- Excel activation and termination
- Excel sheet open
- Work sheet selection and data insertion
- Graph creation
- Printing of selected sheet

The COBOL COM client function supports the early binding and late binding methods to recognize COM server methods and interfaces.

Early binding is used in this sample.

When early binding is used, object property and in-line method invocation can be written. Also, the performance is better than that of late binding. However, the early binding method requires a type library during development and is liable to be influenced by COM server modifications. This program uses the COM server name EXCEL to use the Excel functions. An Excel type library must be specified for this COM server name before compiling the program. Excel97 cannot be operated with an executable file built by referencing an Excel2000 type library. Also, Excel2000 cannot be used with an executable file built by referencing an Excel97 type library.

For details of late binding, see Sample 23, "COM Program to Control Excel (Late Binding)."

Available Programs

- SAMPLE24.PRJ (Project file)
- SAMPLE24.COB (COBOL source file)
- GRAPHDATA.XLS (Excel file for test)

Applicable COBOL Functions

- COM client function
- Project management function

Applicable COBOL Statements

The DISPLAY, IF, INVOKE, PERFORM, and SET statements are used.

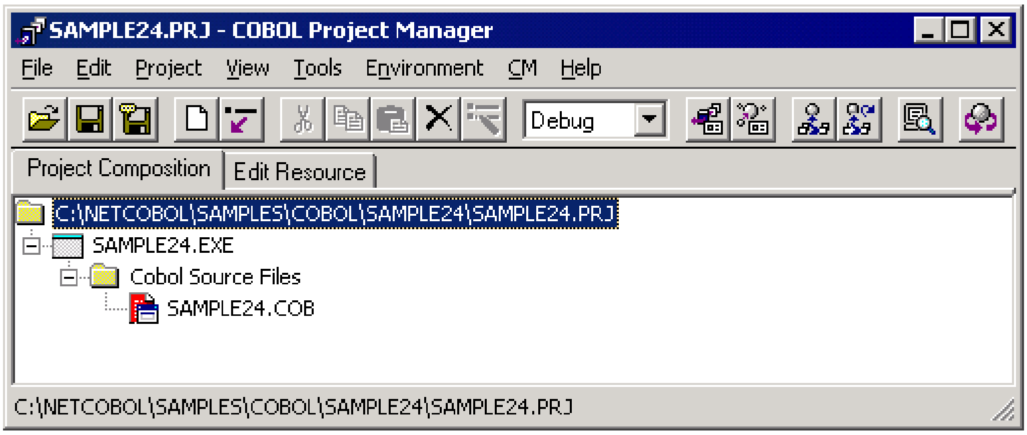
Building and Rebuilding

The Project Manager's Build function is used to compile and link this program.

Folder C:\NetCOBOL is assumed below as the location where NetCOBOL is installed.

Change the folder name C:\NetCOBOL to the name of the folder where NetCOBOL is installed.

1. Start the Project Manager.


SAMPLE24.PRJ - COBOL Project Manager

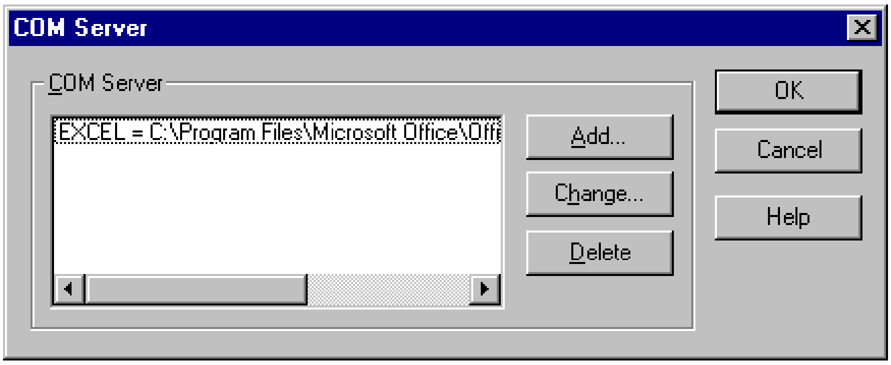
C:\NETCOBOL\SAMPLES\COBOL\SAMPLE24\SAMPLE24.PRJ
SAMPLE24.EXE
Cobol Source Files
SAMPLE24.COB

C:\NETCOBOL\SAMPLES\COBOL\SAMPLE24\SAMPLE24.PRJ

2. Open the project file SAMPLE24.PRJ.
3. Select a project file and select "Compiler Options" from the "Project-Option" menu.

   The "Compiler Options" dialog is displayed.

4. Click the "COM Server Name" button in the "Compiler Options" dialog.



EXCEL = C:\Program Files\Microsoft Office\Offi

The "COM Server Setting" dialog is displayed.

5. Set data as shown below, assuming that type library Excel is installed in the "C:\Program Files\Microsoft Office" with COM server name EXCEL.

```
EXCEL = C:\Program Files\Microsoft Office\Office\Excel8.olb
```

If type library Excel2000 is used (file name: Excel9.olb) or Excel is installed in another folder, click the Change button to change the type library setting.

6. Select "Build" from the "Project" menu.

After build termination, check that SAMPLE24.EXE is created.

Note

When you use Excel2002 or later, following modification is necessary.

- Program

  Modify "sample24.cob" as following.

```
001040        INVOKE OBJ-CELL "SET-VALUE"
001050                        USING PDATA(CELL-COL)
001060* If you use EXCEL2002 or later, replace the above statement with
001061*   this statement:
001070*       INVOKE OBJ-CELL "SET-VALUE"
001080*                       USING OMITTED PDATA(CELL-COL)
```

  - Comment out line 1040 and 1050.

  - Validate the comment line 1070 and 1080.

```
001040*       INVOKE OBJ-CELL "SET-VALUE"
001050*                       USING PDATA(CELL-COL)
001060* If you use EXCEL2002 or later, replace the above statement with
001061*  this statement:
001070        INVOKE OBJ-CELL "SET-VALUE"
001080                        USING OMITTED PDATA(CELL-COL)
```

- Type library

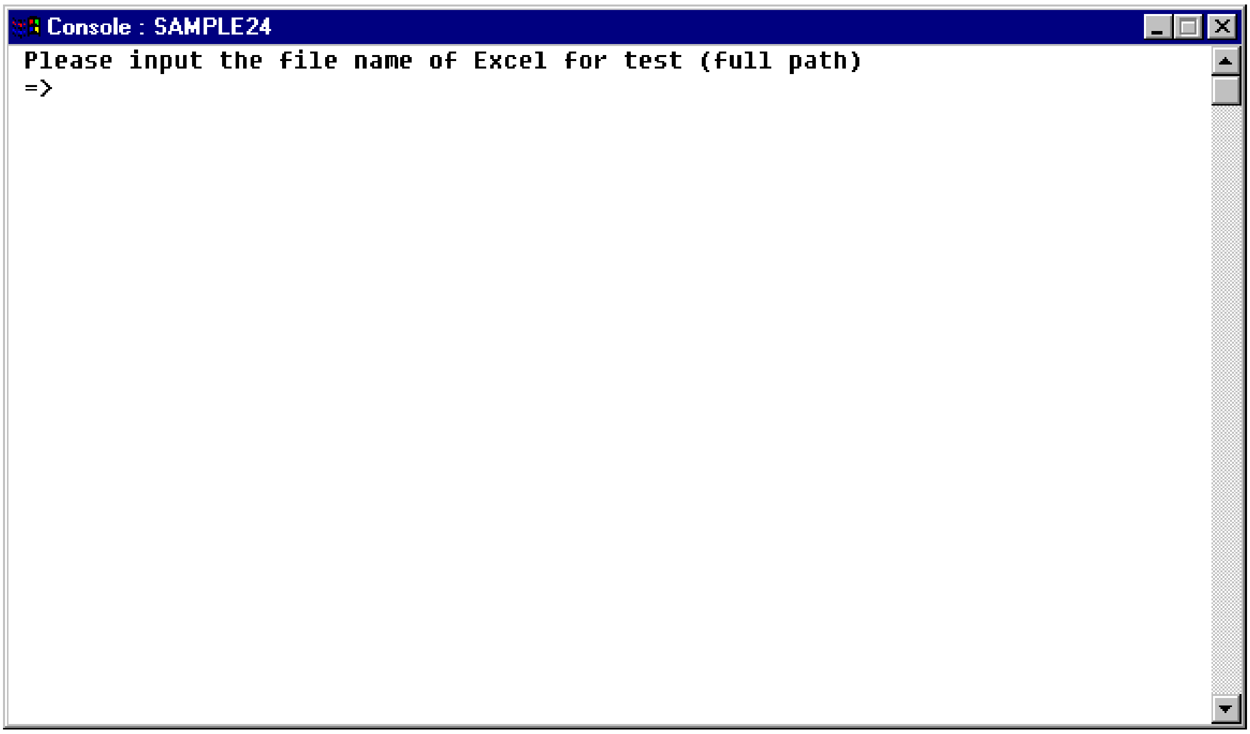
  The type library of Excel2002 or later is Excel.exe. Set COM server name as shown below, assuming that Excel2002 is installed in the "C:\Program Files\Microsoft Office".

```
EXCEL = C:\Program Files\Microsoft Office\Office\Excel.exe
```

Executing the Program

1. Select "Execute" on the "Project" menu.

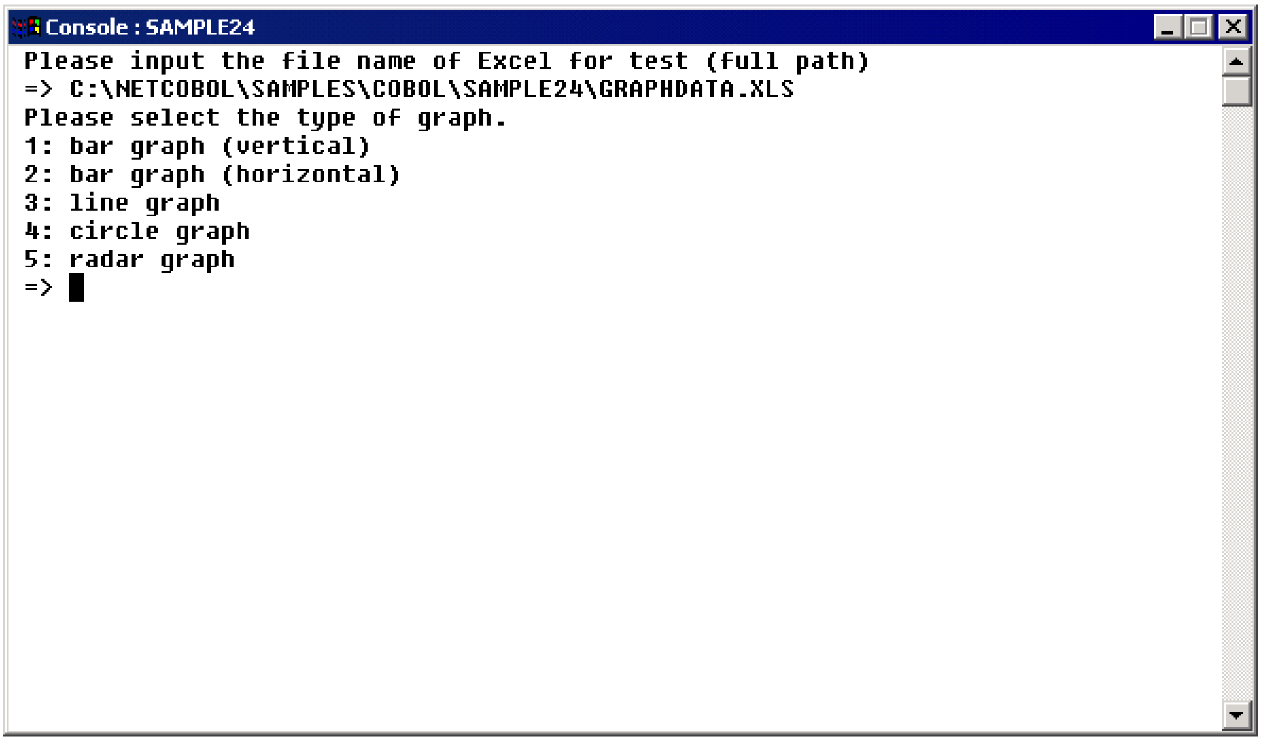
   Excel is started. The following message is displayed on the COBOL console and the system waits for input.



```
Please input the file name of Excel for test (full path)
=>
```

2. Use the full path-name to specify an Excel file for the test.

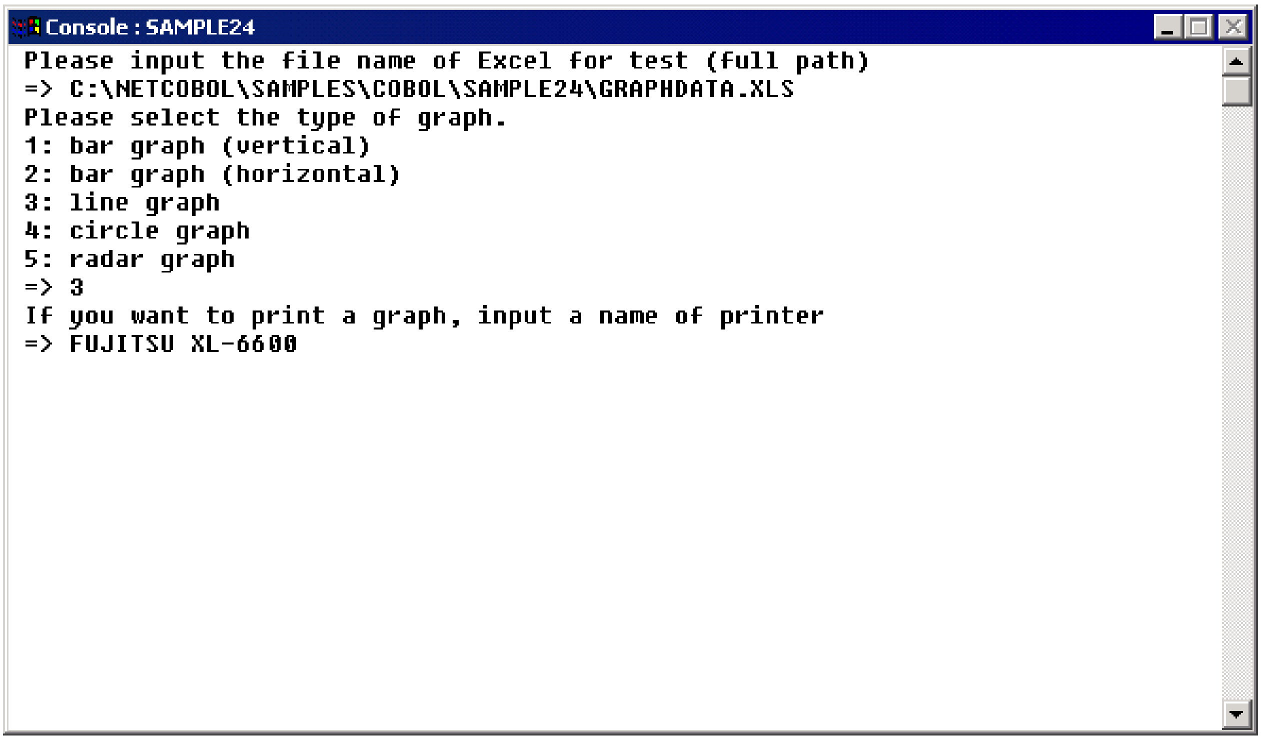
   The specified Excel file is opened and the program controls Excel. The following message is displayed on the COBOL console and the system waits for input again.



```
Please input the file name of Excel for test (full path)
=> C:\NETCOBOL\SAMPLES\COBOL\SAMPLE24\GRAPHDATA.XLS
Please select the type of graph.
1: bar graph (vertical)
2: bar graph (horizontal)
3: line graph
4: circle graph
5: radar graph
=>
```

3. Input the type of a graph to be created.

   The specified graph is plotted. The following message is displayed on the COBOL console and the system waits for input again.



```
Please input the file name of Excel for test (full path)
=> C:\NETCOBOL\SAMPLES\COBOL\SAMPLE24\GRAPHDATA.XLS
Please select the type of graph.
1: bar graph (vertical)
2: bar graph (horizontal)
3: line graph
4: circle graph
5: radar graph
=> 3
If you want to print a graph, input a name of printer
=> FUJITSU XL-6600
```

4. If the graph is to be printed, specify a printer name. If it is not to be printed, press the ENTER key without inputting data.

   Excel is terminated and the program is terminated. The graph is printed before terminating Excel when a printer name is specified.

A.26 Sample 25: Creating a COBOL COM Server Program

Sample 25 shows an example where the COBOL COM server function is used to enable a COBOL program to be used as a COM server.

The COBOL class definition can be shifted to the COM server without change, by using the COBOL COM server function. The selected class is published as the COM server interface simply by setting the necessary information with the Project Manager's COM sever creation function and executing rebuild.

For details on the COBOL COM function, refer to Chapter 24 in the "NetCOBOL User’s Guide".

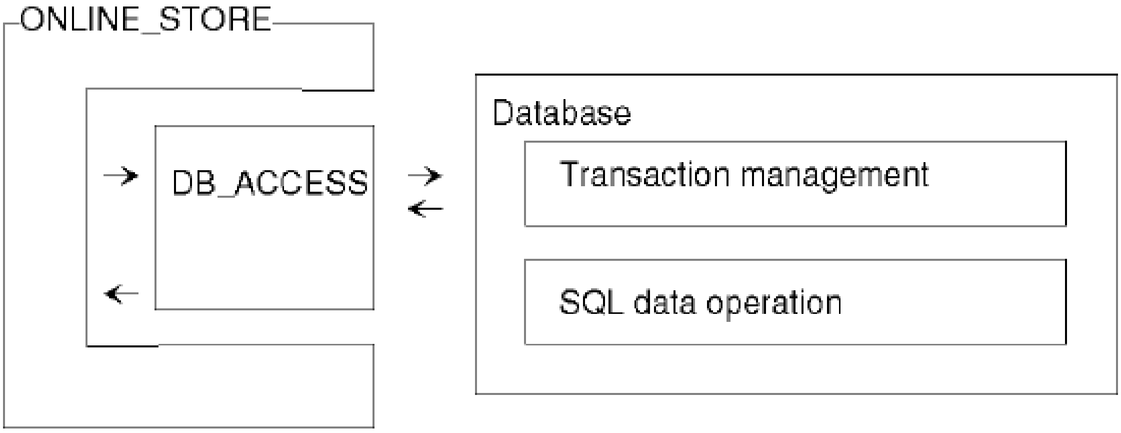
This program accesses a database via an ODBC driver, so the following products are necessary to use this program:

- Database
- Products necessary for database access with ODBC
- ODBC driver
- ODBC driver manager

For details of database access using an ODBC driver, refer to Chapter 19 in the "NetCOBOL User’s Guide".

Overview

The sample program consists of two class definitions.



The COBOL ONLINE_STORE class is the COM server class. The ONLINE_STORE class provides the following functions for constructing an online store application:

- Authorization processing
- Stock check
- Order registration
- Order exact calculation

In addition to COBOL programs, programs generated with Visual C++(R), programs generated with Visual Basic(R), and active server pages (ASP) Visual Basic(R) Scripting Edition (VBScript) can be used as client programs to use these functions. See Sample 26 and Sample 27 for examples of the COBOL client and ASP client.

Available Programs

- DB_ACCESS.COB (COBOL source file)
- ONLINE_STORE.COB (COBOL source file)
- STORESV1.PRJ (project file)
- STORESV1.CBI (compiler option file)
- STORESV1_DLL.CSI (COM server information file)
- STORESV1.DEF (module definition file)

Applicable COBOL Functions

- COM server function
- Remote database access
- *COM-ARRAY class

Applicable COBOL Statements

The IF, INVOKE, INITIALIZE, SET, MOVE, and PERFORM statements are used.

The embedded SQL statements (COMMIT, CONNECT, INSERT, SELECT, UPDATE, ROLLBACK, and DISCONNECT) are used.

Operation necessary before executing the program

Construct an environment where a database can be accessed via the ODBC driver.

Set a default server to be connected and create the four tables shown below in a database on the server.

- CUSTOMER table

  Use the format shown below to create this table.

USERID	PASSWORD
Variable-length character 32 bytes	Variable-length character 32 bytes

Store the data items shown below in the client table.

USERID	PASSWORD
USER0001	USER0001
USER0002	USER0002
USER0003	USER0003
USER0004	USER0004
USER0005	USER0005
USER0006	USER0006
USER0007	USER0007
USER0008	USER0008
USER0009	USER0009
USER0010	USER0010

- STOCK table

Use the format shown below to create this table.

PRODUCTNUMBER	QUANTITY
Fixed-length character 10 bytes	Fixed-length character 10 digits

Store the data items shown below in the stock table.

PRODUCTNUMBER	QUANTITY
FMV2TXH111	900000
FMV2TXH161	100000
FMV2TXH151	500000
FMV2TXF111	45000
FMV2TXF161	300000
FMV2TXF151	60000
FMV2DXH111	90000
FMV2DXH161	55000
FMV2DXH151	990000
FMV2DXF111	10000
FMV2DXF161	777700
FMV2DXF151	200000
FMV2DXD111	690000
FMV2DXD161	870000
FMV2DXD151	619000

PRODUCTNUMBER	QUANTITY
FMV2DXA111	2900000
FMV2DXA161	8760000
FMV2DXA151	100000
FMV3NA3LC0	10000
FMV3NA3LC6	300

- ORDERTABLE

  Use the format shown below to create this table.

ORDERNUMBER	USERID	DATE
Fixed-length character 12 bytes	Variable-length character 32 bytes	Fixed-length character 14 bytes

  No data need be stored in the ORDERTABLE table.

- ORDERDETAIL table

  Use the format shown below to create this table.

ORDERNUMBER	PRODUCTNUMBER	QUANTITY
Fixed-length character 12 bytes	Fixed-length character 10 bytes	Decimal integer 10 digits

  No data need be stored in the ORDERDETAIL table.

- - Use the ODBC information file setting tool (SQLODBCS.EXE) to create an ODBC information file (assumed to be C:\DBMSACS.INF).

Building and Rebuilding

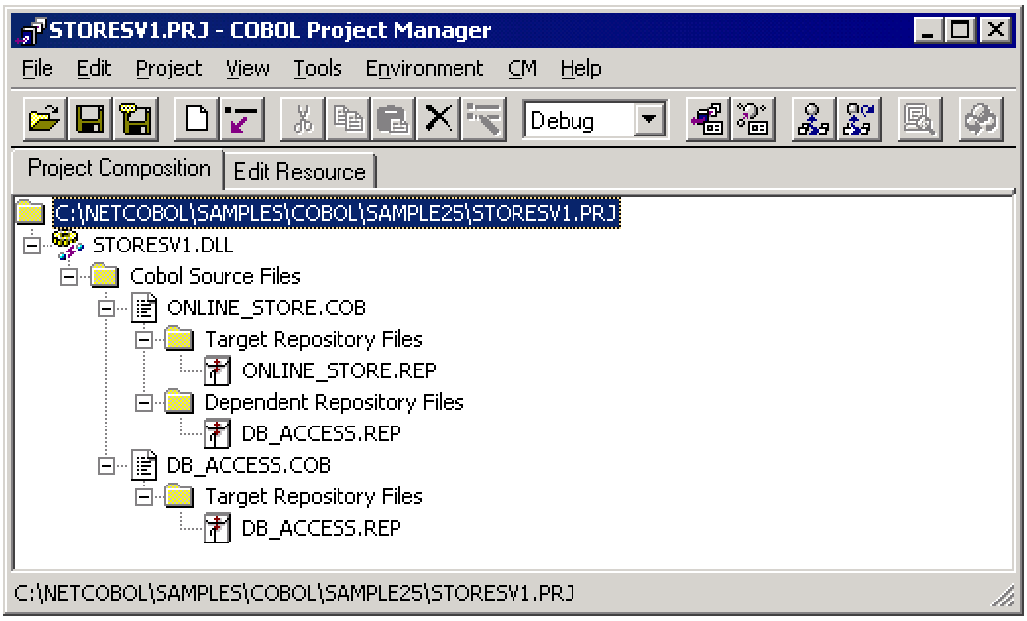
The Project Manager's Build function is used to compile and link this program.

Folder C:\NetCOBOL is assumed below as the location where NetCOBOL is installed.

Change the folder name C:\NetCOBOL to the name of the folder where NetCOBOL is installed.

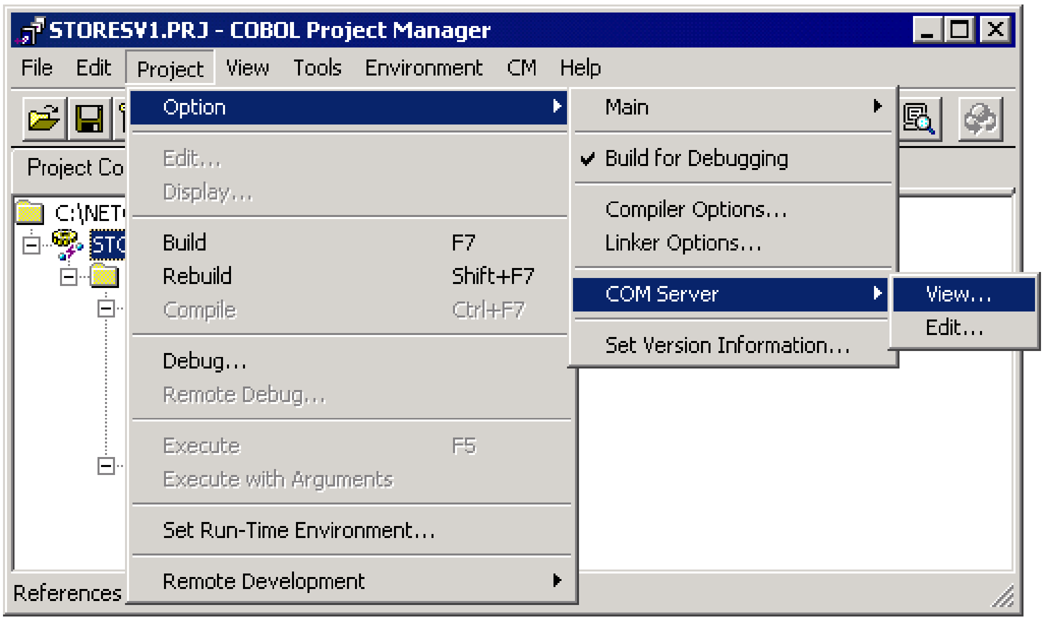
1. Start the Project Manager.

2. Open the project file SAMPLE25.PRJ.

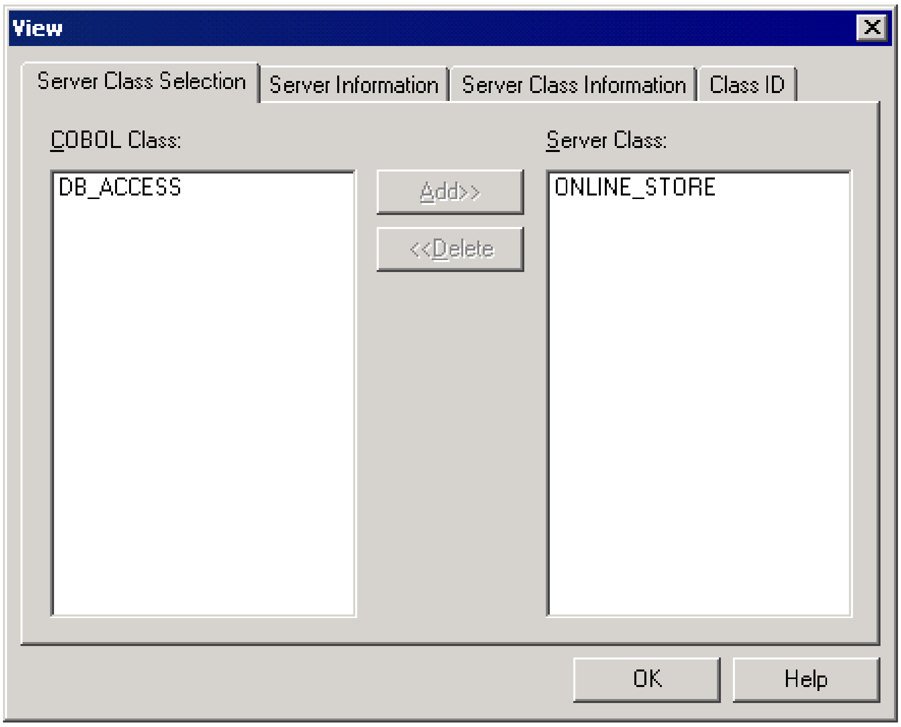

STORESV1.PRJ - COBOL Project Manager
File Edit Project View Tools Environment CM Help
Debug
Project Composition
Edit Resource
C:\NETCOBOL\SAMPLES\COBOL\SAMPLE25\STORESV1.PRJ
STORESV1.DLL
Cobol Source Files
ONLINE_STORE.COB
Target Repository Files
ONLINE_STORE.REP
Dependent Repository Files
DB_ACCESS.REP
DB_ACCESS.COB
Target Repository Files
DB_ACCESS.REP
C:\NETCOBOL\SAMPLES\COBOL\SAMPLE25\STORESV1.PRJ

3. Check the set COM server information.

Select a target file (STORESV1.DLL) and select "View" from the "Project-Option-COM server" menu.



The "View" dialog is opened and the server information can be referenced.



DB_ACCESS

ONLINE_STORE

4. Select "Build" from the "Project" menu.

   After build termination, check that STORESV1.DLL is created.

Setup the Server Program Execution Environment

1. Select "Run-time Environment Setup Tool" from the "Tools" menu of Project Manager.

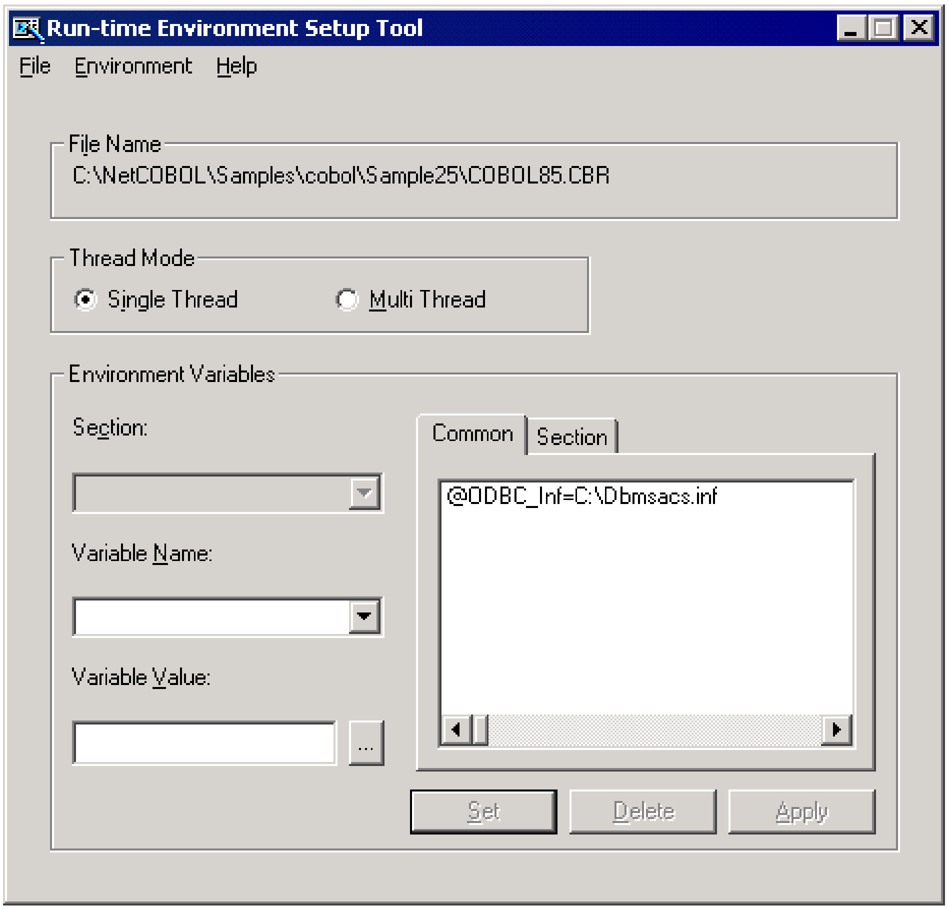
   The run-time environment setup tool is displayed.

2. Select "Open" on the "File" menu and create an object initialization file (COBOL85.CBR) in the folder that contains the dynamic link library (STORESV1.DLL).

3. Select a common tab and set data as shown below:

   - Specify an ODBC information file name in environment variable information

     @ODBC_Inf (ODBC information file specification).



File Name: C:\NetCOBOL\Samples\cobol\Sample25\COBOL85.CBR

Thread Mode: ● Single Thread ○ Multi Thread

@ODBC_Inf=C:\Dbmsacs.inf

4. Click the Apply button.

   The data is saved in the object initialization file.

5. Select File, Exit to terminate the run-time environment setup tool.

Registering the COM Server

The created COBOL application must be registered in the Windows system to use it as a COM server. There are two registration methods depending on how the COM server is used.

- When the COM server and COM client are used on the same machine

  Use REGSVR32.EXE to register the COBOL application in the system registry.

  For details, refer to Section the "Registering or Deleting a COM Server" in the "NetCOBOL User's Guide".

- When the COM server is used from a COM client in a remote network-connected machine.

  Use Microsoft Transaction Server (MTS). Use MTS explorer to register the COBOL application in the system registry and MTS. For details, refer to Section the "Registration in the MTS Environment" in the "NetCOBOL User's Guide".

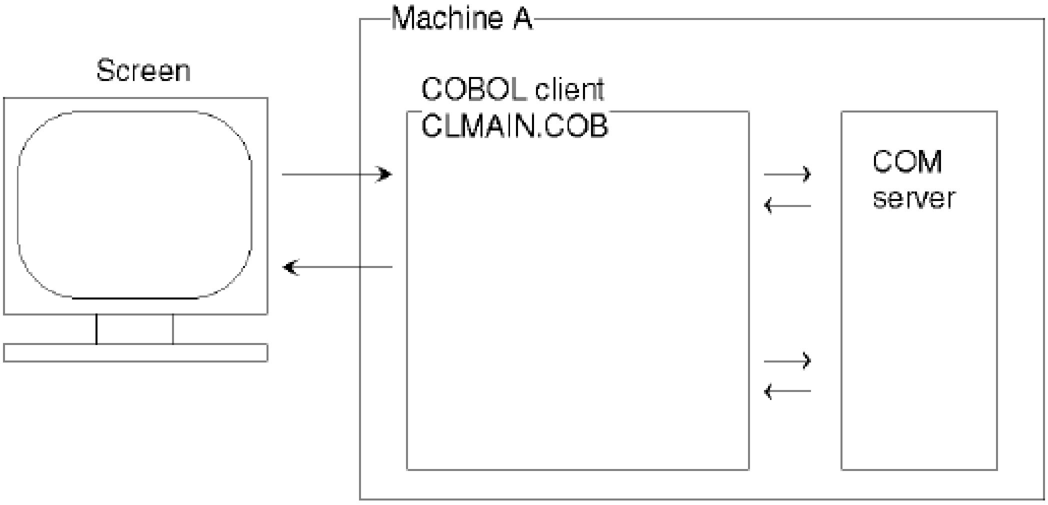
A.27 Sample 26: Using the COM Linkage-COBOL Server Program (COBOL Client)

Sample 26 demonstrates a client program that uses the COBOL server program in Sample 25 with the COBOL COM client function.

For details on the COBOL COM function, refer to Chapter 24 in the "NetCOBOL User's Guide".

Overview

The sample program uses the COBOL server program generated in Sample 25 to create the online store application. The client program accepts data input on the screen by using the screen operation function and requests the server program to process the accepted data. The result of the server program processing is displayed on the screen.



Available Programs

- CLMAIN.COB (COBOL source program)
- ORDERSHEET-INFO.CBL (COBOL library file)
- PRODUCT-TABLE.CBL (COBOL library file)
- SCREENS.CBL (COBOL library file)
- SAMPLE26.PRJ (Project file)
- SAMPLE26.CBI (compiler option file)

Applicable COBOL Functions

- Screen function
- COM client function
- *COM-ARRAY class

Applicable COBOL Statements

The ACCEPT (screen function), DISPLAY (screen function), EVALUATE, INVOKE, IF, PERFORM, and SET statements are used.

Operations Necessary Before Executing the Program

When a COM sever in a remote network-connected machine is used, install the server information in the machine where this program is to be executed as follows:

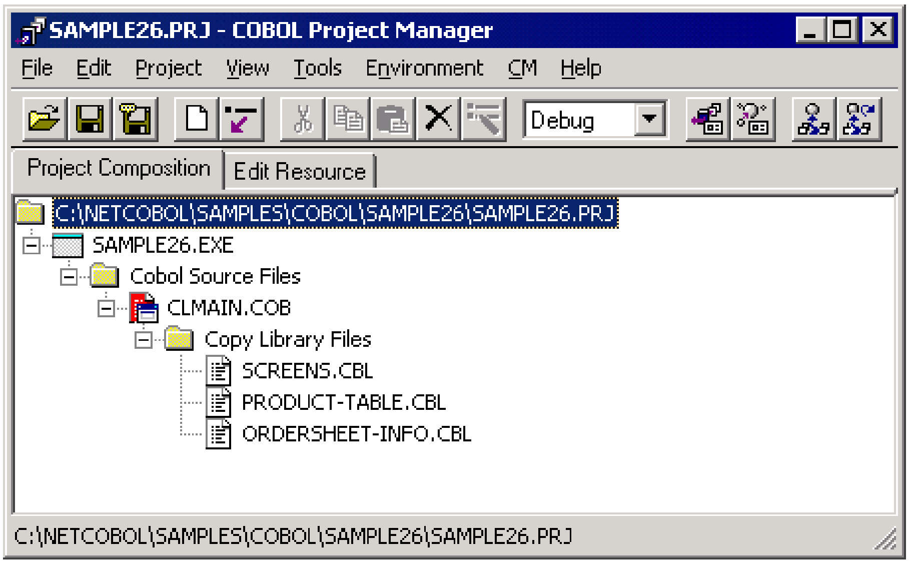
1. Generate a client information installation program in the machine where the COM server is registered. For details on generation of the data, refer to the section "Installation in the Client Machine" in the "NetCOBOL User’s Guide".
2. Execute the client information installation program on the machine where this program is to be executed. If this program is to be executed on the same machine as the COM server, this operation is unnecessary.

Building and Rebuilding

The Project Manager's build function is used to compile and link this program.

In the following screen snapshots, it is assumed that NetCOBOL was installed in folder C:\NetCOBOL. Change the folder name C:\NetCOBOL to the name of the folder where NetCOBOL is installed on your machine.

1. Start the Project Manager.
2. Open the project file SAMPLE26.PRJ.



C:\NETCOBOL\SAMPLES\COBOL\SAMPLE26\SAMPLE26.PRJ
SAMPLE26.EXE
Cobol Source Files
CLMAIN.COB
Copy Library Files
SCREENS.CBL
PRODUCT-TABLE.CBL
ORDERSHEET-INFO.CBL

C:\NETCOBOL\SAMPLES\COBOL\SAMPLE26\SAMPLE26.PRJ

3. Select the project file and select "Compiler Options" from the "Project-Option" menu.

   The "Compiler Options" dialog is displayed.

4. Click the COM Server button in the "Compiler Options" dialog.

   The "COM Server" dialog is displayed.

5. Specify STORESV1.DLL (type library) for COM server name STORESV1. Check the entry, then click the OK button.

   The "Compiler Options" dialog is redisplayed. Click the OK button to redisplay the Project Manager window.

6. Select "Build" from the "Project" menu.

   After build termination, check that SAMPLE26.EXE is created.

Configuring the Server Program Execution Environment

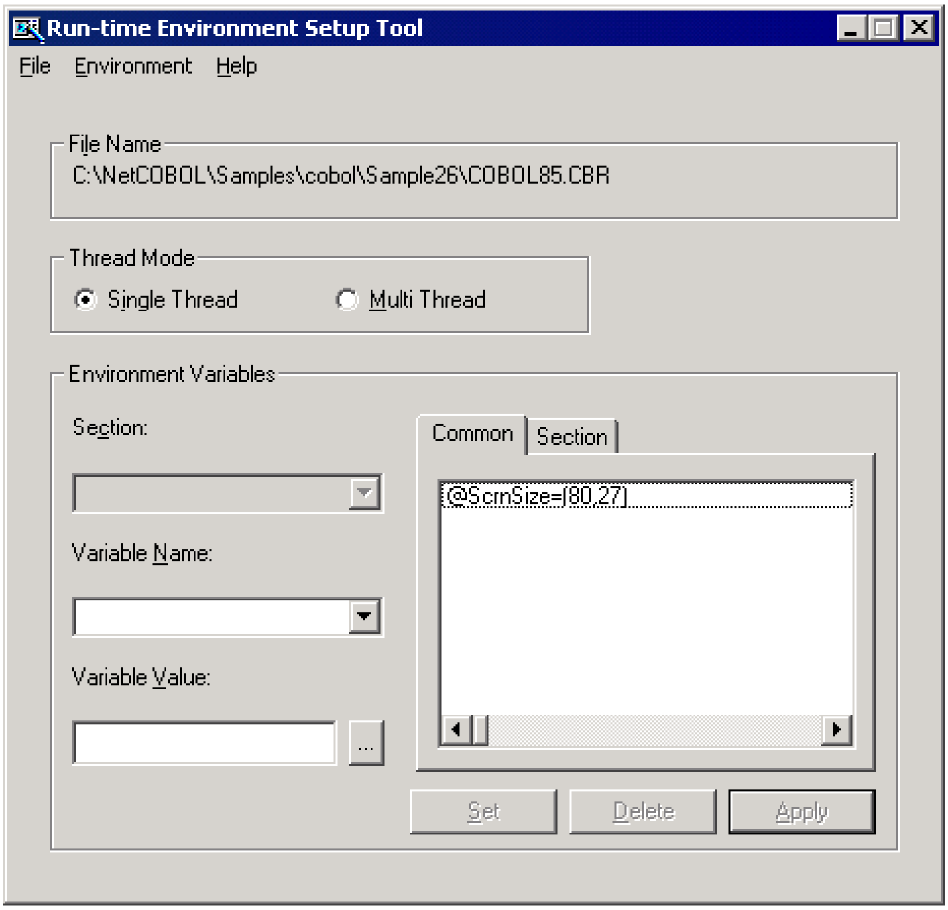
1. Select "Run-time Environment Setup Tool" from the "Tools" menu of the Project Manager.

   The run-time environment setup tool is displayed.

2. Select "Open" on the "File" menu and create an object initialization file (COBOL85.CBR) in the folder that contains the executable program (SAMPLE26.EXE).

3. Select the Common tab and enter data as shown below:

   - Specify "(80,27)" in the environment variable information @ScrnSize (logical screen size in screen operation).



File Name
C:\NetCOBOL\Samples\cobol\Sample26\COBOL85.CBR

Thread Mode
● Single Thread ○ Multi Thread

@ScrnSize=(80,27)

   - If the COBOL client program is executed on the same machine as the server program, also set the server program execution environment information by specifying the ODBC information file name in environment variable information @ODBC_Inf (ODBC information file specification).

4. Click the Apply button.

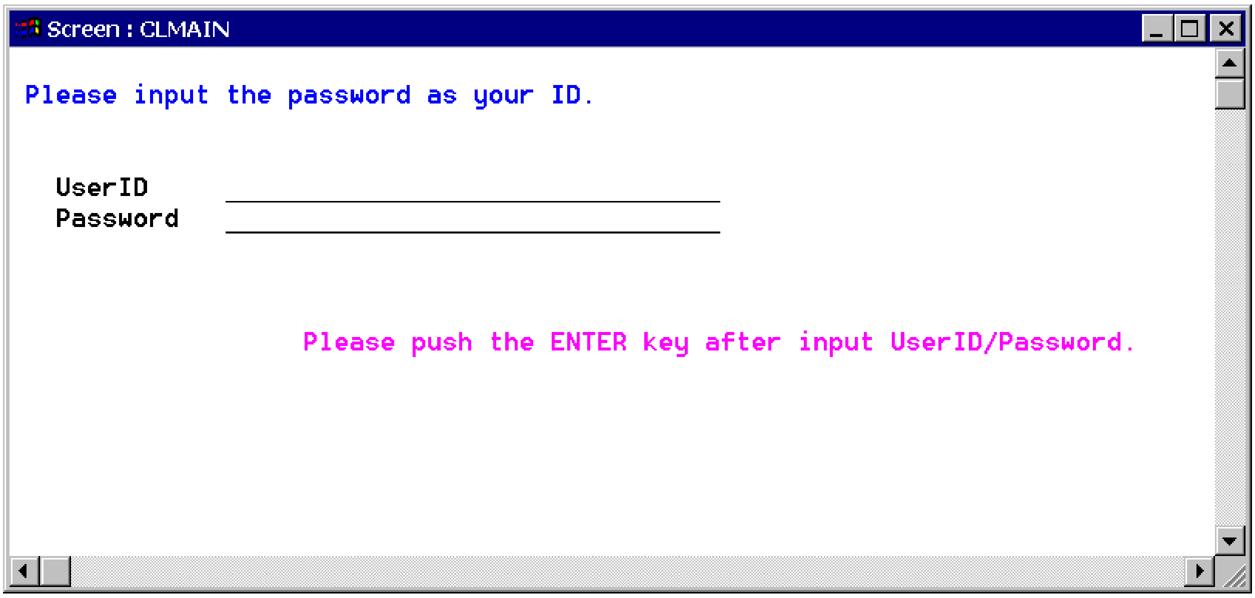
   The data is saved in the object initialization file.

5. Select "Exit" on the "File" menu to terminate the run-time environment setup tool.

Executing

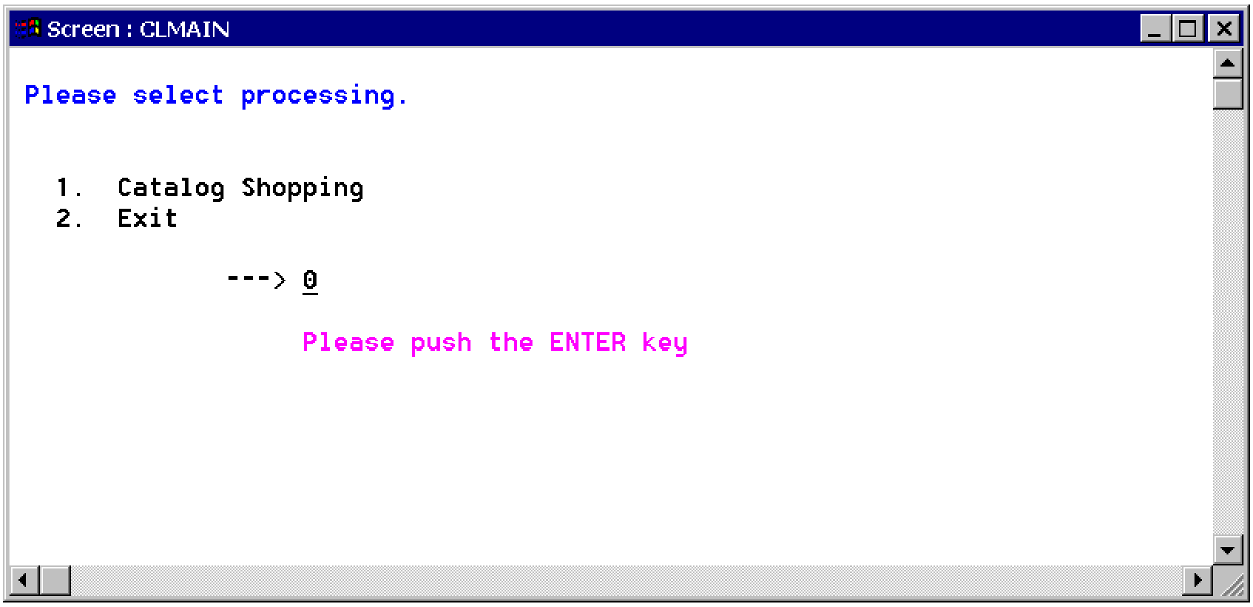
1. Select "Execute" on the "Project" menu.

   The screen shown below is displayed. Input the user ID and password and press the ENTER key (use the CURSOR or TAB key to move among the input fields). Valid user IDs are USER0001 to USER0010 and the password is the same as the user ID. Note that the entered password is not displayed.



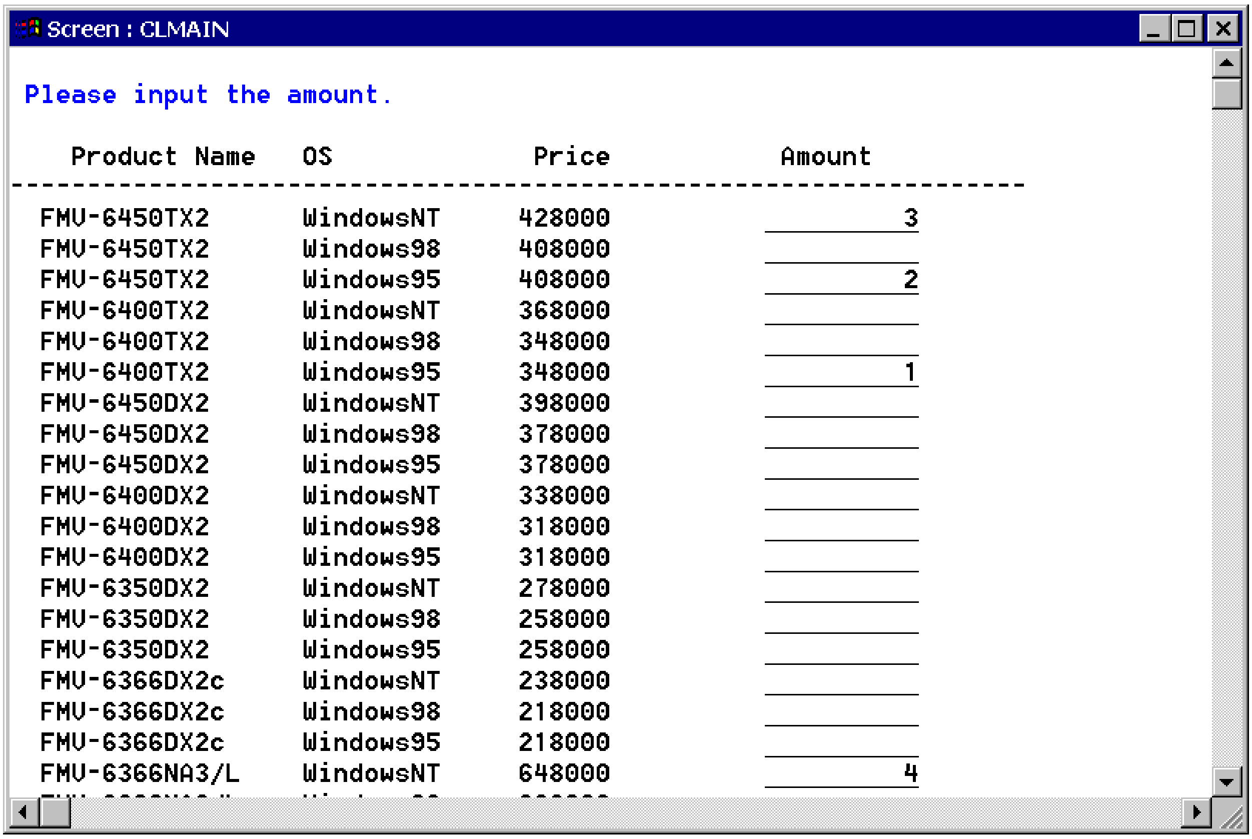
```
Please input the password as your ID.


    UserID       ________________________________
    Password     ________________________________



                                  Please push the ENTER key after input UserID/Password.
```

2. The menu screen is displayed. Input "1" and press the ENTER key.



```
Please select processing.


    1.   Catalog Shopping
    2.   Exit

                 ---> 0

                          Please push the ENTER key
```

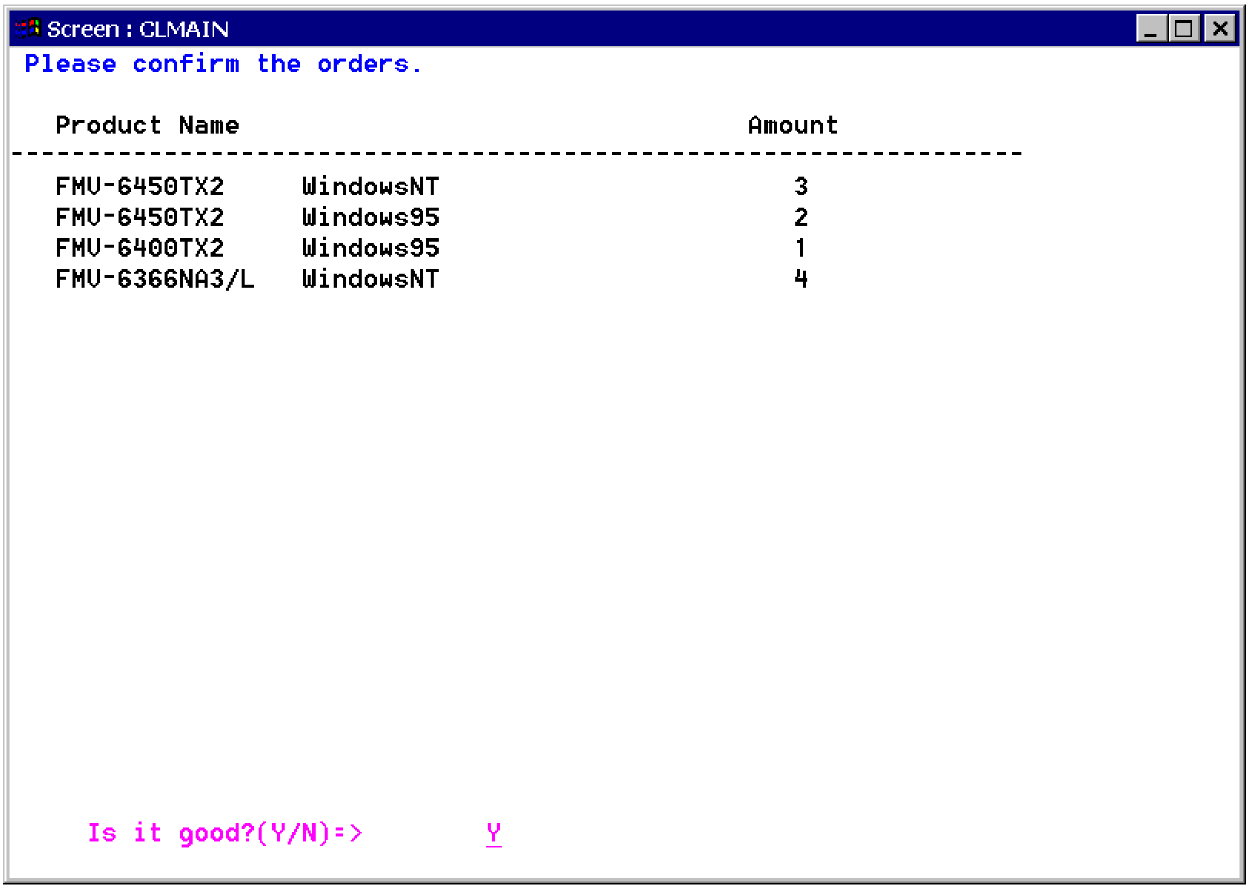
3. The catalog screen is displayed. Input the number of items to be ordered. Use the CURSOR or TAB key to move among the input fields. After entering the data, press the ENTER key.



Please input the amount.

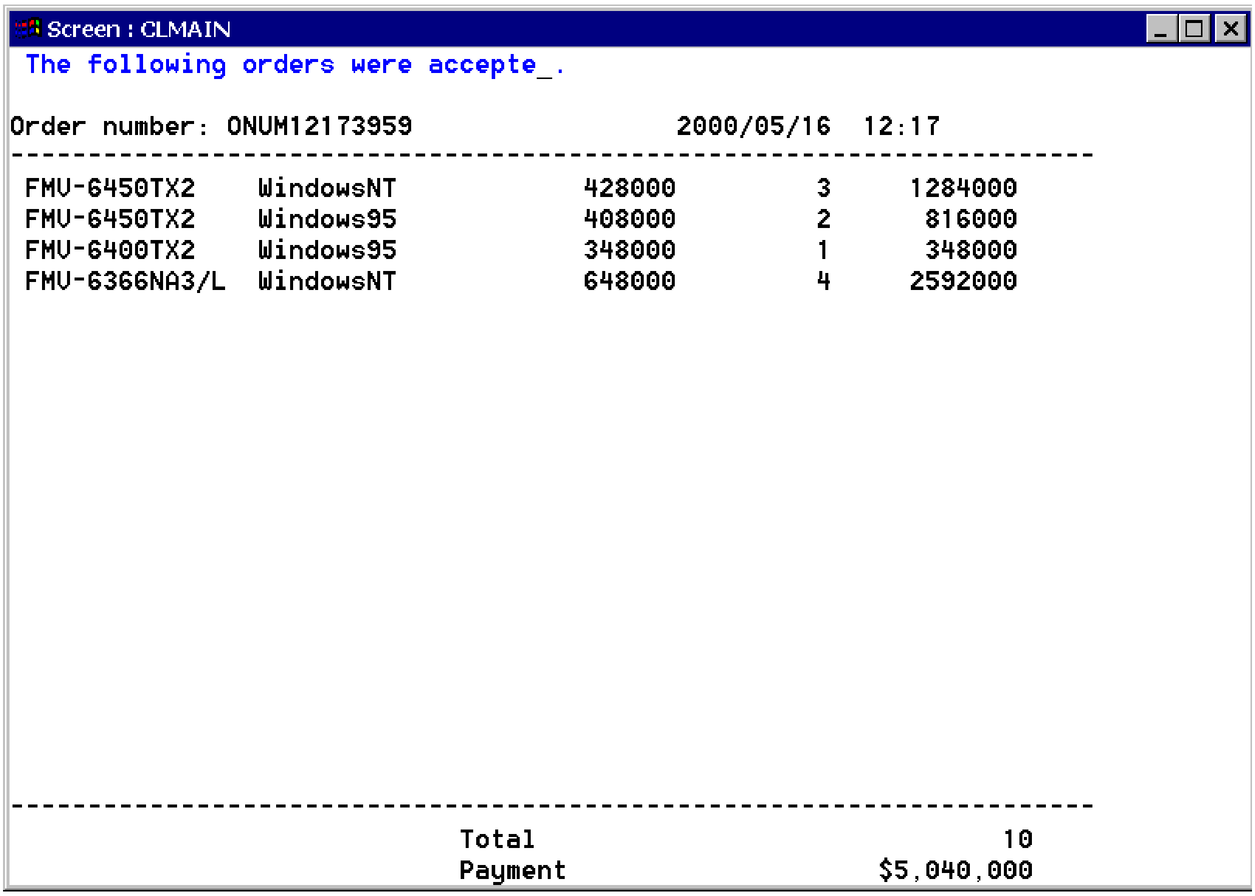
Product Name	OS	Price	Amount
FMV-6450TX2	WindowsNT	428000	3
FMV-6450TX2	Windows98	408000	
FMV-6450TX2	Windows95	408000	2
FMV-6400TX2	WindowsNT	368000	
FMV-6400TX2	Windows98	348000	
FMV-6400TX2	Windows95	348000	1
FMV-6450DX2	WindowsNT	398000	
FMV-6450DX2	Windows98	378000	
FMV-6450DX2	Windows95	378000	
FMV-6400DX2	WindowsNT	338000	
FMV-6400DX2	Windows98	318000	
FMV-6400DX2	Windows95	318000	
FMV-6350DX2	WindowsNT	278000	
FMV-6350DX2	Windows98	258000	
FMV-6350DX2	Windows95	258000	
FMV-6366DX2c	WindowsNT	238000	
FMV-6366DX2c	Windows98	218000	
FMV-6366DX2c	Windows95	218000	
FMV-6366NA3/L	WindowsNT	648000	4

4. The order check screen is displayed. Input "Y" and press the ENTER key.



```
Please confirm the orders.

   Product Name                                        Amount
------------------------------------------------------------------
   FMV-6450TX2     WindowsNT                              3
   FMV-6450TX2     Windows95                              2
   FMV-6400TX2     Windows95                              1
   FMV-6366NA3/L   WindowsNT                              4











     Is it good?(Y/N)=>        Y
```

5. The order list screen is displayed. The (2) menu screen is redisplayed when the ENTER key is pressed.



```
 The following orders were accepte_.

Order number: ONUM12173959                 2000/05/16  12:17
----------------------------------------------------------------------
 FMV-6450TX2    WindowsNT              428000        3      1284000
 FMV-6450TX2    Windows95              408000        2       816000
 FMV-6400TX2    Windows95              348000        1       348000
 FMV-6366NA3/L  WindowsNT              648000        4      2592000

----------------------------------------------------------------------
                             Total                               10
                             Payment                     $5,040,000
```

6. To terminate the processing, input "2" on the menu screen and press the ENTER key.

A.28 Sample 27: Using the COM Linkage-COBOL Server Program (ASP Client)

Sample 27 shows an example where a COM server created using the COBOL COM server function is used by calling it from active server pages (ASP) Visual Basic(R) Scripting Edition (VBScript).

For details on ASP and VBScript, refer to commercially available handbooks.

- To use this program, the following products are necessary:
  - - Microsoft(R) Windows(R) 2000 Server operating system
  - - Microsoft(R) Windows(R) 2000 Advanced Server operating system
  - - Microsoft(R) Windows Server(R) 2003 Standard Edition
  - - Microsoft(R) Windows Server(R) 2003 Enterprise Edition
  - - Microsoft(R) Windows Server(R) 2008 Standard Edition
  - - Microsoft(R) Windows Server(R) 2008 Enterprise Edition
- Microsoft(R) Internet Information Server 5.0 or later

Overview

ASP is one of the methods of constructing dynamic Web applications by embedding script language in the HTML document.

A COM server object can be created in ASP VBScript by using a server object in which the CreateObject method is installed. The method provided by the COM server can be called from the created object.

Use this function to create online store Web applications using the COBOL server program in Sample 25.

Available Programs

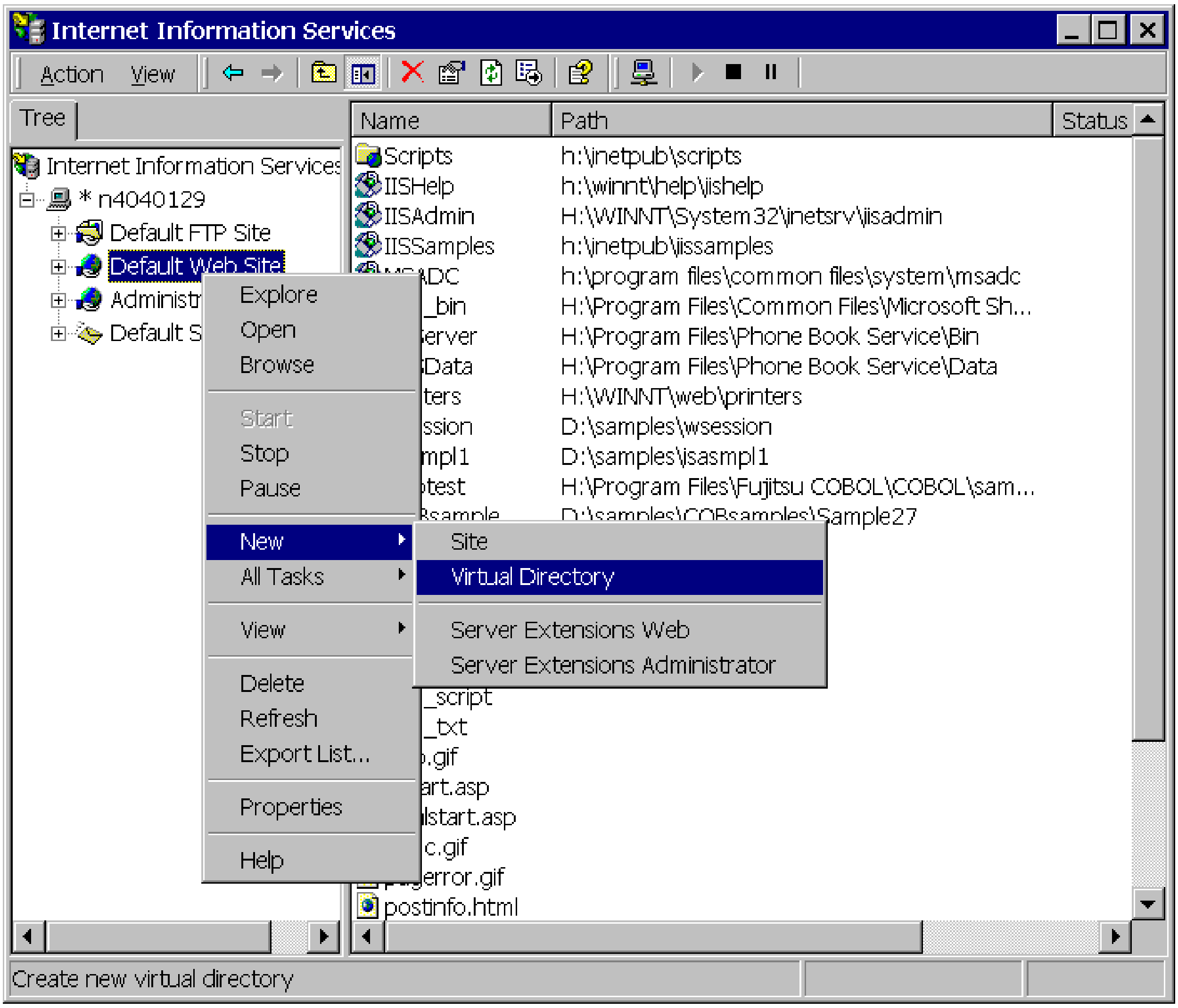
- MENU.ASP (ASP page file)
- AUTH.ASP (ASP page file)
- CATALOG.ASP (ASP page file)
- CONFIRM.ASP (ASP page file)
- ORDER.ASP (ASP page file)
- STYLE.CSS (style sheet file)
- CATALOGTITLE.GIF (image file)
- FJLOGO.GIF (image file)

Operations Necessary Before Execution

The COM server program created in Sample 25 is used in this sample. Build the COM server program, register it as the COM server, and configure the execution environment information.

Next, register Sample 27 with the Internet Service Manager as follows:

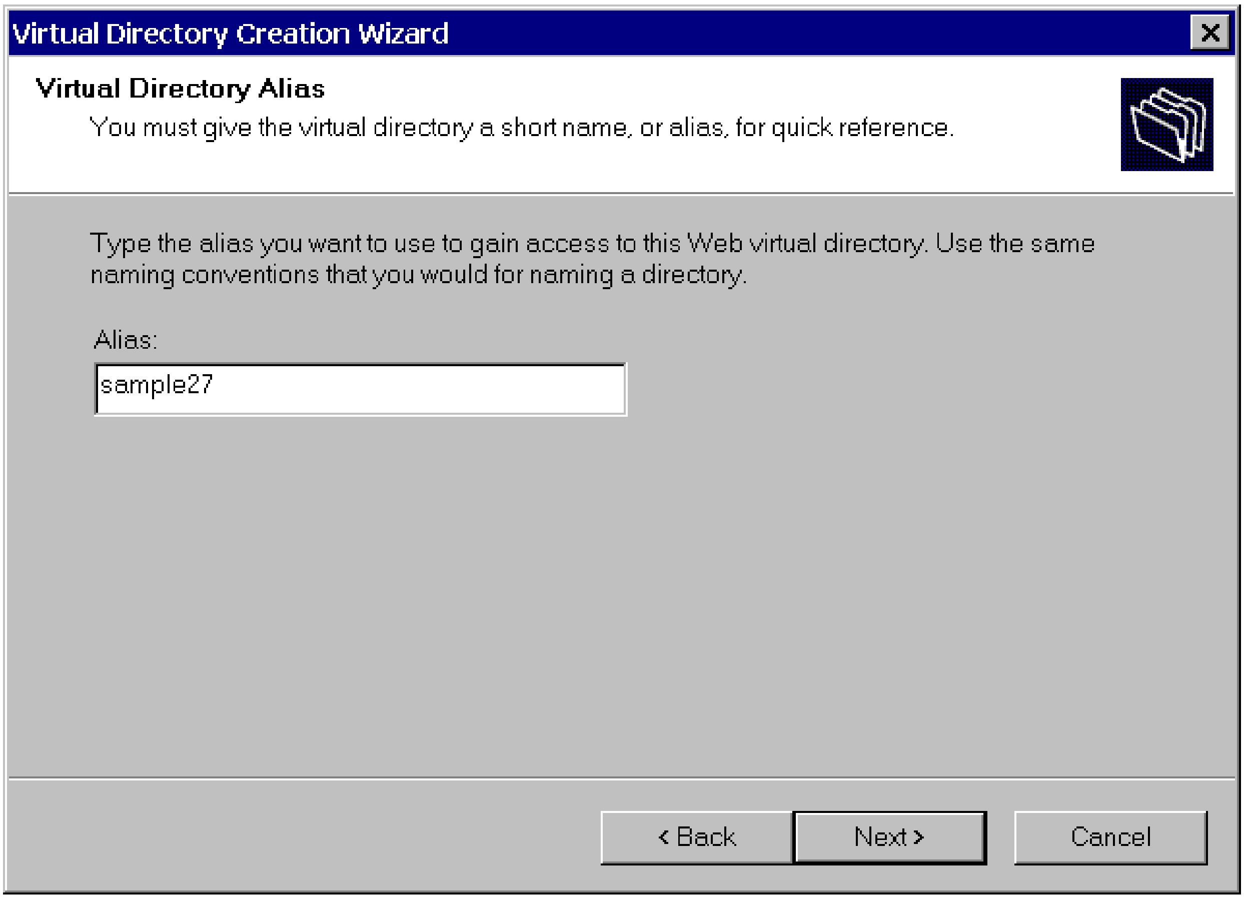
1. Activate the Internet Service Manager, select "specified Web site", and select "Virtual Directory" from the "New Creation" field on the "Context" menu.



Name	Path	Status
Scripts	h:\inetpub\scripts	
IISHelp	h:\winnt\help\iishelp	
IISAdmin	H:\WINNT\System32\inetsrv\iisadmin	
IISSamples	h:\inetpub\iissamples	
MSADC	h:\program files\common files\system\msadc	
_bin	H:\Program Files\Common Files\Microsoft Sh...	
erver	H:\Program Files\Phone Book Service\Bin	
Data	H:\Program Files\Phone Book Service\Data	
ters	H:\WINNT\web\printers	
ssion	D:\samples\wsession	
mpl1	D:\samples\isasmpl1	
test	H:\Program Files\Fujitsu COBOL\COBOL\sam...	
Bsample	D:\samples\COBsamples\Sample27	
_script		
_txt		
.gif		
art.asp		
lstart.asp		
c.gif		
error.gif		
postinfo.html		

Create new virtual directory

2. Input a virtual directory name, and then specify the physical path of Sample 27 with the ASP page file.



Virtual Directory Creation Wizard

Virtual Directory Alias

You must give the virtual directory a short name, or alias, for quick reference.

Type the alias you want to use to gain access to this Web virtual directory. Use the same naming conventions that you would for naming a directory.

Alias: sample27

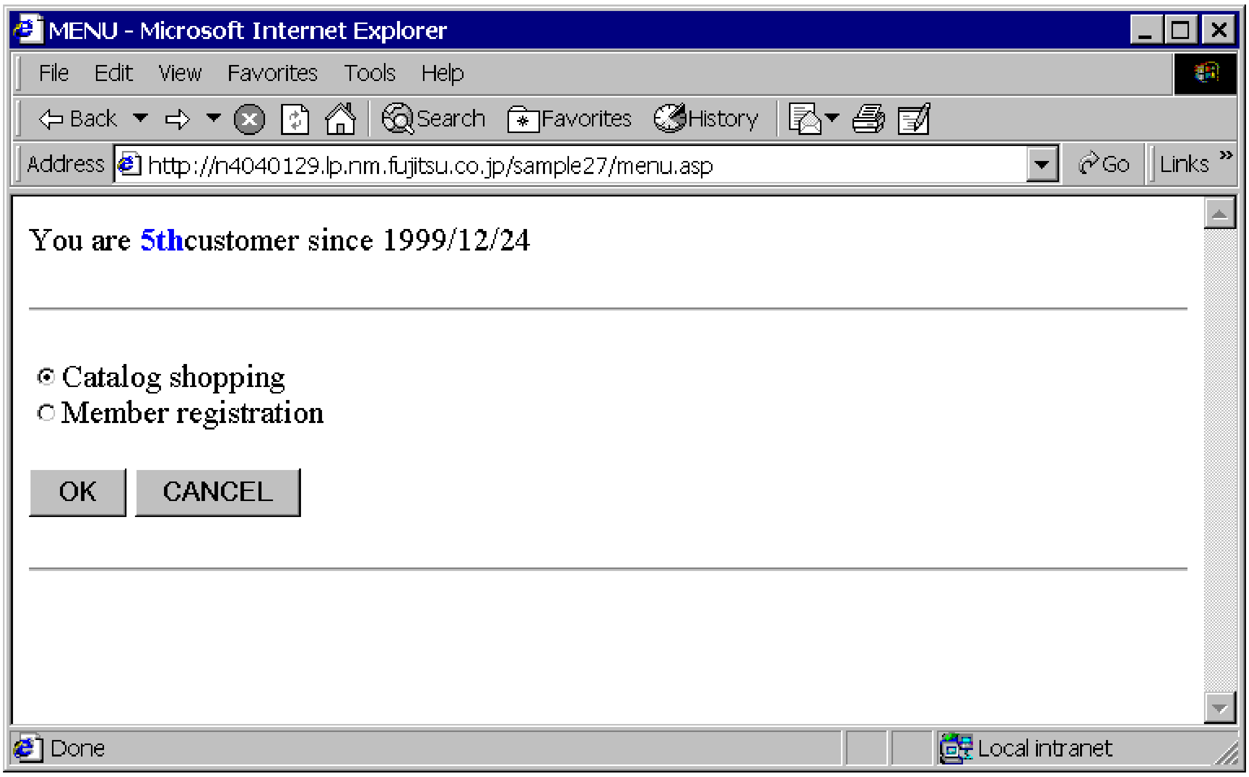
Execution

Domain name "user" and virtual directory name "sample27" are assumed.

Microsoft Internet Explorer is used as the WWW browser.

1. Input the URL as shown below.

   The menu screen is displayed. Select "Catalog Shopping" and click the OK button.


MENU - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Go Links
OK
CANCEL
Done
Local intranet

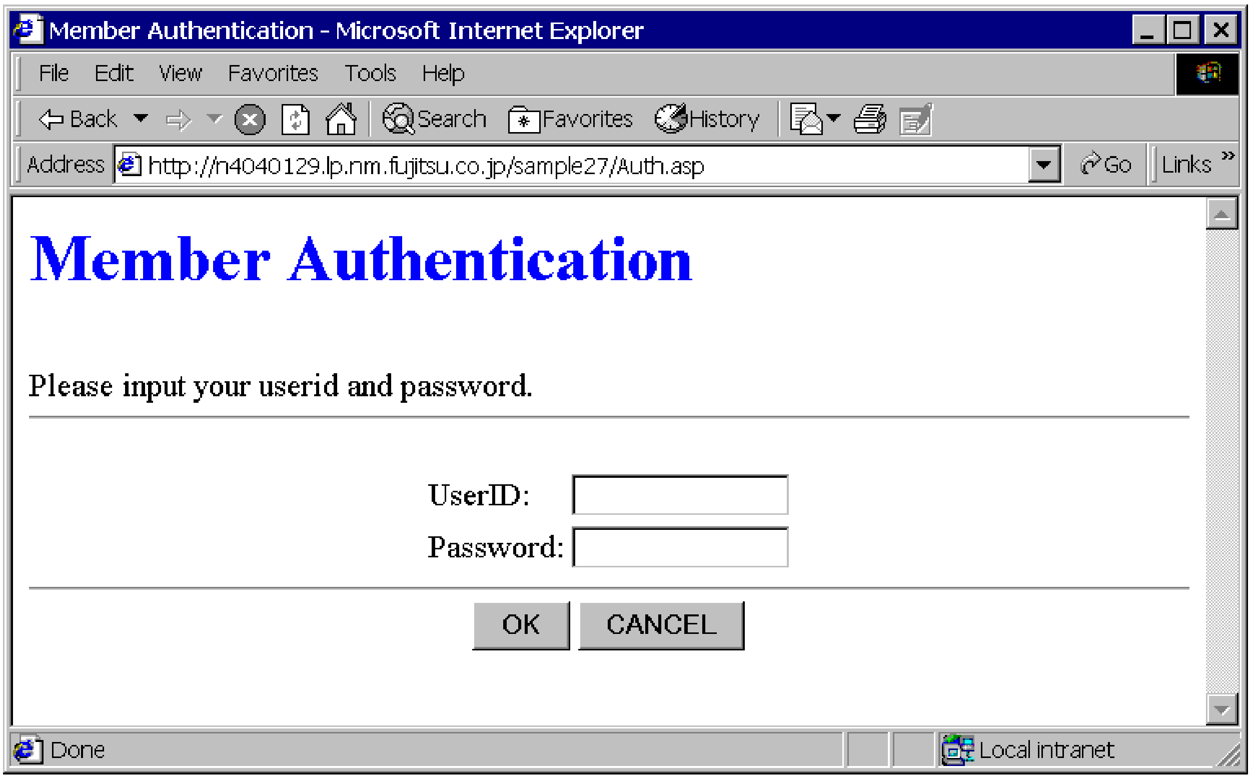
Address: http://n4040129.lp.nm.fujitsu.co.jp/sample27/menu.asp

You are 5thcustomer since 1999/12/24

● Catalog shopping
○ Member registration

2. The member authorization screen is displayed. Input a user ID and password and click the OK button. Valid user ID's are USER0001 to USER0010. The passwords are identical to the user ID.

   Note that the entered password is not displayed.


Member Authentication - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Go Links
OK
CANCEL
Done
Local intranet

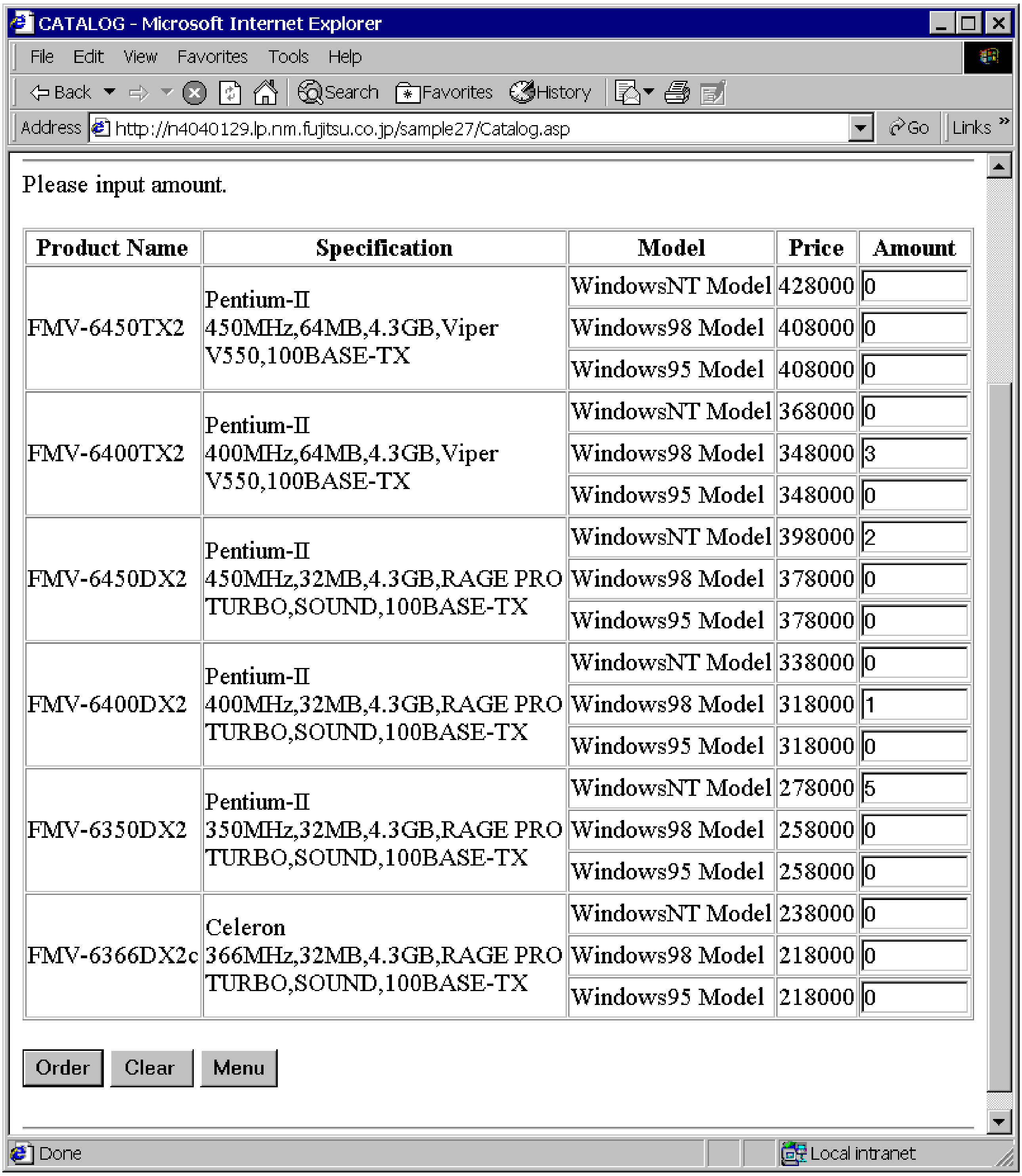
Address: http://n4040129.lp.nm.fujitsu.co.jp/sample27/Auth.asp

Member Authentication

Please input your userid and password.

UserID:
Password:

3. The catalog screen is displayed. Input the number of items to be ordered and click the Order button.

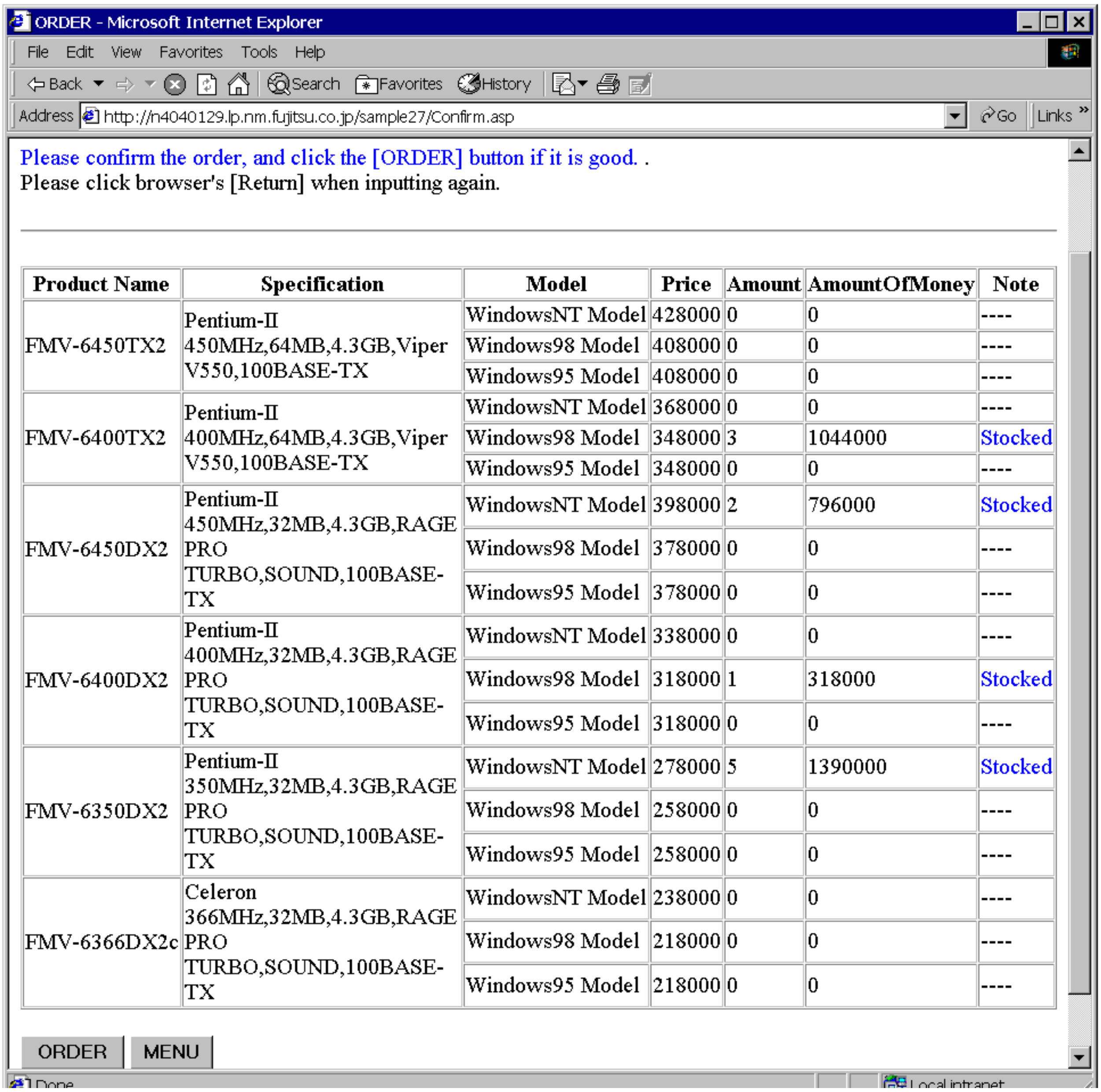


CATALOG - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://n4040129.lp.nm.fujitsu.co.jp/sample27/Catalog.asp
Go Links
Done
Local intranet

Please input amount.

Product Name	Specification	Model	Price	Amount
FMV-6450TX2	Pentium-II 450MHz,64MB,4.3GB,Viper V550,100BASE-TX	WindowsNT Model	428000	0
		Windows98 Model	408000	0
		Windows95 Model	408000	0
FMV-6400TX2	Pentium-II 400MHz,64MB,4.3GB,Viper V550,100BASE-TX	WindowsNT Model	368000	0
		Windows98 Model	348000	3
		Windows95 Model	348000	0
FMV-6450DX2	Pentium-II 450MHz,32MB,4.3GB,RAGE PRO TURBO,SOUND,100BASE-TX	WindowsNT Model	398000	2
		Windows98 Model	378000	0
		Windows95 Model	378000	0
FMV-6400DX2	Pentium-II 400MHz,32MB,4.3GB,RAGE PRO TURBO,SOUND,100BASE-TX	WindowsNT Model	338000	0
		Windows98 Model	318000	1
		Windows95 Model	318000	0
FMV-6350DX2	Pentium-II 350MHz,32MB,4.3GB,RAGE PRO TURBO,SOUND,100BASE-TX	WindowsNT Model	278000	5
		Windows98 Model	258000	0
		Windows95 Model	258000	0
FMV-6366DX2c	Celeron 366MHz,32MB,4.3GB,RAGE PRO TURBO,SOUND,100BASE-TX	WindowsNT Model	238000	0
		Windows98 Model	218000	0
		Windows95 Model	218000	0

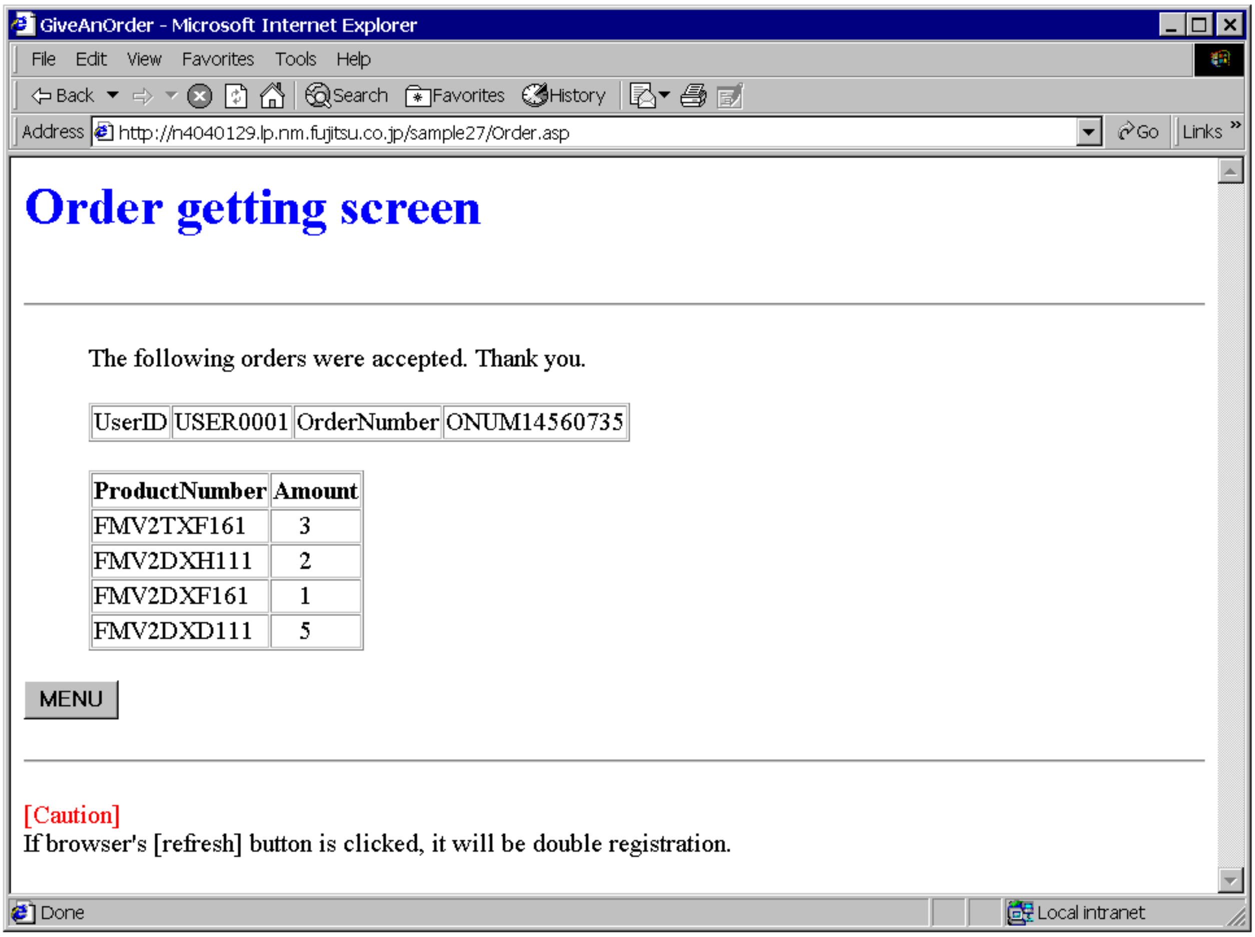
4. The order check screen is displayed. Click the Order Issue button.


ORDER - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://n4040129.lp.nm.fujitsu.co.jp/sample27/Confirm.asp
Go Links
ORDER
MENU
Done
Local intranet

Please confirm the order, and click the [ORDER] button if it is good. .
Please click browser's [Return] when inputting again.

Product Name	Specification	Model	Price	Amount	AmountOfMoney	Note
FMV-6450TX2	Pentium-II 450MHz,64MB,4.3GB,Viper V550,100BASE-TX	WindowsNT Model	428000	0	0	----
		Windows98 Model	408000	0	0	----
		Windows95 Model	408000	0	0	----
FMV-6400TX2	Pentium-II 400MHz,64MB,4.3GB,Viper V550,100BASE-TX	WindowsNT Model	368000	0	0	----
		Windows98 Model	348000	3	1044000	Stocked
		Windows95 Model	348000	0	0	----
FMV-6450DX2	Pentium-II 450MHz,32MB,4.3GB,RAGE PRO TURBO,SOUND,100BASE-TX	WindowsNT Model	398000	2	796000	Stocked
		Windows98 Model	378000	0	0	----
		Windows95 Model	378000	0	0	----
FMV-6400DX2	Pentium-II 400MHz,32MB,4.3GB,RAGE PRO TURBO,SOUND,100BASE-TX	WindowsNT Model	338000	0	0	----
		Windows98 Model	318000	1	318000	Stocked
		Windows95 Model	318000	0	0	----
FMV-6350DX2	Pentium-II 350MHz,32MB,4.3GB,RAGE PRO TURBO,SOUND,100BASE-TX	WindowsNT Model	278000	5	1390000	Stocked
		Windows98 Model	258000	0	0	----
		Windows95 Model	258000	0	0	----
FMV-6366DX2c	Celeron 366MHz,32MB,4.3GB,RAGE PRO TURBO,SOUND,100BASE-TX	WindowsNT Model	238000	0	0	----
		Windows98 Model	218000	0	0	----
		Windows95 Model	218000	0	0	----

5. The order reception screen is displayed. The (2) menu screen is redisplayed when the Menu button is clicked.


GiveAnOrder - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://n4040129.lp.nm.fujitsu.co.jp/sample27/Order.asp
Go Links
Done
Local intranet

Order getting screen

The following orders were accepted. Thank you.

UserID	USER0001	OrderNumber	ONUM14560735

ProductNumber	Amount
FMV2TXF161	3
FMV2DXH111	2
FMV2DXF161	1
FMV2DXD111	5

MENU

[Caution]
If browser's [refresh] button is clicked, it will be double registration.

A.29 Sample 28: Transaction Management with COM Linkage-MTS

Sample 28 demonstrates COBOL COM server program transaction management using the Microsoft(R) Transaction Server (MTS).

See Chapter 24 in the "NetCOBOL User's Guide" for details on COBOL application transaction management using MTS.

To use this program, the following products are necessary:

- One of the following products:
  - Microsoft(R) Windows(R) 2000 Server operating system
  - Microsoft(R) Windows(R) 2000 Advanced Server operating system
  - Microsoft(R) Windows Server(R) 2003 Standard Edition
  - Microsoft(R) Windows Server(R) 2003 Enterprise Edition
  - Microsoft(R) Windows Server(R) 2008 Standard Edition
  - Microsoft(R) Windows Server(R) 2008 Enterprise Edition
- Microsoft(R) Transaction Server 2.0 or later

This program accesses a database via the ODBC driver, so the following products are necessary to run this program:

- Database
- Products necessary for database access with ODBC
- ODBC driver

- ODBC driver manager

For database access using an ODBC driver, refer to Chapter 19 in the "NetCOBOL User's Guide".

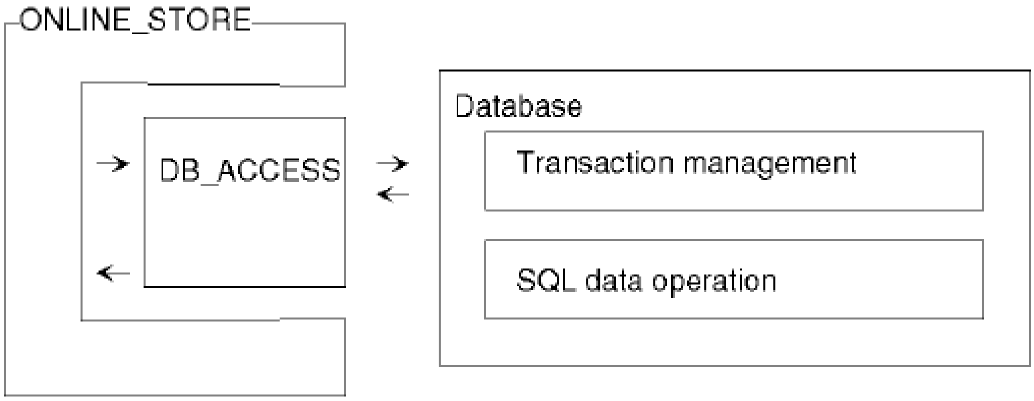
Overview

As in Sample 25, this sample program provides the following functions for constructing an online store application:

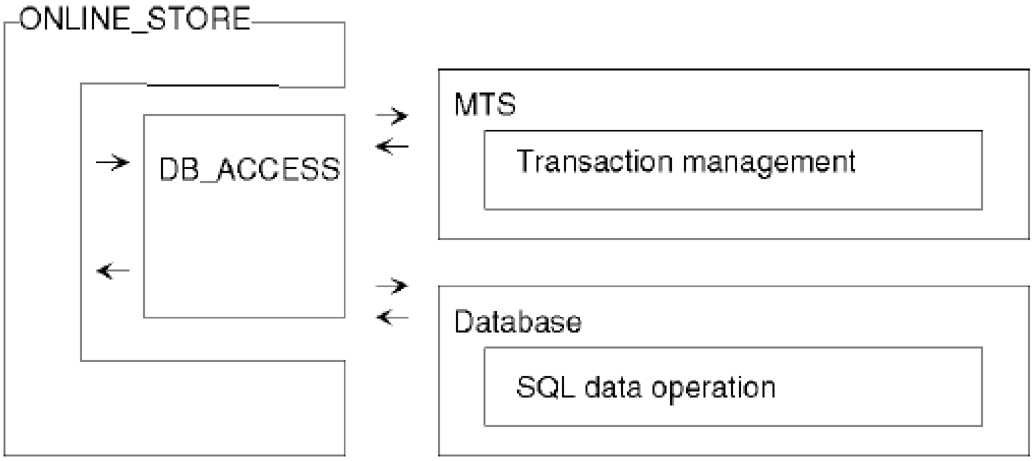
- Authorization processing
- Stock check
- Order registration
- Order calculation

However, this program manages transactions directly from a COBOL program by using the MTS functions.

In Sample 25, the embedded SQL statements COMMIT/ROLLBACK are used and transaction management depends on the database. This is a simple approach for someone who is familiar with embedded SQL statements but increases the database processing load.



This program manages transactions by itself with MTS functions. This reduces the database processing load and enables more detailed transaction management.



There are two methods of using the MTS function for transaction management from a COBOL application:

- Control of transactions where the object is operating from the COM server object
- Control of transactions where the COM server is operating from the COM client

The first method is shown below.

Available Programs

- DB_ACCESS.COB (COBOL source file)
- ONLINE_STORE.COB (COBOL source file)
- STORESV2.PRJ (project file)
- STORESV2.CBI (compiler option file)
- STORESV2_DLL.CSI (COM server information file)
- STORESV2.DEF (module definition file)

Applicable COBOL Functions

- COM server function
- Remote database access
- *COM-ARRAY class
- Object context object

Applicable COBOL Statements

The IF, INVOKE, INITIALIZE, SET, MOVE, and PERFORM statements are used.

The embedded SQL statements (CONNECT, INSERT, SELECT, UPDATE, ROLLBACK, and DISCONNECT) are used.

Operations Necessary Before Execution

- Construct an environment where a database can be accessed via the ODBC driver.
- Set a default server to be connected to and create the five tables listed below in a database on the server. For the formats of the tables and data to be stored, see Sample 25, "Creating a COBOL COM Server Program “.
  - CUSTOMER table
  - STOCK table
  - ORDERTABLE table
  - ORDERDETAIL table
- Use the ODBC information file tool (SQLODBCS.EXE) to create an ODBC information file (assumed to be C:\DBMSACS.INF).

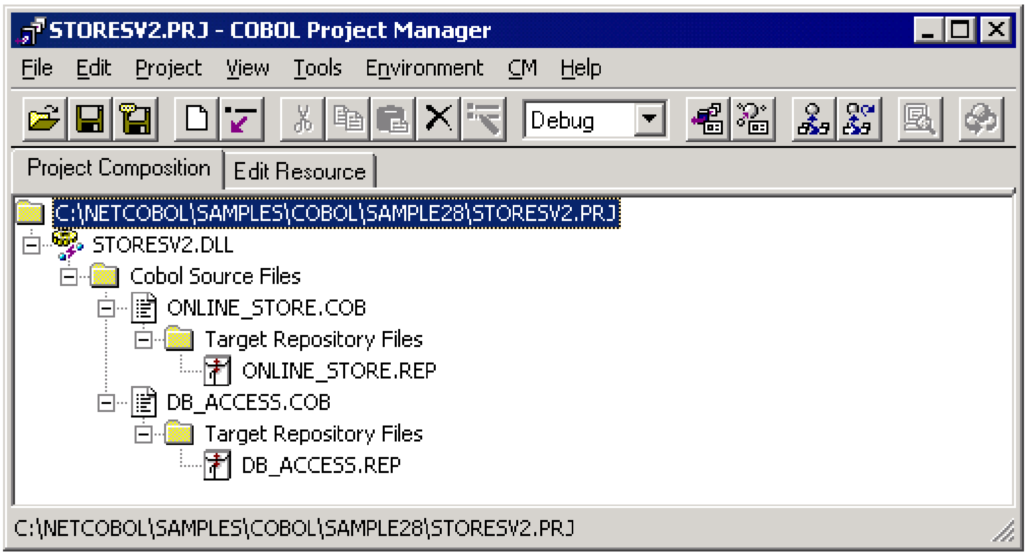
Building and Rebuilding

The Project Manager's build function is used to compile and link this program.

In the screen snapshots below, it is assumed that NetCOBOL was installed in folder C:\NetCOBOL. Change the folder name C:\NetCOBOL to the name of the folder where NetCOBOL is installed on your machine.

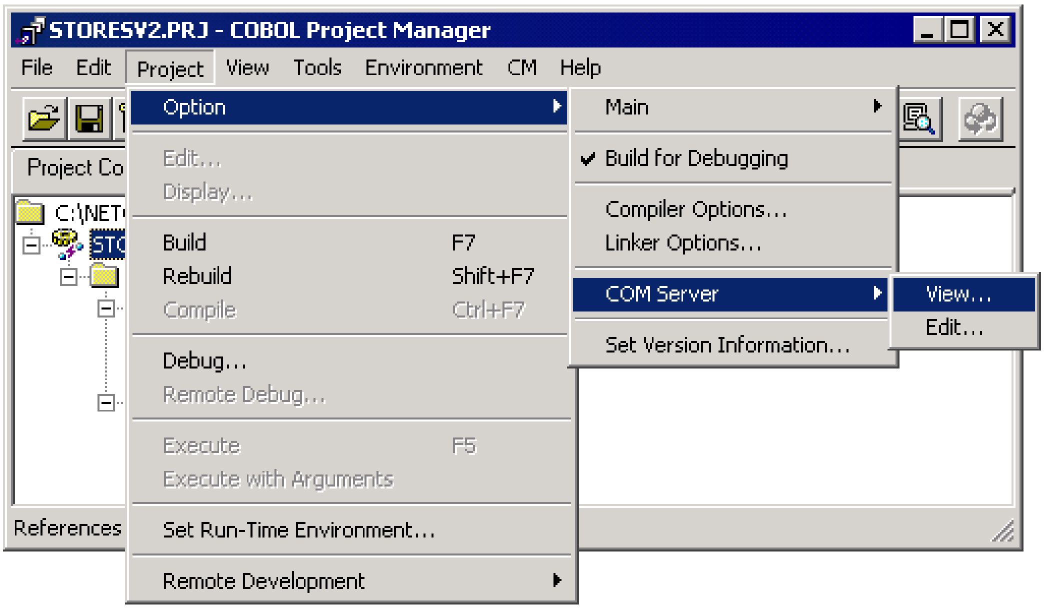
1. Start the Project Manager.

2. Open the project file SAMPLE28.PRJ.

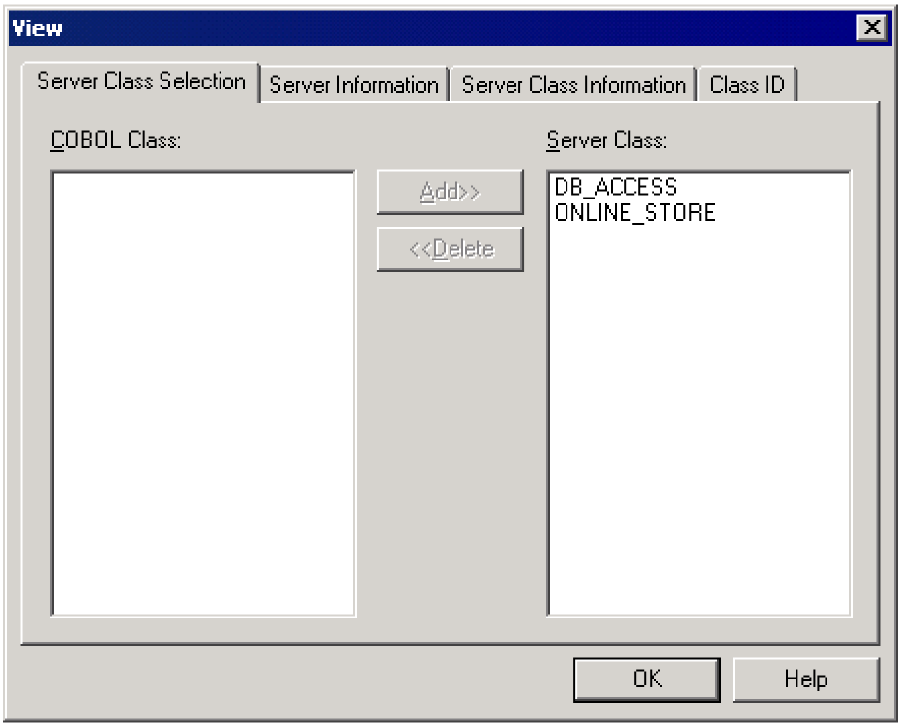

STORESV2.PRJ - COBOL Project Manager
File Edit Project View Tools Environment CM Help
Debug
Project Composition
Edit Resource
C:\NETCOBOL\SAMPLES\COBOL\SAMPLE28\STORESV2.PRJ
STORESV2.DLL
Cobol Source Files
ONLINE_STORE.COB
Target Repository Files
ONLINE_STORE.REP
DB_ACCESS.COB
Target Repository Files
DB_ACCESS.REP
C:\NETCOBOL\SAMPLES\COBOL\SAMPLE28\STORESV2.PRJ

3. Check the COM server information.

Select a target file (STORESV2.DLL) and select "View" from the "Project-Option-COM Server" menu.



The "View" dialog is opened and the server information can be referenced.



DB_ACCESS
ONLINE_STORE

4. Select "Build" from the "Project" menu.

   After build termination, check that SAMPLE28.DLL is created.

Registering the COBOL Application in MTS

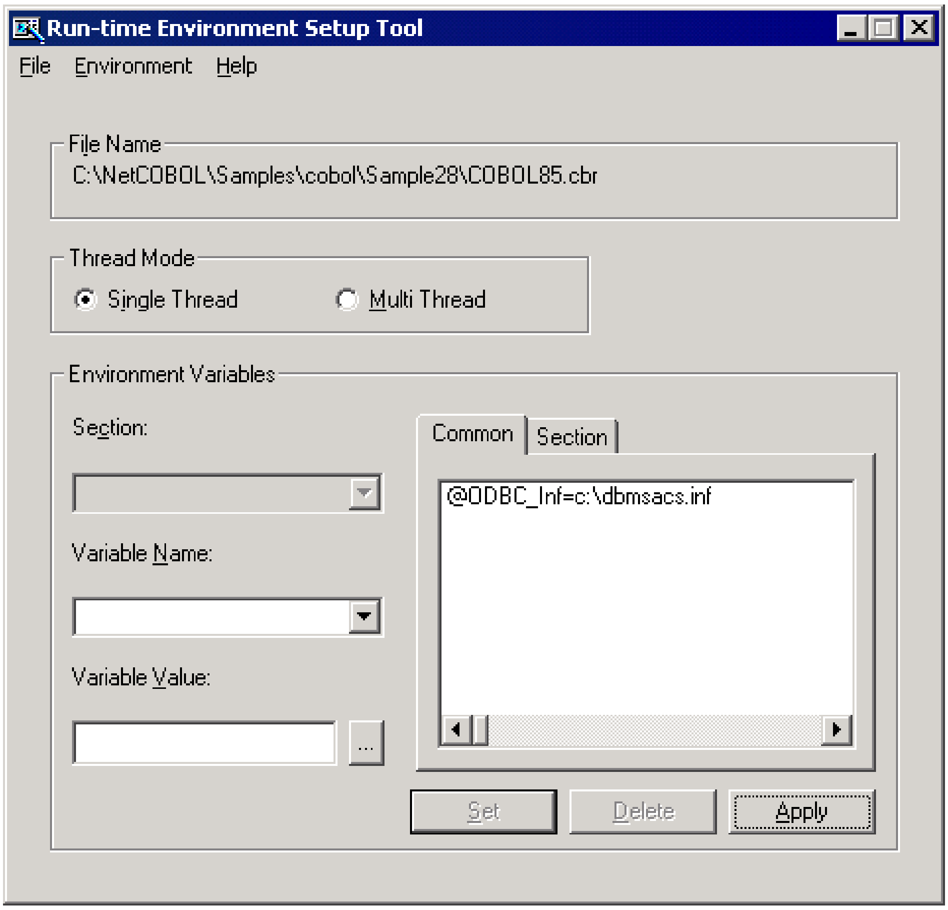
The COBOL application must be registered in the Windows system to use it as a COM server. Also specify a transaction control method in the registration.

Use the MTS explorer to register the COBOL application in the system registry and MTS. For details, refer to the section "Registration in the MTS Environment" in the "NetCOBOL User's Guide".

Setup the Server Execution Environment

1. Select "Run-time Environment Setup Tool" from the "Tools" menu of Project Manager.

   The run-time environment setup tool is displayed.



File Name
C:\NetCOBOL\Samples\cobol\Sample28\COBOL85.cbr

● Single Thread ○ Multi Thread

@ODBC_Inf=c:\dbmsacs.inf

2. Select "Open" on the "File" menu and create an object initialization file (COBOL85.CBR) in the folder that contains the dynamic link library (SAMPLE28.DLL).
3. Select the Common tab and enter data as shown below:

   Specify an ODBC information file name in the environment variable @ODBC_Inf (ODBC information file specification).
4. Click the Apply button.

   The data is saved in the object initialization file.
5. Select "Exit" on the "File" menu to exit the tool.

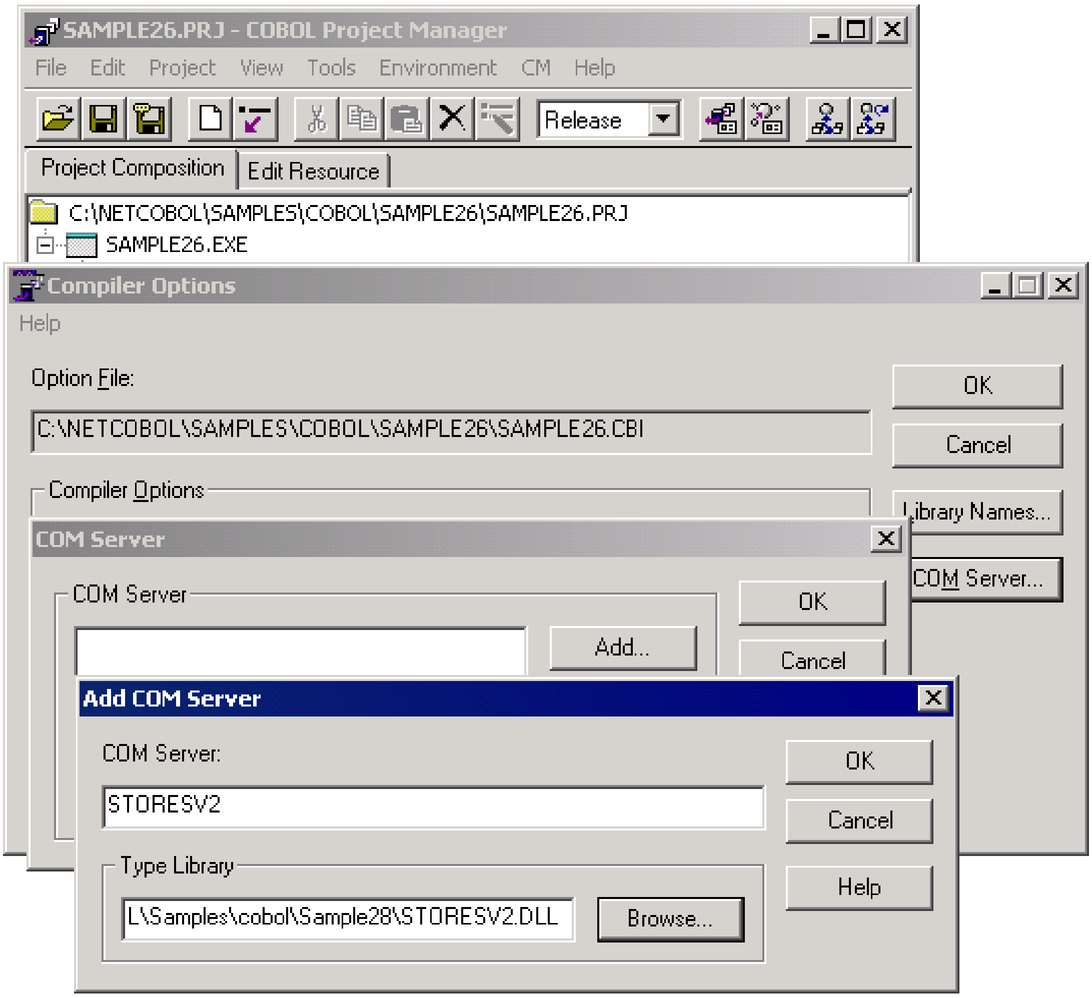
Modifying the Client Program

Samples 26 and 27 can be used as clients of this program by modifying them as shown below.

- Using the COBOL client in Sample 26.
  - Modify the cluster specifier of the CLMAIN.COB repository as shown below.

```
000200 CLASS ONLINE_STORE AS "*COM:STORESV2:ONLINE_STORE"
```

- Open the project file SAMPLE26.PRJ, delete COM server name STORESV1, and enter STORESV2.



Option File: C:\NETCOBOL\SAMPLES\COBOL\SAMPLE26\SAMPLE26.CBI

COM Server: STORESV2

Type Library: L\Samples\cobol\Sample28\STORESV2.DLL

- Rebuild SAMPLE26.EXE.
- Create an installation program for client information on the Sample 28 program and install it in the client.

- Using the ASP client in Sample 27.

Modify the CreateObject argument for COM object creation as shown below.

Catalog.asp

```
Set OLSService = Server.CreateObject("STORESV2.ONLINE_STORE")
```

Confirm.asp

```
Set OLSService = Server.CreateObject("STORESV2.ONLINE_STORE")
```

Order.asp

```
Set obj = Server.CreateObject("STORESV2.ONLINE_STORE")
```

A.30 Sample 29: Inter-application Communication Function

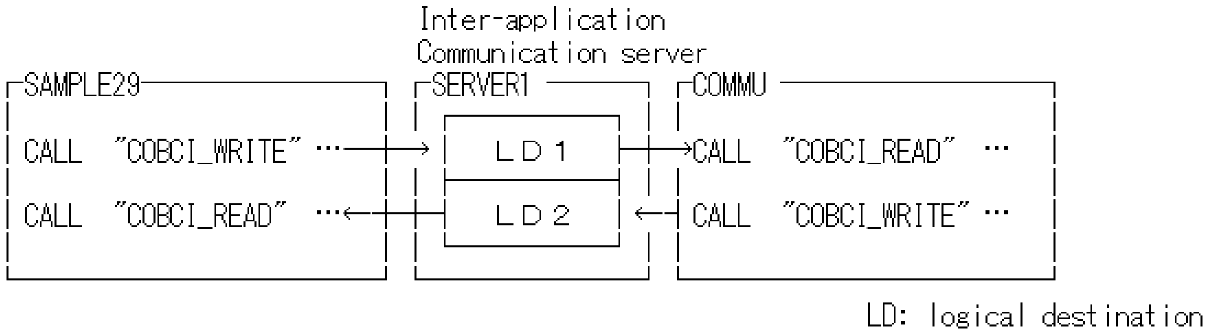
Sample 29 demonstrates how to use the simplified inter-application communication function to read or write messages to or from logical destinations. For details on how to use the simplified inter-application communication function for message transfer, refer to the topic “Using Simplified Inter-application Communication” in Chapter 18 of the “NetCOBOL User’s Guide.”

Overview

The sample program transfers messages between SAMPLE29 and COMMU.

SAMPLE29 establishes a connection to server "SERVER1", writes messages to logical destination "MYLD1", and reads messages from logical destination "MYLD2". If no message comes from logical destination "MYLD2", SAMPLE29 waits for the message arrival for up to 60 seconds. After reading messages from the logical destination "MYLD2", SAMPLE29 reads messages from logical destination "MYLD1" in order of priority.

COMMU establishes a connection to server "SERVER1", reads messages from logical destination "MYLD1", then writes messages to logical destinations "MYLD1" and "MYLD2”.



Available Programs

- SAMPLE29.COB (COBOL source program)
- COMMU.COB (COBOL source program)
- SAMPLE29.PRJ (Project file)
- PRM_REC.CBL (library text)
- SAMPLE29.INI (logical destination definition file)

Applicable COBOL Functions

Using Simplified Inter-application Communication

Applicable COBOL Statements

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

Prerequisites to Execution

To make the Sample 29 programs ready for execution, execute the following utility:

- COBCISRV

After you run this, you can build and execute Sample 29.

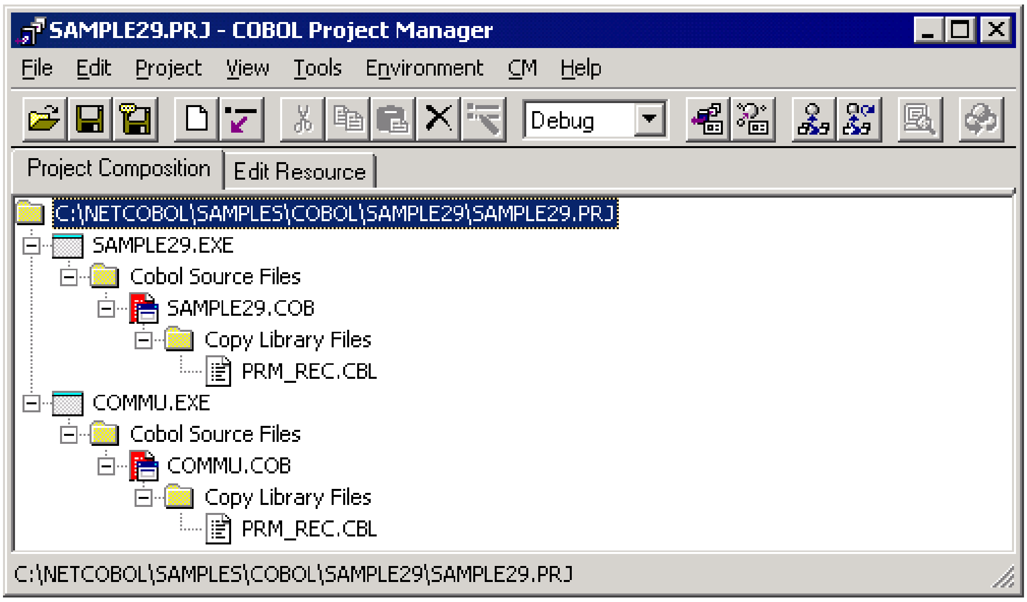
You may refer to Chapter 18, “Communication Functions” in the “NetCOBOL User’s Guide” for more general information on the following topics:

- Activating the server for simplified inter-application communication.
- Creating logical destinations "LD1" and "LD2" on the server for simplified interapplication communication.
- Changing the information on the opponent machine name of the logical destination definition file (SAMPLE29.INI) by using the logical destination definition file creation utility (COBCIU32.EXE). Specify the host name of the server as the opponent machine name and click on the Set button.

Building the Program

The project file for Sample 29 is SAMPLE29.PRJ in the SAMPLE29 directory. Select “Open Project” from the “File” menu to load it into the COBOL Project Manager window.

Fully opened, this simple project includes two separate executable files that share the same copy libraries, as shown in the following figure.



```
C:\NETCOBOL\SAMPLES\COBOL\SAMPLE29\SAMPLE29.PRJ
  SAMPLE29.EXE
    Cobol Source Files
      SAMPLE29.COB
        Copy Library Files
          PRM_REC.CBL
  COMMU.EXE
    Cobol Source Files
      COMMU.COB
        Copy Library Files
          PRM_REC.CBL
```

C:\NETCOBOL\SAMPLES\COBOL\SAMPLE29\SAMPLE29.PRJ

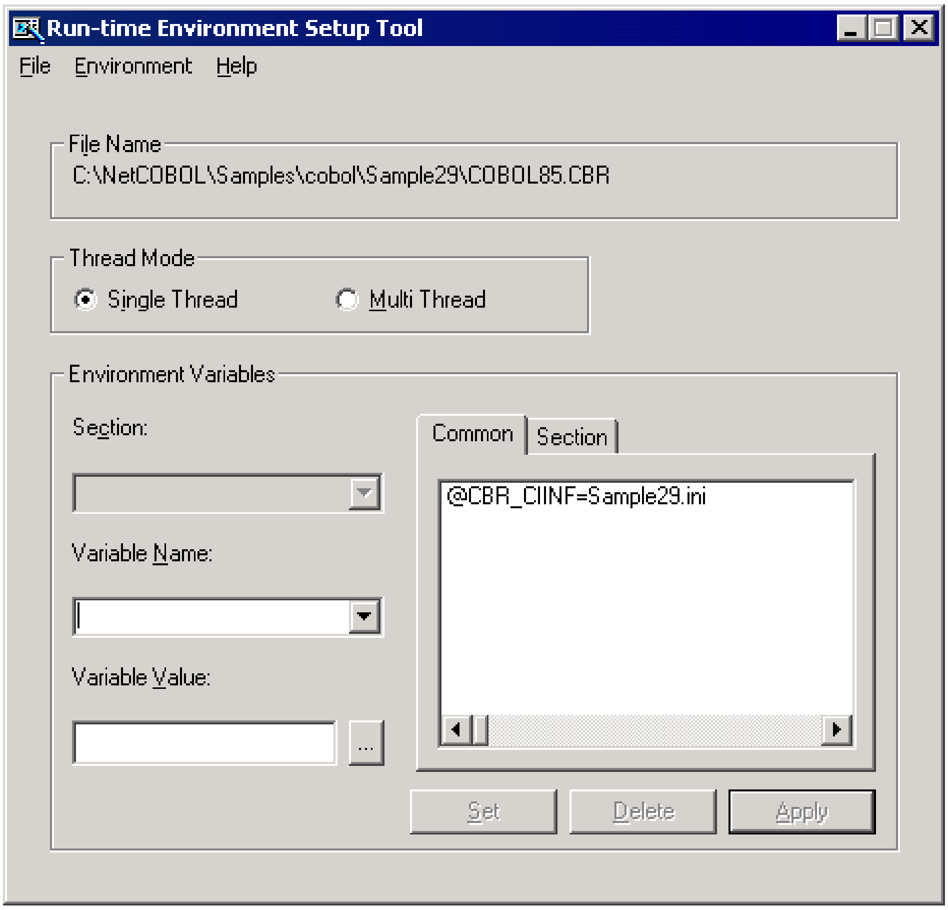
Build the project by selecting Build from the Project menu.

The MAIN and TEST compiler options have been specified.

Setup the Server Program Execution Environment

1. Select "Run-time Environment Setup Tool" from the "Tools" menu of Project Manager.

   The run-time environment setup tool is displayed.



File Name: C:\NetCOBOL\Samples\cobol\Sample29\COBOL85.CBR

Thread Mode: ● Single Thread ○ Multi Thread

@CBR_CIINF=Sample29.ini

2. Select "Open" on the "File" menu and create an object initialization file (COBOL85.CBR) in the folder that contains the executable file (SAMPLE29.EXE).

3. Select the Common tab and enter data as shown below:

   - The path name of the logical destination definition file of the @CBR_CIINF environment variable has been pre-assigned to SAMPLE29.INI. This is the default setting for both the SAMPLE29 and COMMU executable files.

4. Click the Apply button.

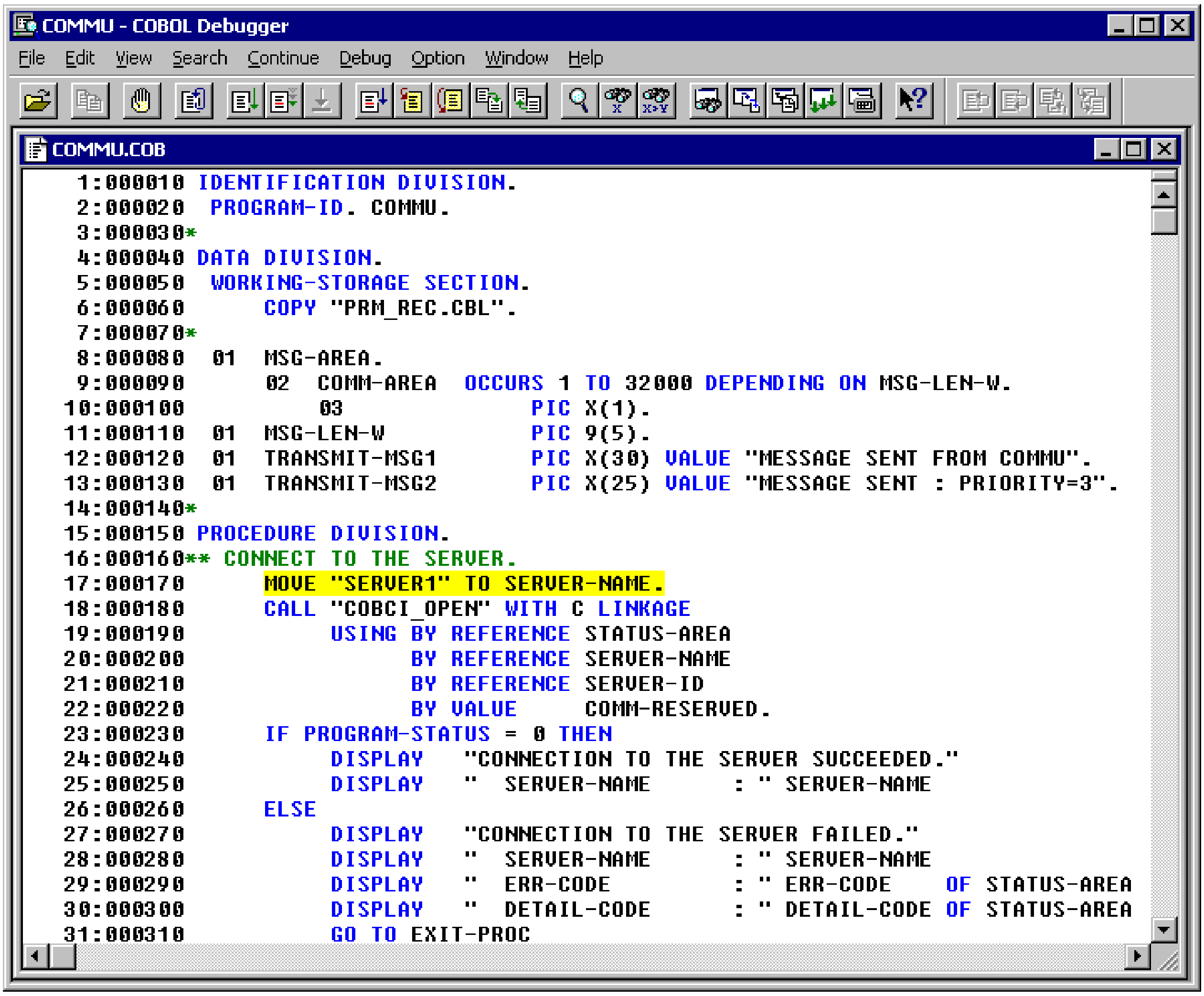
   The data is saved in the run-time initialization file.

5. Select "Exit" on the "File" menu to terminate the run-time environment setup tool.

Debugging the Program

To run these programs under the control of the Debugger, select Debug from the Tools menu. The Start Debugging dialog box appears. Press the ENTER key.

When the source is read into the COBOL Debugger, it will look like the following figure.

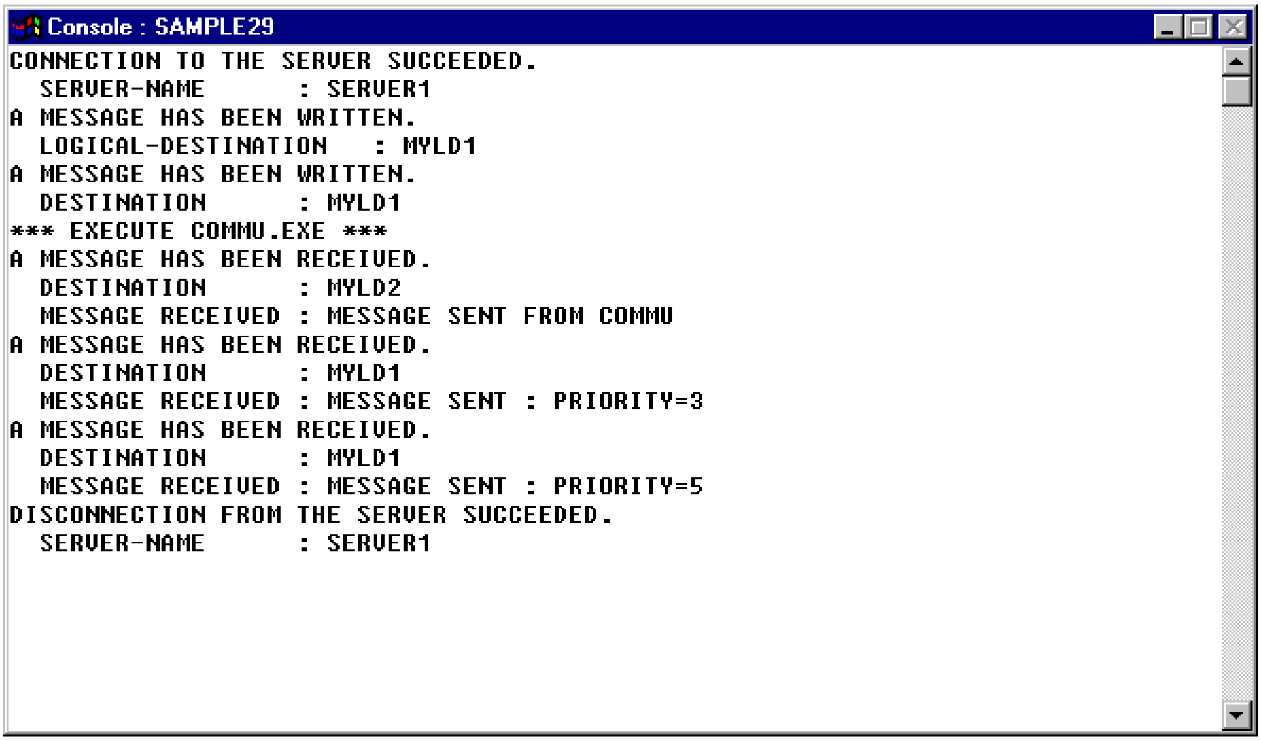


```
000010 IDENTIFICATION DIVISION.
000020  PROGRAM-ID. COMMU.
000030*
000040 DATA DIVISION.
000050  WORKING-STORAGE SECTION.
000060      COPY "PRM_REC.CBL".
000070*
000080  01  MSG-AREA.
000090      02  COMM-AREA  OCCURS 1 TO 32000 DEPENDING ON MSG-LEN-W.
000100          03              PIC X(1).
000110  01  MSG-LEN-W           PIC 9(5).
000120  01  TRANSMIT-MSG1       PIC X(30) VALUE "MESSAGE SENT FROM COMMU".
000130  01  TRANSMIT-MSG2       PIC X(25) VALUE "MESSAGE SENT : PRIORITY=3".
000140*
000150 PROCEDURE DIVISION.
000160** CONNECT TO THE SERVER.
000170      MOVE "SERVER1" TO SERVER-NAME.
000180      CALL "COBCI_OPEN" WITH C LINKAGE
000190           USING BY REFERENCE STATUS-AREA
000200                 BY REFERENCE SERVER-NAME
000210                 BY REFERENCE SERVER-ID
000220                 BY VALUE     COMM-RESERVED.
000230      IF PROGRAM-STATUS = 0 THEN
000240           DISPLAY   "CONNECTION TO THE SERVER SUCCEEDED."
000250           DISPLAY   "  SERVER-NAME      : " SERVER-NAME
000260      ELSE
000270           DISPLAY   "CONNECTION TO THE SERVER FAILED."
000280           DISPLAY   "  SERVER-NAME      : " SERVER-NAME
000290           DISPLAY   "  ERR-CODE         : " ERR-CODE    OF STATUS-AREA
000300           DISPLAY   "  DETAIL-CODE      : " DETAIL-CODE OF STATUS-AREA
000310           GO TO EXIT-PROC
```

Executing the Program

To execute this sample, you must start both executable files─SAMPLE29.EXE and COMMU.EXE. You can do this by double-clicking the mouse on the executable files in the COBOL Project Manager window.

Execution Result



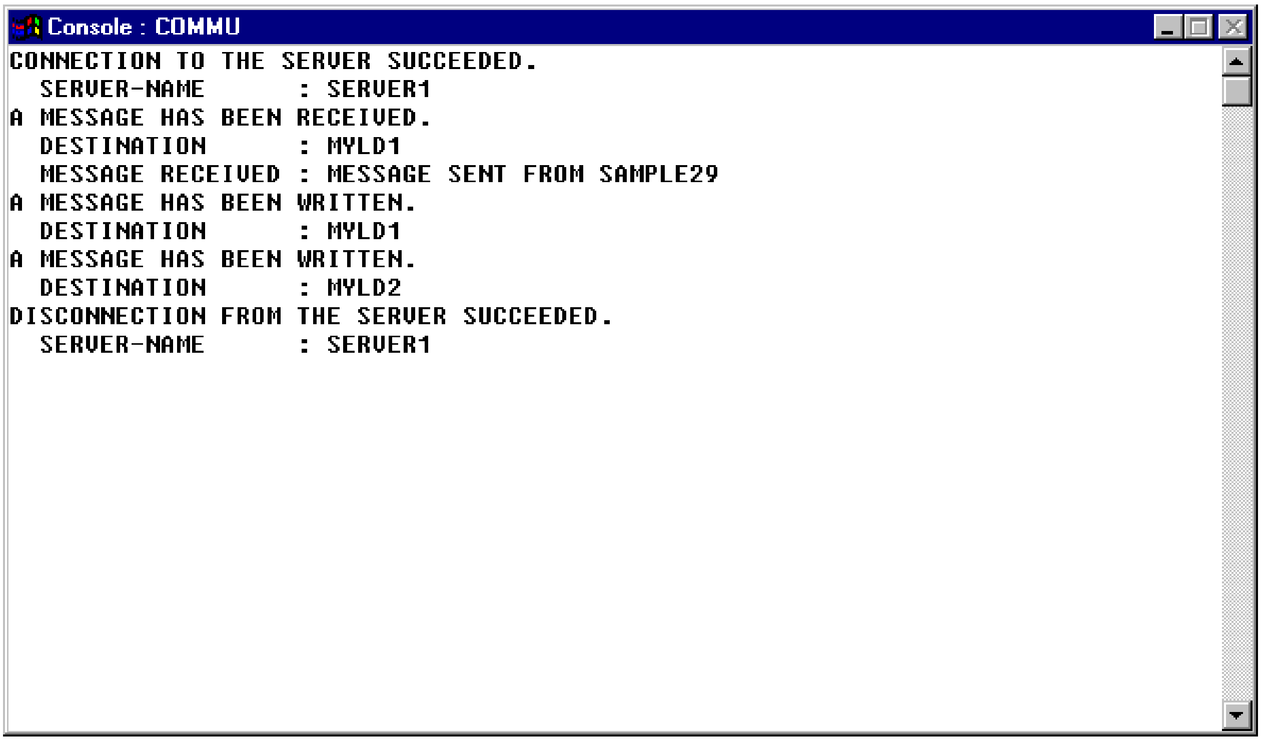
Console : SAMPLE29

```
CONNECTION TO THE SERVER SUCCEEDED.
  SERVER-NAME       : SERVER1
A MESSAGE HAS BEEN WRITTEN.
  LOGICAL-DESTINATION    : MYLD1
A MESSAGE HAS BEEN WRITTEN.
  DESTINATION       : MYLD1
*** EXECUTE COMMU.EXE ***
A MESSAGE HAS BEEN RECEIVED.
  DESTINATION       : MYLD2
  MESSAGE RECEIVED : MESSAGE SENT FROM COMMU
A MESSAGE HAS BEEN RECEIVED.
  DESTINATION       : MYLD1
  MESSAGE RECEIVED : MESSAGE SENT : PRIORITY=3
A MESSAGE HAS BEEN RECEIVED.
  DESTINATION       : MYLD1
  MESSAGE RECEIVED : MESSAGE SENT : PRIORITY=5
DISCONNECTION FROM THE SERVER SUCCEEDED.
  SERVER-NAME       : SERVER1
```

Check the messages displayed on the console to ensure the following:

In the SAMPLE29 console window:

1. Messages from COMMU have been read from logical destination "MYLD2".
2. Messages have been read from logical destination "MYLD1" in order of priority.



Console : COMMU

```
CONNECTION TO THE SERVER SUCCEEDED.
  SERVER-NAME       : SERVER1
A MESSAGE HAS BEEN RECEIVED.
  DESTINATION       : MYLD1
  MESSAGE RECEIVED : MESSAGE SENT FROM SAMPLE29
A MESSAGE HAS BEEN WRITTEN.
  DESTINATION       : MYLD1
A MESSAGE HAS BEEN WRITTEN.
  DESTINATION       : MYLD2
DISCONNECTION FROM THE SERVER SUCCEEDED.
  SERVER-NAME       : SERVER1
```

In the COMMU console window:

1. Messages from SAMPLE29 have been read from logical destination "MYLD1".

2. Messages have been written to logical destination "MYLD1" and logical destination "MYLD2".

A.31 Sample 30: Using UNICODE

Sample 30 is not available in the English version of NetCOBOL.

A.32 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.

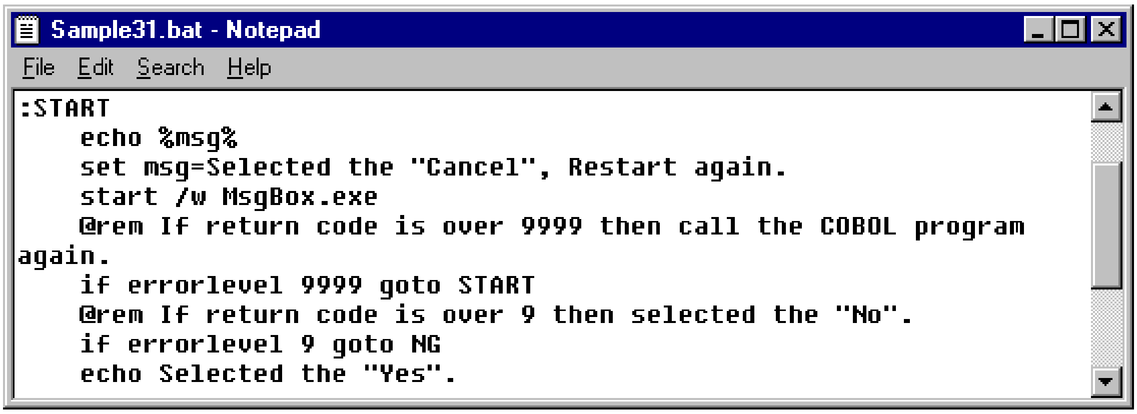
Overview

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, when the suffix is "W".)

The call uses the STDCALL calling convention.

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

It is possible to refer to this return value in a batch file. In a batch file, you can access this return value by the name ERRORLEVEL. SAMPLE31.BAT demonstrates this as follows.



```
: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".
```

Files Included in Sample 31

- MSGBOX.COB (COBOL source program)
- SAMPLE31.PRJ (Project file)
- SAMPLE31.CBI (Compiler option file)
- SAMPLE31.BAT (Batch file for start)

COBOL Statements Used

- Method of calling C program from COBOL program
- STDCALL call convention
- Parameter transfer BY VALUE
- RETURNING phrase of CALL statement
- Special register PROGRAM-STATUS (RETURN-CODE)
- Project Manager

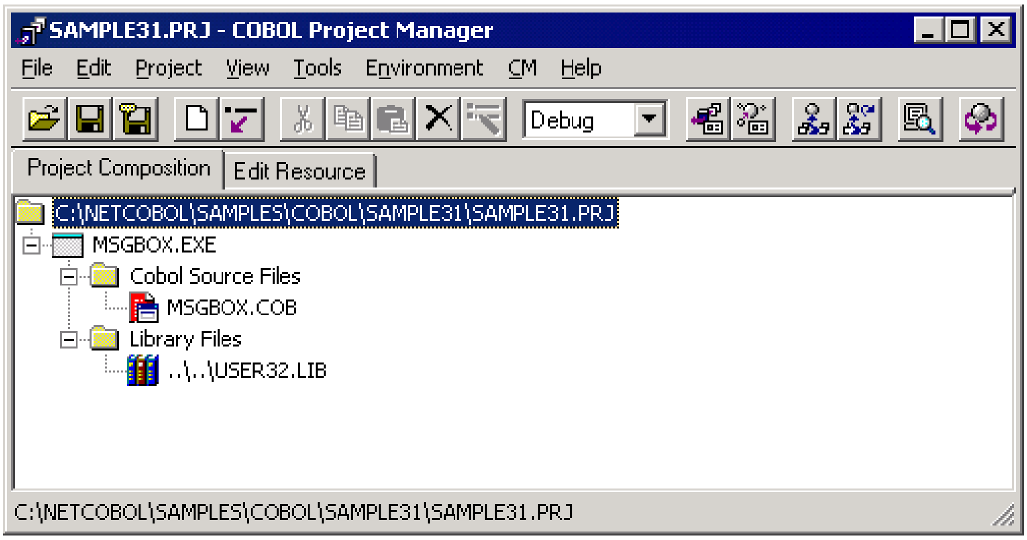
Note

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

Building the Program

Project manager's build function is used to make the executable program.

1. Start the Project Manager.
2. Open project file SAMPLE31.PRJ.



SAMPLE31.PRJ - COBOL Project Manager

C:\NETCOBOL\SAMPLES\COBOL\SAMPLE31\SAMPLE31.PRJ
MSGBOX.EXE
Cobol Source Files
MSGBOX.COB
Library Files
..\..\USER32.LIB

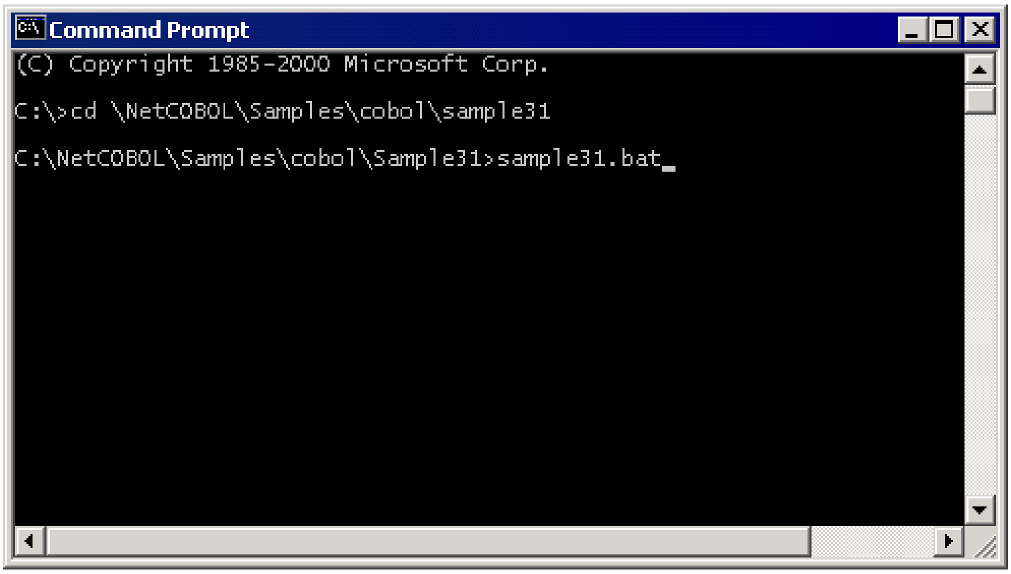
C:\NETCOBOL\SAMPLES\COBOL\SAMPLE31\SAMPLE31.PRJ

Library file "USER32.LIB" is required to link the program. It is used to ensure the linker can resolve the external reference "MessageBoxA". If you installed the product to something other than the default folder structure, you should change the folder name to match the folder structure you are using.

3. Select "Build" from the project menu.

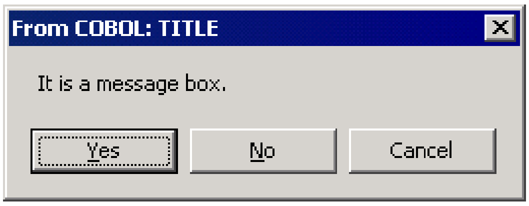
Executing the Program

1. Open a Command Prompt, change directories to the SAMPLE31 folder, and execute SAMPLE31.BAT.



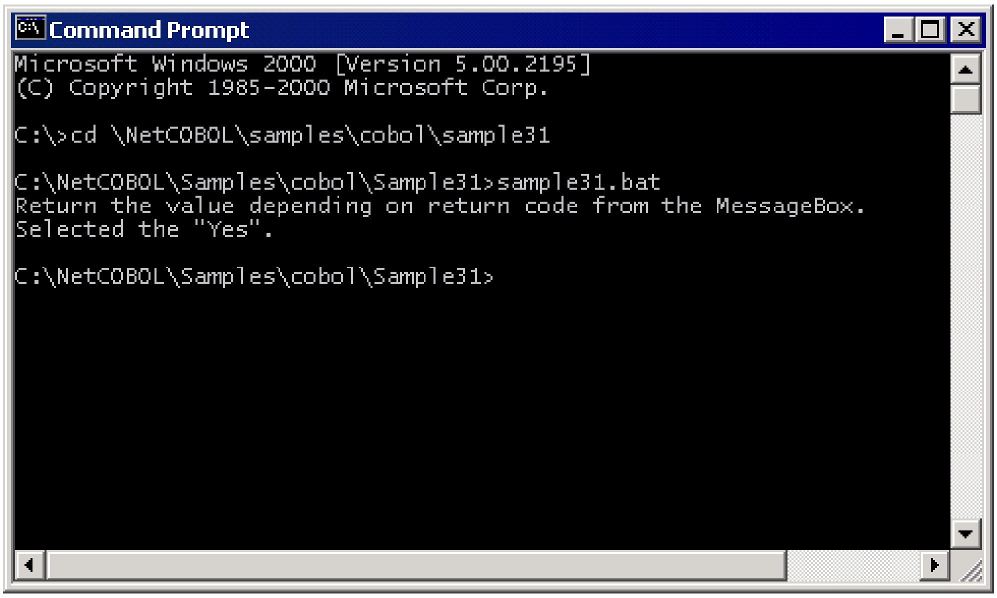
```
(C) Copyright 1985-2000 Microsoft Corp.

C:\>cd \NetCOBOL\Samples\cobol\sample31

C:\NetCOBOL\Samples\cobol\Sample31>sample31.bat_
```

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



It is a message box.

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



```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\>cd \NetCOBOL\samples\cobol\sample31

C:\NetCOBOL\Samples\cobol\Sample31>sample31.bat
Return the value depending on return code from the MessageBox.
Selected the "Yes".

C:\NetCOBOL\Samples\cobol\Sample31>
```

4. When the [Cancel] button is clicked, the program is executed again.

A.33 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)
- SAMPLE32.PRJ (Project file)

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.

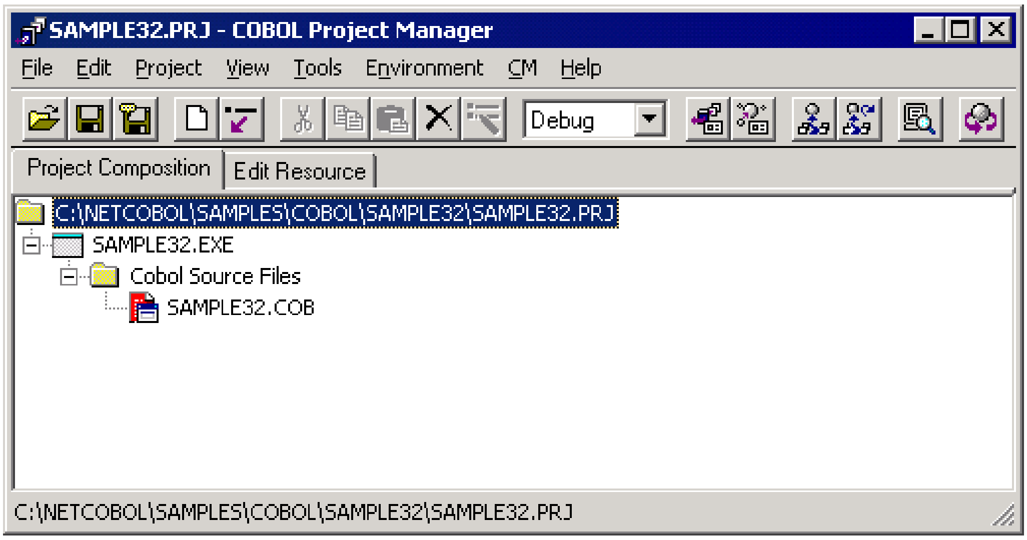
Building and Rebuilding

The Project Manager's build function is used to compile and link this program.

In the screen snapshots below, NetCOBOL was installed in folder C:\NetCOBOL.

Change the folder name C:\NetCOBOL to the name of the folder where NetCOBOL is installed on your machine.

1. Start the Project Manager.
2. Open the project file SAMPLE32.PRJ.


SAMPLE32.PRJ - COBOL Project Manager
File
Edit
Project
View
Tools
Environment
CM
Help
Debug
Project Composition
Edit Resource
C:\NETCOBOL\SAMPLES\COBOL\SAMPLE32\SAMPLE32.PRJ
SAMPLE32.EXE
Cobol Source Files
SAMPLE32.COB
C:\NETCOBOL\SAMPLES\COBOL\SAMPLE32\SAMPLE32.PRJ

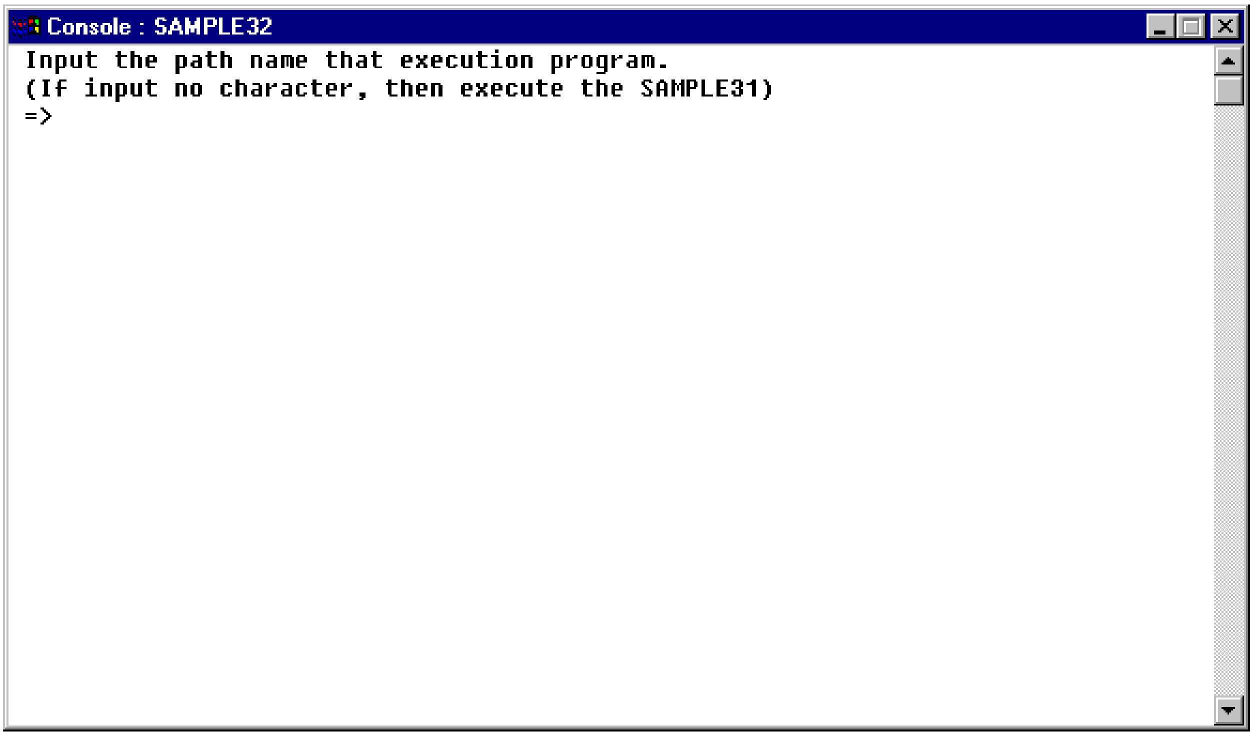
3. Select "Build" from the "Project" menu.

   After build termination, check that SAMPLE32.EXE is created.

Executing the Program

1. Select "Execute" on the "Project" menu.

   The following display appears and waits for your input.

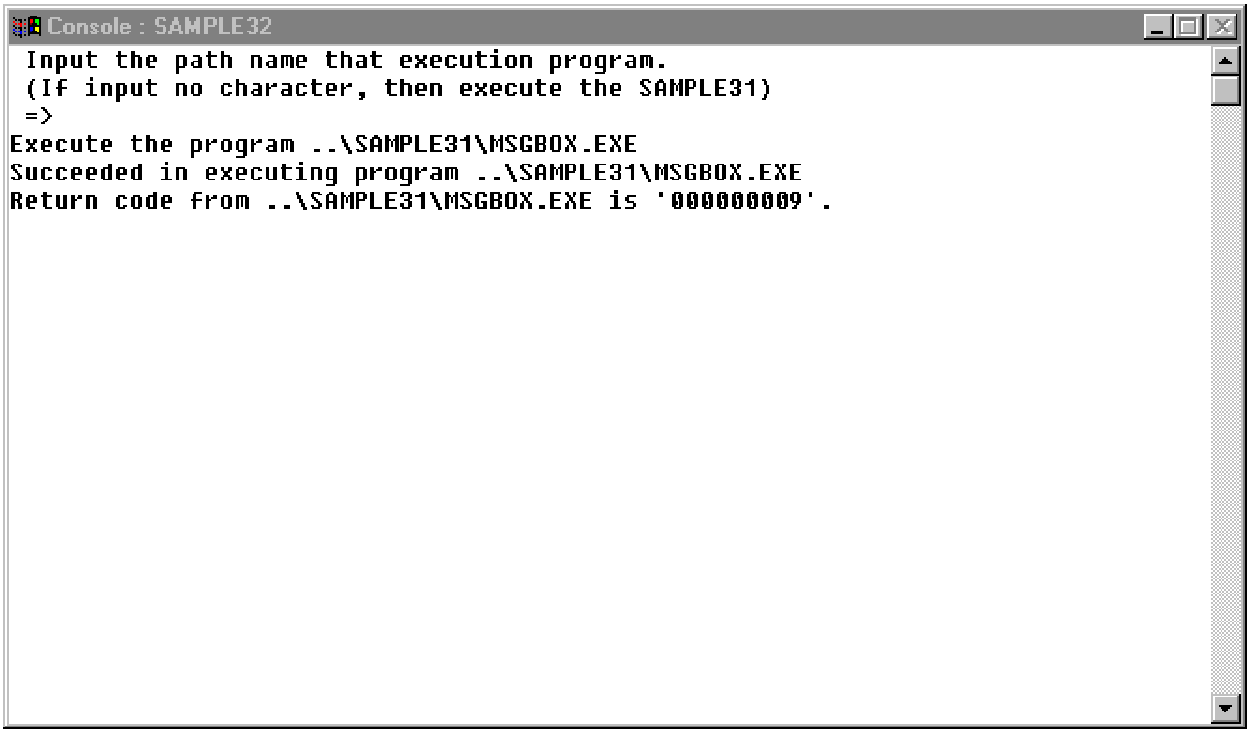


```
Input the path name that execution program.
(If input no character, then execute the SAMPLE31)
=>
```

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.

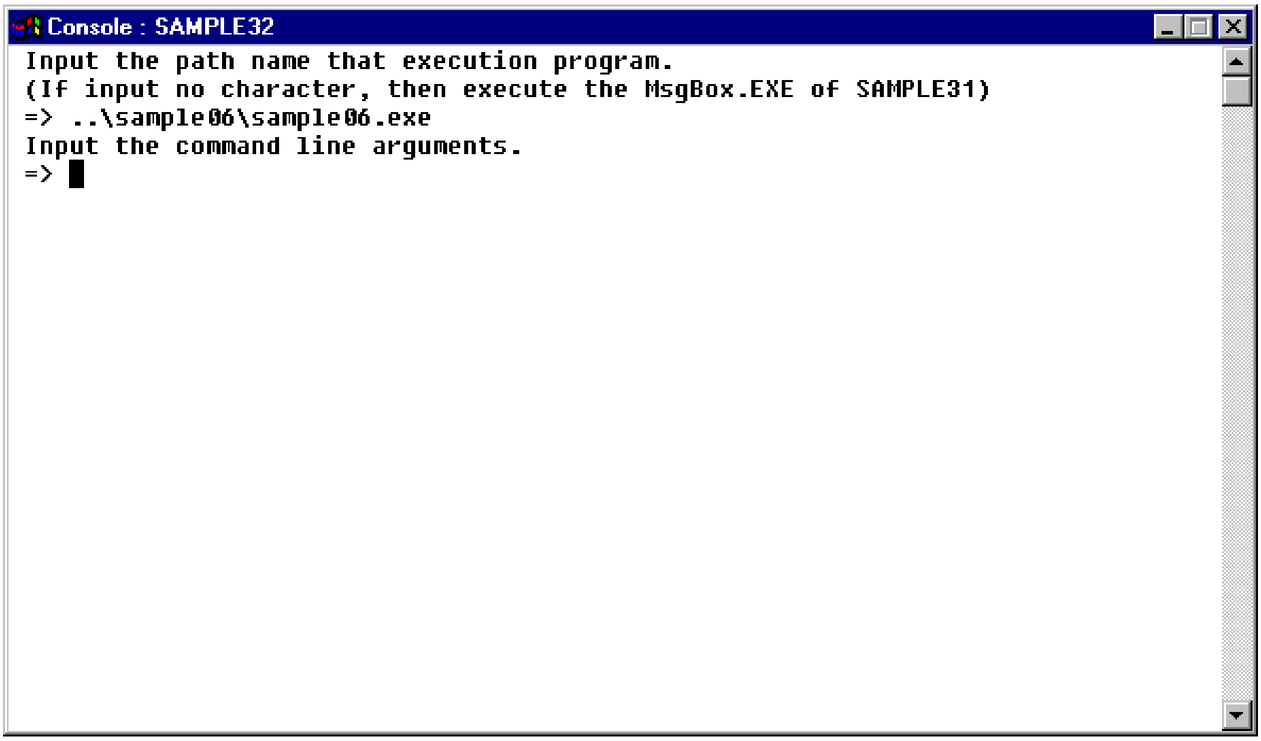
   If nothing is entered, then MSGBOX.EXE of SAMPLE31 is executed. Press the ENTER key.

3. The completion code of MSGBOX.EXE of SAMPLE31 is displayed (indicating what button was pushed). In the following screen, the “No” button was pressed.



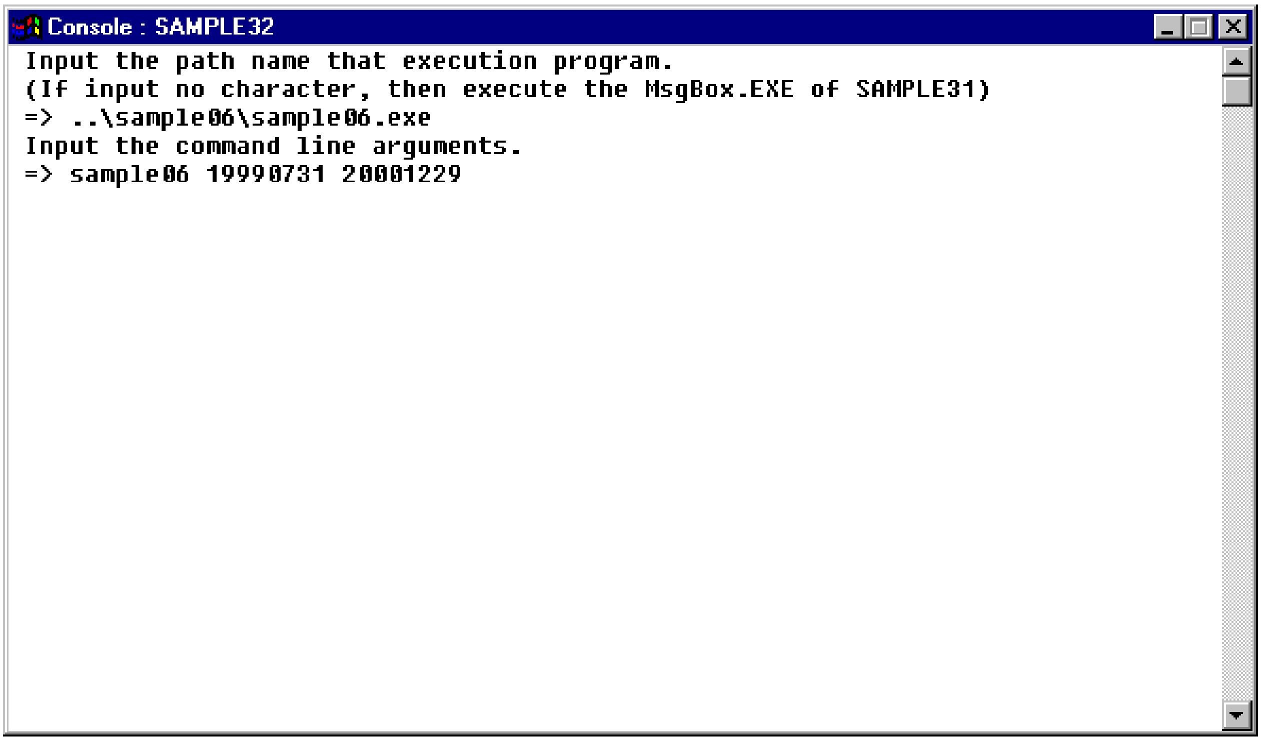
```
 Input the path name that execution program.
 (If input no character, then execute the SAMPLE31)
 =>
Execute the program ..\SAMPLE31\MSGBOX.EXE
Succeeded in executing program ..\SAMPLE31\MSGBOX.EXE
Return code from ..\SAMPLE31\MSGBOX.EXE is '000000009'.
```

4. If SAMPLE32 is reexecuted 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.



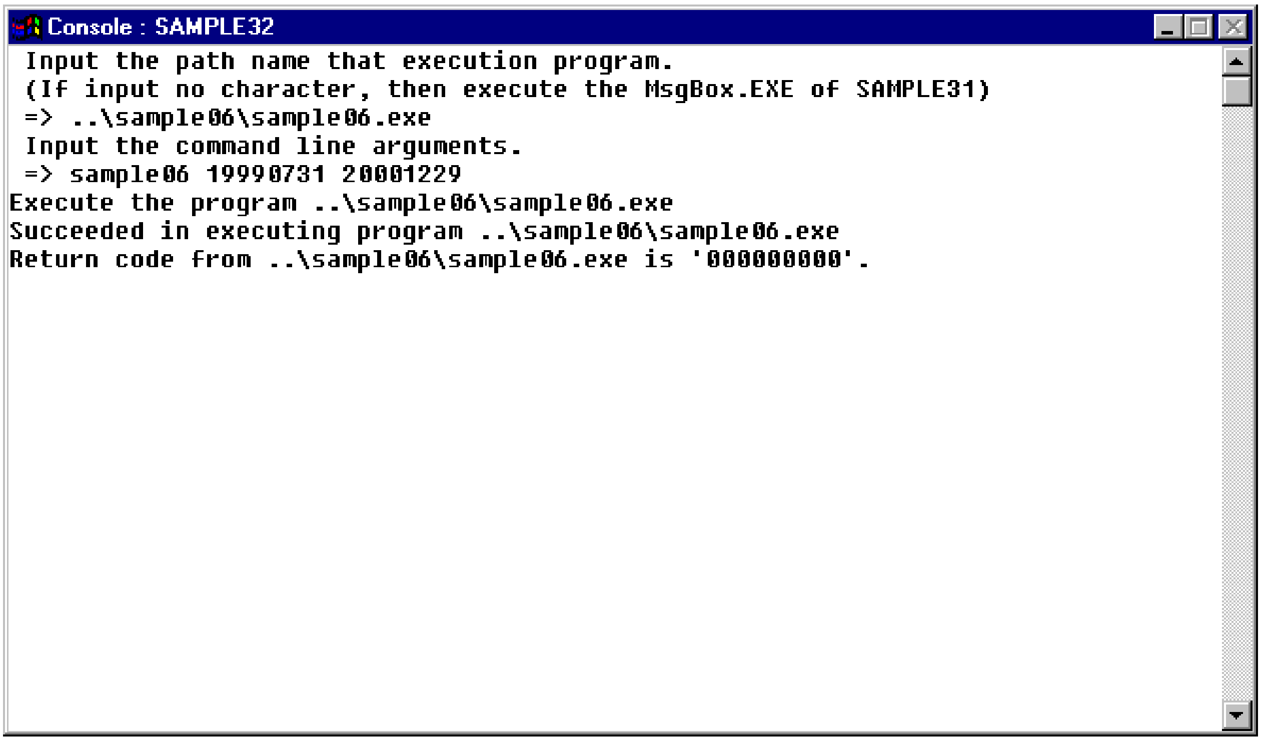
```
 Input the path name that execution program.
 (If input no character, then execute the MsgBox.EXE of SAMPLE31)
 => ..\sample06\sample06.exe
 Input the command line arguments.
 =>
```

5. SAMPLE06.EXE requires two command line arguments. Please note that the command line arguments are specified following the program name.



```
 Input the path name that execution program.
 (If input no character, then execute the MsgBox.EXE of SAMPLE31)
 => ..\sample06\sample06.exe
 Input the command line arguments.
 => sample06 19990731 20001229
```

6. 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\sample06.exe
 Input the command line arguments.
 => sample06 19990731 20001229
Execute the program ..\sample06\sample06.exe
Succeeded in executing program ..\sample06\sample06.exe
Return code from ..\sample06\sample06.exe is '000000000'.
```

Appendix B Tips

1. To build a module or application in Project Manager you must have the executable module or project selected in the project tree view. If any other item is selected in the tree view, the build function is disabled.
2. To set compiler and linker options in Project Manager you must have the project selected in the project tree view.
3. Before invoking the Data function in the debugger, if possible, place the cursor on the data item name you want to inspect. This saves you having to enter the data name in the Data dialog.
4. The Data file I/O function in the debugger is targeted at Fujitsu-specific data file features, only likely to be found in applications being ported from other Fujitsu environments (e.g. mainframe and UNIX machines).
5. To create a class information database you first use the menu bar Tools, Class Database, Options dialog to set up the folders containing the class repositories. You then select the Tools, Class Database, Creation function to create the database.
6. To create a project information database you make sure that "Creation Project Database" is checked on the Project, Option sub-menu, then build the project.
7. For more information on NetCOBOL check the following sources: Softcopy documentation - All the NetCOBOL manuals are available in softcopy form (Adobe Acrobat pdf). You can access the documentation from the Start>Programs>NetCOBOL menu.

   Help - Most components come with on-line help that explains interface details and command formats.