Microsoft Windows®

Microsoft Windows 95™

Microsoft Windows NT™

COBOL85 User's Guide 3.0

FUJITSU

Third Edition: March 1997

Preface

COBOL85 allows you to create, execute, and debug COBOL programs with Microsoft Windows 95, Windows NT and Windows 3.1. This manual describes the functions and operations of COBOL85, but does not detail COBOL syntax. Refer to the "COBOL85 Reference Manual" for details on COBOL syntax.

Throughout this manual, Windows 95 and Windows NT-specific items are denoted with (32); Windows 3.1-specific items are denoted with (16).

Audience

This manual is for users who develop COBOL programs using COBOL85. This manual assumes users possess basic knowledge of COBOL 85 and are familiar with the appropriate Windows platform.

How this Manual is Organized

This manual consists of the following chapters and appendixes:

Chapter	**Contents**
Chapter 1. Overview of COBOL85	The operating environment and functions of COBOL85.
Chapter 2. Creating and Editing a Program	Writing a COBOL program.
Chapter 3. Compiling Programs	Compiling COBOL programs.
Chapter 4. Linking Programs	Linking COBOL programs.
Chapter 5. Executing Programs	Executing a COBOL program that has been compiled and linked.
Chapter 6. Project Management	Registering files and control linker options.
Chapter 7. File Processing	Using files.
Chapter 8. Printing	Printing data and documents.
Chapter 9. Input-Output Using Screens	Transferring data with screens.
Chapter 10. Calling Subprograms (Inter-Program Communication)	Calling subprograms from a COBOL program.
Chapter 11. Using ACCEPT and DISPLAY Statements	The simplified input-output, command line argument operation, and environment variable operation functions, using ACCEPT and DISPLAY statements.
Chapter 12. Using SORT/MERGE Statements (Sort-Merge Function)	Sort-merge processing.
Chapter 13. System Program Description (SD) Functions	Functions such as the pointer and ADDR, used to create system programs.
Chapter 14. Communication Functions	Using simplified inter-application and other communication types.
Chapter 15. Database (SQL)	Remote database access under ODBC.
Chapter 16. Distributed Development Support Functions	Designing and developing COBOL programs in a distributed environment.

Chapter	**Contents**
Appendix A. Compiler Options	The options provided for the COBOL85 compiler.
Appendix B. I-O Status List	The values returned from input-output statement execution, indicating I-O status and their meanings.
Appendix C. Global Optimization	The optimization performed by the COBOL85 compiler.
Appendix D. Built-in Function List	The list of COBOL85 built-in functions.
Appendix E. Special Registers Used with Screen and Form Functions	The values set in the special registers for screen and form functions.
Appendix F. Message Lists	Compilation and execution messages.
Appendix G. Writing Special Literals	Writing literals for Windows.
Appendix H. High-Speed File Processing	Specifications for high-speed file processing.
Appendix I. GS-series Function Comparison	Compares the functions available with the GS- series and this COBOL85.
Appendix J. Command Formats	Compiler and linker command formats.
Appendix K. FCB Control Statement	The specification of the FCB control statement.
Appendix L. Indexed File Recovery	Recovering indexed files (includes examples, codes, and messages).
Appendix M. Using Other File Systems	Using Btrieve and RDM files.
Appendix N. A COBOL-Supported Subroutine	The subroutine for receiving a window handle.

How to Use This Manual

If you are a first-time user of this product, start with Chapter 1.

If you want information about the steps from COBOL program generation to execution, refer to Chapter 2 through Chapter 6.

If you want to know how to use various functions of COBOL85, refer to Chapter 7 through Chapter 13.

Conventions Used in this Manual

This manual uses the following typographic conventions.

Example of Convention	Description
setup	Characters you enter appear in bold.
<u>Program-name</u>	Underlined text indicates a place holder 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 pulldown 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 WITH PASCAL LINKAGE ALL PARAGRAPH-ID COBOL <u>ALL</u>	Commands, statements, clauses, and options you enter or select appear in uppercase. Program section names, and some proper names also appear in uppercase. Defaults are underlined.

`PROCEDURE DIVISION`
`:`
`ADD 1 TO POW-FONTSIZE OF LABEL1.`
`IF POW-FONTSIZE OF LABEL1 > 70 THEN`
`MOVE 1 TOW POW-FONTSIZE OF LABEL1.`
`END-IF.`
	This font is used for examples of program code.

The *sheet* acts as an application creation window.	Italics are occasionally used for emphasis.
"COBOL Reference Manual" Refer to "Compile Options" in Chapter 5.	References to other publications or sections within publications are in quotation marks.

Differences Between Operating Systems

Windows 95 and Windows NT are occasionally denoted with (32); Windows 3.1 is occasionally denoted with (16). COBOL85 supports the following operating systems:

- Windows 95 Operating System
- Windows NT Workstation Operating System 3.5
- Windows NT Server Network Operating System 3.5
- Windows Operating System 3.1

We have tried to make all of our examples position-sensitive. However, given the restrictions of the size of the page, in some examples we have not been able to accomplish this. You should be aware that COBOL85 is a position-sensitive language.

The term *national language* or *national* in this manual indicates double byte character languages, such as Japanese, Korean, or Chinese. Functions that are only available in the national language version of this system are indicated by [*XXXXXX*].

Related Manuals

Other manuals for COBOL85 and related products:

Title	Contents
"COBOL85 Reference Manual"	Detailed explanation of COBOL85 syntax (open system)
"Getting Started with Fujitsu COBOL"	Demonstration of COBOL85 functions using the sample programs as examples
"FUJITSU COBOL Debugging Guide"	Explanation of the debugging functions of COBOL85
"Migration Guide MF to Fujitsu COBOL"	Migration information related to converting COBOL applications from Micro Focus to COBL85
"COBOL Debugging Manual"	Usage information associated with the COBOL85
"Power COBOL User's Guide"	Information related to using Power COBOL graphical programming capabilities
"Power COBOL Programming Guide"	Explanation of how to create graphical COBOL applications
"Power FORM Getting Started"	Explanation of how to create form descriptors

The following products are not supported in the English-language version of this product:

- SequeLink
- MeFt / NET
- MeFt / NET-SV
- BS*NET
- RDB / 7000 Server for Windows NT
- RDB II
- RDB II Esql-COBOL
- PowerAIM

Trademarks

Acknowledgment

The language specifications of COBOL are based on the original specifications developed by the work of the Conference on Data Systems Languages (CODASYL). The specifications described in this manual are also derived from the original. The following passages are quoted at the request of CODASYL.

“COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations. No warranty, expressed or implied, is made by the COBOL Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by the committee, in connection therewith.

“The authors of the following copyrighted material have authorized the use of this material in part in the COBOL specifications. Such authorization extends to the use of the original specifications in other COBOL specifications:

- FLOW-MATIC (Trademark of Sperry Rand Corporation), Processing for the UNIVAC I and II, Data Automation Systems, copyrighted 1958, 1959, by Sperry Rand Corporation.
- IBM Commercial Translator, Form No. F28-8013, copyrighted 1959 by International Business Machines Corporation.
- FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell.”

Contents

Chapter 1. Overview of COBOL85

This chapter explains the functions of COBOL85 and its operating environment. If you are unfamiliar with COBOL85, you should read this chapter before using the product.

This chapter also details how to establish a development environment, set environment variables, and outlines the procedure for developing a program.

COBOL85 Functions

This section explains the COBOL functions and the various utilities provided by COBOL85.

COBOL Functions

COBOL85 has the following COBOL functions:

- Nucleus
- Sequential file
- Relative file
- Indexed file
- Inter-program communication
- Sort-merge
- Source text manipulation
- Presentation file
- Database (SQL)
- System program description (SD)
- Screen handling
- Command line argument handling
- Environment variable operation
- Report writer
- Built-in function
- Floating-point number

For information about how to write COBOL statements to use these functions, refer to the "COBOL85 Reference Manual."

Programs and Utilities Provided by COBOL85

COBOL85 provides the following programs and utilities for developing programs under Windows 95, Windows NT or Windows 3.1.

Table 1. COBOL85 programs and utilities

Name	**Purpose**
COBOL85 compiler	Compiles a described program using COBOL85.
COBOL85 run-time system	Executes a COBOL85 application.
COBOL85 interactive debugger	Allows you to debug a COBOL85 application.
PROGRAMMING-STAFF (P-STAFF)	The COBOL85 development environment.
COBOL85 FILE UTILITY	Processes a COBOL85 file.
COBOL presentation file module test (16)	Activates a COBOL presentation file module test function.
WINCOB command	Activates the COBOL85 compiler.
WINLINK command	Activates the linker.
WINEXEC command	Activates a COBOL85 application.
WINMSG command (16)	Activates the error search function.
Simplified inter-application communication	Exchanges data between applications.

An overview of the programs and utilities follows.

COBOL85 Compiler

The COBOL85 compiler compiles a COBOL source program to create an object program. The compiler provides the following service functions:

- Output compiler listings
- Checks standards and specifications
- Global optimization
- The tag-jump between a compile message and the editor (error search function)
- Provides linkage with FORM (screen and form descriptors) and Power FORM (form descriptors)

Specify these functions in accordance with the compiler options.

COBOL85 Run-time System

When you execute an application program created with COBOL85, the COBOL run-time system is called and operated.

COBOL85 Interactive Debugger

You debug COBOL85 applications created under Windows 95, Windows NT or Windows 3.1 with the COBOL85 interactive debugger. The debugger is started by a simple operation from a window, and can:

- Interrupt and restart program execution
- Reference and change the contents of a data area
- Interrupt when changing the value of a data area
- Display the execution route of a program (16)

- Display the call route of a program
- Rerun the debugging operation

PROGRAMMING-STAFF (P-STAFF)

PROGRAMMING-STAFF (P-STAFF) provides a simple means of editing, compiling, linking, executing, debugging, or maintaining programs without switching windows.

COBOL85 FILE UTILITY

The COBOL85 FILE UTILITY responds to utility commands without referencing the COBOL application.

COBOL Presentation File Module Test Function

Use the presentation file module to conduct a unit test on an application to execute interactive processing when development of an application is being used with the mainframe system designated as the host (GS-series). **Note**: This applies only to users of the GS-series.

WINCOB, WINLINK, and WINEXEC Commands

The WINCOB, WINLINK and WINEXEC commands activate the COBOL85 compiler, linker, and COBOL85 application. Use these commands to enable the COBOL85 compiler, linker, and application to be activated from a window.

WINMSG Command (16)

The WINMSG command opens a message file to be used with the COBOL85 error search function. Only the Windows 3.1 version supports this command.

Simplified Inter-Application Communication

Simplified inter-application communication provides a means for servers to exchange messages between user programs.

COBOL85 Development Environment

The following figure provides an overview of the COBOL85 development environment.

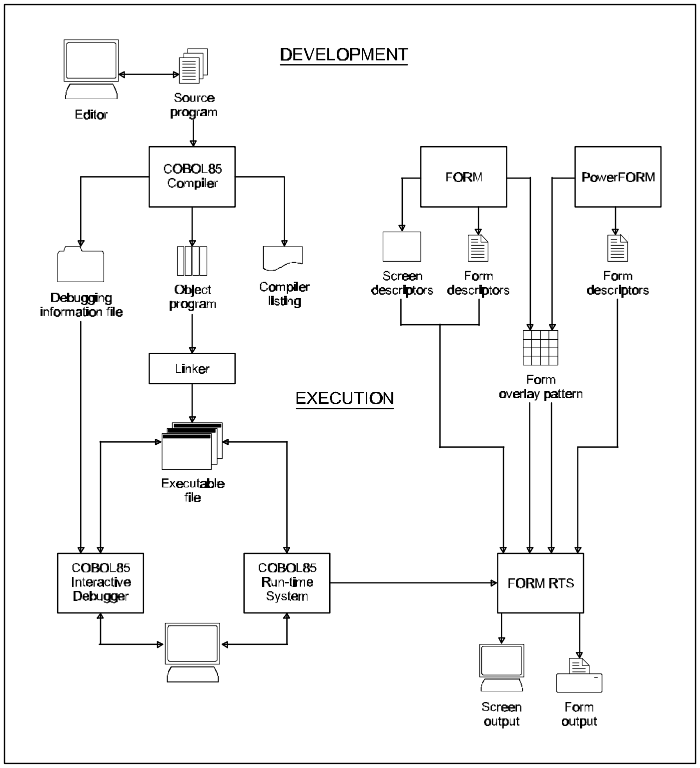


Figure 1. The COBOL85 development environment

Applications created using the Windows 3.1 version of COBOL85 are executed as 16-bit applications. Applications created using the Windows 95 and Windows NT version of COBOL85 are executed as 32-bit applications.

Setting Up Environment Variables

Set the following environment variables according to the COBOL85 functions you want to use.

When using Windows 95, register in AUTOEXEC.BAT by using the text editor.

When using Windows NT, set the variables at the system control panel or command prompt. Refer to “Setting from the Control Panel” and “Setting from the Command Prompt” in Chapter 5.

When using Windows 3.1, set the variables before activating Windows.

Table 2. Setting environment variable

Setup Timing	Environment Variable	Details	Conditions	
Common	PATH	COBOL85 install directory	When using COBOL85	Required
		FORM RTS install directory	When using screen and form descriptors	Required
	TMP	Working directory name	When using P-STAFF	Required
	TEMP	Working directory name	When using P-STAFF	Required
At compilation (32)	SMED_SUFFIX	Extension of screen and form descriptors (any character string)	When changing the extension PMD of screen and form descriptors	Optional
	FORMLIB	Directory name of the file storing screen and form descriptors	When using screen and form descriptors	Optional

Table 2. Setting environment variables (cont.)

Setup Timing	Environment Variable	Details	Conditions	
At compilation (32)	FFD_SUFFIX	Extension of file descriptor (any character string)	When changing the extension FFD of the file descriptor	Optional
	FILELIB	Directory name of file storing the file descriptor	When using the file descriptor	Optional
At linking	LIB	COBOL85 install directory. Directory name of the file to be combined when linking	When linking	Required
	TMP	Working directory name to be used at linking	When linking	Required
At execution	BSORT_TMPDIR	Working directory name	When using the sort/merge function	Optional
			When using the COBOL85 FILE UTILITY	Optional
	MEFTDIR	Directory name of window information file and printer information file	When using screen and form descriptors	Optional
	TEMP	Working directory name	When using the sort/merge function	Required
			When using the COBOL85 FILE UTILITY	Required

When Using Screen and Form Descriptors (32)

SMED_SUFFIX=NONE

Specifying 'None' indicates there is no extension.

FORMLIB=E:\FORM

Specifies the storage location for screen and form descriptors.

When Using the File Descriptor (32)

FFD_SUFFIX = NONE

Specifying 'None' indicates there is no extension.

FILELIB = E:\FILE

Specifies the storage location for the file descriptor.

Related Products

COBOL85 supports the following products.

Table 3. Related products

Name of Product	Function
FORM	Design screen and form layouts
Power FORM	Design form layouts
FORM overlay option	Design overlay pattern descriptors
FORM RTS	Process input/output of screens and forms

An overview of each product is given below.

FORM and Power FORM

FORM designs screens and forms to be displayed and printed by the COBOL program. Use FORM interactively to design the layout of screens and forms.

Power FORM designs the forms to be printed by the COBOL program. Use Power FORM interactively to design the layout of forms.

FORM Overlay Option

The FORM overlay option is a FORM product for designing the overlay pattern to be printed by a COBOL program. You can interactively design the layout of the overlay pattern.

FORM RTS

FORM RTS (sometimes referred to as MeFt) is used implicitly when a program that reads and writes screens and forms is executed. FORM RTS edits the format after receiving a screen or form I-O request from the program.

Developing a Program

The standard procedure for developing a program using P-STAFF is explained in the following figure.

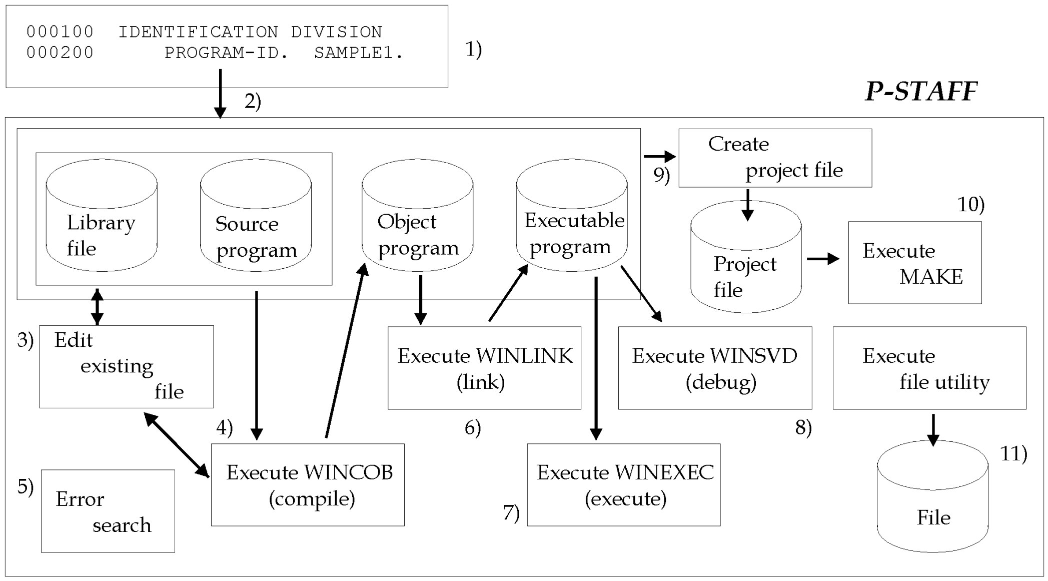


Figure 2. Developing a program with P-STAFF

1) Describe the COBOL source program.

2) Activate P-STAFF.

3) Edit the program after P-STAFF has been activated. ((16): PowerFRAMEVIEW editor / (32): P-STAFF editor)

4) Execute WINCOB to compile the program. Refer to Chapter 3, “Compiling Programs.”

5) Use the error search function to correct a COBOL statement where a compile error has occurred. Refer to “Correcting a Compile Error” in Chapter 3.

6) Execute WINLINK to link a program. Refer to Chapter 4, “Linking Programs.”

7) Execute WINEXEC to execute a program. Refer to Chapter 5, “Executing Programs.”

8) When the program does not execute as expected, or when the unit test is being executed on a program (16) , debug a program with the interactive debugger. Refer to the “Fujitsu COBOL Debugging Guide” for more details.

9) When an application consists of several programs, use the project management function for developing and maintaining the application. Refer to Chapter 6, “Project Management Function.”

10) When a program contained within a project-managed application has been corrected, re-create the application. In this case, execute the rebuild command. Refer to Chapter 6, “Project Management Function.”

11) To process a COBOL file, execute the COBOL85 FILE UTILITY. Refer to “COBOL85 FILE UTILITY” in Chapter 7.

COMPILE and LINK can be done without relying on P-STAFF. Refer to “Using Commands to Compile,” in Chapter 3, “Using Commands to Link,” in Chapter 4, and Appendix J, “Command Formats.”

Chapter 2. Creating and Editing a Program

This chapter explains how to create and edit a program, how to create library text and the program format, and describes compiler directing statements.

Creating the Program

You create COBOL source programs and library text with editors. This section explains how to create the COBOL source program and library text.

Creating and Editing a COBOL Source Program

Simply, you create a COBOL source program by:

- Activating the editor
- Creating and entering text
- Storing the program

Activating the Editor

To display the edit screen, activate the editor.

Creating and Editing the Program

You format COBOL source programs using the COBOL reference format. Enter the line number, COBOL statement and procedure-name in the positions stipulated in the reference format.

Enter the line number, COBOL statement, and procedure identifier in the edit screen, as shown in the following example.

```
                    (1) (2)(3)   (4)

Column position  12345678901234567890
                 000001 IDENTIFICATION DIVISION.          (5)
                 000002  PROGRAM-ID. PROG1.
                                 :
                                 :
                 000012  01  A  PIC X(5) VALUE "AB D".    (6)
                                 :
                                 :
                 000021     DISPLAY A.
```

(1) Sequence number area (columns 1 to 6)
Specify the line number in the sequence number area. The line number can be omitted.

(2) Indicator area (column 7)
Use the indicator area when continuing a line or to change a line to a comment line. In all other cases, be sure to enter a space.

(3) Area A (columns 8 to 11)
Normally, COBOL divisions, sections, paragraphs and end program headers are described in this area. Data items whose level-number is 77 or 01 are also described in this area.

(4) Area B (column 12 and on)
Normally, COBOL statements, comment-entries and data items whose level-number is not 77 or 01 are described in this area.

(5) Line feed character (end of line)
Enter a line feed character at the end of each line.

(6) TAB character
A TAB character can be specified as the nonnumeric literal in a COBOL source program. The TAB character takes up 1 byte within the non-numeric literal.

Storing the Program and Quitting the Editor

After creating and editing the source program, store the program in a file, then quit (exit) the editor. Normally, you affix the extension COB or CBL to the name of the file.

When using Windows 95 or Windows NT, you can also use the extension COBOL. Affixing the extension COB, CBL or COBOL facilitates specification of the file name when compiling the program.

Creating Library Text

You create library text, fetched by the COBOL source program with COPY statements, in the same manner as you create COBOL source programs. However, the reference format (fixed, variable or free) for the library text does not have to be the same as the format of the COBOL source program that fetches the library text.

As with the COBOL source program, you specify compiler options to set up the reference format for the library text. However, when a single program fetches several library texts, the reference formats for all the library texts must be identical.

Normally, append the extension CBL or COB to the name of the file in which the library text is to be stored. When using Windows 95 or Windows NT, the extension COBOL can be used.

Screen and form descriptors can use any method of reference format.

Program Format

Each line in the COBOL source program is delimited with a line feed character, in accordance with the rules governing the reference format.

There are three types of reference formats: fixed, variable and free. You specify the type with a compiler option before compiling the program.

The line feed character that delimits each line is not regarded as part of the line.

Fixed Format

In a fixed format, each line in the COBOL source program has a fixed length of 80 bytes.

(Column position)

1 6	7	8	12 72	73 80
Sequence number area	*	Area A	Area B	**

* Indicator area

** Program identification number area

Variable Format

In a variable format, each line in the COBOL source program can be up to 251 bytes long.

(Column position)

1 6	7	8	12 ≤ 251
Sequence number area	*	Area A	Area B

* Indicator area

Free Format

In a free format, you do not need to distinguish between the sequence number area, the indicator area, Area A, Area B, or the program identification number area. Each line can be up to 251 bytes long.

(Column position)
1 $\leq$ 251

Compiler Directing Statement

A compiler directing statement indicates the compiler options. Normally, compiler options are specified in the Compiler Options dialog box of the WINCOB window that runs compilation, but the options can also be defined within the source program.

The description format for the compiler directing statement is shown below.

```
@OPTIONS [compiler-option [,compiler-option]...]
```

- Enter "@OPTIONS" starting at column 8.
- Enter at least one space between "@OPTIONS" and the compiler options.
- Each compiler option must be delimited by a comma.
- A compiler directing statement indicates the starting position of each separately compiled program. The compiler options specified in the compiler directing statement apply only to the corresponding separately compiled programs in the compiler directing statement.

For example:

```
000100 @OPTIONS MAIN, APOST
000200 IDENTIFICATION DIVISION.
          :
          :
```

Do not use a tab as a separator in an @OPTIONS compiler directing statement.

The compiler option values that can be specified in the compiler directing statement are restricted. For details, refer to Appendix A, “Compiler Options.”

Chapter 3. Compiling Programs

This chapter explains how to compile programs, using the sample programs provided as examples. It describes the resources required for compilation, and gives a sample compilation procedure. This chapter also explains how to correct compile errors and use commands to run the compiler.

Compiling Sample Programs

This section explains the steps required to compile the sample programs located in the SAMPLES directory (in the directory where the COBOL85 compiler is installed). Each sample file is stored separately.

Windows 95 and Windows NT

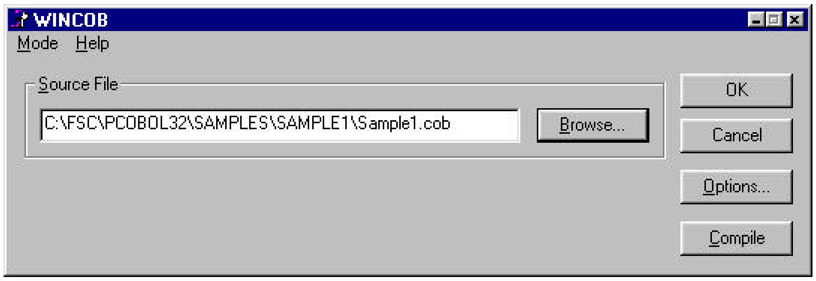
In the following explanation, it is assumed the sample program is stored in:

```
C:\FSC\PCOBOL32\SAMPLES\SAMPLE1\SAMPLE1.COB
```

Start PROGRAMMING-STAFF by clicking on Start, Programs, Fujitsu COBOL and Programming-Staff 32.
The PROGRAMMING-STAFF (P-STAFF) window opens.

Select WINCOB [Compile] from the Tools menu in the P-STAFF window. The WINCOB window opens.

Click on the Browse button, and navigate to the directory where the sample program is stored. Alternatively, you can specify the file name in the edit box by entering it directly.



Source File: C:\FSC\PCOBOL32\SAMPLES\SAMPLE1\Sample1.cob

Figure 3. The WINCOB window

Click on the Options button of the WINCOB window. The Compiler Options dialog box is displayed.

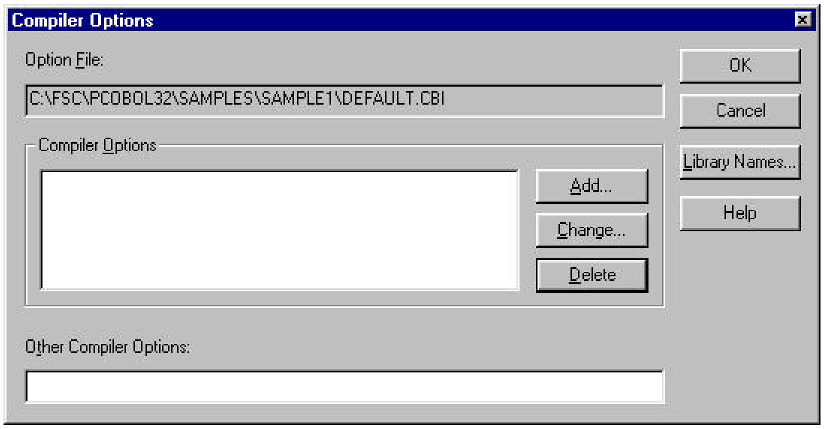


Figure 4. The Compiler Options dialog box

Click on the Add button. The (Add) Compiler Options dialog box, containing a list of available options, is displayed.

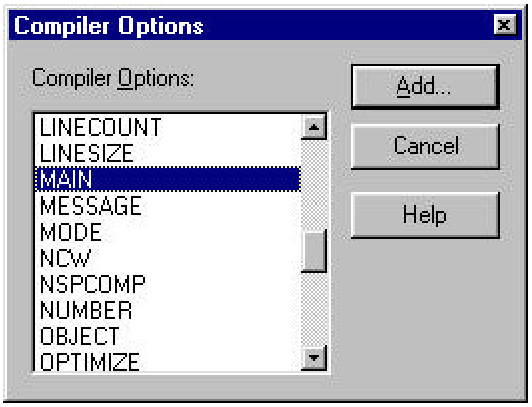
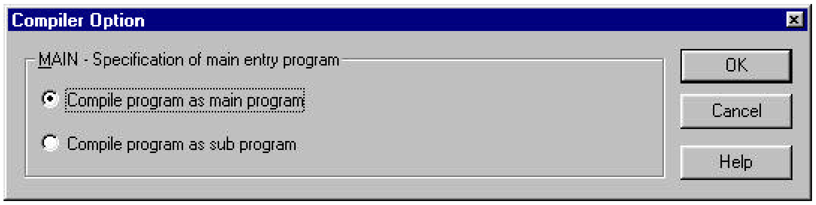


Figure 5. The (Add) Compiler Options dialog box

Select MAIN in the Compiler Options list box and click on the Add button. The (Details) Compiler Option dialog box is displayed. **Note**: The main program must be defined in order to distinguish it from the subprograms.


Compiler Option

MAIN - Specification of main entry program

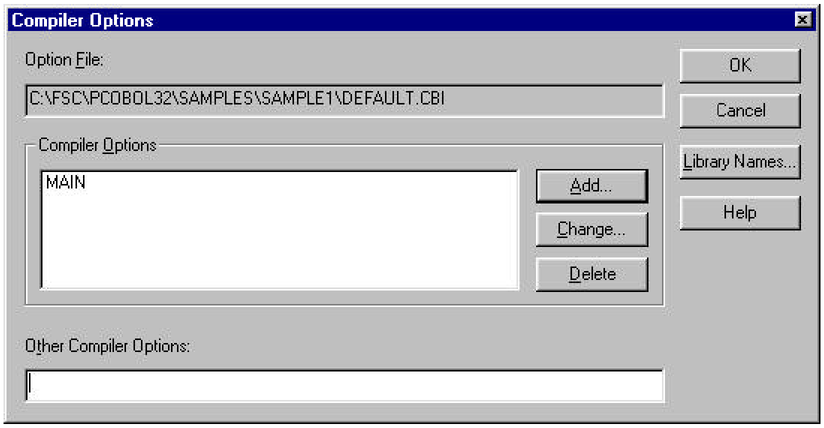
[✓ Compile program as main program]

Compile program as sub program

Figure 6. The (Details) Compiler Option dialog box

Select Compile program as main program and click on the OK button. The (Details) Compiler Options dialog box is redisplayed.

Since there are no more options to be added, click on the Cancel button. The (Add) Compiler Options dialog box closes. The Compiler Options dialog box is redisplayed.


Compiler Options

Option File: C:\FSC\PCOBOL32\SAMPLES\SAMPLE1\DEFAULT.CBI

Compiler Options: MAIN

Other Compiler Options:

Figure 7. The Compiler Options dialog box with MAIN specified

Verify that MAIN appears in the Compiler Options list box of the Compiler Options dialog box, then click on the OK button. The

Compiler Options dialog box closes and the WINCOB window is displayed.

Click on the Compile button in the WINCOB window. Compilation starts, and a window showing compilation progress is displayed.

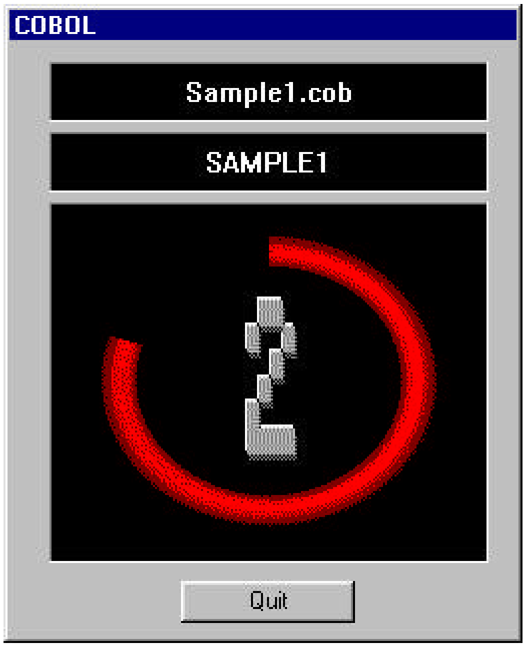
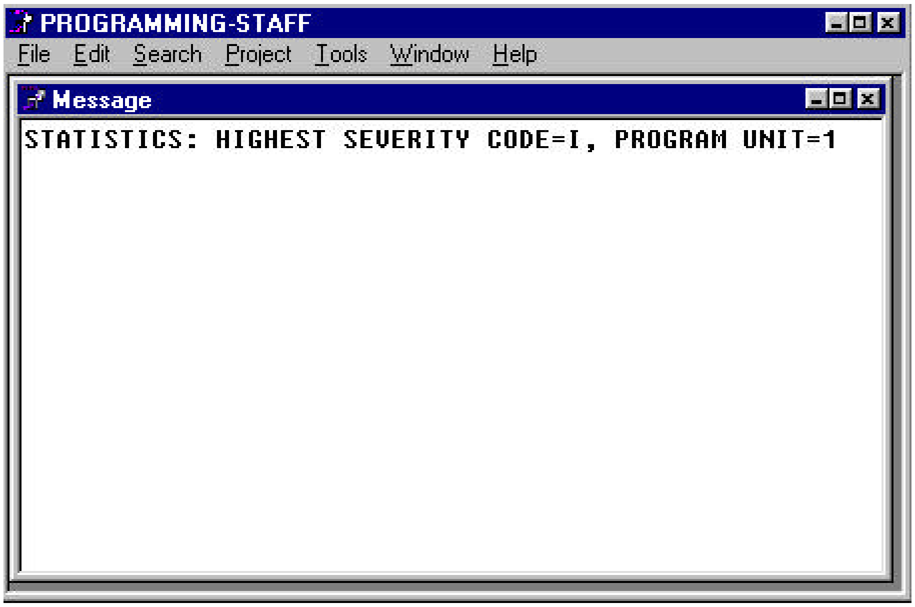


Figure 8. The compilation progress dialog box

Compilation ends. Compiler messages are displayed in the P-STAFF window.



```
STATISTICS: HIGHEST SEVERITY CODE=I, PROGRAM UNIT=1
```

Figure 9. The PROGRAMMING-STAFF window

Verify that compilation completed normally and then click on the Cancel button in the WINCOB window. The WINCOB window closes. Click on Exit in the File menu to close the P-STAFF window.

Compilation of the sample program is now complete. Verify that the object program (SAMPLE1.OBJ) was created in the same directory (C:\FSC\PCOBOL32\SAMPLES\SAMPLE1 in this example) as the directory where the sample program was stored.

If compilation does not complete normally, the program may not have been installed properly. Verify that the program was installed properly.

Windows 3.1

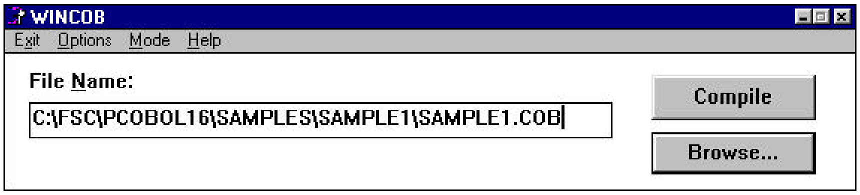
In the following explanation, it is assumed the sample program is stored in:

```
C:\FSC\PCOBOL16\SAMPLES\SAMPLE1\SAMPLE1.COB
```

Double-click on the PROGRAMMING-STAFF icon. The PROGRAMMING-STAFF (P-STAFF) window opens.

Select WINCOB from the Utility menu in the P-STAFF window. The WINCOB window opens.

Click on the Browse button, and navigate to the directory where the sample program is stored. Alternatively, you can specify the file name in the edit box by entering it directly.



File Name: C:\FSC\PCOBOL16\SAMPLES\SAMPLE1\SAMPLE1.COB

Figure 10. The WINCOB window

Select Options in the WINCOB window. The Compiler Options dialog box is displayed.

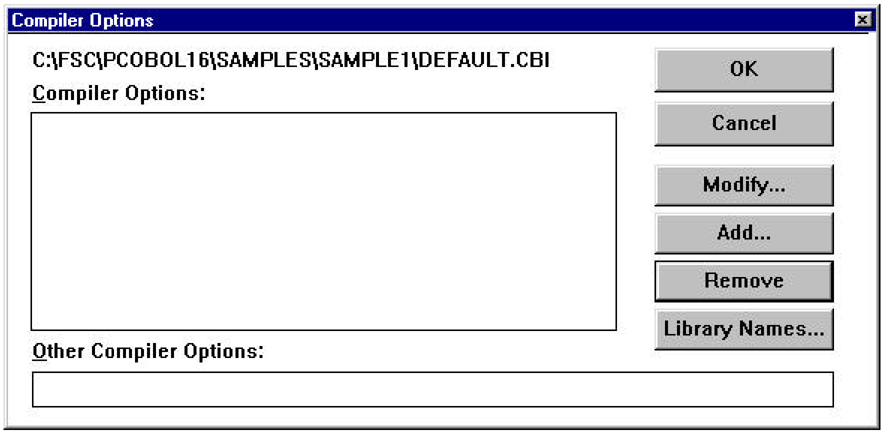


Figure 11. The Compiler Options dialog box

Click on the Add button. The (Add) Compiler Options dialog box, containing a list of available options, is displayed.

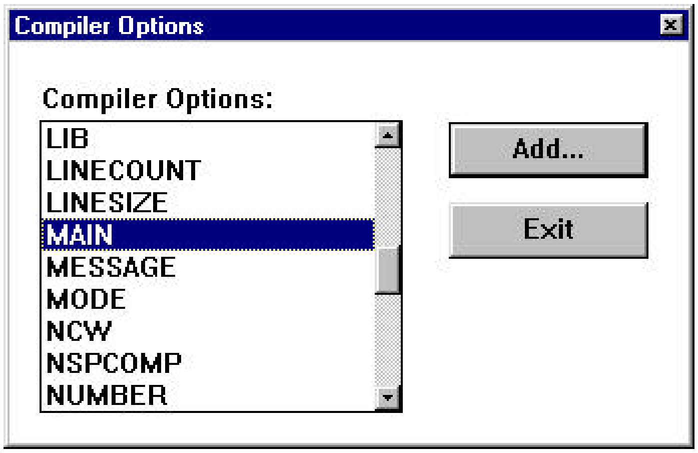
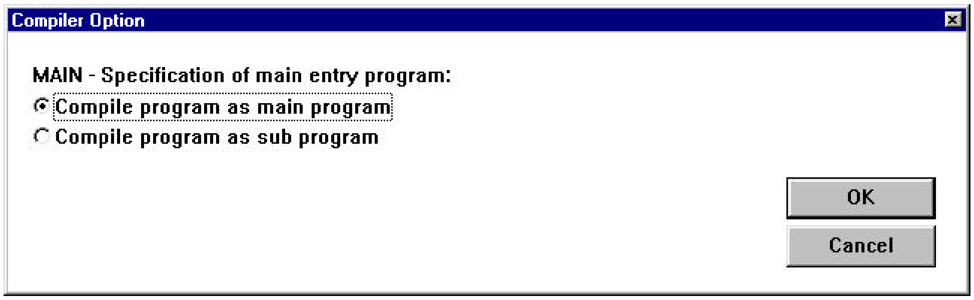


Figure 12. The (Add) Compiler Options Dialog Box

Select MAIN in the Compiler Options list box and click on the Add button. The (Details) Compiler Option dialog box is displayed.


Compiler Option

MAIN - Specification of main entry program:

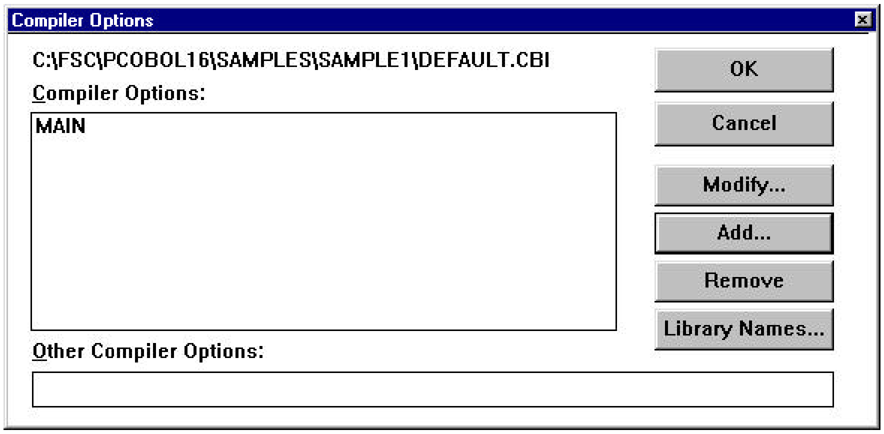
(●) Compile program as main program

( ) Compile program as sub program

Figure 13. The (Details) Compiler Option dialog box

Select Compile program as main program and click on the OK button. The (Details) Compiler Options dialog box is redisplayed.

Since there are no options to be added, click on the Exit button. The (Details) Compiler Options dialog box closes. The Compiler Options dialog box is redisplayed.


Compiler Options

C:\FSC\PCOBOL16\SAMPLES\SAMPLE1\DEFAULT.CBI

Compiler Options:

MAIN

Other Compiler Options:

Figure 14. The Compiler Options dialog box with MAIN specified

Verify that MAIN appears in the Compiler Options list box of the Compiler Options dialog box, then click on the OK button. The Compiler Options dialog box closes and the WINCOB window is displayed.

Click on the Compile button in the WINCOB window. Compilation starts, and a window showing compilation progress is displayed.

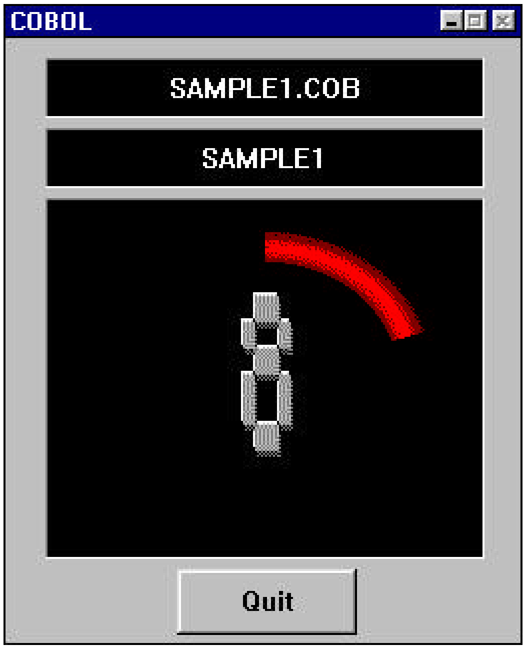
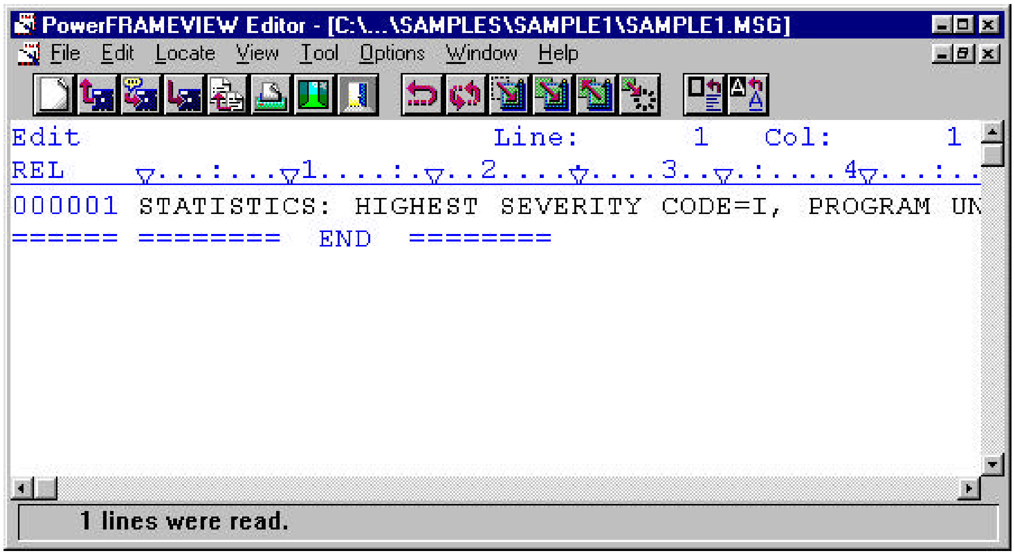


Figure 15. The compilation progress window

Compilation ends. Compiler messages are displayed in the PowerFRAMEVIEW Editor.


PowerFRAMEVIEW Editor - [C:\...\SAMPLES\SAMPLE1\SAMPLE1.MSG]
File Edit Locate View Tool Options Window Help

```
Edit                          Line:        1    Col:        1
REL     ▽...:...▽1....:.▽..2....▽....3..▽.:....4▽...:..
000001 STATISTICS: HIGHEST SEVERITY CODE=I, PROGRAM UN
====== ======== END ========
```

1 lines were read.

Figure 16. The PowerFRAMEVIEW Editor

Verify that compilation completed normally and then select Exit in the WINCOB window. The WINCOB window closes.

Compilation of the sample program is now complete. Verify that the object program (SAMPLE1.OBJ) was created in the same directory (C:\FSC\PCOBOL16\SAMPLES\SAMPLE1 in this example) as the directory where the sample program was stored.

If compilation does not complete normally, the program may not have been installed properly. Verify that the program was installed properly.

Resources Necessary for Compilation

This section explains the files used by the COBOL85 compiler and the information that it produces.

Files Used by the COBOL85 Compiler

The COBOL85 compiler uses the following files:

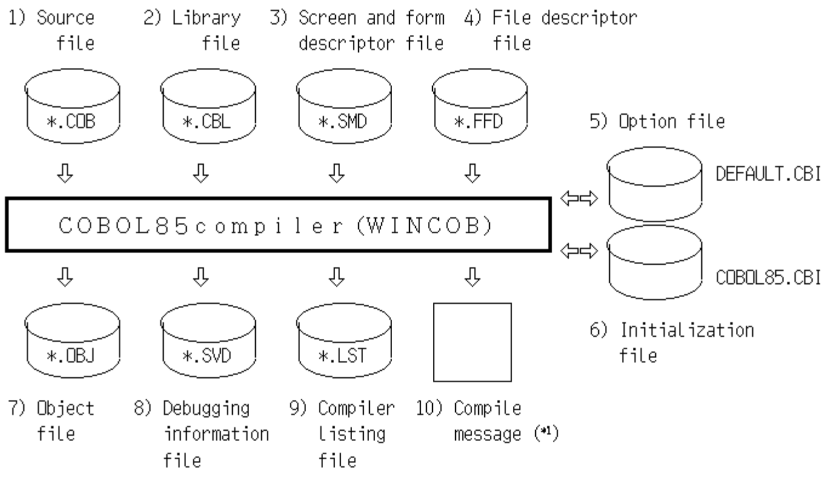


Figure 17. Files used by the COBOL85 compiler

*1 A compile message is written in the editor when compilation terminates.

The following table lists the files used by the COBOL85 compiler.

Table 4. Files used by the COBOL85 compiler

	File Contents	**File Name Format**	**I/O**	**Condition**	**Option**
1	Source programs	Any (Normally, use the extension COB, CBL or COBOL)	I	Required	
2	Library text	library-text-name.CBL(*1)	I	Follow the specification method when compiling a source program that uses the COPY statement ⇒ (*3)	LIB
3	Screen and form descriptors	screen-and-form -descriptor.PMD (*2)	I	Follow the specification method when compiling a source program that uses screen and form descriptors ⇒ (*4)	FORMLIB FORMEXT
4	File descriptor	file-descriptor.FFD (*2)	I	Follow the specification method when compiling a source program that uses a file descriptor ⇒ (*4)	FILELIB FILEEXT

Table 4. Files used by the COBOL85 compiler (cont.)

	File Contents	File Name Format	I/O	Condition	Option
5	Character-strings indicating compiler options	DEFAULT.CBI or Any name (Refer to Appendix J)	I	When specifying compiler options stored in a file using the WINCOB window or COBOL32 command	
			O	When compiler options are set up in the Compiler Options dialog box for the WINCOB window	
6	Information used by the COBOL85 compiler	COBOL85.CBI	I/O	Created in the directory which COBOL85 was installed	
7	Object program	source-file-name.OBJ	O	When the source program compilation completes normally	OBJECT
8	Debugging information	source-file-name.SVD	O	When the compiler option TEST is specified	TEST
9	Compiler listings	source-file-name. LST	O	When compiler listings is output	PRINT
10	Compile message			Is displayed	

Notes:

(*1) The extension COB or COBOL may be used instead of CBL. The system searches the extension for the file to be compiled in the following order:

- CBL
- COB
- COBOL (32)

(*2) Each extension can be changed to any character string by using the compiler option (FORMEXT / FILEEXT) or environment variable (SMED_SUFFIX / FFD_SUFFIX).

Refer to "Setting Up Environment Variables" in Chapter 1 and Appendix A, "Compiler Options."

(*3) When COPY statements have been entered without IN / OF in the COBOL source program either:

- Specify the directory of the library text file in compiler option LIB.
- Store the library text file in the current directory. (32)
  When compiling from the WINCOB window, the current directory is the directory where the COBOL source program is stored.
- Store the library text files in the same directory as the source file. (16)

When COPY statements have been entered with IN / OF in the COBOL source program:

- Associate the storage directory of the library text file with the library specified by IN / OF. Failure to create an association with the library causes a compile error to occur during compilation.

(*4) Specify the directory of the file (screen and form descriptors / file descriptor) in the compiler option (FORMLIB / FILELIB).

- Specify the directory of the file (screen and form descriptors / file descriptor) in the environment variable (FORMLIB / FILELIB). (32)
- Store the file (screen and form descriptors / file descriptor) in the current directory. (32)
  When compiling from the WINCOB window, the current

directory is the directory where the COBOL source program is stored.

- Store the file (screen and form descriptors / file descriptor) in the same directory as the source file. (16)

Information Provided by the COBOL85 Compiler

The COBOL85 compiler reports program compilation results as a diagnostic message. The diagnostic message appears in the window when compilation terminates.

To store the message in a file, either save the displayed message as a file or specify the compiler option PRINT before compiling.

Specifying the compiler option MESSAGE writes option and statistical information listings for separately compiled programs. These lists indicate the compiler options and provide information on compiled programs. These lists are also called compiler listings.

The formats of the compiler listings written when compiler option PRINT and MESSAGE specified are shown below.

Option Information Listing

The numbers in parentheses () correspond to the notes that follow these examples.

```
COBOL85 V30L10                     TUE JAN 30 19:00:45 1996 0001
                                                 (1)          (2)
 ** OPTIONS SPECIFIED **
MAIN
MESSAGE
TERM                                                     (3)
** OPTIONS ESTABLISHED **
ALPHAL   LINECOUNT(60) SDS                               (4)
BINARY(WORD, MLBON) LINESIZE(136) NOSOURCE
  :   :   :
  :   :   :
```

Diagnostic Message Listing

```
COBOL85 V30L10 program-name  TUE JAN 30 19:00:45 1996 0002 (5)
  ** DIAGNOSTIC MESSAGE ** (program-name)
  C:\TEST.COB 4: JMN2503I-S USER WORD 'A' IS UNDEFINED.  (6)
```

Compile Unit Statistical Information Listing

```
COBOL85 V30L10 program-name TUE JAN 30 19:00:45 1996 0003
  ** STATISTICS **
  FILE NAME = C:\TEST. COB
  DATE AND TIME = TUE JAN 30 1996 19:00:45
    SOURCE STATEMENTS   = 5 RECORDS                      (7)
    PROGRAM SIZE (CODE)     = 0 BYTES                    (8)
    CONTROL LEVEL   = 1 LEVEL                            (9)
    CPU TIME = 4.72 sec.                                 (10)
  HIGHEST SEVERITY CODE = S                              (11)
```

Notes:

(1) Indicates the compilation date.

(2) Indicates the page number.

(3) Indicates the compiler options specified by the user.

(4) Indicates a list of compiler options established by the COBOL85 compiler.

(5) Indicates the program name.

(6) Indicates the diagnostic message output by the COBOL85 compiler. For details on the format of the diagnostic message, refer to Appendix F, "Message Lists."

(7) Indicates the number of records in the source program input by the COBOL85 compiler. When a library has been fetched, the number of fetched records is included.

(8) Indicates the size of the object program. When the compilation operation was completed normally and the object program has been output, the DATA size is also output.

(9) Indicates the level of the compiler used.

(10) Indicates the time required for compilation.

(11) Indicates the highest severity code among output diagnostic message codes.

Compiling a COBOL Source Program

There are two methods to compile a COBOL source program. One method involves running WINCOB commands from a window, and the other method involves running the compiler from the command line. This section explains the procedure for compiling the program using the WINCOB window.

For details on how to compile the program using the commands, see “Using Commands to Compile.”

A source program containing several compilation units within one file cannot be compiled.

Compiling a Single Source File

To compile files:

1. Activate the WINCOB window. See “Activating the WINCOB Window.”
2. Specify the name of the source file to be compiled in the File Name edit box. See “Specifying a File Name” (for single compilation).
3. Establish the compiler options. See “Setting Up Compiler Options.” Specify the name of the source file before establishing the compiler options.
4. Establish the library name. See “Setting a Library Name.”

5. Start compilation. See “Starting and Quitting Compilation.”
6. Correct any compile errors. When compilation terminates, a compile message appears in the window of P-STAFF editor. When a compile error is detected, correct the program by using the error search function, then restart compilation. See “Correcting a Compile Error.”
7. Quit compilation. See “Starting and Quitting Compilation.”

- To continue compilation, close the message and repeat the operation from Step 2.
- To quit compilation, click on the Cancel button (32) or select Exit (16) in the WINCOB window.

When a compile ends normally, an object program is created. The object program has to be linked before it can be executed. Linking is explained in Chapter 4, “Linking Programs.”

Compiling Several Source Files

To compile several files consecutively with WINCOB:

1. Activate the WINCOB window. See “Activating the WINCOB Window.”
2. Specify continuous compilation. See “Specifying Continuous Compilation.”
3. Specify the names of the source files to be compiled in the WINCOB (Continuous Compilation mode) List box. See “Specifying a File Name” (for continuous compilation).
4. Establish the compiler options. See “Setting Up Compiler Options.” Select and establish compiler options and a library name separately for each directory of files selected from the list box.

5. Establishing a library name. See “Setting a Library Name.” Select and establish compiler options and a library name separately for each directory of files selected from the list box. When running continuous compilation, the same compiler options must be specified for source files in the same directory. To specify different compiler options, enter a compiler directing statement within the COBOL source program. Refer to “Compiler Directing Statement” in Chapter 2.
6. Specify the main program. See “Specifying the Main Program.” The options set up at this step have priority over options specified at the dialog box even if MAIN was specified.
7. Start compilation. When compilation of one file ends, start compilation of the next file. See “Starting and Quitting Compilation.”
8. Correct any compile errors. When compilation ends, a compile message appears in the window. When a compile error is detected, a compile message for all files appears in the window after continuous compilation ends. Correct the program by using the error search function, then restart compilation. See “Correcting a Compile Error.”
9. Quit compilation. See ‘Starting and Quitting Compilation.”

- To continue compilation, close the message and repeat compilation from Step 3.
- To quit compilation, click on the Cancel button (32) or select Exit (16) in the WINCOB window.

If no errors are found in the COBOL source program and compilation ends normally, an object program is created.

The object program cannot be run as it is. The object program must be linked to make it executable.When compilation ends, link the program. For details on linking, refer to Chapter 4, “Linking Programs.”

WINCOB Window

The WINCOB window runs compilation. This section explains the WINCOB window. For further details about using the window, refer to the online help.

Activating the WINCOB Window (32)

The WINCOB window can be started by either:

- Selecting WINCOB from the Tools menu of the P-STAFF window

  or

- Executing “WINCOB.EXE”.

The WINCOB (Standard Compilation mode) window is displayed.

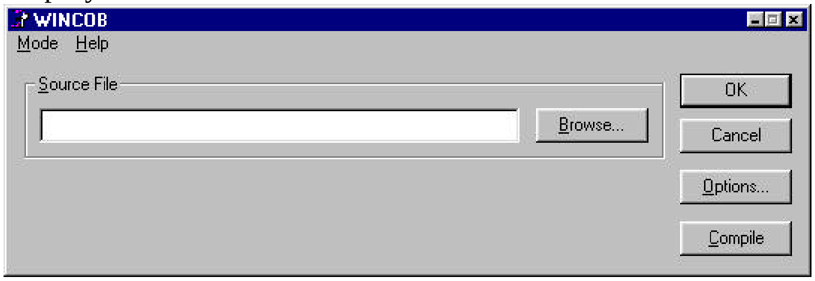


Figure 18. The WINCOB window in Standard Compilation mode

The WINCOB window contains the following elements:

Mode

Specify whether a single program (standard) or multiple programs (continuous) are to be compiled.

Help

Access the online help.

Source file edit box

Specify the COBOL source file to be compiled.

Browse button

Click to browse directories to select a COBOL source file.

OK button

Click to start compilation.

Cancel button

Click to close the WINCOB window.

Options button

Click to specify compile options.

Compile button

Click to start compilation.

Specify the COBOL source program to be compiled by either entering it in the source file edit box, or by using the Browse button to select a file.

Specifying Continuous Compilation

To compile more than one program, select Continuous compilation from the Mode menu in the WINCOB window. The WINCOB window expands to include the continuous compilation options. Specify the file names to compile in the WINCOB (Continuous Compilation mode) window.

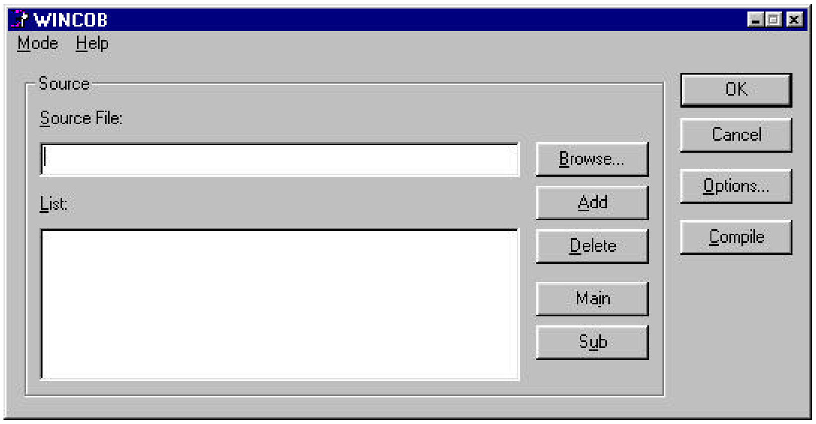


Figure 19. The WINCOB window in Continuous Compilation mode

The WINCOB (Continuous Compilation mode) window contains the following elements:

Mode

Specify whether a single program (standard) or multiple programs (continuous) are to be compiled.

Help

Access the online help.

Source file edit box

Specify the COBOL source file to be added to the List.

List (list box)

List of COBOL source files to be compiled.

Browse button

Click to browse directories to select a COBOL source file.

Add button

Click to add the Source File to the List.

Delete button

Click to remove the selected file(s) from the List.

Main button

Click to indicate that the selected file is a main program.

Sub button

Click to indicate that the selected file is a sub program.

OK button

Click to start compilation.

Cancel button

Click to close the WINCOB window.

Options button

Click to specify compile options.

Compile button

Click to start compilation.

Specifying a File Name

The method of specifying a file name differs for single compilation and continuous compilation. Both methods are explained below.

- To specify the file name to be used for standard (single) compilation:

  Specify the name of the file to be compiled in the Source File edit box either by entering it directly from the keyboard or selecting it using the Browse button.

- To specify the file names to be used for continuous (multiple) compilation:
  - As with single compilation, specify the name of each file to be compiled in the Source File edit box.
  - Click on the Add button.

Setting Compiler Options

You specify compiler options by selecting the Options button and setting the options in the displayed Compiler Options dialog box. See “Compiler Options Dialog Box.”

Specify the source file name before setting up the compiler options.

Specifying the Main Program

To specify a main program, the compiler option MAIN must be specified.

There are levels of priority for the compiler option MAIN, depending on whether the option was selected for single or continuous compilation.

The priority for compiler option MAIN is as follows:

1. Specified by a compiler directing statement
2. Specified by clicking on the Main button
3. Specified in the Compiler Options dialog box

To specify the main program for single compilation, specify the compiler option MAIN from the (Add) Compiler Options dialog box.

To specify the main program for continuous compilation, select the Main button in the WINCOB (Continuous Compilation mode) window:

1. Select the file name to be compiled as the main program from the Compilation list box. Multiple files can be selected.
2. Click on the Main button. The small icon next to the selected file turns red.

The setting of Main or Subprogram in the WINCOB (Continuous Compilation mode) window takes priority over the setting in the Options dialog box.

When selecting files from the compilation list, failure to click on the Main button causes the selected file name to be regarded as a subprogram. Be sure to set up the file as the main program by clicking on the Main button.

Starting and Quitting Compilation

To start compilation, click on the Compile button. Compilation starts, and a window showing compilation progress is displayed.

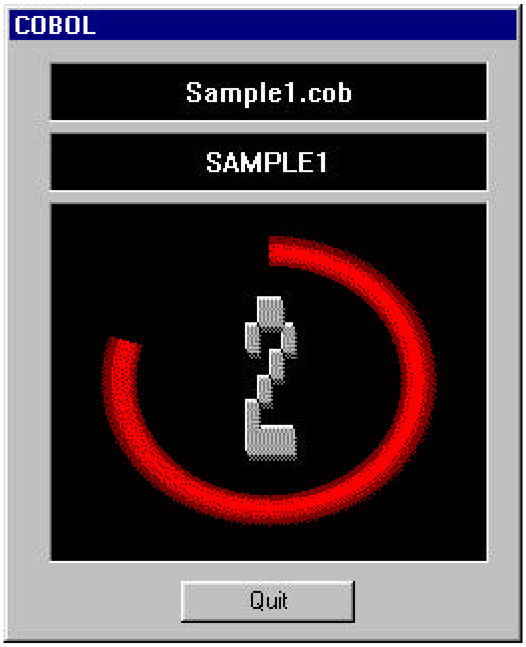


Figure 20. The compilation progress window

Note: The countdown begins at number 9.

Click on the Quit button to interrupt the compile.

When compilation ends, a compile message appears in the window of P-STAFF editor. If a compile error is detected, correct the program by using the error search function. See “Correcting a Compile Error.”

To quit compilation, click on the Cancel button to close the WINCOB window.

Compiler Options Dialog Box

This section explains how to set the compiler options and library names. You must define the source file name to be compiled in the WINCOB window before setting the compiler options.

To set compiler options, click on the Options button in the WINCOB window to open the Compiler Options dialog box, then set the options. When necessary, open the Library Names dialog box from the Compiler Options dialog box and define the library names.

For details on the compiler options that can be set, refer to Appendix A, “Compiler Options.” For additional details on how to use the windows, refer to the online help.

Setting Compiler Options

Set the compiler options from the Compiler Options list box or the Other Compiler Options edit box.

When setting compiler options in the Compiler Options list box, use the Add, Change and Delete buttons. When setting compiler options in the Other Compiler Options edit box, enter the compiler options as character strings directly from the keyboard.

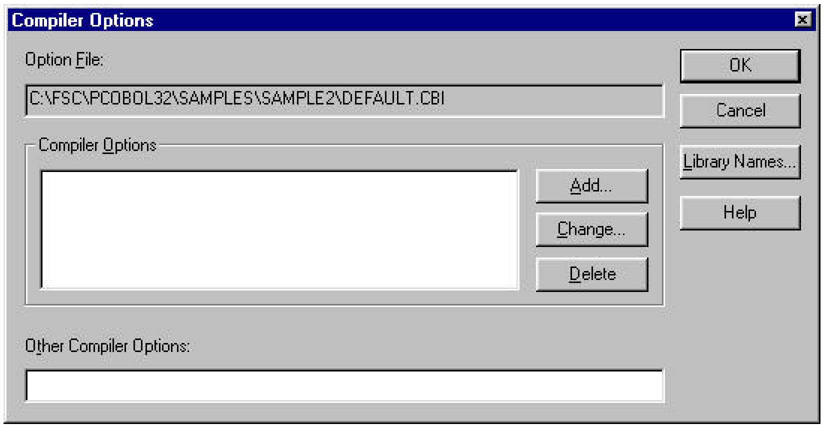


Figure 21. The Compiler Options dialog box

The Compiler Options dialog box contains the following elements:

Option File display field

Displays the file in which option settings will be saved.

Compiler Options list box

Displays compiler options selected by the Add button.

Add button

Click to select from a list of compiler options.

Change button

Click to change the setting of the selected compiler option(s).

Delete button

Click to remove the select compiler option(s) from the list box.

Other Compiler Options edit box

Allows the compile options to be keyed in directly.

OK button

Click to confirm the compiler option settings or changes.

Cancel button

Click to cancel any compiler option settings or changes and return to the state before the dialog box was opened.

Library Names button

Click to specify the library text files.

Help button

Click to access the online help.

Previously specified compiler options, as well as those you add, are displayed in the compiler options list field.

Adding a Compiler Option

To set a compiler option that is not shown in the Compiler Options list box:

1. Click on the Add button. The (Add) Compiler Options dialog box appears.
2. Select the compiler option to be added from the Compiler Options list box and click on the Add button. The (Details) Compiler Option dialog box then appears.
3. Specify the details to be set for the compiler option. The details in the (Details) Compiler Option dialog box differ depending on the compiler option. Refer to the online help or “Compiler Options” in Appendix A for more information.
4. Click on the OK button in the (Details) Compiler Option dialog box. The (Details) Compiler Options dialog box closes and the (Add) Compiler Options dialog box is redisplayed. To delete a compiler option setup, click on the Cancel button.

5. To continue adding compiler options, repeat the procedure from Step 2.
6. When all the compiler options to be added are set, click on the Cancel button. The (Add) Compiler Options dialog box closes and the Compiler Options dialog box is displayed.
7. The compiler options which were set are listed in the Compiler Options list box in the Compiler Options dialog box.

Changing a Compiler Option

To change the details of a compiler option listed in the Compiler Options list box:

1. Select the compiler option whose setup details are to be changed from the Compiler Options list box, then click on the Change button. The (Details) Compiler Options dialog box is displayed.
2. Change the settings.
3. After all modifications have been made, click on the OK button. To cancel the change to the compiler setup, click on the Cancel button. The Compiler Options dialog closes.
4. The changed details of the compiler option are listed in the Compiler Options list box in the Compiler Options dialog box.

Deleting a Compiler Option

To delete a compiler option that appears in the Compiler Options list box:

1. Select the compiler option to be deleted from the Compiler Options list box
2. Click on the Delete button.

Setting a Library Name

When compiling a COBOL source program containing COPY statements that specify library names, use the Library Names button to associate the library names with the directories where the library files are kept.

When compiling a COBOL source program containing COPY statements that do not specify library names, specify the directory where the libraries are kept in the compiler option LIB.

To associate a library name with a directory, click on the Library Names button in the Compiler Options dialog box, then set the library details in the Library Names dialog box.

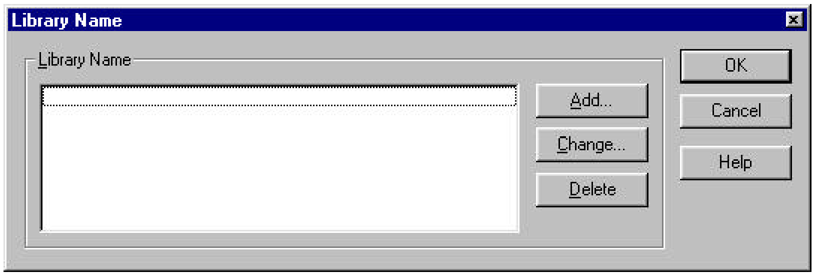


Figure 22. The Library Name dialog box

The Library Name dialog box contains the following elements:

Library Name list box

Displays the library text files and associated directories that have already been specified.

Add button

Click to enter a library name and associated directory.

Change button

Click to change the directory associated with the selected library.

Delete button

Click to remove the selected library from the list box.

OK button

Click to confirm the library settings or changes.

Cancel button

Click to cancel any library settings or changes and return to the state before the dialog box was opened.

Help button

Click to access the online help.

Adding a Library Name

To specify a library name that does not appear in the Library Name list box:

1. Click on the Add button. The Add Library Name dialog box is displayed.
2. Specify the library name entered in the COPY statement of the COBOL source program and the directory name of the library storage file.

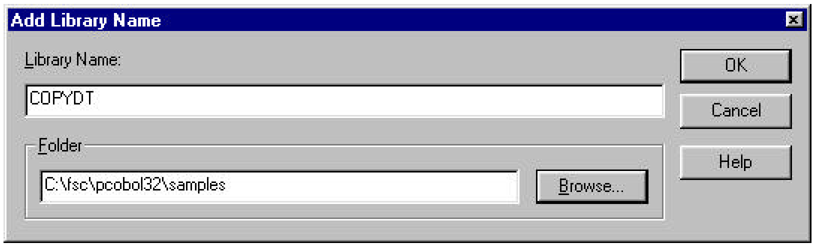


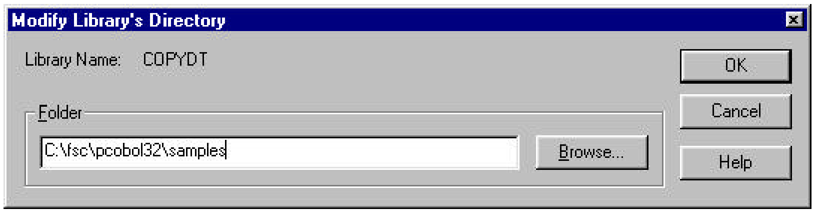
Figure 23. The Add Library Name dialog box

3. Click on the OK button. To cancel the specified details, click on the Cancel button. The Add Library Name dialog closes. Closing the dialog box without clicking on the OK button looses the specified details.
4. The specified library name, and associated directory, appears in the Library Names list box in the Library Name dialog box.

Changing a Library Name

To change the directory associated with a library displayed in the Library Name list box:

1. Select the library, for which the associated directory is to be changed, from the Library Name list box, then click on the Change button. The Modify Library's Directory dialog box is displayed.



Library Name: COPYDT

Folder: C:\fsc\pcobol32\samples

Figure 24. The Modify Library's Directory dialog box

2. Change the directory name.
3. After changing the directory name, click on the OK button. The Modify Library's Directory dialog box closes.
4. The new directory name is displayed in the Library Name list box in the Library Name dialog box.

Deleting a Library Name

To delete a library name displayed in the Library Name list box:

1. Select the library name to be deleted from the Library Name list box.
2. Click on the Delete button.

Quitting Library Name Setup

When setup of a library name is completed, click on the OK button in the Library Name dialog box. The Compiler Options dialog box is redisplayed.

Storing Setup Options

When setup of compiler options and library names are completed, click on the OK button in the Compiler Options dialog box to save the details in the options file.

The options file creates the file name "DEFAULT.CBI" in the same directory as the source file specified in the WINCOB window. If the file "DEFAULT.CBI" already exists, the file is updated with the new details.

To cancel setup before new options have been stored, click on the Cancel button, or close the dialog without clicking on the OK button.

When running continuous compilation, the same compiler options must be specified for source files being stored in the same directory. To specify different compiler options, enter a compiling directive statement within the COBOL source program. Refer to “Compiler Directing Statement” in Chapter 2.

Correcting a Compile Error

If a compile error is detected after compilation is activated from the WINCOB window, a compile message (diagnostic message) is displayed in the relevant window of the P-STAFF editor when compilation terminates.

The error search function causes the display to tag-jump from the diagnostic message displayed in the message window to the line where the compile error was detected in the edit window of the COBOL source program.

The edit window of the COBOL source program automatically opens at the first tag-jump, so you do not have to activate the editor beforehand.

Using the error search function automates the search for faulty statements, making program correction highly efficient.

Compile Messages

When compilation terminates, a compile message appears in the window of the P-STAFF editor (32) or the WINMSG window (16). To save the compile message as a file, save the file from the window.

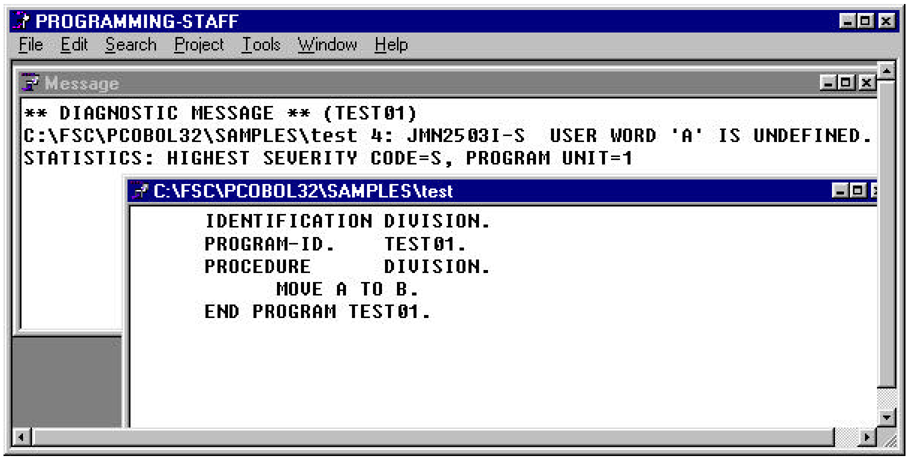
When compiler option MESSAGE has been specified, the compile message consists of the option information listing, the diagnostic message listing, and the compile unit statistical listing.

Using Error Search

This section explains how to use error search and the P-STAFF editor.

Error Search Procedures

1. Compile from the WINCOB or P-STAFF editor window.
2. When compilation has ended, a message appears on the P-STAFF editor. If the message already exists, open the P-STAFF editor to display the message.
3. Move the cursor to the line of the diagnostic message and press the F11 key or select Tag Jump from the Search menu. The COBOL source program is displayed, and the cursor moves to the line causing the message.


PROGRAMMING-STAFF
File Edit Search Project Tools Window Help
Message
C:\FSC\PCOBOL32\SAMPLES\test

```
** DIAGNOSTIC MESSAGE ** (TEST01)
C:\FSC\PCOBOL32\SAMPLES\test 4: JMN2503I-S  USER WORD 'A' IS UNDEFINED.
STATISTICS: HIGHEST SEVERITY CODE=S, PROGRAM UNIT=1
```

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.    TEST01.
       PROCEDURE      DIVISION.
              MOVE A TO B.
       END PROGRAM TEST01.
```

Figure 25. The P-STAFF editor

4. Correct the program in the edit window. To switch from the edit window to the message window, press the Shift and F11 keys or select Back Tag Jump from the Search menu.

The error search function is effective only for compile messages. This function cannot be activated from messages written to the compiler listings (when PRINT has been specified as the compiler option).

Error search cannot be used when the compiler option NUMBER has been specified.

If the total length (underlined portion) of the file name, spaces and line numbers in the diagnostic message exceeds 63 bytes, the first part of the file name is truncated and the message is written as 63 bytes. This means error search cannot be used.

For example:

```
C:\TEST.COB 4:  JMN25031-S USER WORD "A" IS
UNDEFINED.
```

Using the P-STAFF Editor

The error search function displays the COBOL85 P-STAFF edit window of the COBOL source program.

This section describes the functions of each item in the P-STAFF editor menu bar. The menu bar used in the P-STAFF editor has three items: File, Edit and Find. Refer to the online help for details.

[File]

Opens, saves and closes a file, specifies the font, and sets compiler options and runs compilation (valid only for currently open files).

[Edit]

Allows you to cut and paste, copy and delete, specify a range, and control the line number.

[Find]

Finds and replaces, moves the cursor, and controls tag-jump and back tag-jump.

When the line number is renumbered, the new number overwrites any characters in the first six columns.

Alternative Editors

You can direct P-STAFF to use an editor other than the P-STAFF editor. To do this:

1. From the File menu, click Customize Editor and click Set from the cascading menu. The Customize Editor dialog box is displayed.
2. Enter the command line of the editor to be used and click the OK button. **Note**: All open P-STAFF editor windows will be closed when the customize editor function is selected. The reason for this is to provide a clean break from using the P-STAFF editor to using the newly configured editor.
3. To return to using the P-STAFF editor, click on Customize Editor in the File menu and click on Reset from the cascading menu.

Compiler messages created by the Project Make function or by compiling using WINCOB are displayed using the editor specified with the Customize Editor function.

Note: When other editors are selected, the error message search function of P-STAFF cannot be used. The presence of the tag-jump function depends on the function of the selected editor.

Using Commands to Compile

COBOL32 Commands

With COBOL32 commands, you can compile from the command prompt.

Specification Example and Output Format

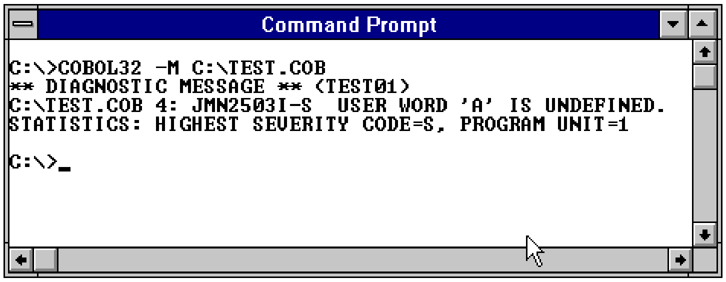
COBOL32 commands return compilation information such as the results of compilation and diagnostic messages to the command prompt screen.

For details on the format of COBOL32 commands and the available options, refer to Appendix J, “Command Formats.”

The following example shows how to compile with a COBOL32 command, and the output.

General Format

A compile end message is normally written if compile option MESSAGE has not been specified. If a compile error occurs, the corresponding diagnostic message is generated.

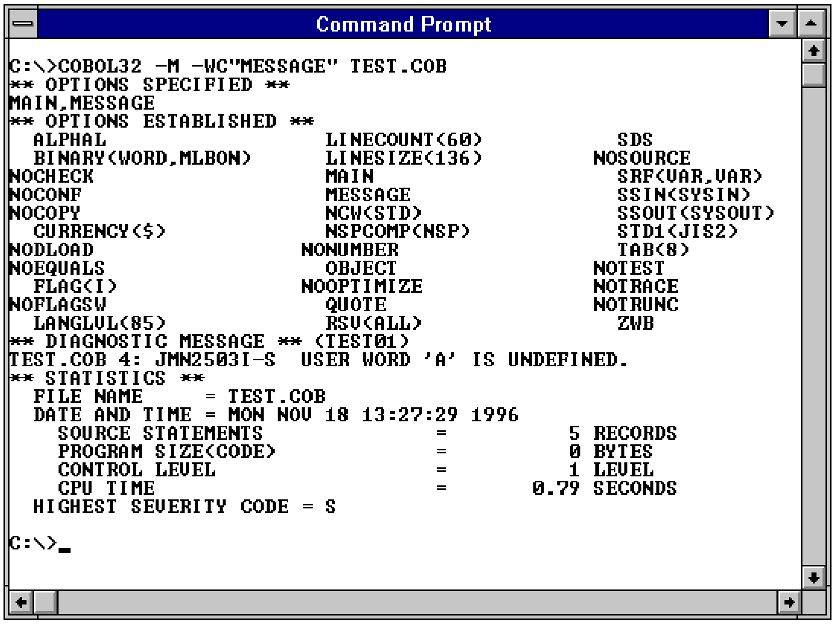


```
C:\>COBOL32 -M C:\TEST.COB
** DIAGNOSTIC MESSAGE ** (TEST01)
C:\TEST.COB 4: JMN2503I-S  USER WORD 'A' IS UNDEFINED.
STATISTICS: HIGHEST SEVERITY CODE=S, PROGRAM UNIT=1

C:\>_
```

Figure 26. The compile end message

Output Information

When the compiler option MESSAGE has been specified, the option information listing and compile unit statistical information listing are written. If a compile error occurs, the corresponding diagnostic message is written.



```
C:\>COBOL32 -M -WC"MESSAGE" TEST.COB
** OPTIONS SPECIFIED **
MAIN,MESSAGE
** OPTIONS ESTABLISHED **
  ALPHAL                 LINECOUNT(60)          SDS
  BINARY(WORD,MLBON)     LINESIZE(136)        NOSOURCE
NOCHECK                  MAIN                   SRF(VAR,VAR)
NOCONF                   MESSAGE                SSIN(SYSIN)
NOCOPY                   NCW(STD)               SSOUT(SYSOUT)
  CURRENCY($)            NSPCOMP(NSP)           STD1(JIS2)
NODLOAD                NONUMBER                 TAB(8)
NOEQUALS                 OBJECT               NOTEST
  FLAG(I)              NOOPTIMIZE             NOTRACE
NOFLAGSW                 QUOTE                NOTRUNC
  LANGLVL(85)            RSV(ALL)               ZWB
** DIAGNOSTIC MESSAGE ** (TEST01)
TEST.COB 4: JMN2503I-S  USER WORD 'A' IS UNDEFINED.
** STATISTICS **
  FILE NAME     = TEST.COB
  DATE AND TIME = MON NOV 18 13:27:29 1996
    SOURCE STATEMENTS              =         5 RECORDS
    PROGRAM SIZE(CODE)             =         0 BYTES
    CONTROL LEVEL                  =         1 LEVEL
    CPU TIME                       =      0.79 SECONDS
  HIGHEST SEVERITY CODE = S

C:\>_
```

Figure 27. Listings when the compiler option MESSAGE is specified

Differences in Compilation Run from WINCOB

Compilation operations run by COBOL32 command differ from compilation operations run from WINCOB in the following ways:

- No compile progress screen is displayed when compilation is run from COBOL32 commands.
- To interrupt compilation, follow the same method as for DOS commands.

Return Codes for COBOL32 Commands

The return codes for COBOL32 commands are set according to the highest severity code when the program is compiled. The relation between the highest severity code and the return code is shown below.

Highest Severity Code	Return Code
I	0
W	
E	1
S	2
U	3

Chapter 4. Linking Programs

This chapter describes the resources required for linking, program structure, linkage procedures, how to use windows for linking, linking with link commands, and link messages.

Linking Sample Programs

Compilation creates an object program that must be linked before it can be executed.

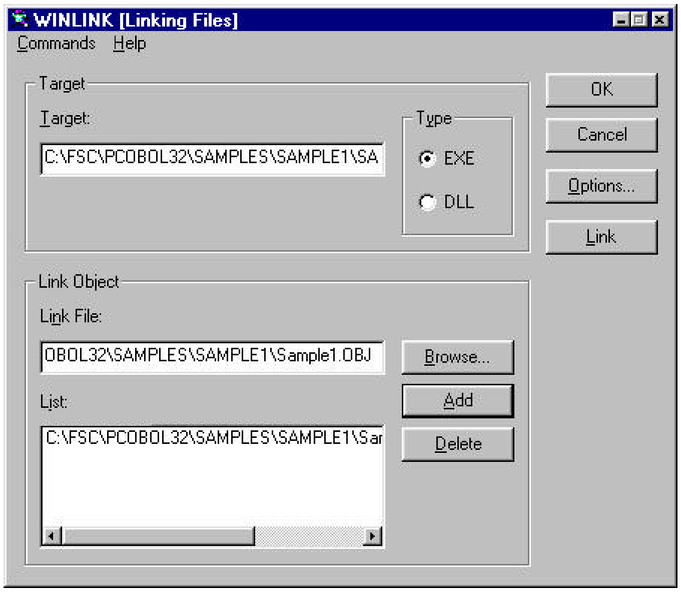
Windows 95 and Windows NT

This section shows how to use WINLINK to link the Sample 1 program. This section assumes the file containing the object program to be:

```
C:\FSC\PCOBOL32\SAMPLES\SAMPLE1\SAMPLE1.OBJ
```

To perform a link:

- Select WINLINK [Link] from the P-STAFF Tools menu.
  The WINLINK [Linking Files] window appears.



C:\FSC\PCOBOL32\SAMPLES\SAMPLE1\SA

OBOL32\SAMPLES\SAMPLE1\Sample1.OBJ

C:\FSC\PCOBOL32\SAMPLES\SAMPLE1\Sar

Figure 28. The WINLINK [Linking Files] window

- Select EXE in the Type box.
- Use the Browse button to locate and select the object file to link in the Link File edit box.
- Click on the Add button. The object name is displayed in the list box. The executable program to be created is displayed in the Target edit box.
- Click on the Link button. The system displays a command prompt screen showing the linker messages.
- The system completes linking. The command prompt window displays a message stating that the system has completed linking. Make sure that linking has completed successfully, then close the command prompt window.

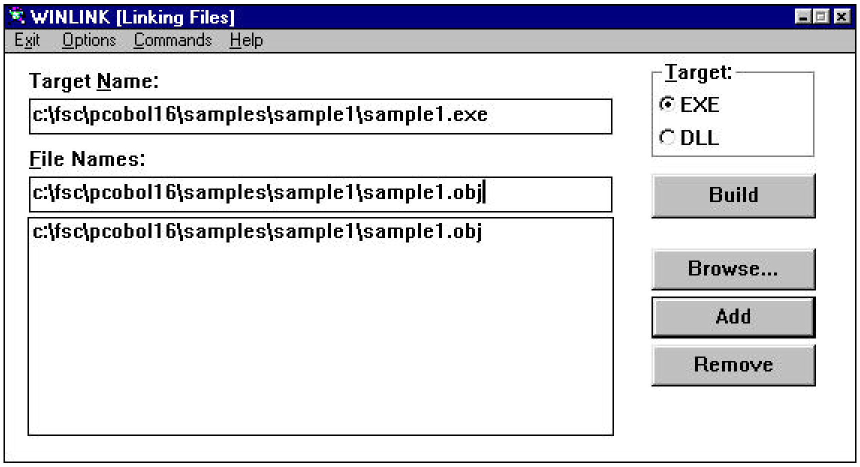
Windows 3.1

This section shows how to use WINLINK to link the Sample 1 program. This section assumes the file containing the object program to be:

```
C:\FSC\PCOBOL16\SAMPLES\SAMPLE1\SAMPLE1.OBJ
```

The section describes how to link the example dynamically.

1. Select WINLINK from the Utilities menu in the P-STAFF window. The WINLINK [Linking Files] window appears.



Target Name: c:\fsc\pcobol16\samples\sample1\sample1.exe

File Names: c:\fsc\pcobol16\samples\sample1\sample1.obj

c:\fsc\pcobol16\samples\sample1\sample1.obj

Target: [✓ EXE] DLL

Figure 29. The WINLINK [Linking Files] window

2. Set the required information.

   Select EXE in the Target box. Enter the object file.

   Click on the Add button. The object name is displayed in the list box. The executable program to be created by linking is displayed.

   Enter the module definition file. Click on the Add button.

3. Click on the Build button in the WINLINK [Linking Files] window.

   A link message is displayed in the message window.

4. The system completes linking.

   A message stating that the system completed linking is displayed in the message box. Make sure that linking has completed successfully, then close the message box.

This completes linking of the sample program. Make sure that an executable program (SAMPLE1.EXE) is created in the directory containing the sample program (in this example, C:\FSC\PCOBOL16\SAMPLES\SAMPLE1).

Resources Required for Linking

Object programs compiled from COBOL source programs are usually linked using WINLINK. WINLINK allows you to create libraries, import libraries, dynamic link libraries (DLLs), and executable files through simple screen operations. This section briefly describes the files required for linking.

You must take into account the type of link required and the program structure. See “Linkage Types and Program Structure.”

Before linking using WINLINK, the work disk drive must be set in environment variable TMP.

This following examples illustrate the files used and created by WINLINK.

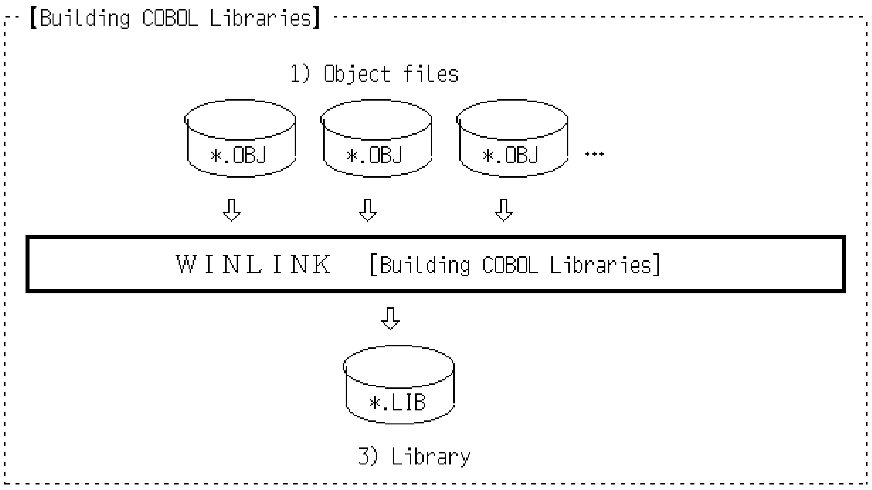

[Building COBOL Libraries]
1) Object files
*.OBJ
*.OBJ
*.OBJ
...
WINLINK [Building COBOL Libraries]
*.LIB
3) Library

Figure 30. Building COBOL libraries with WINLINK

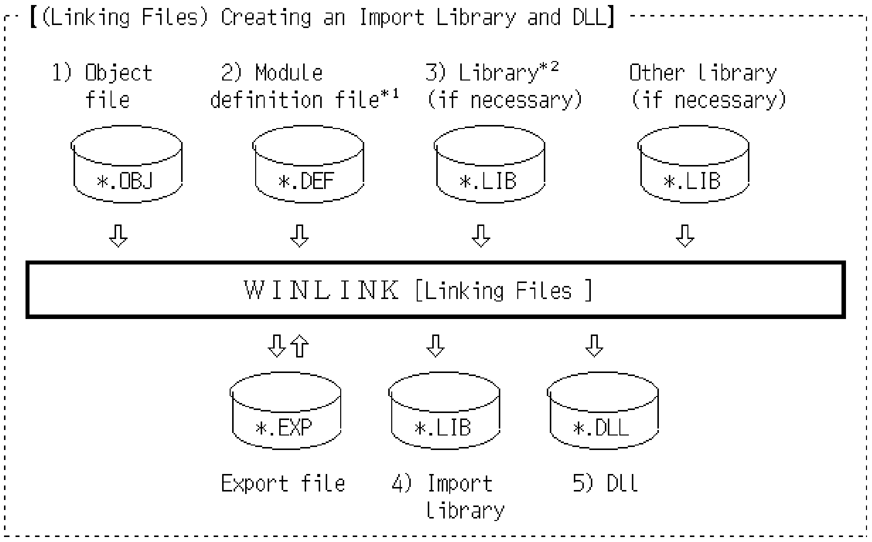

[(Linking Files) Creating an Import Library and DLL]
1) Object file
2) Module definition file*1
3) Library*2 (if necessary)
Other library (if necessary)
*.OBJ
*.DEF
*.LIB
*.LIB
WINLINK [Linking Files]
*.EXP
*.LIB
*.DLL
Export file
4) Import library
5) DLL

Figure 31. Linking files (creating an Import Library and DLL) with WINLINK

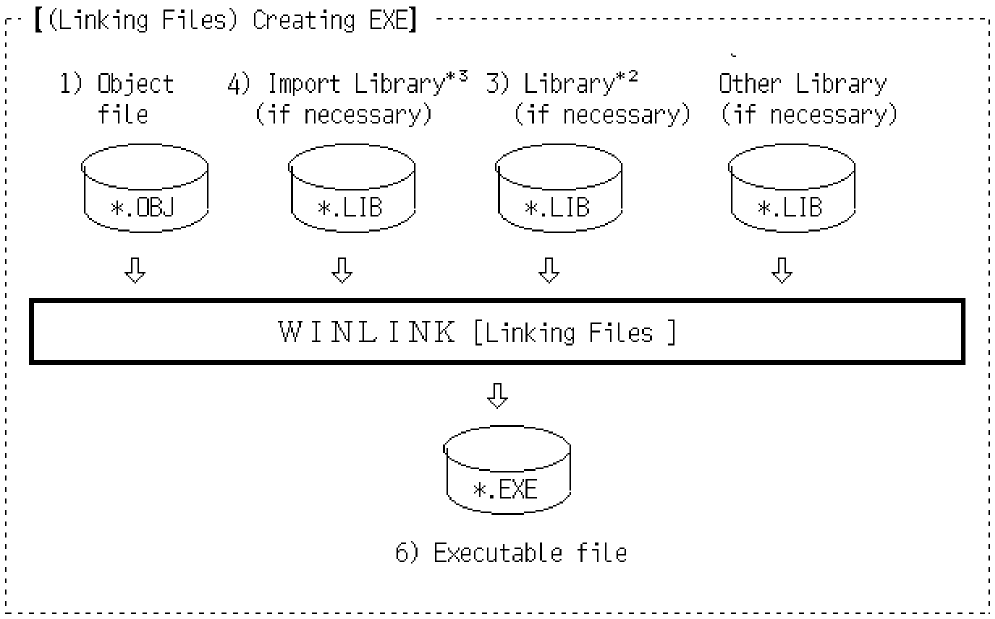

[(Linking Files) Creating EXE]
1) Object file
4) Import Library*3 (if necessary)
3) Library*2 (if necessary)
Other Library (if necessary)
*.OBJ
*.LIB
*.LIB
*.LIB
WINLINK [Linking Files]
*.EXE
6) Executable file

Figure 32. Linking files (creating an .EXE) with WINLINK

Notes on the above figures:

*1 The module definition file is generated automatically and is not specified in the WINLINK window.

*2 Standard libraries can be built using the WINLINK Building COBOL Libraries function.

*3 Import libraries are required for the dynamic link structure. The DLL/Import library should be made beforehand.

The following table describes the files used by WINLINK.

Table 5. Files used by WINLINK

	File Contents	**File Name Format**	**I/O**	**Condition to Use or Create**
1	Object file	Source-file-name. OBJ	I	Use the object program compiled from a COBOL source program
2	Module definition statement (Required for subprograms)	Optional-name. DEF	I	[16] Required to create an import library, export file, or DLL [32] Required to create an import library, DLL or EXE
			O	Created automatically if omitted in the WINLINK window
3	Standard library (Object code library)	Optional-name. LIB	I	Define when required to create a DLL or executable file
			O	Created by the WINLINK Building COBOL Libraries function.

Table 5. Files used by WINLINK (cont.)

	File Contents	File Name Format	I/O	Condition to Use or Create
4	Import library	Optional-name. LIB	I	Specify to create an executable file having the dynamic link structure.
			O	[16] Created when DLL is selected in WINLINK [Linking Files] and checked by selecting Building Import Libraries.
				[32] Created automatically when DLL is selected in WINLINK Linking Files
5	Dynamic link library (DLL)	Object-file-name.DLL or Optional-name. DLL	O	Created when linking has completed successfully.
6	Executable program	Object-file-name.EXE or Optional-name. EXE	O	Created when linking has completed successfully.

Executable Files

The executable files created are Microsoft Windows applications and will run in the following environments:

- Microsoft Windows 95 operating system
- Microsoft Windows NT Workstation operating system Version 3.51(or above) , Microsoft Windows NT Server Network operating system Version 3.51(or above).
- Microsoft Windows 3.1 operating system

DLLs

DLLs are executable modules containing functions called to perform any processing from Windows applications. A DLL is linked with an application at run time, instead of during linking.

A DLL allows multiple applications to use the same library. In a multitasking environment such as Windows, DLLs are very efficient because multiple applications can share the same DLL.

Import Library

The import library contains information used to set a dynamic link between an active application and a DLL during application execution. This library is required to create an executable file having a dynamic link structure.

The system copies information indicating where a required library is stored from the import library to the executable program. In other words, the import library provides an interface between the application and the DLL.

Module Definition File

The contents of the module definition file depend on whether you are using Windows 95 and Windows NT or Windows 3.1.

Module Definition File Contents (32)

The module definition file is required to create the import library. Create the file with a text editor.

Even if a module definition file is not specified in the WINLINK [Linking Files] window, it is created automatically during linking.

The module definition file is used indirectly to create an executable file having a dynamic link structure, because the import library created from information in the file must be specified.

The module definition statements in the module definition file define the contents of the DLL and requirements to the system.

The following table lists the module definition statements that can be used.

Table 6. Module definition statements (32)

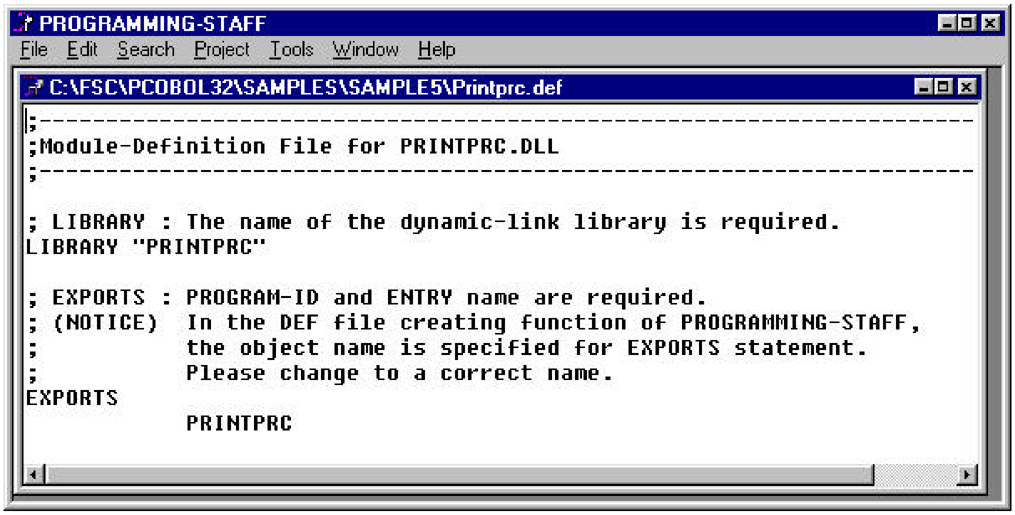
Module Statement	**Description**
NAME	Defines the module name of an application.
LIBRARY	Specifies the module name of a dynamic link library.
DESCRIPTION	Describes the modules briefly.
STACKSIZE	Specifies the stack size in bytes.
SECTIONS	Sets the attribute of a specific section.
EXPORTS	Specifies the PROGRAM-ID and ENTRY names.
VERSION	Embeds the version number.

The module definition file must contain the following statements:

- LIBRARY
- EXPORTS

When the file is created automatically, the specified object name is set as EXPORTS.

The following figure shows the contents of the module definition file created from the WINLINK [Linking Files] window.


PROGRAMMING-STAFF
File Edit Search Project Tools Window Help
C:\FSC\PCOBOL32\SAMPLES\SAMPLE5\Printprc.def

```
;----------------------------------------------------------------------
;Module-Definition File for PRINTPRC.DLL
;----------------------------------------------------------------------

; LIBRARY : The name of the dynamic-link library is required.
LIBRARY "PRINTPRC"

; EXPORTS : PROGRAM-ID and ENTRY name are required.
; (NOTICE)  In the DEF file creating function of PROGRAMMING-STAFF,
;           the object name is specified for EXPORTS statement.
;           Please change to a correct name.
EXPORTS
            PRINTPRC
```

Figure 33. A module definition file created by WINLINK

Module Definition File Contents (16)

The module definition file used under Windows 3.1 is required for linking. Create the file with a text editor.

Even if no module definition file is specified in the WINLINK [Linking Files] window, it is created automatically during linking.

The module definition statements in the module definition file define the contents of the application or DLL and requirements to the system.

The following table lists the module definition statements that can be used.

Table 7. Module definition statements (16)

Module Statement	**Description**
NAME	Defines the module name of an application.
LIBRARY	Specifies the module name of a dynamic link library.
DESCRIPTION	Describes the modules briefly.
CODE	Defines the attribute of the code segment.
DATA	Defines the attribute of the data segment.
STACKSIZE	Specifies the local stack size in bytes.
STUB	Specifies the MS-DOS executable file used to display an alarm message when any application is ran outside the Windows environment.
EXPORTS	Specifies the module functions that will be called by Windows or other applications.
IMPORTS	Specifies functions of other applications that will be called from this module.
HEAPSIZE	Specifies the local heap size in bytes.
PROTMODE	Sets the executable program to run in Windows protected mode only (standard or 386 extended).
EXETYPE WINDOWS 3.0	Sets applications to run under the Windows environment. This statement must not be omitted

Different formats are used for the module definition file depending on whether you create an application or DLL.

The following examples show the module definition files created in the WINLINK [Linking Files] window for applications and DLL. Information in the module definition file was set with the module definition file templates.

Module Definition File for Applications

```
   NAME  'SAMPLE1'                                        (1)
   EXETYPE   WINDOWS 3.1                                  (2)
   STUB  'COBSTUB'
   PROTMODE                                               (3)
   DATA MULTIPLE MOVEABLE
   PRELOAD
   CODE MOVEABLE
   PRELOAD DISCARDABLE
   HEAPSIZE     1024
   STACKSIZE    8192                                      (4)
```

(1) Set the program name.

(2) Set the application to run under Windows.

(3) Set protected mode.

(4) Set required sizes.

Module Definition File for DLLs

```
LIBRARY  'INSATSU'                                        (1)
EXETYPE  WINDOWS 3.1
STUB 'COBSTUB'                                            (2)
PROTMODE
DATA SINGLE MOVEABLE PRELOAD
CODE MOVEABLE PRELOAD
DISCARDABLE
EXPORTS                                                   (3)
 WEP                                                      (4)
 INSATSU
```

(1) Define the module name of the DLL.

(2) Display an alarm message if you run a Windows application under DOS.

(3) Set the Windows End Procedure (WEP) callback function which processes the DLL before unloading the library. The WEP is required and must not be omitted.

(4) Set the specified object name.

Linkage Types and Program Structure

This section describes the structure of an executable program created by linking.

There are two types of linkage, static and dynamic.

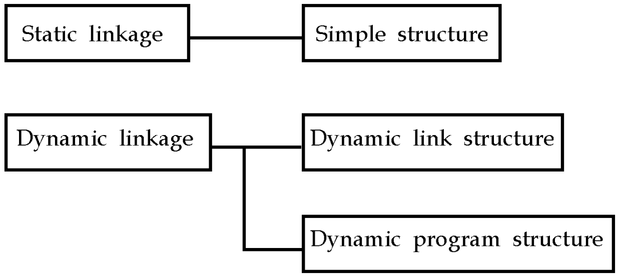
Static Linkage

The calling program and called program are all linked during linking.

Dynamic Linkage

The called program is linked to the calling program at run time.

The following shows the relationship of the program structures created by static and dynamic linkage. Throughout this section, the main program is the program initially run and the subprogram is the program called from the main program. A program called from a subprogram is also called a subprogram.



Simple Structure

Simple structure means that multiple object programs are linked to a single executable program by a static linkage. Therefore, the main program and subprograms are all loaded to virtual storage at the beginning of execution, thus allowing programs to call subprograms efficiently.

You need all subprograms during linking when you create an executable file having a simple structure.

Dynamic Link Structure

Dynamic link structure means that the main program object program and the import library containing subprogram information are dynamically linked to a single executable program.

Unlike the simple structure, no subprograms are linked to the executable file for the dynamic link structure. Subprograms are loaded to virtual storage when the main program is loaded.

Programs are loaded by the dynamic linker of the system using subprogram information created in the executable file during dynamic linking. To create an executable file having the dynamic link structure, the import library containing all subprograms called by the program is required during linking.

Dynamic Program Structure

Dynamic program structure uses only the main program object program as an executable program in a dynamic link. Therefore, unlike the dynamic link structure, no subprogram information is included in the executable program.

With dynamic program structure, subprograms are loaded by the calling program from the COBOL85 run-time system. For this reason, the loading function of the system is used.

The program structure is determined by the format of the CALL statement, options specified at compilation, and type of link. The following table lists the relationship between the program structure, CALL statement, compiler options, and link types. For compiler option DLOAD, refer to Appendix A, “Compiler Options.”

Table 8. Relationship between program structure, CALL statement, compiler options, and linkage types

Program Structure	**Call Statement**	**Compiler Option**	**Linkage Type**
Simple structure	CALL "program-name"	NODLOAD	Static linkage
Dynamic link structure	CALL "program-name"	NODLOAD	Dynamic linkage
Dynamic program structure	CALL data-name	____________	Dynamic linkage
	CALL "program-name" CALL data-name (coexisting)	DLOAD	
	CALL "program-name"	DLOAD	

Link Procedures

Object programs compiled from COBOL source programs (COBOL object programs) can be linked through window operation with the WINLINK command, or from the command prompt with link commands.

This section describes the procedures for linking with the WINLINK command. Linking with commands is explained in “Using Commands to Link.”

Linking a Single Object Program

To create an executable program that does not call any subprograms:

- Activate the WINLINK [Linking Files] window. See “Activating the WINLINK Window.”
- Select EXE in the Type box.
- Enter the target name. See “Entering the Target Name.”
- Enter the other files (object file, module definition file) required for linking in the File Names list box. See “Entering Files.”
- Set linker options. See “Setting Linker Options.”
- Start linking. See “Starting and Quitting Linking.”
- Quit linking. See “Starting and Quitting Linking.”

Creating a DLL

To create a DLL by linking a subprogram object program:

- Activate the WINLINK [Linking Files] window. See "Activating the WINLINK Window."
- Select DLL in the Type box.
- Enter the target name. See "Entering the Target Name." Do not enter any object files created by specifying compiler option MAIN during compilation.
- Enter the other files (object file, module definition file) required for linking in the Link File list box. See "Entering Files."
- Set linker options. See "Setting Linker Options."
- Start linking. See "Starting and Quitting Linking."
- Quit linking. See "Starting and Quitting Linking."

Creating an Executable Program with a Simple Structure

The COBOL object program must be compiled from a COBOL source program containing no dynamic CALL statements (those using data names to specify the called program name), or be compiled after specifying the compiler option NODLOAD.

To create an executable program having a simple structure by linking a COBOL object program that calls a subprogram:

- Create the subprogram as an object program or library. See "Procedures for Creating a Library."
- Activate the WINLINK [Linking Files] window. See "Activating the WINLINK Window."

- Select EXE in the Type box.
- Enter the target name. See “Entering the Target Name.”
- Enter the other files required for linking in the File Names list box. See “Entering Files.”
  - Object file
  - Module definition file
  - Subprogram created in Step 1
  - Object file (when the subprogram is an object program)
  - Library (when the subprogram is a library)
- Set linker options. See “Setting Linker Options.”
- Start linking. See “Starting and Quitting Linking.”
- Quit linking. See “Starting and Quitting Linking.”

Creating an Executable Program with a Dynamic Link Structure

The COBOL object program must be compiled from a COBOL source program containing no dynamic CALL statements (those using data names to specify the called program name), or be compiled after specifying the compiler option NODLOAD.

To create an executable program having a dynamic link structure by linking a COBOL object program that calls a subprogram:

- Activate the WINLINK [Linking Files] window. See “Activating the WINLINK Window.”
- First create a DLL from the subprogram:
  - Select DLL in the Type box.
  - Enter the target name. See “Entering the Target Name.”

- Enter the other files required to link the subprogram in the File Names list box. See “Entering Files.”
  - Object file (Subprogram)
  - Module definition file (Subprogram)
- Set linker options. See “Setting Linker Options.”
- Create an import library (16).
  For Windows 3.1, check "Building Import Libraries" in the Command menu.
- Start linking the subprogram. See “Starting and Quitting Linking.” The DLL, the import library (DLL name.LIB) and export file of the subprogram are created.

  If the import library already exists a message will be displayed.

Then create the executable for the main program:

- Select EXE in the Type box.
- Enter the target name. See “Entering the Target Name.”
- Enter the other files required to link the main program in the File Names list box. See “Entering Files.”
  - Object file (Main program)
  - Module definition file
  - Import library (created when you started linking the program)
- Set linker options. See “Setting Linker Options.”
- Start linking. See “Starting and Quitting Linking.”
- Quit linking. See “Starting and Quitting Linking.”

Creating an Executable Program with a Dynamic Program Structure

The COBOL object program used here must be compiled from a COBOL source program containing CALL statements in data name specification, or be compiled after specifying the compiler option DLOAD.

To create an executable program having a dynamic program structure by linking a COBOL object program that calls a subprogram:

- Create the subprogram as a DLL.
- Activate the WINLINK [Linking Files] window. See “Activating the WINLINK Window.”
- Select EXE in the Type box.
- Enter the target name. See “Entering the Target Name.”
- Enter the other files (object file(main program)) required for linking in the Link File list box. See “Entering Files.”
- Set linker options. See “Setting Linker Options.”
- Start linking. See “Starting and Quitting Linking.”
- Quit linking. See “Starting and Quitting Linking.”

Creating a Library

To create a library by linking object programs of multiple subprograms:

- Activate the WINLINK [Linking Files] window. See “Activating the WINLINK Window.”
- Switch to the WINLINK [Building COBOL Libraries] window. Select Building COBOL Libraries from the Commands menu.
- Enter the COBOL library name. See “Entering a Target Library Name.” If an existing file name is entered, the existing file is deleted and a new file is created.
- Enter the other files to be included in the library in the Object File list box. See “Entering Files.”
- Start library building. See “Starting and Quitting Library Building.”
- Quit library building. See “Starting and Quitting Library Building.”

Creating an Import Library

WINLINK also creates an import library when it creates a DLL. To create an import library:

- Create the subprogram as a DLL.
- Specify creation of an import library. (16)
  For Windows 3.1, check "Building Import Libraries" from the command menu. For Windows 95 and Windows NT, the import library is created automatically.
- Start linking. After linking, the following files are created:

- DLL
- Import library of subprogram (DLL-name.LIB)
- Export file (32)

If an import library already exists, an alarm message is displayed.

WINLINK

WINLINK is a utility that links programs compiled by COBOL85 to create executable programs and DLLs, libraries, and import libraries. WINLINK provides two windows.

- WINLINK [Linking Files] window:

  Creates import libraries, DLLs, and executable programs.

- WINLINK [Building COBOL Libraries] window:

  Creates libraries.

Switch these windows by selecting the Commands menu on the menu bar in each window. For details about how to use the windows, refer to the online help.

Activating the WINLINK Window

To activate the WINLINK window, do one of the following:

- Select WINLINK from the Tools menu in P-STAFF (32).
- Select WINLINK.EXE in the File Manager directory window (16).

WINLINK [Linking Files] Window

The WINLINK [Linking Files] window is used to create import libraries, DLLs, and executable programs by linking.

Use the menu bar to exit the window, set linker options, or switch between windows.

Select the target type as either EXE or DLL.

The following sections describe how to use the window.

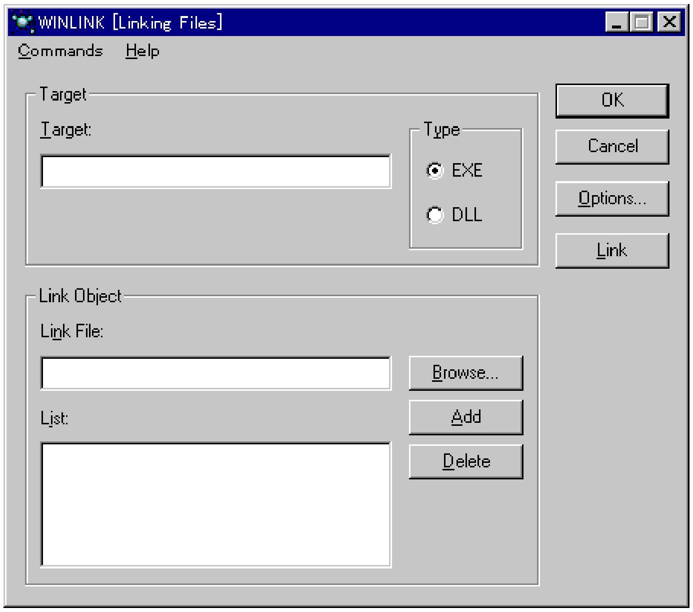


Figure 34. The WINLINK window

The WINLINK window contains the following elements:

Commands

Switches the window between linking files mode and building COBOL libraries mode.

Help

Access the online help.

Target edit box

Specify the name of the file to be created.

Type radio buttons

Specify whether an EXE or DLL file should be created.

Link File edit box

Specify the name of a file to be added to the List.

Link Object List

Displays a list of the input files to the link.

Browse button

Click to select a file to be included in the link.

Add button

Click to add the file in the Link File edit box to the List.

Delete button

Click to remove the selected file(s) from the List.

OK button

Click to start the link and close the WINLINK window.

Cancel button

Click to close the WINLINK window without performing a link.

Options button

Click to specify the link options.

Link button

Click to start the link and keep the WINLINK window open for further links.

Entering the Target Name

The name of the file to be created by linking can be entered in the Target edit box using one of two methods.

Method 1

1. Select whether the target should be EXE or DLL in the Type box.
2. Enter, or browse and select, an object file in the Link File edit box.
3. Click the Add button. The object file name is added to the Link Object List. A target name is automatically created by combining the extension specified in the Type box with the base of the object file name.

Method 2

Enter the name of the file to be created directly in the Target edit box.

If the Type is selected after a file name is entered in the Target edit box, the extension of the entered file name will change to the extension of the file selected by the option button.

Entering Files

Enter the names of files required to link in the Link Object List box by following the procedures below.

1. Click on the Browse button, then select the files required to link from the Browse Files dialog box. Files required to link include:
   - Object file
   - Module definition file [(16): required] [(32): when creating a DLL]
   - Library
   - Import library (when creating an executable program having a dynamic link structure)
2. Click on the OK button. The selected file names are displayed in the Link File edit box. Alternatively the file name(s) can be keyed directly into the Link File edit box.
3. Click on the Add button. The file names are added to the File Names list box.
4. Repeat Steps 1 to 3 until all files to link are entered.

Building an Import Library

In Windows 95 and Windows NT, an import library and an export file are automatically created when a DLL is created.

To create an import library under Windows 3.1, check "Building Import Library" in the WINLINK [Linking Files] Command menu.

Setting Linker Options

Different Linker Options screens are used for (32) Windows 95 and Windows NT and (16) Windows 3.1.

Windows 95 and Windows NT

Linker options are set using the Linker Options dialog box displayed by clicking the Options button in the WINLINK [Linking Files] window.

A main link option (Debugging is excluded) is automatically set.

See the LINK command options table in Appendix J.

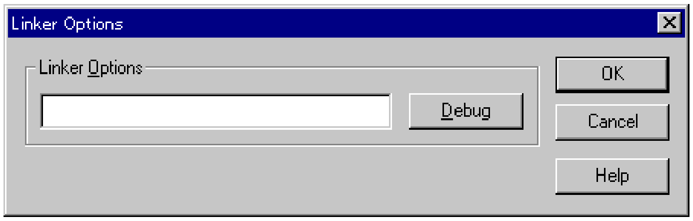


Figure 35. The Linker Options dialog box

The Linker Options dialog box contains the following elements:

Linker Options edit box

Enter link options.

Delimit multiple options with one or more spaces.

Debug button

Click to include linker debug options in the Linker Options edit box.

OK button

Click when finished specifying linker options.

Cancel button

Click to cancel any linker option settings or changes and return to the state before the dialog box was opened.

Help button

Click to access the online help.

Windows 3.1

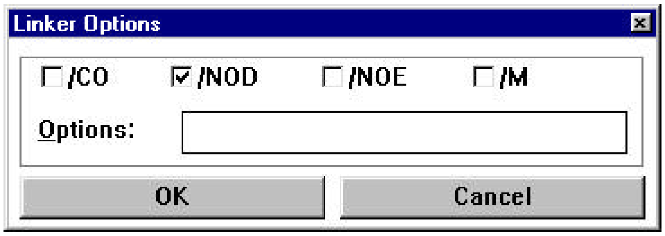


Figure 36. The Linker Options dialog box

The Linker Options dialog box contains the following elements:

CO

Prepares for debugger operation. Use this option when compiler option TEST was specified during compilation of source programs.

NOD

Restricts the linker to use only standard libraries. Always use this option when linking object programs compiled by the COBOL85 compiler.

NOE

Disables the library extended dictionary.

M

Outputs a list of public symbols to the map file.

Options

Allows direct key entry of other linker options.

OK button

Click when finished specifying linker options.

Cancel button

Click to cancel any linker option settings or changes and return to the state before the dialog box was opened.

Starting and Quitting Linking

After all information required for linking has been set, start linking. To start linking, select the Link button.

The system brings up a command prompt window that displays the linker messages. (32)

A link message is displayed in the message window. (16)

A link exit message is displayed in the command prompt window. Make sure that linking has completed successfully, then close the command prompt window. (32)

A link exit message is displayed in the message window. Make sure that linking has completed successfully, then close the message box. (16)

If a link error occurs, see “Linker Messages” and take action.

To continue linking, repeat the procedure from the entering a target name step.

To quit linking, click on the Cancel button in the WINLINK [Linking Files] window. This closes the window.

Please note the following points as they are causes of common link errors:

- Always use the linker corresponding to the object.
- Do not create an executable program only from object programs containing the compiler option NOMAIN (that is, no compiler option MAIN was used during compilation).
- Do not create an executable program from multiple programs containing the compiler option MAIN.

WINLINK [Building COBOL Libraries] Window

Use the WINLINK [Building COBOL Libraries] window to link multiple object programs to a library.

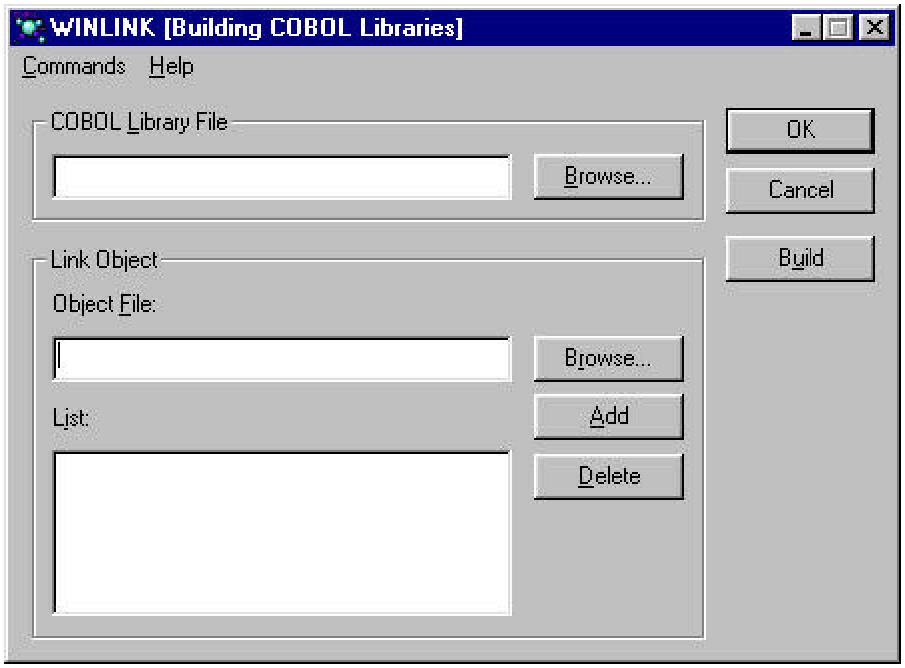


Figure 37. The WINLINK [Building COBOL Libraries] window

The WINLINK [Building COBOL Libraries] window contains the following elements:

Commands

Switches the window between linking files mode and building COBOL libraries mode.

Help

Accesses the online help.

COBOL Library File edit box

Specify the name of the library file to be created.

[COBOL Library File] Browse button

Click to select the name of a library file to be created.

Object File edit box

Specify the name of a file to be added to the List.
Do not enter programs containing compiler option MAIN.

Link Object List

Displays a list of input files to include in the link.

[Link Object] Browse button

Click to select a file to be included in the link.

Add button

Click to add the file in the Object File edit box to the List.

Delete button

Click to remove the selected file(s) from the List.

OK button

Click to start the link and close the WINLINK [Building COBOL Libraries] window.

Cancel button

Click to close the WINLINK [Building COBOL Libraries] window without performing a link.

Build button

Click to start the link and keep the WINLINK [Building COBOL Libraries] window open for further links.

The following sections describe how to use the window to build libraries.

Entering a Target Library Name

Enter the name of a target library in the COBOL Library File edit box by using either of the procedures below.

Method 1

Enter the name of an object file in Object File. The entered file name having the extension LIB is displayed in the COBOL Library File edit box.

Method 2

Enter the name of the target file in the COBOL Library File edit box.

If an existing library name is entered here, the library is deleted and a new library is created.

Entering Files

Enter the names of object files to be linked in the Link Object List with the procedures below.

1. Click on the [Link Object] Browse button, then select required files from the Browse Files dialog box.
2. Click on the Open button. The selected file names are displayed in the Object File edit box. Alternatively the file name(s) can be keyed directly into the Object File edit box.
3. Click on the Add button. The file names are added to the Link Object List.
4. Repeat Steps 1 to 3 until all object files to be linked are entered.

Starting and Quitting Library Building

After all information required for linking has been set, start building the libraries. To start building, click on the Build button.

Windows 95 and Windows NT: The system brings up a command prompt window that displays the linker messages.

Windows 3.1: A building library message is displayed in the message window.

Windows 95 and Windows NT: A building library end message is displayed in the command prompt window. Make sure that linking has completed successfully, then close the command prompt window.

Windows 3.1: A building library exit message is displayed in the message window. Make sure that linking has completed successfully, then close the message window.

To continue building, repeat the procedure from the “entering a target library name” step.

To quit building, click on the Cancel button in the WINLINK [Building COBOL Libraries] window. This closes the window.

Using Commands to Link

Object programs can also be linked with commands. This section describes link operation with commands.

There are two commands, the LINK command and the LIB command.

LINK Command (32)

In Windows 95 and Windows NT, the LINK command performs the link operation. Refer to “LINK Command” in Appendix J for details of the format of the command.

When executing the LINK command, the following libraries must be specified:

- F3BICIMP.LIB
- LIBC.LIB
- KERNEL32.LIB
- USER32.LIB

Some examples of how to use the LINK command follow.

LIB Command (32)

In Windows 95 and Windows NT, the LIB command performs the creation of standard libraries and import libraries. Refer to “LIB Command” in Appendix J for details of the format of the command.

Examples of Using the LINK and LIB commands (32)

Some examples of using the LINK command and the LIB command are shown here.

When Linking an Object Program

```
LINK A.OBJ F3BICIMP.LIB LIBC.LIB KERNEL32.LIB USER32.LIB
/OUT:A.EXE
```

When Creating a DLL

To create an export file:

```
LIB /DEF:SUB.DEF /OUT:SUB.LIB /MACHINE:IX86 SUB.OBJ
```

To create a DLL:

```
LINK SUB.OBJ SUB.EXP F3BICIMP.LIB LIBC.LIB KERNEL32.LIB
USER32.LIB /DLL /OUT:SUB.DLL
```

When Creating an Executable Program with a Dynamic Link Structure

To create an import library:

```
LIB /DEF:BBB.DEF /OUT:BBB.LIB /MACHINE:IX86 BBB.OBJ
```

To create a DLL:

```
LINK BBB.OBJ BBB.EXP F3BICIMP.LIB LIBC.LIB KERNEL32.LIB USER32.LIB /DLL
/OUT:BBB.DLL
```

To create an executable program:

```
LINK AAA.OBJ F3BICIMP.LIB LIBC.LIB KERNEL32.LIB USER32.LIB BBB.LIB
/OUT:AAA.EXE
```

LINK Command (16)

Windows 3.1 provides the LINK, LIB, and IMPLIB commands for linking.

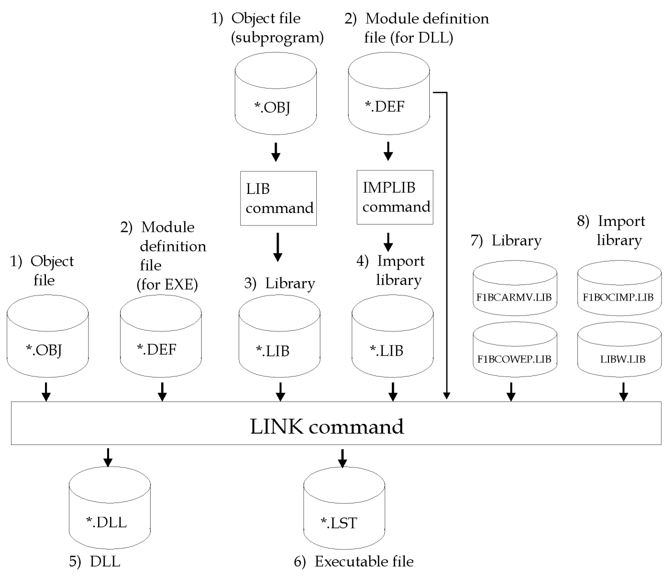


Figure 38. Files used by the LINK command

Files or libraries 1) to 6) correspond to 1) to 6) in Table 5. Files 7) and 8) are explained below.

- F1BCARMV.LIB of 7): Arithmetic library provided by COBOL85. Always specify this library when linking COBOL programs.
- F1BCOWEP.LIB of 7): Library provided by COBOL85. Always specify this library when building DLLs by linking COBOL programs.
- F1BCCIMP.LIB and LIBW.LIB of 8): Run-time system provided by COBOL85 and import library of Windows 3.1. Specify these files when linking COBOL programs.

Linker Messages

This following sections describe linker messages displayed during COBOL85 operation in Windows 95 and Windows NT (32) and Windows 3.1 (16).

Windows 95 and Windows NT

The following linker messages may be displayed in Windows 95 and Windows NT.

LNK1104

Explanation:
There is not enough space on the disk or root folder.
Operator response:
Delete files to make space.

LNK1123

Explanation:
An attempt was made to link Microsoft Windows operating system Version 3.1 objects with the 32 bit linker.
Operator response:
Use 32 bit Objects.

LNK1561

Explanation:
An attempt was made to create an executable program (EXE) by compiling the main program without the compiler option MAIN.
Operator response:
Specify the compiler option MAIN before compiling the main program.

LNK2001

Explanation:
- No internal name of the export routine is defined in the EXPORTS definitions in the module definition file.
- No internal name of the export routine is set in the library or object file.

Operator response:
Make sure that the export name in the EXPORTS statement of the module definition file is set in the library or object file.

LNK2005

Explanation:
Tried to make a single executable file from two or more programs compiled with the MAIN option.
Operator response:
Specify only the MAIN compiler option for the main program.

LNK4006

Explanation:
- EXPORTS is duplicated in the module definition file.
- The internal name of existent data was found in the library or object file.

Operator response:
Make sure that the export name in the EXPORTS statement of the module definition file is duplicated in the library or object file.

Linker Messages (16)

The following linker messages may be displayed in Windows 3.1:

L1052

Explanation:
Too many libraries were linked.
Operator response:
- Link libraries to reduce the number of libraries.
- Use a module requiring less libraries.

L1081

Explanation:
There is not enough space on the disk or root directory.
Operator response:
- Delete files to make space.
- Move files to make space.

L1082

Explanation:
The directory containing COBSTUB.EXE was not set in the environment variable PATH.
Operator response:
Set the install directory of the COBOL system in the environment variable PATH.

L1101

Explanation:
An attempt was made to link Windows NT objects with the Windows 3.1 linker.
Operator response:
Use the Windows NT linker.

L2022

Explanation:
- No internal name of the export routine is defined in the EXPORTS definitions in the module definition file.
- No internal name of the export routine is set in the library or object file.

Operator response:
Make sure that the export name in the EXPORTS statement of the module definition file is set in the library or object file.

L2023

Explanation:
- EXPORTS is duplicated in the module definition file.
- The internal name of existent data was found in the library or object file.

Operator response:
Make sure that the export name in the EXPORTS statement in the module definition file is duplicated in the library or object file.

L4038

Explanation:
An attempt was made to create an executable program (EXE) by compiling the main program without the compiler option MAIN.
Operator response:
Specify the compiler option MAIN before compiling the main program.

Chapter 5. Executing Programs

This chapter describes the procedures for executing programs, setting run-time environment information, and operating windows for execution.

Executing Sample Programs

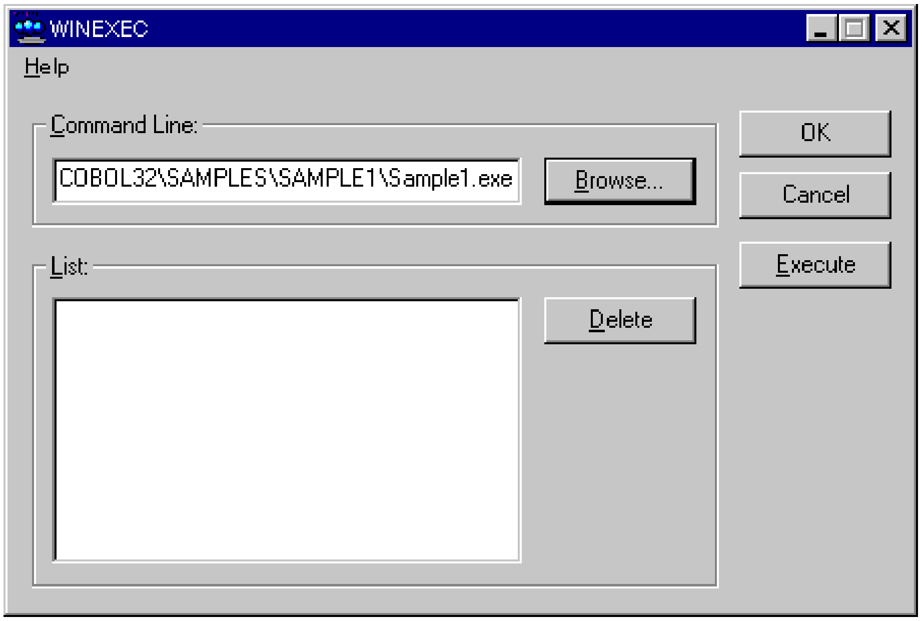
This section shows how to execute the program in Sample 1 with WINEXEC in Windows 95 / Windows NT and Windows 3.1.

Windows 95 and Windows NT

Throughout this section, assume the run-time information to be:

```
C:FSC\PCOBOL32\SAMPLES\SAMPLE1\SAMPLE1.EXE
```

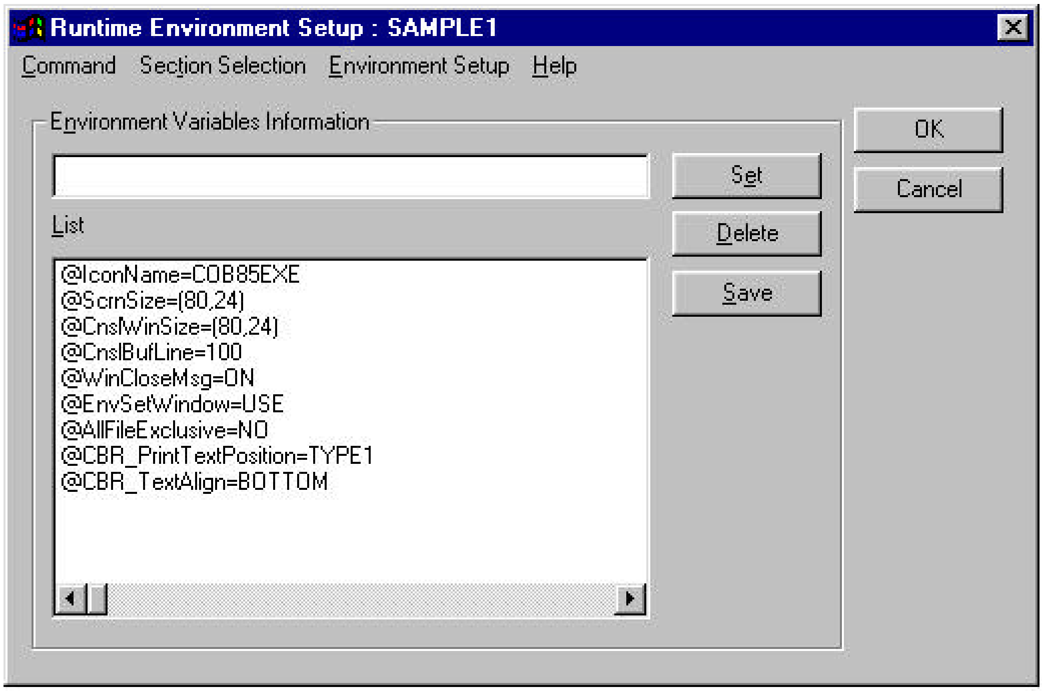
Select WINEXEC from the Tools menu of the P-STAFF window. The WINEXEC window appears.



COBOL32\SAMPLES\SAMPLE1\Sample1.exe

Figure 39. The WINEXEC window

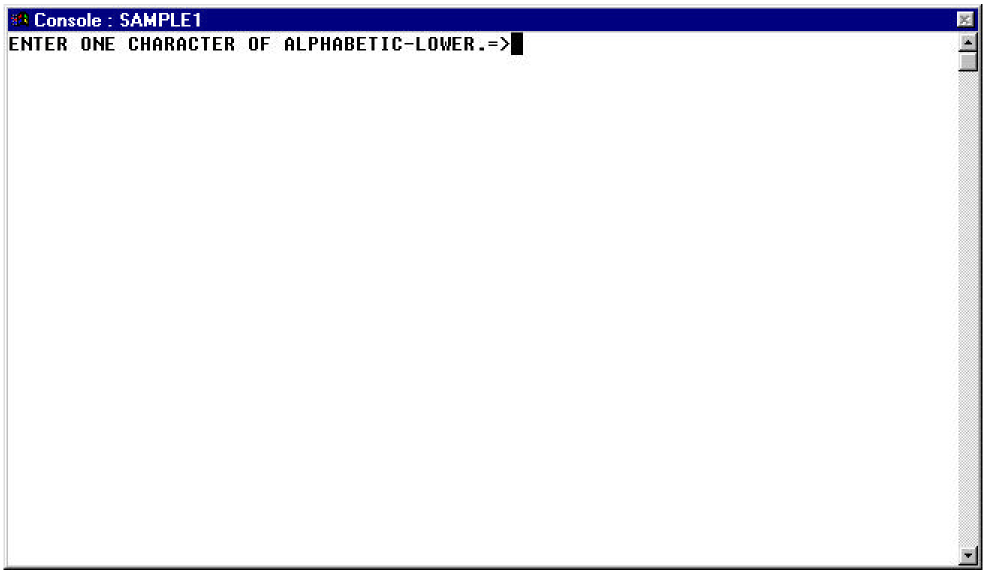
- Enter the name of the file to execute (SAMPLE1.EXE) in the Command Line edit box.
- Click on the Execute button. The Run-time Environment Setup window is displayed.



```
@IconName=COB85EXE
@ScrnSize=(80,24)
@CnslWinSize=(80,24)
@CnslBufLine=100
@WinCloseMsg=ON
@EnvSetWindow=USE
@AllFileExclusive=NO
@CBR_PrintTextPosition=TYPE1
@CBR_TextAlign=BOTTOM
```

Figure 40. The Run-time Environment Setup window

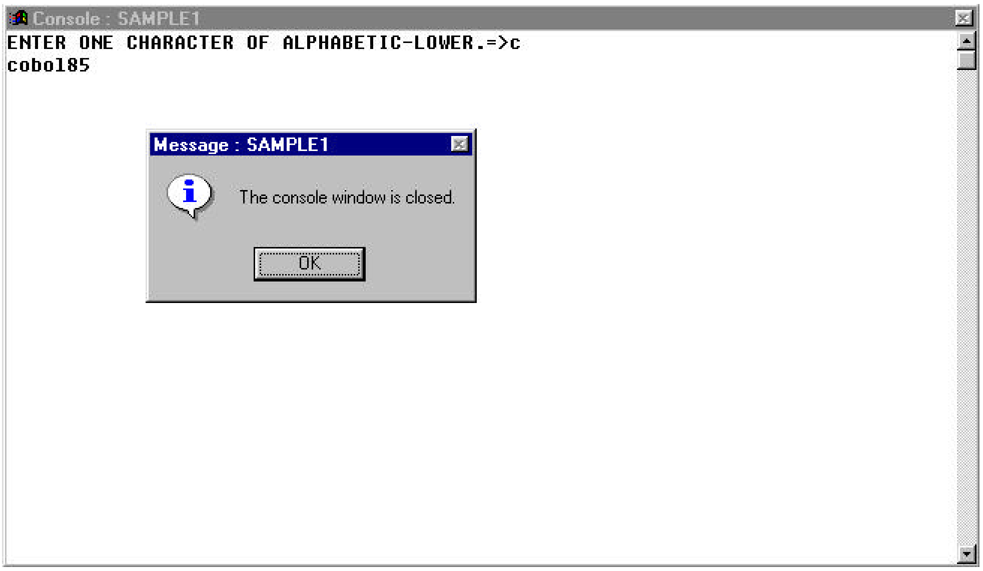
- For Sample 1, there is no information to set. Click on the OK button in the Run-time Environment Setup window. The system starts executing the program.
- For Sample 1, the system inputs or outputs data using the COBOL ACCEPT / DISPLAY function. A console window is displayed.



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

Figure 41. The COBOL85 console window

- Enter a lowercase letter, then press the ENTER key. A lowercase letter is displayed along with corresponding text. In the following example, cobol85 is displayed based on the specified letter "c".



```
ENTER ONE CHARACTER OF ALPHABETIC-LOWER.=>c
cobol85
```

The console window is closed.

Figure 42. The COBOL85 console window and message window

- Check the results, then click on the OK button in the message window. The message window and console window close.

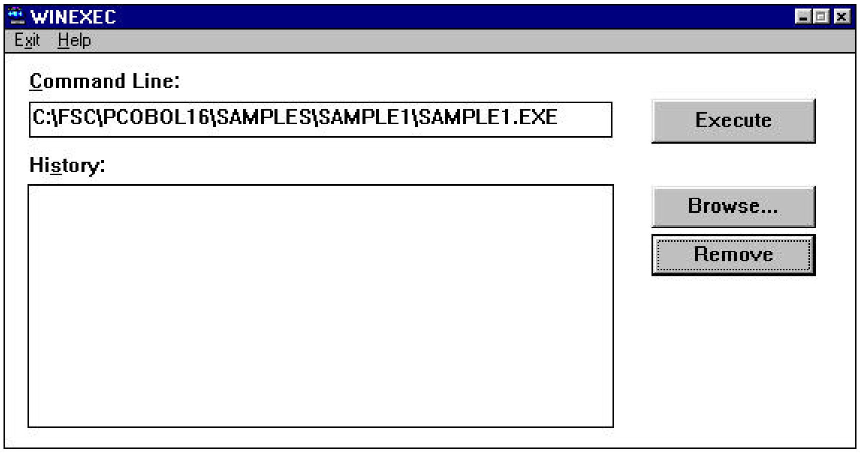
This completes execution of the sample program.

Windows 3.1

Throughout this section, assume the run-time information to be:

```
C:FSC\PCOBOL16\SAMPLES\SAMPLE1\SAMPLE1.EXE.
```

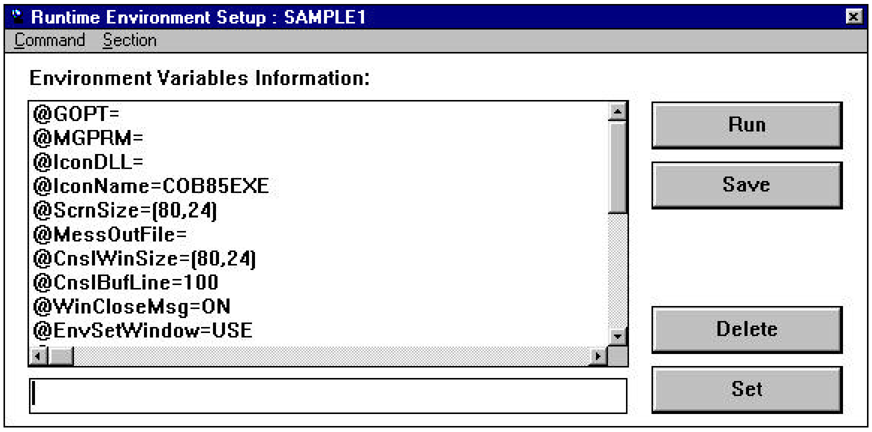
Select WINEXEC from the Utilities menu in the P-STAFF window. The WINEXEC window appears.



C:\FSC\PCOBOL16\SAMPLES\SAMPLE1\SAMPLE1.EXE

Figure 43. The WINEXEC window

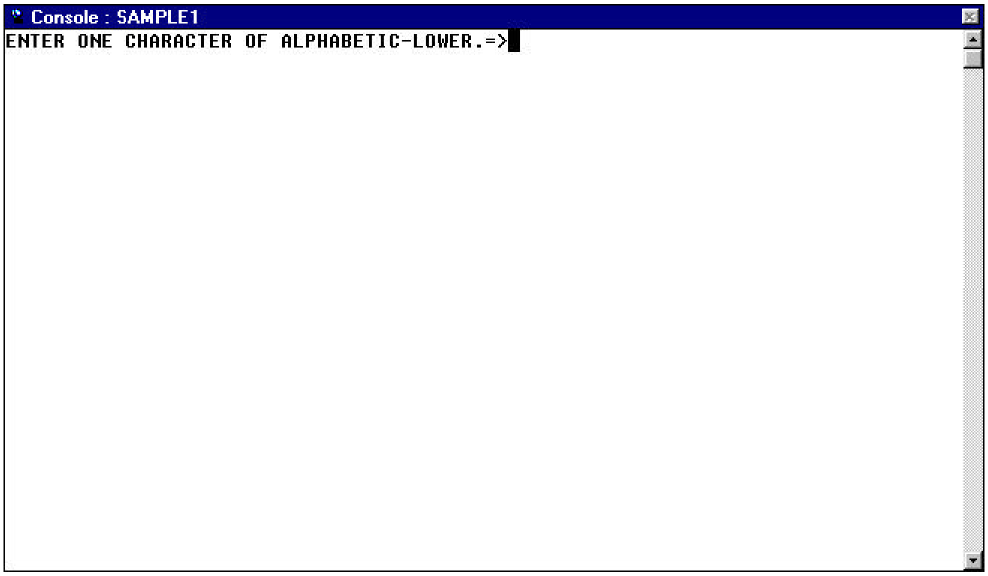
- Enter the name of the file to execute (SAMPLE1.EXE) in the Command Line edit box.
- Click on the Execute button. The Run-time Environment Setup window is displayed.


Runtime Environment Setup : SAMPLE1
Command
Section
Environment Variables Information:
Run
Save
Delete
Set

```
@GOPT=
@MGPRM=
@IconDLL=
@IconName=COB85EXE
@ScrnSize=(80,24)
@MessOutFile=
@CnslWinSize=(80,24)
@CnslBufLine=100
@WinCloseMsg=ON
@EnvSetWindow=USE
```

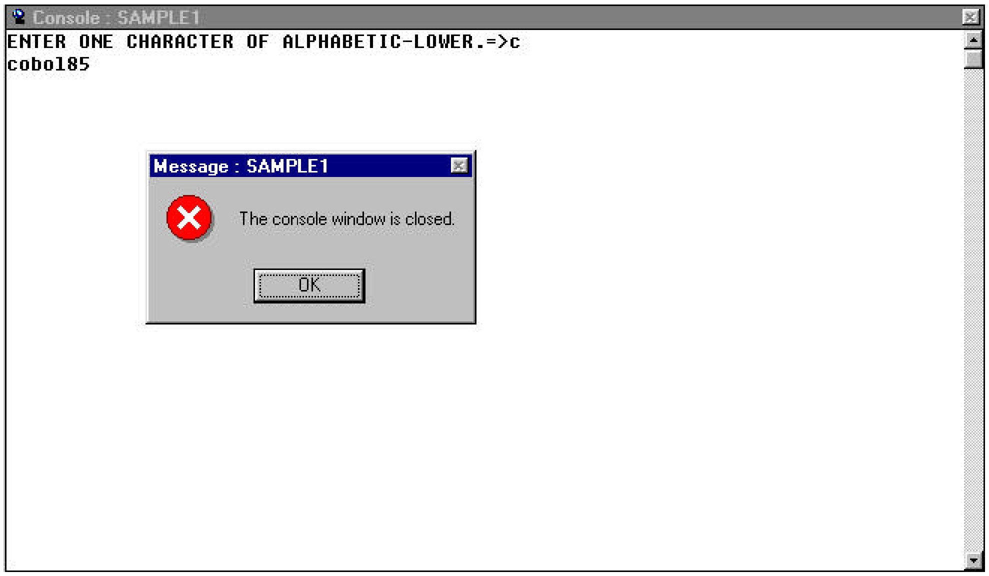
Figure 44. The Run-time Environment Setup window

- For Example 1, there is no information to set. Click on the Run button in the Run-time Environment Setup window. The system starts executing the program.
- For Example 1, the system inputs or outputs data using the COBOL ACCEPT / DISPLAY function. A console window is displayed. Enter a lowercase letter, then press the ENTER key. A lowercase letter is displayed along with corresponding text. In the following example, cobol85 is displayed based on the specified letter "c".



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

Figure 45. The COBOL85 console window



```
ENTER ONE CHARACTER OF ALPHABETIC-LOWER.=>c
cobol85
```

The console window is closed.

Figure 46. The COBOL85 console window and message window

- Check the results, then click on the OK button in the message window. The message window and console window close.

This completes execution of the sample program.

Execution Procedures

Executable programs compiled and linked from COBOL programs can be executed in the same manner as normal Windows applications(*) with the WINEXEC command. This section describes how to run COBOL programs.

(*) The applications work with:

- Windows 95
- Windows NT Version 3.51 (both server and workstation versions)
- Windows 3.1

Before Executing COBOL Programs

Before executing COBOL programs, you must set run-time environment information.

COBOL85 calls resources and data to be assigned for executing COBOL program run-time environment information. For information about setting the run-time environment, see “Setting Run-time Environment Information.”

Before Executing COBOL Programs (16)

Before executing a COBOL program under Windows 3.1, ensure that SHARE was loaded in MS-DOS command mode to enabled. This allows file sharing and exclusive systems to be included before running Windows. The format of the SHARE command is shown below.

```
SHARE /L:50 /F:2048
```

The L option: sets exclusive control option for files and records. Use the value determined from the following expression as standard when executing a COBOL program.

When executing multiple COBOL programs at the same time (also for different execution units), add the value determined for each program:

> Value to be entered = 20 + number of files opened concurrently. (Number of files * 2 for index files only) + number of records exclusively controlled (or locked)

The F option: sets the number of files to be opened. To open many files concurrently, enter a large value. To execute COBOL programs, specify a value of 2048 or more, depending on the length of the path name of files to be opened.

Once loaded, the SHARE command cannot be selected again. If any SHARE option must be changed, restart the system, then execute the SHARE command with the revised command line.

Executing COBOL Programs

COBOL programs can be executed from the WINEXEC window or with a batch file. This section describes how to use the WINEXEC window to execute COBOL programs.

1. Activate the WINEXEC window. See “Activating the WINEXEC Window.”
2. Enter the name of the file to be executed in the Command Line edit box. See “Entering a File Name.”
3. When any argument is specified, directly type it in the Command Line edit box in the same command line form.
4. Start execution by selecting the Execute button in the WINEXEC window.

5. To re-execute a program already executed, double-click on the file name displayed in the History list box.
6. Set the run-time environment information. When a COBOL program is executed, the Run-time Environment Setup window appears. Set run-time environment information if necessary, then close the Run-time Environment Setup window. See “Run-time Environment Setup Window”.
7. Quit execution. To quit execution and close the WINEXEC window, select Exit in the WINEXEC window.

Setting Run-time Environment Information

This section explains the relationship between the types of run-time environment information and setup procedures, and how to set each item in the run-time environment.

Types of Run-time Environment Information

Information required to execute COBOL applications is called run-time environment information.

There are two types of run-time environment information, environment variable information and entry information. Environment variable information includes items such as the console window size, console font, and file identifier.

Entry information includes the dynamic program structure.

The following table lists the types of run-time environment information. For details on run-time environment information, see “Format of Run-time Environment Information.”

Table 9. Types of run-time environment information

Run-time Environment Information	**Description**	**Run-time Environment Information**	**Description**
Environment variable information		@AllFileExclusive	Set exclusive control of files
@GOPT	Set run-time options	@CBR_CIINF	Set the logical destination definition file
@MGPRM	Set the GS-series format run-time parameter	@CBR_ENTRYFILE(32)	Set the entry information file
@IconDLL	Set the DLL-name of an icon resource	@CBR_PSFILE_xxx	Set the connected-product-name used from the presentation file
@IconName	Set the identifier of an icon resource	@NoMessage	Set to suppress run-time message
@ScrnSize	Set the size of the logical screen for screen handling	File-identifier	Set the name of the file used by the program
@MessOutFile	Set the message output file	File-identifier	Set the information file used by the program
@CnslWinSize	Set the size of console window	SYSIN-access-name	Set the input file for the ACCEPT/ DISPLAY function
@CnslBufLine	Set the buffers count for the console window	SYSOUT-access-name	Set the output file for the ACCEPT/ DISPLAY function
@WinCloseMsg	Set a display message when the window closes	TERMINATOR	Set the function keys for screen handling

Table 9. Types of run-time environment information (cont.)

Run-time Environment Information	**Description**	**Run-time Environment Information**	**Description**
@EnvSetWindow	Set whether the Run- time Environment Setup window is used	FCBxxxx	Set FCB control statements
@PrinterFontName	Set the font used for print files	FOVLDIR	Set the directory containing form overlay patterns
@CnslFont	Set the console window font	OVD_SUFFIX	Set the extension of the form overlay pattern file
@ScrnFont	Set the font used for screen handling	FOVLTYPE	Set the format of the form overlay pattern file
@ODBC_Inf	Set the ODBC information file	@PRN_FormName_xxx(32)	Form name
@CBR_PrintTextPosition(32)	Method of calculating character arrangement coordinates	@DefaultFCB_Name	Name of default FCB
@CBR_TextAlign(32)	Matching the top and bottom when arranging character lines	@CBR_PrinterANK_Size(32)	Size of ANK character
@SQL_CLI (16)	Set database linkage software	@SequeLink_Inf(16)	Set the SequeLink information file
		@ODBC_Inf (16)	Set the ODBC information file
Entry information			
Subprogram-name	DLL-file-name	Secondary-entry-point-name	Subprogram-name
Subprogram-name2	Subprogram-name		

Windows 95 and Windows NT can have run-time environment information as environment variables, you can set up COBOL environment variable information directly in the user environment variables.

Windows NT can also use COBOL run-time environment variables and user environment variables for COBOL execution with the initialization file and the Run-time Environment Setup window. Information in the initialization file is reflected in user's environment variables upon execution of a COBOL program, then reset when the program ends.

How to Set Run-time Environment Information

Run-time environment information can be set as follows:

- By editing AUTOEXEC.BAT with a text editor. (Windows 95) or from the System control panel. (Windows NT)
- From the command prompt in the same manner as normal environment variables. (32)
- In the initialization file.
- In the Run-time Environment Setup window.
- From the command line.

If run-time environment information is duplicated, precedence is given in the following order:

1. Run-time Environment Setup window
2. Command line
3. Initialization file
4. Command prompt (32)
5. AUTOEXEC.BAT/ System of Control Panel (Windows NT)

Setting in AUTOEXEC.BAT (Windows 95)

Environment variables can be set when the operating system starts by editing AUTOEXEC.BAT. Each variable is set on a separate line by entering:

```
environment-variable=value
```

Where the environment-variable is one of the strings from the previous table, and value is the value to be set.

Setting from the Control Panel (Windows NT)

Set environment variables common to multiple applications by the Control Panel before execution. For details about setting from the Control Panel, refer to the online help.

Setting from the Command Prompt (32)

Use the SET command to set environment variables from the Command Prompt before executing programs.

A batch file can also be used. Environment information set here is effective only within the specific command prompt session used.

Setting in the Initialization File

Use an initialization file to set application-specific information required every time you run an application.

Create the initialization file in the directory containing the programs using the file name "COBOL85.CBR." If any other file name is given, you must enter the initialization file name in the command line upon execution.

The contents of the initialization file can be modified with an editor or by saving the contents set in the Run-time Environment Setup window. Programs can be executed even if no initialization file is available.

Contents of the Initialization File

There are two types of sections in initialization files, one for setting environment variable information, and the other for entry information. The section is identified by using a section name enclosed in brackets ("[ ]"). A section continues until the next section name is found.

The format of the initialization file is shown below:

```
[program-name]                                               ...(1)
run-time environment information =                           ...(2)
entered contents

[program-name.ENTRY]                                         ...(3)
run-time environment information =                           ...(4)
entered contents
```

1) Section name of run-time environment information:
   This indicates the start of environment variable information. Set the name of the program to execute as the section name of environment variable information. Run-time environment information set in the section is effective for the program defined as the section name as well as for programs called from the defined program.

2) Environment variable information:
   This defines environment variable information. For the format of each environment variable information, see "Environment Variables Information." Only one environment variable can be defined on each line.

3) Entry information section name:
   This indicates the start of entry information. Define the program name plus ".ENTRY" as the section name of entry information.

4) Entry information:
   This defines entry information. For the format of each entry information, see “Entry Information.” Only one entry can be defined on a single line.

Entry Example of the Initialization File

```
 [PROG1]
@EnvSetWindow=USE
@CnslWinSize=(80,24)
@CnslBufLine=100
@WinCloseMsg=ON
@IconName=COB85EXE

[PROG1.ENTRY]
SUB1=SUB1.DLL
ENTRY1=PROG1
```

Do not insert any spaces between the brackets and section name.

Saving the information set in the Run-time Environment Setup window modifies the contents of the initialization file.

Setting in the Run-time Environment Setup Window

Setting run-time environment information in the Run-time Environment Setup window allows you to easily change information for every test. See “Run-time Environment Setup Window”.

If the information is not saved in the initialization file, the run-time environment information set in the Run-time Environment Setup window is only effective for that execution of the program.

Setting from the Command Line

This approach specifies run-time environment information as arguments of a command when starting a program from the command prompt. You can specify the run-time parameter in the GS-series (M-host) format (run-time environment information name @MGPRM), initialization file, and run-time options (run-time environment information name @GOPT).

The format is shown below:

```
executable-file-name [run-time parameter] [-CBR initialization file-name] [-
CBL run-time option]
```

-CBR and -CBL can be in any order.

Specifying the Run-time Parameter in the GS-series Format

The first argument following the command name is the GS-series format run-time parameter. For example:

```
PROG1.EXE "ABCDE"
```

ABCDE is specified as the GS-series format run-time parameter.

Note: This method is applicable only to users of the GS-series.

Specifying the Initialization File Name

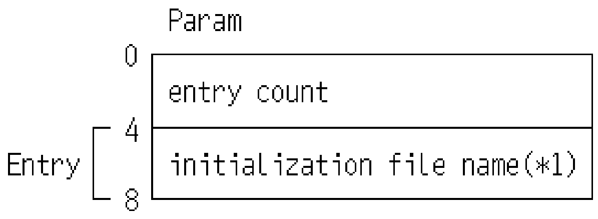
- When COBOL is the main program

  Specify the initialization file following identifier -CBR or /CBR.

- When C is the main program

  Call the following function immediately after JMPCINT2 is called:

```
int JMPCINTC (int 0, void &param)
```



(*1) Pointer to a NULL terminated string containing the initialization file name including path name (may be omitted). The maximum length of the string which can be specified is 127 characters (including the terminator).

JMPCINTC return value	**Meaning**
0	Normal termination
-1	Insufficient work area / Non-call of JMPCINT2 / Specified mistake of function code.
1	Multiple calls: i.e. JMPCINTC was called after it had already been called. The effective initialization file name is that specified in the first execution of JMPCINTC.

- When Visual Basic is the main program

  Make the declarations described below.
  Call JMPCINTB immediately after the call to JMPCINT2.

  Description in DECLARE phrase:

```
Private Declare Sub JMPCINT2 Lib "F3BIPRCT.DLL" ()
Private Declare Sub JMPCINT3 Lib "F3BIPRCT.DLL" ()
Private Declare Function Sub JMPCINTB Lib "F3BIPRCT.DLL"(*)
  Alias "JMPCINTB@8" (ByVal a As Long, ByVal D As String) As long
```

  (*) Note this line is continued in the next line, and is actually only a single line.

  Example of calling JMPCINTB:

```
Static DATA As String * 14
Dim ans As Long

Data = "c:\test.cbr" & chr(0)
ans = JMPCINTB(0, DATA)
```

Where the meaning of the arguments and return value of JMPCINTB are as follows.

JMPCINTB argument value	**Meaning**
First argument value	Function code. (Change of CBR name, 0 is set.)
Second argument value	Add Initialization file name string (Chr(0) terminator) for execution. The maximum length of the string which can be specified Is 127 characters (including the terminator).

JMPCINTB return value	**Meaning**
0	Normal termination
-1	Insufficient work area / Non-call of JMPCINT2 / Specified mistake of function code.
1	Multiple calls: i.e. JMPCINTB was called after it had already been called. The effective initialization file name is that specified in the first execution of JMPCINTB.

Do not call JMPCINT2 / JMPCINT3 / JMPCINTC / JMPCINTB more than once in the same application. Behavior cannot be guaranteed on second and subsequent calls.

Specifying Run-time Options in the Command Line

Run-time options are specified following identifier -CBL or /CBL. For the format of the run-time options, see “Format of Run-time Options.” For example:

```
PROG1.EXE -CBL r20 c20
```

r20 and c20 are specified as run-time options.

Format of Run-time Environment Information

This section describes the format of run-time environment information. The parameters for some fields can be case-sensitive so be sure to enter values for the parameters as shown.

Environment Variables

The following table lists environment variable information. The numbers refer to the detailed descriptions that follow the table.

Table 10. Environment Variable Information

Function	**Environment Variable Information**	**No.**
Environment Variables Related to Run-time Options		
Set run-time options	@GOPT	1
Set the GS-series format run-time parameter	@MGPRM	2
Environment variables related to windows		
Set the DLL-name of an icon resource	@IconDLL	3
Set the identifier of an icon resource	@IconName	4
Set the size of the logical screen for screen handling	@ScrnSize	5
Set the size of the console window	@CnslWinSize	7
Set the buffer count for the console window	@CnslBufLine	8
Environment variables related to messages		
Set the message output file	@MessOutFile	6
Set a display message when the window closes	@WinCloseMsg	9
Set to suppress run-time messages	@NoMessage	20
Environment variables related to fonts		
Set the console window font	@CnslFont	12
Set the font used for screen handling	@ScrnFont	13
Set the font used for print files	@PrinterFontName	14

Table 10. Environment Variable Information (cont.)

Function	Environment Variable Information	No.
Environment variables related to files / related to SQL		
Set exclusive control of files	@AllFileExclusive	11
Set the logical destination definition file	@CBR_CIINF	17
Set the entry information file (32)	@CBR_ENTRYFILE	18
Set the file used by the program	File-identifier	26
Set the input file for the ACCEPT/DISPLAY function	SYSIN-access-name	28
Set the output file for the ACCEPT/DISPLAY function	SYSOUT-access-name	29
Set the directory containing form overlay patterns	FOVLDIR	32
Set the extension of the form overlay pattern file	OVD_SUFFIX	33
Set the format of the form overlay pattern file	FOVLTYPE	34
Environment variables related to presentation files		
Set the connected product name used from the presentation file (by destination)	@CBR_PSFILE_xxx	19
Set the information file used by the program	File-identifier	27
Set the connected product name used from the presentation file (by file)	File-identifier	27
Environment variables related to printing		
Specify method of calculating character arrangement coordinates (32)	@CBR_PrintTextPosition	21
Specify alignment of print characters with either top or bottom of line (32)	@CBR_TextAlign	22
Specify paper size (32)	@PRN_FormName_xxx	23
Specify default FCB name	@DefaultFCB_Name	24
Specify ANK character size (32)	@CBR_PrinterANK_Size	25
Environment variables related to database		
Set the ODBC information file	@ODBC_Inf	15
Set the database type linkages software(16)	@SQL_CLI	16
Other environment variables		
Set if the Run-time Environment Setup window is used	@EnvSetWindow	10
Set the function key for screen handling	TERMINATOR	30
Set FCB control statements	FCBxxxx	31

1 @GOPT (Set Run-time Options)

```
@GOPT= list of run-time options
```

Specify run-time options as a list of run-time options. For the format of the run-time options, see “Format of Run-time Options.” For example:

```
@GOPT=r20 c20
```

2 @MGPRM (Set the GS-series Format Run-time Parameter)

```
@MGPRM= "string of run-time parameter"
```

Specify a string passed from the program in quotation marks ("). The specified string is passed to other programs in the same manner as the program is executed on the system of the GS-Series.

For more information about the GS-Series format run-time parameter, see Appendix I, ”GS-series Function Comparison.” For example:

```
@MGPRM="A2042CDE"
```

3 @IconDLL (Set the DLL name of an Icon Resource)

```
@IconDLL= DLL-name of icon resource
```

Specify the DLL name of an icon resource if no icon resource is included in the executable file. For using an icon provided by COBOL85, also set the following file:

- F3BIICON.DLL

4 @IconName (Set the Identifier of an Icon Resource)

```
@IconName= icon resource identifier
```

Specify the identifier of an icon resource when changing the icon. COBOL85 provides "COB85EXE". SDK is required when you create icons.

5 @ScrnSize (Set the Size of the Logical Screen for Screen Handling)

```
@ScrnSize=[{(columns, lineage) | (80, 24)}]
```

Specify the logical screen size of the window used by the screen handling function. Specify columns and lineage in the range from 1 to 999. If (columns + 1) * lineage exceeds 16250, an error occurs upon execution of the program.

6 @MessOutFile (Set the Message Output File)

```
@MessOutFile= file-name
```

Specify the name of the file containing the contents displayed in the message box as file name. When a file name is specified, no message box is displayed.

7 @CnslWinSize (Set the Size of the Console Window)

```
@CnslWinSize=[{(columns, lineage) | (80, 24)}]
```

Specify the size of the console window used for the ACCEPT/DISPLAY function. Specify the number of columns and lines in the range from 1 to 999. The minimum and maximum values of the window size are the system values. If a value outside the system value range is specified, it is adjusted to the system minimum or maximum value.

8 @CnslBufLine (Set the Buffer Count for the Console Window)

```
@CnslBufLine=[{buffer-lines | 100}]
```

Specify the number of buffers for the console window used by the ACCEPT/DISPLAY function. Specify buffer lines in the range from 1 to 9999.

If (console window columns + 1) * (buffer lines) exceeds 65000, the specified value is decreased. For example, buffer-lines can be up to 802 when the console window columns is 80; and 706 when 91.

9 @WinCloseMsg (Set a Display Message When the Window Close)

```
@WinCloseMsg={ON | OFF}
```

Specify whether a confirmation message is displayed (ON) or not (OFF) when the console window used by the ACCEPT/DISPLAY function or the window used by the screen handling function closes.

10 @EnvSetWindow (Set if Run-time Environment Setup Window is Used)

```
@EnvSetWindow=[{USE | UNUSE}]
```

Specify whether the Run-time Environment Setup window is displayed (USE) or not (UNUSE) when a program is executed. To change to USE once UNUSE is set, modify the contents of the initialization file.

11 @AllFileExclusive (Set Exclusive Control of Files)

```
@AllFileExclusive={YES | NO}
```

Specify whether all files are exclusively controlled (YES) or not (NO) when a program is executed. When YES is selected, files used by the program cannot be accessed from any other program (open error). File exclusive control can reduce file access time.

12 @CnslFont (Set the Console Window Font)

```
@CnslFont= (font-name, font-size)
```

Specify the font of the console window used for the ACCEPT / DISPLAY function.

13 @ScrnFont (Set the Font Used for Screen Handling)

```
@ScrnFont= (font-name, font-size)
```

Specify the font used for screen handling.

14 @PrinterFontName (Set the Font Used for Print Files)

```
@PrinterFontName= (Minchou-font-name, Gothic-font-name)
```

Specify the font used for print files.

- Specify the printer font name using up to 32 single-byte alphanumeric characters.
- Spaces before and after the printer font name are included in the printer font name.
  - Minchou font name: Specify the font name used for printing the data item that is specified MINCHOU or

MINCHOU-HANKAKU font in a PRINTING MODE clause. If omitted, "CourierNew" is assumed.

- Gothic font name: Specify the font name used for printing the data item that is specified G, GOTHIC or GOTHIC-HANKAKU font in a PRINTING MODE clause. If omitted, "CourierNew" is assumed.

15 @ODBC_Inf (Set the ODBC Information File)

```
@ODBC_Inf= ODBC-information-file-name
```

Specify the name of the file containing information required by the run-time system to use ODBC. Information in this file is mainly used to connect the client and server (with the CONNECT statement).

16 @SQL_CLI (Set the Type of Database Linkage Software) (16)

```
@SQL_CLI= database-linkage-software-name
```

Specify the linkage software name.

- RDB2 When the linkage software is RDBII Esql-COBOL

This environment variable is used only when “RDBII Esql-COBOL” is specified as the database access mode. (It is specified either at installation or within P-STAFF).

17 @CBR_CIINF (Set the Logical Destination Definition File)

```
@CBR_CIINF= definition-file-name
```

Specify the name of the file containing the logical destination definition when using the simplified inter-application communication facility.

18 @CBR_ENTRYFILE (Set the Entry Information File) (32)

```
@CBR_ENTRYFILE= entry information file-name
```

To specify entry information other than the entry description section of the initialization file (COBOL85.CBR) create an entry information file and specify it in the run-time environment @CBR_ENTRYFILE entry. See “Entry Information.”

The order of precedence for entry information is as follows:

1. Specification in the Run-time Environment Setup window.
2. Specification in the initialization file.
3. Specification in the entry information file.

19 @CBR_PSFILE_xxx (Set the Connected Product Name Used from the Presentation File (by Destination))

```
@CBR_PSFILE_xxx= connected-product-name
```

For xxx, specify the destination name described in the SYMBOLIC DESTINATION clause of the presentation file. xxx can be one of the following: DSP / PRT / ACM / APL.

For the connected-product-name enter a string indicating the associated product name to be used. The following table shows the strings that can be specified.

Table 11. Supported @CBR_PSFILE strings

Specification in SYMBOLIC DESTINATION Clause	Product Used	String Specified
DSP or SYMBOLIC DESTINATION clause may be omitted	FORM RTS	MEFT or omitted
	MeFt/NET	MEFTNET
PRT	FORM RTS	MEFT or omitted
	MeFt/NET	MEFTNET
ACM	BS*NET	ACM
	RDB/7000 Server for Windows NT	ACM
APL	IDCM	IDCM or omitted

If a connected product name is specified as a file identifier in the presentation file (refer to No. 26), the specified connected product name is not effective.

If no connected product name is specified in the run-time environment information and "ACM" is specified, an error message is displayed during execution.

If the SYMBOLIC DESTINATION clause is omitted, always specify "DSP."

20 @NoMessage (Set to Suppress Run-time Messages)

```
@NoMessage= YES
```

Suppresses the following run-time messages:

- Run-time messages other than U level
- Run-time messages not requiring operator response

To suppress the message displayed when the window closes, specify "@WinCloseMsg=OFF" (refer to No. 9).

21 @CBR_PrintTextPosition (Specify method of calculating character arrangement coordinates) (32)

```
                              TYPE1
@CBR_PrintTextPosition=
                              TYPE2
```

Specify the method of calculating coordinates (x,y) for arranging printed characters when the FORMAT clause is omitted from the file control entry.

An effective range of this specification is in the execution unit.

You specify whether (TYPE2) or not (TYPE1) to correct the character arrangement coordinates.

- TYPE1

  The x coordinate is calculated by dividing the DPI (dots per inch) by the CPI (characters per inch) and then multiplying the remainder (which is rounded down) by the line number. The y coordinate is calculated by dividing the DPI by the LPI (lines per inch) and then multiplying the remainder (which is rounded down) by the line number. **Note**: Errors caused by rounding down the remainder may cause the PrintTextPosition to shift.

- TYPE2

  The x coordinate is calculated by multiplying the line number by a defined constant, then dividing by the CPI, then multiplying by the DPI and then dividing the remainder by the defined constant. The y coordinate is calculated by multiplying the line number by a defined constant, then dividing by the LPI, then multiplying by the DPI, and then dividing the remainder by the defined constant. **Note**: As with TYPE1, the remainder is rounded down. However, in this case, the error is corrected every one inch and actual PrintTextPosition does not shift.

22 @CBR_TextAlign (Specify alignment of print characters with either top or bottom of line) (32)

```
                    TOP
@CBR_TextAlign=
                    BOTTOM
```

Specify the alignment of print characters in a line when the FORMAT clause is omitted from the file control entry. When the printer line height is greater than the character height, you can specify alignment of the printed characters to the top of the line or the bottom of the line.

23 @PRN_FormName_xxx (Specification of paper size) (32)

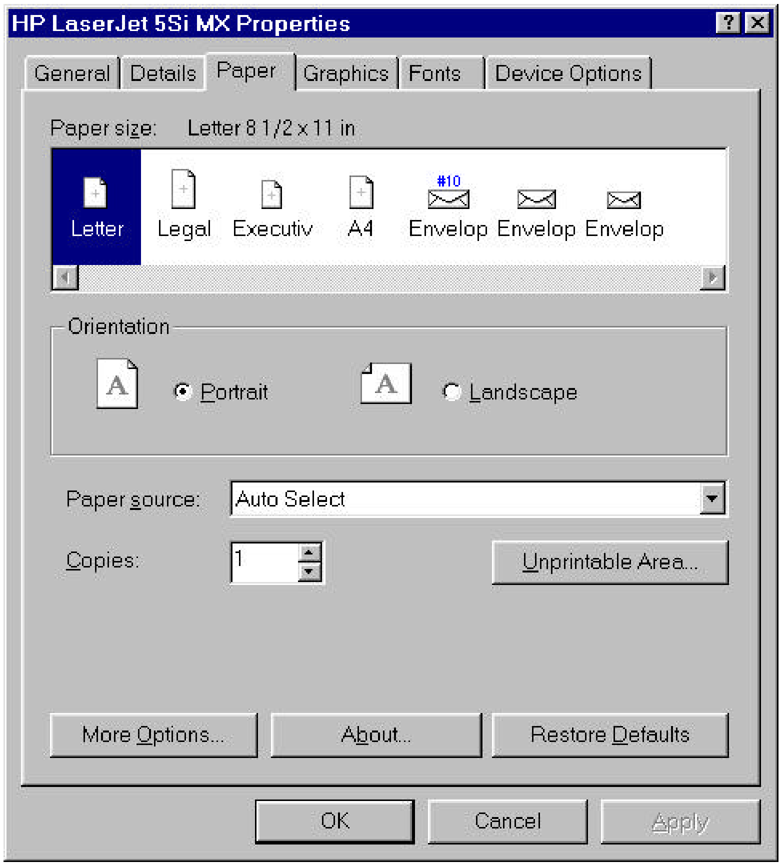
Dynamically specify the paper size to be used at run-time when the FORMAT clause is omitted from the file control entry and the I control record is used for printing (refer to “Using Print File 2” in Chapter 8). In the I control record, the FSIZE field is set to a user-defined string of three characters or less. The user-defined string replaces "XXX".

Environment variable @PRN_FormName_xxx is set equal to the string that defines the paper size. The values for the paper size strings are found in the Windows system printer defaults.

Windows 95:

Use the information in the Paper tab of the printer Properties dialog box to obtain paper sizes. Click on the Start button and select Settings→Printers. Highlight the target printer and select Properties from the File menu. Available paper sizes are shown as icons. Click on an icon to display the specifications (following Paper Size).

The following figure shows the printer Properties dialog box.



HP LaserJet 5Si MX Properties

General | Details | Paper | Graphics | Fonts | Device Options

Paper size: Letter 8 1/2 x 11 in

Letter, Legal, Executiv, A4, #10 Envelop, Envelop, Envelop

Orientation: [✓ Portrait] Landscape

Paper source: Auto Select

Copies: 1

Unprintable Area...

More Options... | About... | Restore Defaults

OK | Cancel | Apply

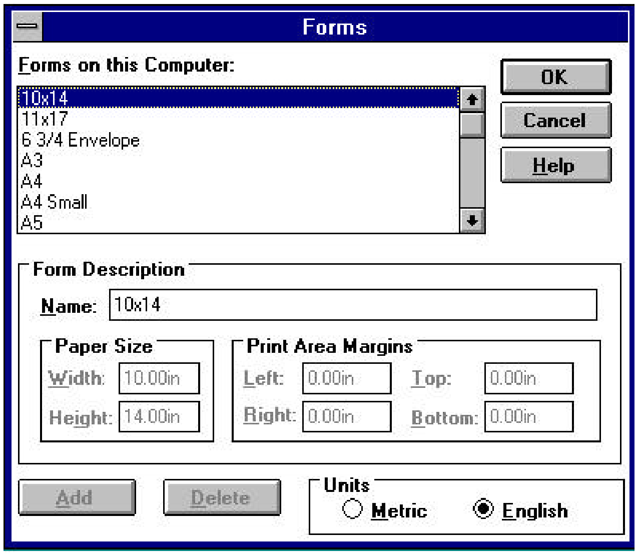
Figure 47. A Windows 95 printer Properties dialog box

Windows NT:

Use the information in the Forms dialog box to obtain paper sizes. Activate Print Manager and select Forms from the Printer menu. Available paper sizes are listed in the Forms on this

Computer list box. Click on a paper size and the specifications are displayed in the [Form Description] Name edit box.

Note: The range of support differs between printers. Be sure to confirm that the paper size is supported by the target printer driver.



Forms on this Computer:
10x14
11x17
6 3/4 Envelope
A3
A4
A4 Small
A5

Form Description

Name: 10x14

Paper Size
Width: 10.00in
Height: 14.00in

Print Area Margins
Left: 0.00in
Top: 0.00in
Right: 0.00in
Bottom: 0.00in

Units
Metric [✓ English]

Figure 48. Windows NT Forms dialog box

For example:

Specification of SIZE field on I control record: ABC

Environment variable name after xxx is substituted: @PRN_FormName_ABC

Association of environment variable name and paper size: @PRN_FormName_ABC=Letter 8 ½ x 11 in

24 @DefaultFCB_Name (Specification of default FCB name)

```
@DefaultFCB_Name=Default FCB name
```

Specify the default FCB name to be used when the FCB name is set to spaces in the I control record. The example below demonstrates setting the default FCB name to "6LPI" which has a line interval of 6 lines per inch, 66 lines per page, starts printing at line 1, and is 11 inches long. The default FCB name in this example is "FCB6LPI".

For example:

```
@DefaultFCB_Name=FCB6LPI
FCB6LPI=LPI((6,66)),CH1(1),SIZE(110)                    (*1)
FCB8LPI=LPI((8,88)),CH1(1),SIZE(110)
FCBA4LD=LPI((12)),CH1(1),FORM(A4,LAND)                  (*1)
```

(*1) Other FCB names are accessed by using the FCB field in the I control record. Refer to “Using Print File 2” in Chapter 8 for additional details.

25 @CBR_PrinterANK_Size (Specification of ANK character size) (32)

```
                        TYPE-M
@CBR_PrinterANK_Size=   TYPE-PC
                        TYPE-G
```

Specify the ANK character size in the cases where the following are NOT defined in the program:

- CHARACTER TYPE phrase
- PRINTING POSITION phrase
- print file FORMAT clause

Settings are:

- TYPE-M : Default ANK character size is brought close to the size in the GS-series. Print size is 9.6 points.
- TYPE-PC : Default ANK character size is assumed to be a PC standard size. Print size is 10.5 points.

- TYPE-G : Default ANK character size is brought close to the size in the SX/G-series. Print size is 10.8 points.

If this environment variable is omitted, the print size is 7.0 points.

26 File identifier (Set the File Used by the Program)

```
File-identifier= file-name [,access-type or file-system-type]
```

Specify the file identifier entered in the ASSIGN clause of the COBOL source program as the file identifier, and the name of the file to be processed as file name.

This environment variable assigns the file in the program with the file to be processed. Specify run-time environment information in uppercase letters even if the file identifier was defined in lowercase letters in the program.

For example:

- Description of the ASSIGN clause in the COBOL source program

  ASSIGN TO OUTFILE

- Name of file processed

  F:\WORK.DAT

- Run-time environment information

  OUTFILE=F:\WORK.DAT

Access type or file system type can be the following strings depending on the type used.

- BSAM: Provides fast file processing
- RDM: Processes the RDM file (RDB/7000 Server for Windows NT)

- BTRV: Processes the Btrieve file

27 File Identifier (Set the Information File Used by the Program) (Set the Connected Product Name Used from the Presentation File (by File))

```
File-identifier= [information-file-name] [,connected-product-
name]
```

Specify the file identifier entered in the ASSIGN clause of the COBOL source program. When using FORM RTS, specify the FORM RTS window information file name or printer information file name used as data file name.

Specify a string indicating the connected product name actually used as the connected product name (refer to No. 20). If omitted, the value specified in @CBR_PSFILE_xxx is effective. If the connected product name in the run-time environment information is also omitted, the default destination in the presentation file is assumed.

Specify run-time environment information in uppercase letters even if the file identifier was entered in lowercase letters in programs.

For example:

- Description of the ASSIGN clause in the COBOL source program

  ASSIGN TO GS-DSPFILE

- SYMBOLIC DESTINATION clause

  SYMBOLIC DESTINATION IS "DSP"

- Name of file data actually used

  F:\WORK.WRC

- Connected product actually used

FORM RTS

- Run-time environment information

  DSPFILE=F:\WORK.WRC,MEFT

28 SYSIN Access Name (Set the Input File for the ACCEPT/DISPLAY Function)

```
SYSIN-access-name= file-name
```

Specify the environment variable name specified in the compiler option SSIN as SYSIN access name. Specify the name of the file used as the data input destination when the ACCEPT statement of the ACCEPT/DISPLAY function is run.

For example:

- Compiler option SSIN

  SSIN(INFILE)

- Name of input file of ACCEPT statement

  A:\INDATA.TXT

- Run-time environment information

  INFILE=A:\INDATA.TXT

29 SYSOUT Access Name (Set the Output File for the ACCEPT/DISPLAY Function)

```
SYSOUT-access-name= file-name
```

Specify the environment variable name specified in the compiler option SSOUT as SYSOUT access name. Specify the name of the file used as the data output destination when the DISPLAY statement of the ACCEPT/DISPLAY function is run.

For example:

- Compiler option SSOUT

  SSOUT(OUTFILE)

- Name of output file of DISPLAY statement

  A:\OUTDATA.TXT

- Run-time environment information

  OUTFILE=A:\OUTDATA. TXT

30 TERMINATOR (Set the Function Keys for Screen Handling)

```
TERMINATOR= [!] keyword [[,[!] keyword ] ... ]
```

Specify the PF1 to PF24 keys in keywords in the form of PFmm - PFnn (a range from mm to nn, where nn represents 01 to 24 and 01 to 09 equal to 1 to 9.) "!" indicates that the function keys specified in the keyword are disabled as input.

For example:

```
TERMINATOR=PF1-PF3, !PF4-PF24
```

31 FCBxxxx (Set FCB Control Statements)

```
FCBxxxx= FCB-control-statement
```

Specify the FCB name specified in the I control record in xxxx and an FCB control statement in FCB control statement. For the format of the FCB control statement, refer to Appendix K, “FCB Control Statement.”

32 FOVLDIR (Set the Directory Containing Form Overlay Patterns)

```
FOVLDIR=directory-name
```

Specify the directory containing form overlay patterns by using an absolute path name. If omitted, no form overlay patterns are printed. Only one directory can be specified in directory name.

33 OVD_SUFFIX (Set the Extension of the Form Overlay Pattern File)

```
OVD_SUFFIX= extension
```

Specify an extension string in extension if the default extension of the form overlay pattern file "OVD" is not used. When the file name has no extension, specify "None".

34 FOVLTYPE (Set the Format of the Form Overlay Pattern File)

```
FOVLTYPE= format
```

Specify the first four characters of the form overlay pattern file name if they are not "KOL5" (default).

Entry Information

Information about program entry points is required only for programs having dynamic program structure. Entry information can be placed in either:

- The initialization file for execution (COBOL85.CBR).
- A file whose name is given in the @CBR_ENTRYFILE environment variable.

See “Environment Variables” for details on the @CBR_ENTRYFILE variable.

Entry Information for Separately Linked Subprograms

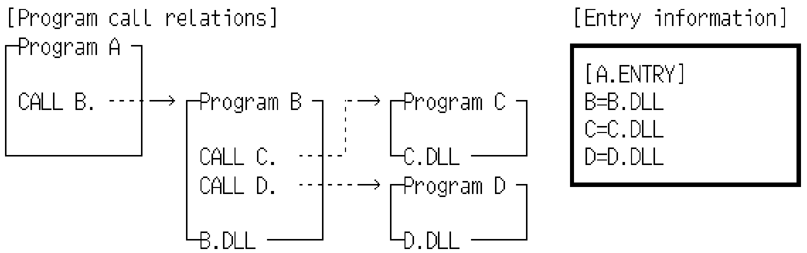
```
Subprogram-name=DLL-file-name
```

Relates the name used in the COBOL CALL statement to the name of the DLL file containing the subprogram.

Entry statements are grouped under the tag:

```
[main-program.ENTRY]
```

For example:



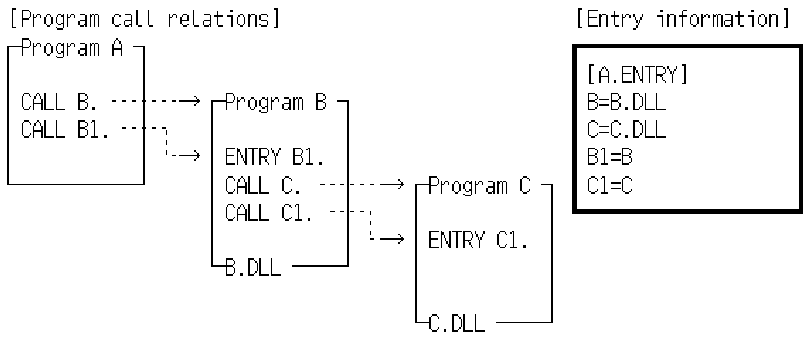
Entry Information for Secondary Entry Points

When a subprogram contains multiple entry points, the secondary entry points need to be associated with the primary entry point. The format is:

```
Secondary-entry-name=Primary-entry-name
```

The secondary-entry-name is the name defined in the ENTRY statement of the COBOL source program. The primary-entry-name is the name defined in the PROGRAM-ID paragraph.

For example:



If entry information is not specified, the system adds the extension DLL to the program name specified in the CALL statement as the DLL file name.

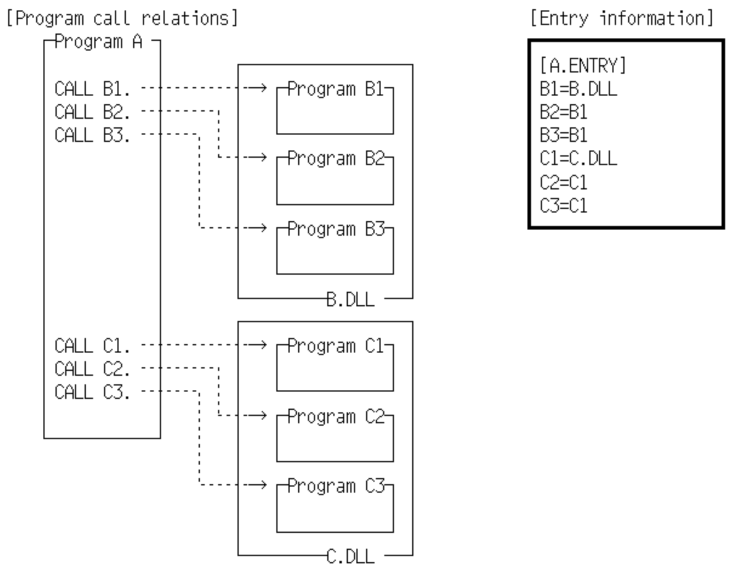
Entry Information for Subprograms Linked in the Same DLL

When subprograms are linked together in a single DLL, the first program name is mapped to the DLL file name (as documented above), and the subsequent programs are mapped to the first program. The format is:

```
Sub-program-name-2=Sub-program-name
```

Where sub-program-name is the PROGRAM-ID name of the first program in the DLL, and sub-program-name-2 is the PROGRAM-ID name of a subsequent program linked in the same DLL.

For example:



```
[A.ENTRY]
B1=B.DLL
B2=B1
B3=B1
C1=C.DLL
C2=C1
C3=C1
```

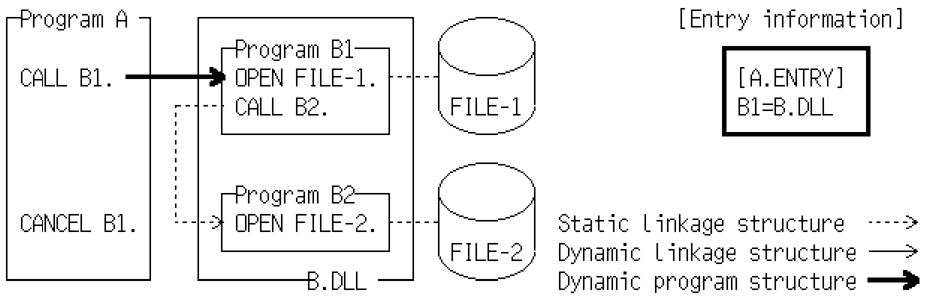
Notes on Canceling Subprograms:

The DLL containing the program that is the target of the CANCEL statement is deleted from the virtual memory space.

In the above example, if any of B1,B2, or B3 is specified in the CANCEL statement, B.DLL is deleted from the virtual memory space.

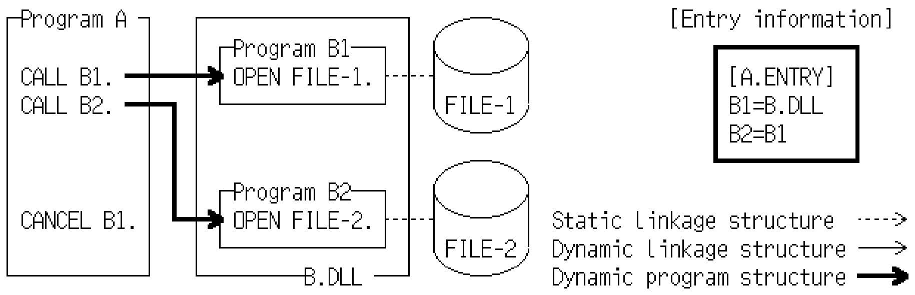
Note: Files opened in subprograms may not be closed if the subprogram is not explicitly canceled (CANCEL). In these circumstances you must make sure your programs close the affected files before a CANCEL is performed. The following three examples illustrate this concept.

Example 1: When a canceled subprogram does a static call.



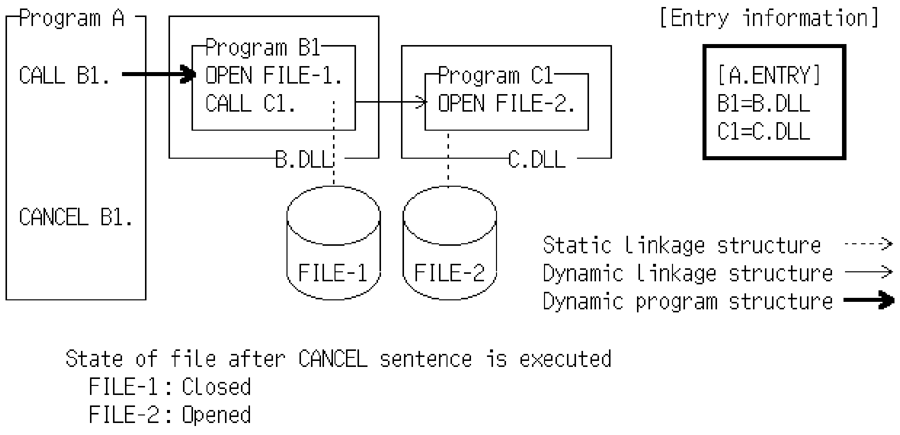
State of file after CANCEL sentence is executed
FILE-1 : Closed
FILE-2 : Opened

Example 2: When a canceled subprogram is in a DLL containing other subprograms.



State of file after CANCEL sentence is executed
FILE-1 : Closed
FILE-2 : Opened

Example 3: When a canceled subprogram does a dynamic call.

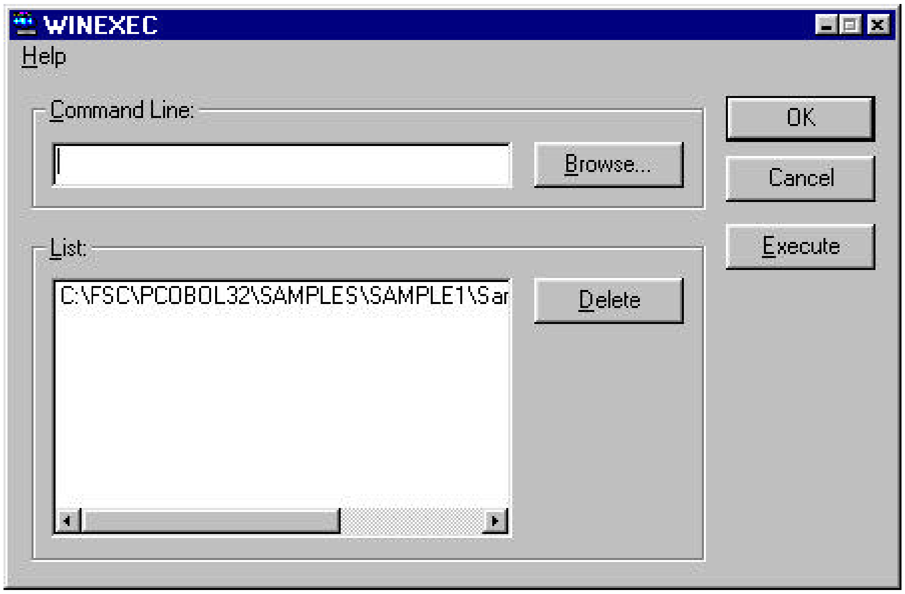


WINEXEC

WINEXEC is used for program execution. This section describes the WINEXEC window. For further details on how to use the window, refer to the online help.

Activating the WINEXEC Window (32)

In P-STAFF, select WINEXEC from the Tools menu to activate the WINEXEC Window.



C:\FSC\PCOBOL32\SAMPLES\SAMPLE1\Sar

Figure 49. The WINEXEC window

The WINEXEC window contains the following elements:

Help

Access the P-STAFF online help.

Command Line edit box

Specify the name of the file to be executed.

Browse button

Click to select a file to be included in the link.

List list box

Displays a list of the most recently executed programs.

Delete button

Click to remove the selected file(s) from the List.

OK button

Click to start execution and close the WINEXEC window.

Cancel button

Click to close the WINEXEC window without performing an execution.

Execute button

Click to start execution, and keep the WINEXEC window open for further execution.

Activating the WINEXEC Window (16)

Activate the WINEXEC window for Windows 3.1 by one of the following methods:

- Select WINEXEC from the Utilities menu of P-STAFF.
- Select WINEXEC.EXE in the File Manager directory window.
- Define WINEXEC.EXE as an icon, then double-click on the icon.
- Select the Execute command from Program Manager or File Manager, then enter the WINEXEC command in the proper command form.

The following sections describe how to use the WINEXEC window.

Entering a File Name

Enter the name of the file to execute in the Command Line edit box using either of the following methods:

- Enter or use the Browse button to select the name of the file to execute.

- Select the name of the file to execute from the List to re-execute a program that was already executed.

The List displays up to the last ten files that were executed.

Starting and Quitting Execution

To start execution, click on the Execute button. After execution starts, the system may display the Run-time Environment Setup window for COBOL programs. Set information where necessary.

For details about setting run-time environment information, see “Run-time Environment Setup Window (16).”

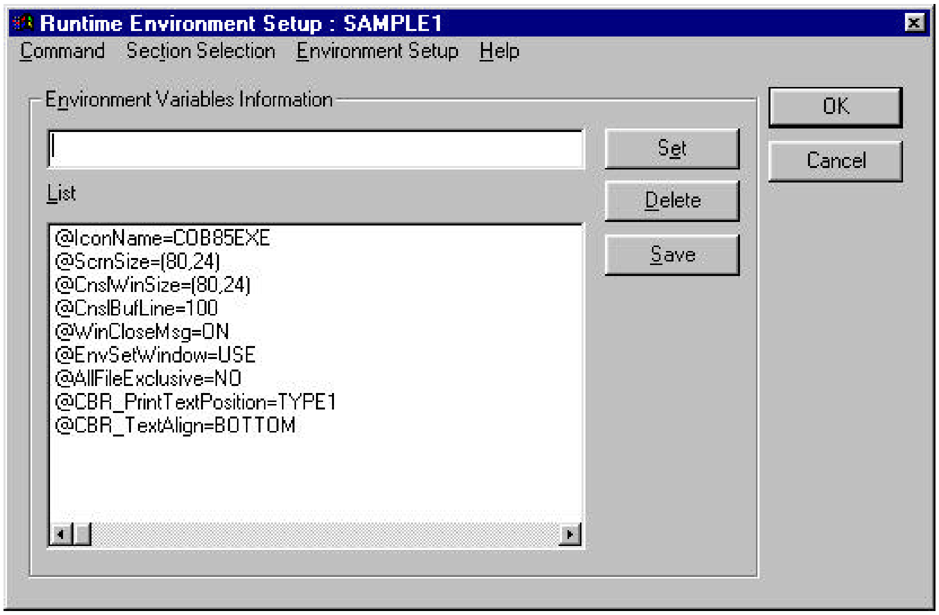
After execution is completed, click on the Cancel button in the WINEXEC window to close the window.

Run-time Environment Setup Window (32)

The Run-time Environment Setup window is used to set run-time environment information that is displayed when a COBOL program is executed. The Run-time Environment Setup window displays run-time environment information defined in the initialization file as initial values.

Information set in the Run-time Environment Setup window can also be saved in the initialization file. To prevent display of the Run-time Environment Setup window at run-time, define "@EnvSetWindow=UNUSE" in the initialization file.

Note: "UNUSE" must be in upper case.


Runtime Environment Setup : SAMPLE1
Command Section Selection Environment Setup Help
Environment Variables Information
List
Set
Delete
Save
OK
Cancel

```
@IconName=COB85EXE
@ScrnSize=(80,24)
@CnslWinSize=(80,24)
@CnslBufLine=100
@WinCloseMsg=ON
@EnvSetWindow=USE
@AllFileExclusive=NO
@CBR_PrintTextPosition=TYPE1
@CBR_TextAlign=BOTTOM
```

Figure 50. The Run-time Environment Setup window (32)

The Run-time Environment Setup window contains the following elements:

Command

Provides the functions available on the Set, Delete, Save and OK buttons, and the option to configure the Console, Screen and Printer fonts.

Section Selection

Switches the display between environment variables and entry information.

Environment Setup

Provides access to lists of the current environment variables, COBOL environment variables and printer names. Values can be selected from these lists for inclusion in the Environment Variables Information.

Help

Access the online help.

Environment Variables Information edit box

Information is entered and edited in this box then set into the list of environment variables to be used for this execution.

List list box

Displays the environment variables that will be used in this execution.

Set button

Click to register the contents of the edit box in the List.

Delete button

Click to remove the currently selected item(s) from the List.

Save button

Click to save the information in the List to the initialization file (COBOL85.CBR) for use in multiple executions.

OK button

Click to continue execution.

Cancel button

Click to discontinue execution.

Setting Environment Variable Information

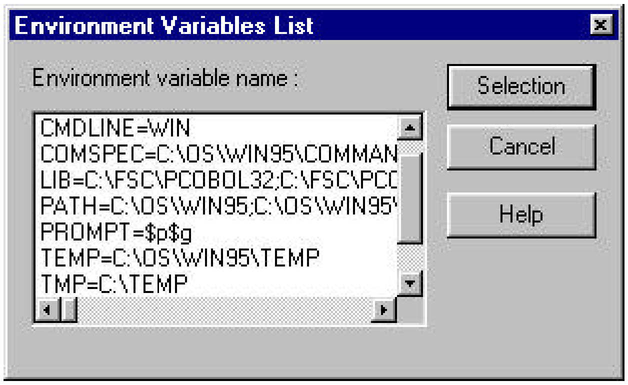
Environment information is set using the Environment Variables Information edit box of the Run-time Environment Setup window.

Adding Environment Variable Information

To add specified or new environment variables, use the procedures that follow.

Adding Currently Set Environment Variables

1. Select Environment Variables from the Environment Setup menu in the Run-time Environment Setup window. The Environment Variables List dialog box is displayed.



```
CMDLINE=WIN
COMSPEC=C:\OS\WIN95\COMMAN
LIB=C:\FSC\PCOBOL32;C:\FSC\PCC
PATH=C:\OS\WIN95;C:\OS\WIN95\
PROMPT=$p$g
TEMP=C:\OS\WIN95\TEMP
TMP=C:\TEMP
```

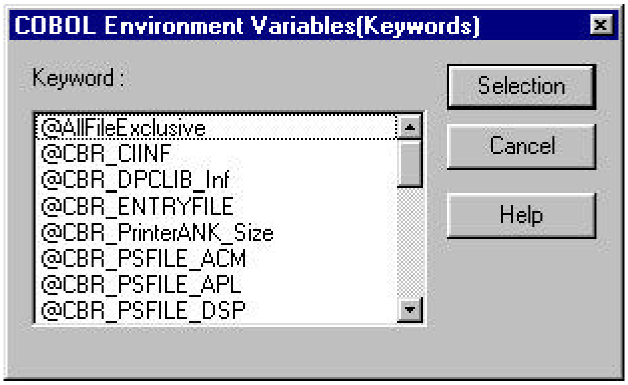
Figure 51. The Environment Variables List dialog box

2. Select the environment variable to add from the list box, then click on the Selection button. The selected variable is displayed in the Environment Variables Information edit box.
3. If necessary, modify the contents of the selected variable.

4. Click on the Set button. The changed information is added to the Environment Variables Information list box.

Adding New Environment Variables

1. Select Keyword from the Environment Setup menu in the Run-time Environment Setup window. The COBOL Environment Variables (Keywords) dialog box is displayed.



@AllFileExclusive
@CBR_CIINF
@CBR_DPCLIB_Inf
@CBR_ENTRYFILE
@CBR_PrinterANK_Size
@CBR_PSFILE_ACM
@CBR_PSFILE_APL
@CBR_PSFILE_DSP

Figure 52. The COBOL Environment Variables (Keywords) dialog box

2. Select the environment variable to add from the list box, then click on the Selection button. The selected information is displayed in the Environment Variables Information edit box.
3. Complete the information required by the environment variable.
4. Click on the Set button. The changed information is added to the Environment Variables Information list box.

Changing Environment Variable Information

To change the environment variable information or environment variables displayed in the Environment Variables Information list box:

1. Select the environment variable information or environment variables to change from the list box. The selected information is displayed in the Environment Variables Information edit box.
2. Change the contents.
3. Click on the Set button. The changed environment variable information or environment variables are displayed in the Environment Variables Information list box.

Deleting Environment Variable Information

To delete environment variable information or environment variables from the Environment Variables Information list box:

1. Select the environment variable information or environment variables to delete from the list box.
2. Click on the Delete button. The selected environment variable information or environment variables are deleted from the list box.

Setting Entry Information

Select Section Information from the Section Selection menu in the Run-time Environment Setup window. The Entry Information list box is displayed. Enter the required entry information and add it to the List by clicking on the Set button.

Entry information can also be set without using the Run-time Environment Setup window. Create an entry information file and specify the entry information file name in the environment variable @CBR_ENTRYFILE.

For the format, see “Format of Run-time Environment Information.”

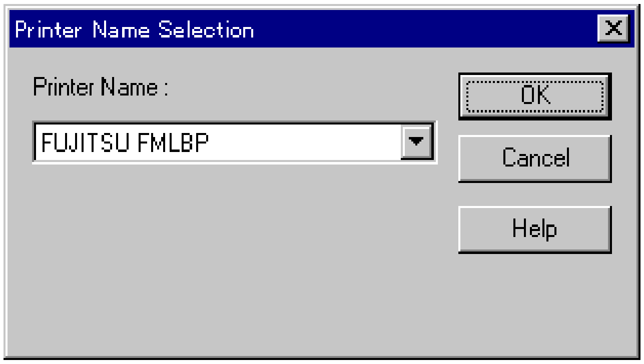
Saving to the Initialization File

When the Save button is clicked, the information in the list box is saved in the initialization file. Saving in the initialization file means that the same run-time environment information can be used the next time the program is executed.

Saving modifies the existing information.

Setting Printers

When using a printer, printer information can be set. Select Printer from the Environment Setup menu of the Run-time Environment Setup window, then set required information in the Printer Name Selection dialog box.



Printer Name : FUJITSU FMLBP

Figure 53. The Printer Name Selection dialog box

Exiting the Run-time Environment Setup Window

Click on the OK button to close the Run-time Environment Setup window and start the program. Run-time environment information set in the Run-time Environment Setup window is effective for the program only if it is not saved in the initialization file.

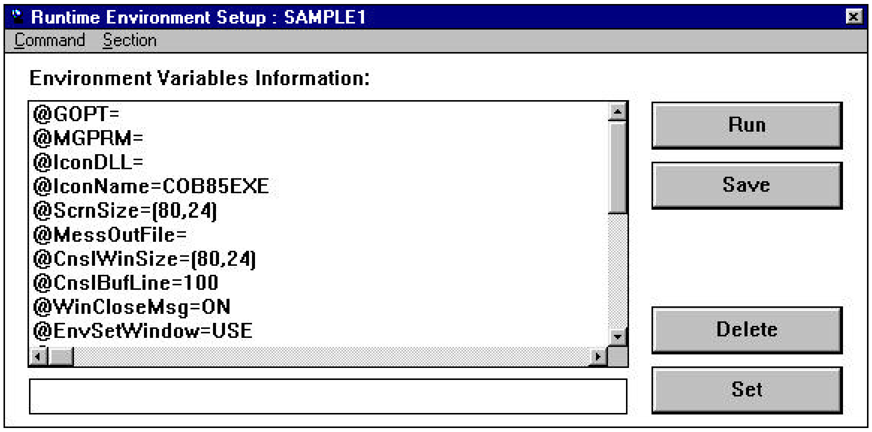
Run-time Environment Setup Window (16)

The Run-time Environment Setup window is used to set run-time environment information that is displayed when a COBOL program is executed. The Run-time Environment Setup window displays run-time environment information defined in the initialization file as initial values.

Contents set in the Run-time Environment Setup window can also be saved in the initialization file.

To prevent display of the Run-time Environment Setup window at run-time, define "@EnvSetWindow=UNUSE" in the initialization file.

This section describes the Run-time Environment Setup window. For details on the format of defined run-time environment information, see “Format of Run-time Environment Information.” For details about how to use the window, refer to the online help.


Runtime Environment Setup : SAMPLE1
Command Section
Environment Variables Information:
Run
Save
Delete
Set

```
@GOPT=
@MGPRM=
@IconDLL=
@IconName=COB85EXE
@ScrnSize=(80,24)
@MessOutFile=
@CnslWinSize=(80,24)
@CnslBufLine=100
@WinCloseMsg=ON
@EnvSetWindow=USE
```

Figure 54. The Run-time Environment Setup window (16)

You can select the operation or font from the Command menu. The Section menu allows you to switch between environment variable information and entry information.

Setting Environment Variable Information

Run-time environment variable information is set using the Environment Variables Information edit box in the Run-time Environment Setup window.

Adding Environment Variable Information

To set environment variable information not displayed in the list box:

1. Directly enter run-time environment information in the edit box.

2. Click on the Set button. The environment variables information is added to the Environment Variables Information list box.

Changing Environment Variable Information

To change run-time environment information displayed in the Environment Variables Information list box:

1. Select the run-time environment information to change from the list box. The selected information is displayed in the Environment Variables Information edit box.
2. Change the contents.
3. Click on the Set button. The changed run-time environment information is displayed in the list box.

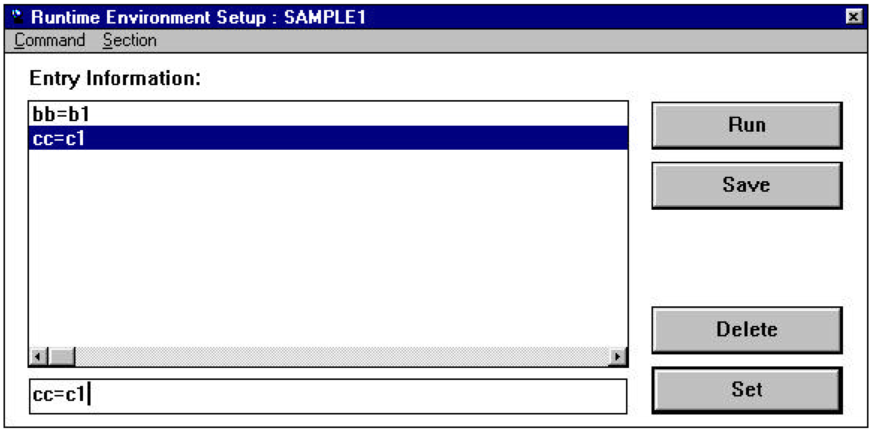
Deleting Environment Variable Information

To delete run-time environment information displayed in the Environment Variables Information list box:

1. Select the run-time environment information to delete from the list box .
2. Click on the Delete button. The selected run-time environment information is deleted from the list box.

Setting Entry Information

Select Section Information from the Section menu in the Run-time Environment Setup window. The Entry Information list box is displayed in the window. Select the required entry information.



bb=b1
cc=c1

cc=c1

Figure 55. The Run-time Environment Setup window with changed Entry Information

Saving to the Initialization File

Click on the Save button to save current information (information displayed in the list box) in the initialization file. Because information is saved in the initialization file, the same run-time environment information can be used when executing the program at another time.

Saving modifies the existing information.

Exiting from the Run-time Environment Setup Window

Clicking on the Run button causes the system to close the Run-time Environment Setup window and start executing the program. Run-time environment information set in the Run-time Environment Setup window is effective for the program only if it is not saved in the initialization file.

Format of Run-time Options

Run-time options specify information or operations to COBOL programs at run-time.

Define run-time options depending on the functions used in the COBOL program or the options specified when compiling the COBOL source program. Run-time options can be entered:

- in the text editor (Windows 95)
- in the control panel system (Windows NT)
- in the initialization file
- in the Run-time Environment Setup window
- or from the command line

The following table lists the available functions and formats of the run-time options. When multiple run-time options are defined, they must be delimited by a comma (,).

Table 12. Run-time options

Function	**Format**
Set the number of trace data	[r count]
Set the process count at error detection	[c count | noc]
Set the external switch value	[s value]

[r COUNT]

Set this option to change the amount of trace information produced by the TRACE function. Specify the amount of trace information in the range from 1 to 999999.

This option is effective in programs defined with the -Dr option or the compiler option TRACE during compilation.

[c COUNT | NOC]

Set this option to change the process when an error is detected by the CHECK function. Specify the process count in the range from 9 to 999999.

A value of 0 assumes no limit. A value of noc suppresses the CHECK function.

This option is effective only in the program defined with the -Dk option or the compiler option CHECK during compilation.

[s VALUE]

Set this option to set a value for external SWITCH-0 to SWITCH-7, specified in the SPECIAL-NAMES paragraph in the COBOL program. Enter eight consecutive switch values from SWITCH-0 sequentially.

The values can be 0 or 1. If omitted, "S00000000" is assumed.

SWITCH-8 is equivalent to SWITCH-0. When SWITCH-8 is used, therefore, the switch values correspond to SWITCH-8, SWITCH-1 to SWITCH-7 from the left.

Chapter 6. Project Management

This chapter describes the project management function, details resources needed for the function, and explains procedures for using it. Additionally, this chapter describes the project window.

What is the Project Management Function?

This section outlines the project management function.

PROGRAMMING-STAFF provides a project management function for COBOL85 application development and maintenance. The project management function manages target COBOL85 applications consisting of multiple files (COBOL source programs and libraries) as a single project.

Information on dependent files is stored in the project file, and the compiler and linker options are saved in the option file. If the project file or option file managed under the project is updated, the project management function can automatically recompile and relink files dependent on the updated file. This operation is called "building" or "making".

The project manager can also recompile and relink all programs registered in the project to renew applications, called "rebuilding".

Use the project management function:

- To process all steps from compilation to execution of a program in one operation
- To call a subprogram or use a program library

Sample programs using the project management function are provided with COBOL85. For compiling, linking, and executing sample programs using the project management function, refer to “Getting Started with Fujitsu COBOL.”

Resources Required for Project Management

Project management uses the following files:

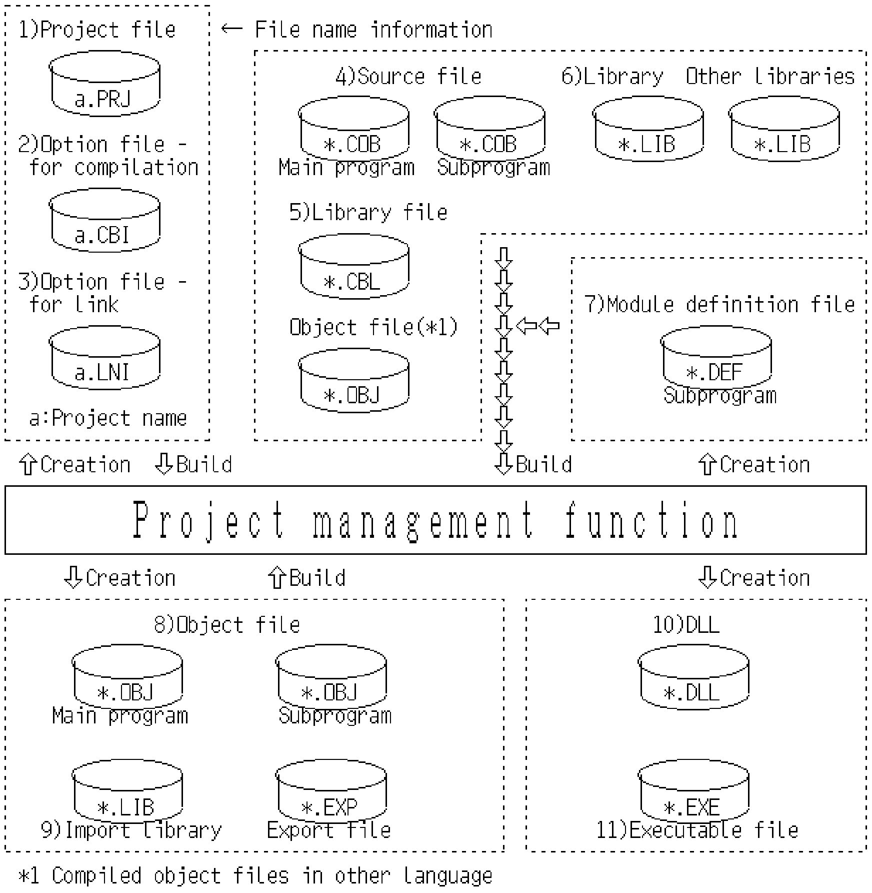


Figure 56. Files used by the Project Management function

Table 13. Files used by the Project Management function

	File Contents	File Name Format	I/O	Condition to Use or Create
1	Information for project management - Dependent files - Target-names - Source-file-names - Library-names - Other-required file-names	project-name.PRJ	I	Information saved in the project file is referenced for building or rebuilding
			O	This file must be created for project management
2	Compiler options	project-name.CBI	I	Used to compile source programs during building or rebuilding
			O	Created when compiler options are set by project management
3	Linker options	project-name.LNI	I	Used to link object programs during building or rebuilding
			O	Created automatically for project management
4	COBOL source program	optional-name.COB	I	Used to compile source programs during building or rebuilding
5	Library text	library-text-name.CBL	I	Used to compile source programs during building or rebuilding
6	Standard libraries (object code libraries)	optional-name.LIB	I	Used to link object programs during building or rebuilding
7	Module definition statement (required for subprograms)	target-name.DEF	I	Required to link during building or rebuilding
				Required to create the import library or DLL during building or rebuilding
			O	Created automatically if omitted
8	Object program	source-file-name.OBJ	I	Used to link object programs during building or rebuilding
			O	Created when source programs are compiled successfully

Table 13. Files used by the Project Management function (cont.)

	File Contents	File Name Format	I/O	Condition to Use or Create
9	Import library	target-name.LIB	I	Used to create an executable file saving the dynamic link structure
			O	Created when building or rebuilding
10	Dynamic link library(DLL)	target-name.DLL	I	Used to create an executable program with a dynamic link structure
			O	Created when building or rebuilding
11	Executable program	target-name.EXE	O	Created when building or rebuilding

Project Management Procedures

This section describes the procedures for project management.

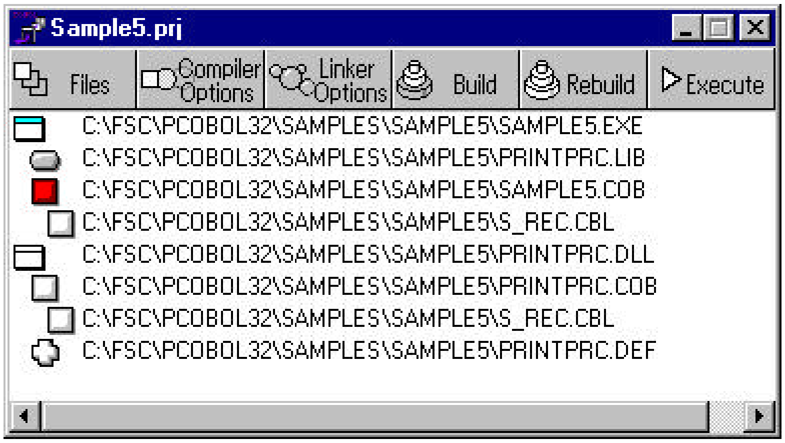
1. Select Open from the Project menu in the P-STAFF window. The Open Project dialog box is displayed.
2. Enter the project file name. See “Creating the Project File.” To create a new project file, the Target Files dialog box is displayed. If an existing project file is opened, the Project window appears. To change project information, select a button in the Project window. See “Project Window.”
3. Register the target file name to be created by the project then click on the OK button. See “Registering Target Files.” The Dependent Files dialog box is displayed.
4. Register a source program managed under the project. See “Registering Source Files.”
5. Select the main program from source programs registered in Step 4. See “Setting the Main Program.” If you are creating a DLL, do not select a main program.
6. Register libraries. See “Registering Library Files.”

7. Register any other required files. See “Registering Other Required Files.”
8. End registration. After all registration has been completed, click on the OK button in the Dependent Files dialog box. Registered file names are displayed in the Project window.
9. Set compiler options. See “Setting Compiler Options.”
10. Set linker options. See “Setting Linker Options.”
11. Build or rebuild the project. See “Building and Rebuilding the Project.”
12. If an error is detected, modify the file contents. The contents of the files registered to a program (COBOL source programs, library text, and module definition statements) can be modified by using an editor. Double-clicking on a file name displayed in the Project window opens the P-STAFF editor edit screen. See “Modifying File Contents.”
13. Execute the program. Click on the Execute button to execute the created application program from the Project window. See “Executing Application Programs.”

Project Window

The Project window is displayed when the project file is opened. In the Project window, you can:

- Edit the project file (Register files, set compiler options, and set linker options)
- Build and rebuild the project
- Edit registered files
- Execute created application programs



```
C:\FSC\PCOBOL32\SAMPLES\SAMPLE5\SAMPLE5.EXE
C:\FSC\PCOBOL32\SAMPLES\SAMPLE5\PRINTPRC.LIB
C:\FSC\PCOBOL32\SAMPLES\SAMPLE5\SAMPLE5.COB
C:\FSC\PCOBOL32\SAMPLES\SAMPLE5\S_REC.CBL
C:\FSC\PCOBOL32\SAMPLES\SAMPLE5\PRINTPRC.DLL
C:\FSC\PCOBOL32\SAMPLES\SAMPLE5\PRINTPRC.COB
C:\FSC\PCOBOL32\SAMPLES\SAMPLE5\S_REC.CBL
C:\FSC\PCOBOL32\SAMPLES\SAMPLE5\PRINTPRC.DEF
```

Figure 57. The Project window

File names registered in the project are displayed. Each file is marked with a small icon that indicates its designated category.

The Project Window contains the following elements:

Files button

Edit the dependent files list.

Compiler Options button

Display, enter and edit the compiler options.

Linker Options button

View and edit the linker options.

Build button

Start a build of all the programs in the project.

Rebuild button

Start a rebuild of all the programs in the project.

Execute button

Start an execution of the project program.

Creating the Project File

The project file registers information for project management. One project file is required per project. To create the project file:

1. Start P-STAFF. The P-STAFF window is displayed.
2. Select Open from the Project menu. The Open Project dialog box is displayed.
3. Enter the name of project file in the File Name edit box.
4. Click on the OK button. The project file is created. After creation, the Target Files dialog box automatically appears to enter the target name. After creating the project file, register the files.

Set project-name.PRJ as the project file name.

Registering Files

For project management, files managed under the project must be registered. Information on registered files is saved in the project file. Register the following files for the project:

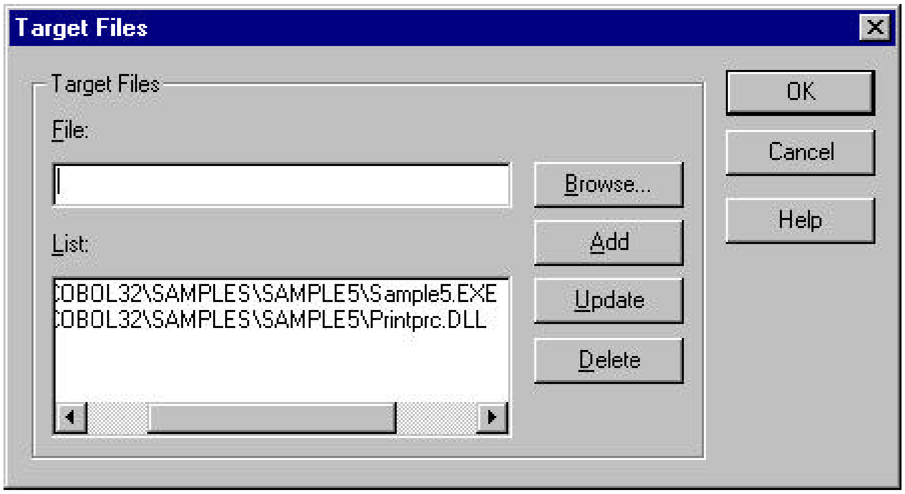
- Target file
- Source files
- Library files
- Other required files

Registering Target Files

Target files are registered in the Target Files dialog box. Multiple target files can be registered. To register target files:

1. When a new project is created, create the project file, and the Target Files dialog box is displayed with project-name.EXE displayed in the File edit box.

   For an existing project file, select the Files button in the Project window. The Dependent Files dialog box is displayed. Click on the Set button and the Target Files dialog box is displayed.



COBOL32\SAMPLES\SAMPLE5\Sample5.EXE
COBOL32\SAMPLES\SAMPLE5\Printprc.DLL

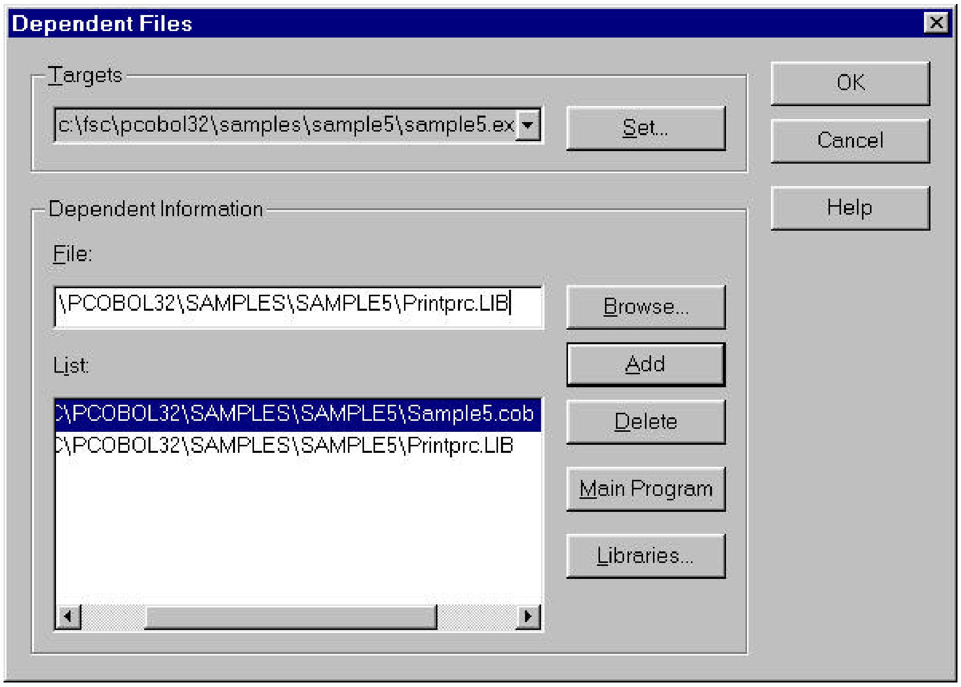
Figure 58. The Target Files dialog box

2. Enter a name in the File edit box and click on the Add button. To update, select a target name to update from the Target Files list box, enter the file name in the File edit box, and click on the Update button.
3. Repeat Step 2 until all target files have been registered.
4. After all target files have been registered, click on the OK button. The Dependent Files dialog box is displayed.

Registering Source Files

Source files are registered in the Dependent Files dialog box. To register source files:

1. For a new project, click on the OK button in the Target Files dialog box. For an existing project file, click on the Files button on the Project window.



c:\fsc\pcobol32\samples\sample5\sample5.ex

\PCOBOL32\SAMPLES\SAMPLE5\Printprc.LIB

C:\PCOBOL32\SAMPLES\SAMPLE5\Sample5.cob
C:\PCOBOL32\SAMPLES\SAMPLE5\Printprc.LIB

Figure 59. The Dependent Files dialog box

2. Select a file from Targets.
3. Click on the Browse button; the Browse Files dialog box is displayed and you can select the source files required for the target.
4. Click on the OK button. The selected source file is displayed in the File edit box of the Dependent Files dialog box.
5. Click on the Add button. The selected source file is added to the Dependent Files list box.
6. When the added source file is the main program, set the main program. See “Setting the Main Program.”
7. Repeat Steps 2 to 6 for all targets.

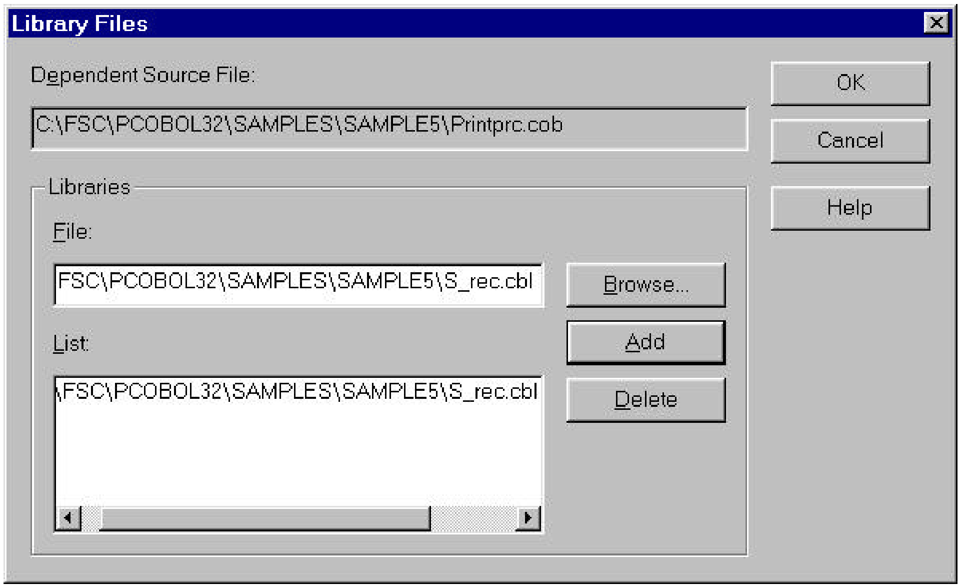
Setting the Main Program

When the source file registered is the main program, the file must be set as the main program. To set the main program, select the file to be used as the main program from the files in the Dependent Files list box, then click on the Main Program button. The small icon next to the selected file turns red.

Registering Library Files

Source file libraries are registered in the Library Files dialog box. To register a library file:

1. Select the source file of the program defining the library from the Dependent Files list box of the Dependent Files dialog box.
2. Click on the Libraries button. The Library Files dialog box is displayed.



Dependent Source File:

C:\FSC\PCOBOL32\SAMPLES\SAMPLE5\Printprc.cob

Libraries

File:

FSC\PCOBOL32\SAMPLES\SAMPLE5\S_rec.cbl

List:

\FSC\PCOBOL32\SAMPLES\SAMPLE5\S_rec.cbl

Figure 60. The Library Files dialog box

3. Click on the Browse button, the Browse Files dialog box is displayed. Select the library required by the dependent source file selected in Step 1.
4. Click on the OK button. The library file name is displayed in the File edit box of the Library Files dialog box.
5. Click on the Add button. The library file name is added to the Library Files list box.
6. After all library files are registered, click on the OK button. The Dependent Files dialog box is redisplayed.
7. Repeat Steps 1 to 6 for all source programs using the library.

Registering Other Required Files

Other required files are registered in the Dependent Files dialog box. The file selection process in the Dependent Files dialog box is the same as for source files.

Register required files to the project after determining the type of link and program structure.

The following files can be registered to the project:

- Module definition file

  Required when creating a DLL. See “Creating the Module Definition File.”

- Object file

  Object files created in COBOL or other language can be specified.

- Import library

  An import library created by the project is required when executable programs have a dynamic link structure. The import library must have the name DLL-name.LIB. When a DLL is created, an import library (target-name.LIB) is created. If this file already exists, the existing file is replaced with a new one.

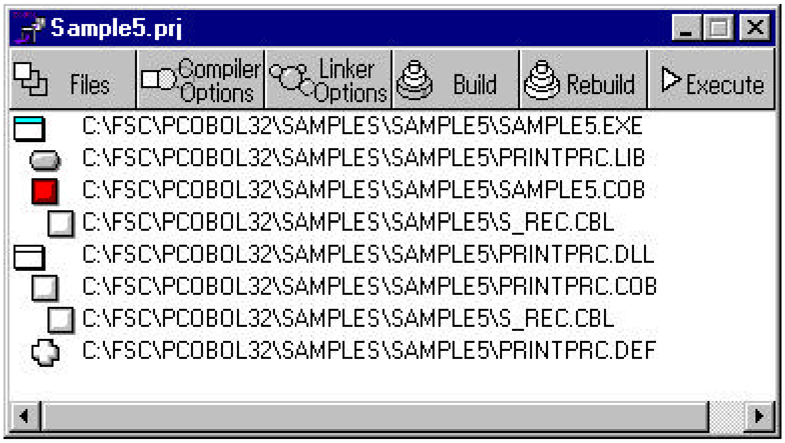
- Other libraries

  Other libraries (excluding libraries provided by COBOL85) can be specified.

Quitting Registration

After registration is completed, click on the OK button in the Dependent Files dialog box. The Dependent Files dialog box closes; control is then returned to the project window. The Project window then displays all registered files by target.

- The Project window is displayed outside the P-STAFF window. (32)
- The Project window is displayed inside the P-STAFF window. (16)



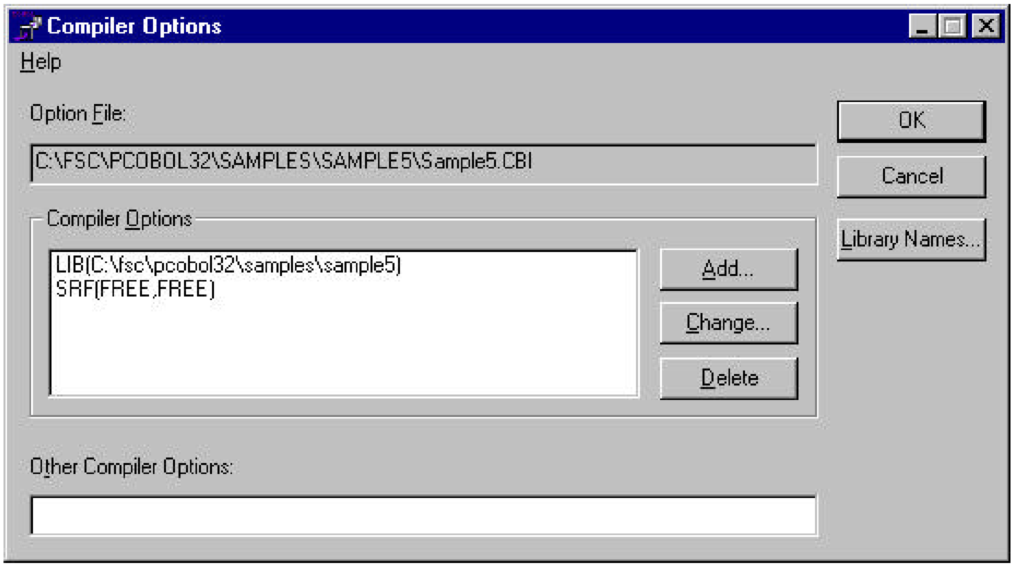
```
C:\FSC\PCOBOL32\SAMPLES\SAMPLE5\SAMPLE5.EXE
  C:\FSC\PCOBOL32\SAMPLES\SAMPLE5\PRINTPRC.LIB
  C:\FSC\PCOBOL32\SAMPLES\SAMPLE5\SAMPLE5.COB
    C:\FSC\PCOBOL32\SAMPLES\SAMPLE5\S_REC.CBL
C:\FSC\PCOBOL32\SAMPLES\SAMPLE5\PRINTPRC.DLL
  C:\FSC\PCOBOL32\SAMPLES\SAMPLE5\PRINTPRC.COB
    C:\FSC\PCOBOL32\SAMPLES\SAMPLE5\S_REC.CBL
  C:\FSC\PCOBOL32\SAMPLES\SAMPLE5\PRINTPRC.DEF
```

Figure 61. The Project window

Setting Compiler Options

Compiler options are set before compiling source files managed under the project. Saved compiler options are saved in the option file (project-name.CBI). To set compiler options:

1. Click on the Compiler Options button on the Project window. The Compiler Options dialog box is displayed. Refer to “Compiler Options Dialog Box” in Chapter 3 for more details.
2. Set the required information.
3. Click on the OK button.



Option File:

C:\FSC\PCOBOL32\SAMPLES\SAMPLE5\Sample5.CBI

Compiler Options

LIB(C:\fsc\pcobol32\samples\sample5)
SRF(FREE,FREE)

Other Compiler Options:

Figure 62. The Compiler Options dialog box

In this example, the compiler option MAIN is not effective even when specified because MAIN in the main program takes precedence.

When using libraries, always set the compiler option LIB.

The same compiler option must be set for all registered targets.

Compiler options set here are effective in the project only. These compiler options are not effective when programs are compiled from the WINCOB window.

Setting Linker Options

Different linker option screens are provided for Windows 95/Windows NT and Windows 3.1. For details on the Linker Options dialog box, refer to “Setting Linker Options” in Chapter 4.

The linker options are saved in the option file (project-name.LNI).

1. Click on the Linker Options button on the Project window. The Linker Options dialog box is displayed.
2. Set the required information.
3. Click on the OK button.

When the compiler option TEST is specified, linker options are automatically set. In this case, linker options for debugging need not be set. (32)

Creating the Module Definition File

The module definition file can be created by building or rebuilding the project. The procedures to follow differ depending on whether you are using Windows 95/Windows NT and Windows 3.1.

Module Definition File (32)

The module definition file is required to create the import library. To create the module definition file templates:

1. Select Create Module-Definition Files from the Project menu in the P-STAFF window.

2. Build or rebuild the project. A module definition file having the DLL-name.DEF is created in the same folder as DLL and is added to the project file.

Assign the name target-name.DEF to module definition files.

The EXPORTS module definition statement in the DLL is automatically set with the file name of the source file and object file depending on the target excluding their extension. If the program identifier differs from the file name, modify the value of EXPORTS.

Module Definition File (16)

The module definition file used under Windows 3.1 is required for linking. The template for a module definition file can be created when using the project. To create the module definition file:

1. Select Create Module-Definition Files from the Project menu in the P-STAFF window.
2. Select Template of Module-Definition File from the Project menu in the P-STAFF window.
3. Select the target as EXE or DLL. A dialog for creating templates appears.
4. Set the required information.
5. Build or rebuild the project. A module definition file containing the contents specified in Step 4 and with a target-name.DEF is added to the project file.

Assign the name target-name.DEF to module definition files.

The EXPORTS module definition statement in DLL is automatically set with the file name of the source file and object file depending on the target excluding their extension. If the program identifier differs from the file name, modify the value of EXPORTS.

Building and Rebuilding the Project

You must either build or rebuild the project when any file registered to the project is updated, or when you change compiler or linker options.

To build or rebuild the project, click on the Build or Rebuild button in the Project window. Compile errors can be corrected by jumping to the editor from the message window.

The following table lists system processing executed with Build. Rebuilding means that the system recompiles and relinks all registered programs.

Table 14. Build processing

Update Contents	Building
COBOL source program	1) Recompile the updated program 2) Relink dependent on the updated
Library file	1) Recompile all programs defining the library 2) Relink all programs dependent on the programs recompiled in 1)
Compiler option	Recompile and relink all programs
Linker option	Relink all programs

Modifying File Contents

Contents of files registered to programs (COBOL source programs, library text, and module definition statements) can be modified by using an editor.

To open the editor, double-click on a file name displayed in the Project window.

Refer to “Using the P-STAFF Editor” in Chapter 3.

Executing Application Programs

To execute created application programs from the Project window:

1. Select an application to execute. When multiple applications are created, select the appropriate application from the project window.
2. When a run-time parameter is defined, select Argument at Execution from the Project menu in the P-STAFF window, then enter the parameter in the edit box.
3. Click on the Execute button. If the program is a COBOL application, the Run-time Environment Setup window may be displayed. Set the required information and start execution. Refer to “Run-time Environment Setup Window” in Chapter 5.

Chapter 7. File Processing

This chapter explains how to read and write data files. Among the topics covered in Chapter 7 are file organization types, using record and line sequential files, using relative and indexed files, error processing, file processing, and the COBOL85 FILE UTILITY.

File Organization Types

This section explains file organization types and characteristics, and details how to design records and process files.

File Organization Types and Characteristics

The following files can be processed by using sequential, relative, and indexed file functions:

Table 15. File types

Sequential file function	Record sequential files Line sequential files Print files
Relative file function	Relative files
Indexed file function	Indexed files

The following table lists the characteristics of each file.

Table 16. File organization types and characteristics

File Types	**Record-Sequential File**	**Line-Sequential File**	**Print File**	**Relative File**	**Indexed File**
Record processing	Record storage sequence			Relative record number	Record key value
Usable data medium	Hard disk Floppy disk	Hard disk Floppy disk	Printer	Hard disk Floppy disk	Hard disk Floppy disk
Usage example	Saving data Work file	Text file	Printing data	Work file	Master file

File organization is determined when a file is generated, and cannot be changed. Before generating a file, therefore, fully understand the characteristics of the file and be careful to select the file organization that most agrees with its use. Each file organization is explained below.

Record Sequential Files

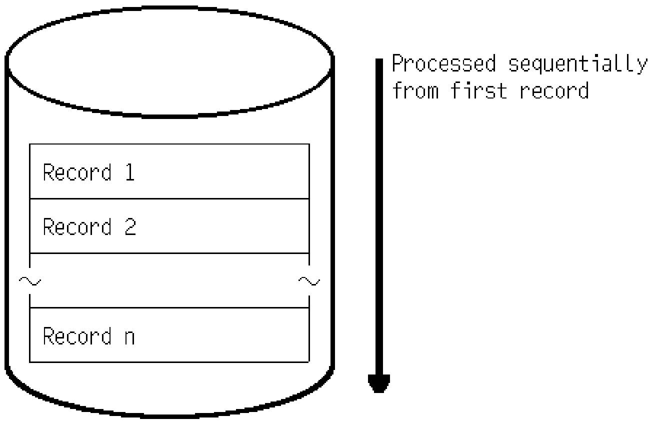


Figure 63. Record sequential files

In a record sequential file, records are read and updated in order, starting with the first record in the file.

A record sequential file is the easiest to create and maintain. Record sequential files are generally used to store data sequentially and maintain a large amount of data.

Line Sequential Files

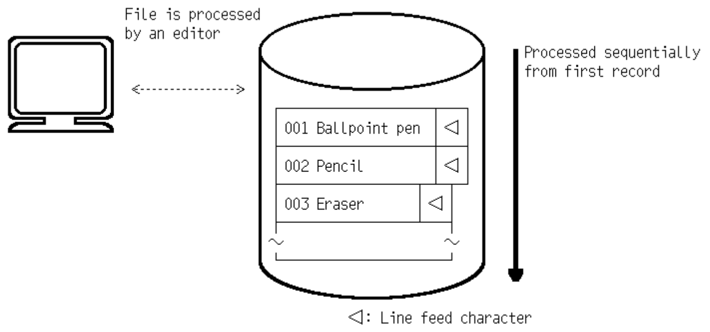


Figure 64. Line sequential files

Records can also be read from the first record in the physical order they are placed in a line sequential file.

Line feed characters are used as record delimiters in a line sequential file.

For example, a line sequential file can be used when handling a file created by an editor.

Print Files

A print file is used for printing with the sequential file function; there is no special file name for a print file. For details of a print file, refer to Chapter 8, “Printing.”

Relative Files

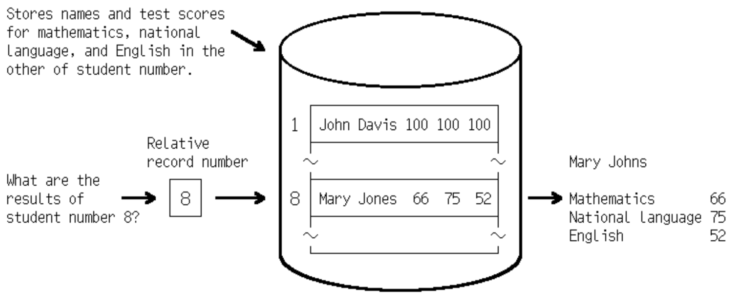

Stores names and test scores for mathematics, national language, and English in the other of student number.
What are the results of student number 8?
Relative record number
8
1
John Davis 100 100 100
8
Mary Jones 66 75 52
Mary Johns
Mathematics 66
National language 75
English 52

Figure 65. Relative files

Records can be read and updated by specifying their relative record numbers (the first record is addressed by relative record number 1) in a relative file.

For example, a relative file can be used as a work file accessed by using relative record numbers as key values.

Indexed Files

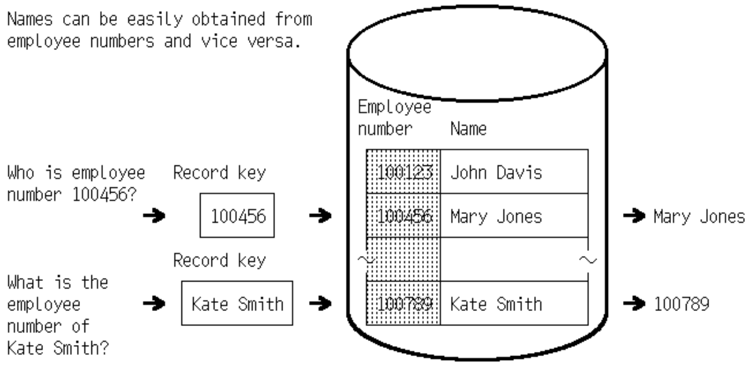

Names can be easily obtained from employee numbers and vice versa.
Employee number
Name
Who is employee number 100456?
Record key
100456
100123
John Davis
100456
Mary Jones
Mary Jones
Record key
What is the employee number of Kate Smith?
Kate Smith
100789
Kate Smith
100789

Figure 66. Indexed files

Records can be read and updated by specifying the values of items (record keys) in an indexed file. Use an indexed file as a master file when retrieving information associated with the values of items in a record.

Designing Records

This section explains the types and characteristics of record formats and the record keys of indexed files.

Record Formats

There are two types of record formats, fixed length and variable length.

- Fixed length record

  In a fixed length record, all records in a file contain the same number of character positions.

- Variable length record

  In a variable length record, records contain a varying number of character positions. The record size is determined when a record is placed in a file. Because each record can be written with the required size, the variable length record format is useful if you want to minimize the file capacity.

Record Keys of Indexed Files

When designing a record in an indexed file, a record key must be determined. Multiple record keys can be specified for an item in the record.

There are primary record keys (primary keys) and alternate record keys (alternate keys). Records in the file are stored in the ascending order of primary record keys.

Specifying a primary or alternate key determines which record is processed in the file. In addition, records can be processed in ascending order, starting from any record.

To process the same file with multiple record organization, primary keys must be at the same position and of the same size in each record organization.

In the variable length record format, the record key must be set at a fixed position.

Processing Files

There are six types of file processing:

- File creation: Writes records to files.
- File extension: Writes records after the last record in a file.
- Record insertion: Writes records at arbitrary positions in files.
- Record reference: Reads records from files.
- Record updating: Rewrites records in files.
- Record deletion: Deletes records from files.

File processing depends on the file access mode. There are three types of access modes:

- Sequential access: Allows serial records to be processed in a fixed order.
- Random access: Allows arbitrary records to be processed.
- Dynamic access: Allows records to be processed in sequential and random access modes.

The following table indicates the processing available with each file type.

Table 17. File organization types and processing

File Types	Access Mode	Processing					
		Creation	**Extension**	**Insertion**	**Reference**	**Updating**	**Deletion**
Record sequential file	Sequential	o	o	x	o	o	x
Line sequential file	Sequential	o	o	x	o	x	x
Print file	Sequential	o	o	x	x	x	x
Relative file	Sequential	o	o	x	o	o	o
Indexed file	Random	o	x	o	o	o	o
	Dynamic	o	x	o	o	o	o

o: Can be processed. x: Cannot be processed.

Disable access by locking records in use. This is called exclusive control of files. For details of exclusive control of files, see "Exclusive Control of Files."

Using Record Sequential Files

This section explains how to define record sequential files, and process and define records.

```
IDENTIFICATION DIVISION.
PROGRAM-ID program.name.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
 FILE CONTROL.
  SELECT file-name
     ASSIGN TO file-reference identifier
     [ORGANIZATION IS SEQUENTIAL].
DATA DIVISION.
 FILE SECTION.
 FD file-name
   [RECORD record-size].
 01 record-name.
 ┌────────────────────────────┐
 │ Contents of records        │
 │ 02 data-name ... .         │
 │ :                          │
 └────────────────────────────┘
PROCEDURE DIVISION.
  OPEN open-mode file-name.
  [READ file-name.]
  [REWRITE record-name.]
  [WRITE record-name.]
  CLOSE file-name.
END PROGRAM program-name.
```

Defining Record Sequential Files

This section explains the file definitions required to use record sequential files in a COBOL program.

File Name and File-Reference-Identifier

Determine file names in conformance to the rules of COBOL user-defined words, then write them in the SELECT clause.

Determine a file-reference-identifier, then write them in the ASSIGN clause. For the file-reference-identifier, specify a file-identifier, file-identifier-literal, data-name, or character string DISK.

Use the file-reference-identifier to associate the file name specified in the SELECT clause with the file of the actual input-output medium.

How you associate the file name in the SELECT clause to the file of the actual input-output medium depends on what is specified for the file-reference-identifier.

The recommended method to determine what to specify for the file-reference-identifier is:

- When the file name of the input-output medium is determined at COBOL program generation and is not subsequently changed, specify a file-identifier-literal or character string DISK.
- When the file name of the actual input-output medium is undetermined at COBOL program generation, or to determine the file name at every program execution, specify a file identifier.

- To determine the file name in a program, specify a data name.
- If the file is temporary and will be unnecessary after the program terminates, specify character string DISK.

Files can be processed at high speed in record sequential or line sequential files. For the specification method, refer to Appendix H, "High-speed File Processing."

The following table lists description examples of SELECT and ASSIGN clauses.

Table 18. Description examples of SELECT and ASSIGN clauses

Type of File-Reference-Name	**Description Example**	**Remarks**
File-identifier	SELECT file-1 ASSIGN TO INFILE	Must be related to the actual input-output medium at program execution.
Data-name	SELECT file-2 ASSIGN TO data name	The data name must be defined in the WORKING-STORAGE section in the DATA DIVISION
File-identifier literal	SELECT file-3 ASSIGN TO "c.dat"	
DISK	SELECT data1 ASSIGN TO DISK	The file name cannot be specified with an absolute path name

File Organization

Specify SEQUENTIAL in the ORGANIZATION clause. If the ORGANIZATION clause is omitted, SEQUENTIAL is used as the default.

Defining Record Sequential File Records

This section explains record formats, lengths, and organization.

Record Formats and Lengths

Record sequential files are either fixed or variable length.

The record length in the fixed length record format is the value specified in the RECORD clause, or the maximum value in the record description entries if the RECORD clause is omitted.

In the variable length record format, the length when a record is written is the length of the record. The length of an output record can be set for the data name written in DEPENDING ON data-name in the RECORD clause.

You can obtain the record length when entering a record by using this data name.

Record Organization

Define the attribute, position, and size of data in a record in record description entries.

The following are examples of record definitions:

[A fixed length record]

data-1	data-2
100 bytes	100 bytes

```
FD data-storage-file.
01 data-record.
    02 data-1 PIC X(100).
    02 data-2 PIC X(100).
```

[A variable length record]

data-1	data-2
100 bytes	1 to 100 bytes

```
FD     data-storage-file
       RECORD IS VARYING IN SIZE
       FROM 101 TO 200 CHARACTERS
       DEPENDING ON record-length.
01     data-record.
       02 data-1 PIC X(100).
       02 data-2.
            03     PIC X
                       OCCURS 1 to 100 TIMES
                       DEPENDING ON length-1.
                       :
WORKING-STORAGE SECTION.
01    record-length     PIC 9  (3)  BINARY.
01    length-1          PIC 9  (3)  BINARY.
```

Processing Record Sequential Files

Use input-output statements to create, extend, reference, and update record sequential files. This section explains the types of input-output statements used in record sequential file processing, and outlines each type of processing.

Types of Input-Output Statements

OPEN, CLOSE, READ, REWRITE, and WRITE statements are used for input and output.

Using Input-Output Statements

The section provides explanation and examples of how to use input-output statements.

OPEN and CLOSE Statements

Execute an OPEN statement only once, at the beginning of file processing, and a CLOSE statement only once at the end of file processing.

The open mode specified in the OPEN statement depends on the kind of file processing. For example, for creation, use the OPEN statement in the OUTPUT mode (OUTPUT specified).

Other Statements

To read records in a file, use a READ statement. To update records in a file, use a REWRITE statement. To write records to a file, use a WRITE statement.

Execution Order of Input-Output Statements

The execution order of input-output statements for creating, extending, referencing, and updating are as follows:

For file creation:

```
OPEN OUTPUT file-name.
Editing-records
WRITE record-name ... .
CLOSE file-name.
```

For file extension:

```
OPEN EXTEND file-name.
Editing-records
WRITE record-name ... .
CLOSE file-name.
```

For record reference:

```
OPEN INPUT file-name.
READ file-name ... .
CLOSE file-name.
```

For record updating:

```
OPEN I-O file-name.
READ file-name... .
Editing-records
REWRITE record-name ... .
CLOSE file-name.
```

Processing Outline

Creating

To generate a record sequential file, open a file in OUTPUT mode, then write records to the file with a WRITE statement.

If an attempt was made to generate a file that already exists, the file is regenerated and the original contents are lost.

Extending

To extend a record sequential file, open a file in EXTEND mode, then write records to the file with a WRITE statement. Records are added to the end of the last record in the file.

Referencing

To refer to records, open the file in INPUT mode, then read records in the file from the first record with a READ statement.

When OPTIONAL is specified in the SELECT clause in the file control entry, the OPEN statement is successful even if the file does not exist during execution of the OPEN statement, and the at end condition is satisfied upon execution of the first READ statement.

For information on the at end condition, see “Input-Output Error Processing.”

Updating

To update records, open the file in I-O mode, read the records from the file by using a READ statement, then rewrite them by using a REWRITE statement.

When a REWRITE statement is executed, records read by the last READ statement are updated.

The record length cannot be changed even in the variable-length record format.

Using Line Sequential Files

This section explains how to define files and records, and process files.

```
IDENTIFICATION DIVISION.
PROGRAM-ID program.name.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
  FILE CONTROL.
    SELECT file-name
       ASSIGN TO file-reference identifier
       [ORGANIZATION IS LINE SEQUENTIAL].
DATA DIVISION.
  FILE SECTION.
  FD file-name
    [RECORD record-size].
  01 record-name.
   Contents of records
   02 data-name ... .
   :
PROCEDURE DIVISION.
   OPEN open-mode file-name.
   [READ file-name.]
   [WRITE record-name.]
   CLOSE file-name.
END PROGRAM program-name.
```

Defining Line Sequential Files

This section explains the file definitions required to use line sequential files in a COBOL program.

File Name and File-Reference-Identifier

As with a record sequential file, specify a file name and file-reference-identifier for a line sequential file. For more information about specifying these items, see “Defining Record Sequential Files.”

File Organization

Specify LINE SEQUENTIAL in the ORGANIZATION clause.

Defining Line Sequential File Records

This section explains record formats, lengths, and organization.

Record Formats and Lengths

The explanations of the record formats and lengths of line sequential files are the same as for record sequential files. See “Defining Record Sequential File Records” for more information.

Because a record is delimited by a line feed character in a line sequential file, the end of a record is always a line feed character regardless of the record format. A line feed character, however, is not included in the record length.

Record Organization

Define the attribute, position, and size of record data in the record description entries. You do not have to define line feed characters in the record description entries because they are added when records are written.

The following is an example of record definitions in the variable length records format:

Text character-string ◁	◁ : Line feed character

A text character string can be up to 80 alphanumeric characters.

```
FD            text-file
              RECORD IS VARYING IN SIZE FROM 1 TO 80 CHARACTERS
                           DEPENDING ON record-length.
01            text-record.
              02 text-character-string.
                  03 character-data PIC X OCCURS 1 TO 80 TIMES
                            DEPENDING ON record-length.
              :
              :
WORKING-STORAGE SECTION.
01            record-length PIC 9(3) BINARY.
```

Processing Line Sequential Files

In a line sequential file processing, creation, extension, and reference can be done with input-output statements. This section explains the types and use of input-output statements in line sequential file processing, and outlines the processing.

Using Input-Output Statements

OPEN, CLOSE, READ, and WRITE statements are used in line sequential file processing.

OPEN and CLOSE Statements

Execute an OPEN statement only once at the beginning of file processing, and a CLOSE statement only once at the end of file processing.

The open mode specified in the OPEN statement depends on the kind of file processing. For example, for creation, use the OPEN statement in the OUTPUT mode (OUTPUT specified).

Other Statements

To read records in a file, use a READ statement. To write records to a file, use a WRITE statement.

Execution Order of Input-Output Statements

The execution order of input-output statements for creating, referencing, and extending are as follows.

For file creation:

```
OPEN OUTPUT file-name.
Editing-records
WRITE record-name ... .
CLOSE file-name.
```

For file extension:

```
OPEN EXTEND file-name.
Editing-records
WRITE record-name ... .
CLOSE file-name.
```

For record reference:

```
OPEN INPUT file-name.
READ file name ... .
CLOSE file-name.
```

Processing Outline

Creating

To create a line sequential file, open a file in OUTPUT mode, then write records to the file with a WRITE statement.

If an attempt was made to create a file that already exists, the file is recreated and the original file are lost.

The contents of the record area and line feed character are written at record write.

Extending

To extend a line sequential file, open a file in EXTEND mode, then write records sequentially to the file with a WRITE statement. Records are added to the end of the last record in the file.

Referencing

To refer to records, open the file in INPUT mode, then read records in the file from the first record by using a READ statement.

When OPTIONAL is specified in the SELECT clause in the file control entry, the OPEN statement is successful even if the file does not exist, and the at end condition is satisfied upon execution of the first READ statement. For information on the at end condition, see “Input-Output Error Processing.”

If the size of a read record is greater than the record length, data with the same length as the record length is set in the record area when a READ statement is executed. The continuation of the data of the same record is then set in the record area when the next READ statement is executed. If the size of data to be set is smaller than the record length, spaces are written to the remaining portions of the record area.

Using Relative Files

This section explains how to define and process files, and how to define records for relative files.

```
IDENTIFICATION DIVISION.
PROGRAM-ID program.name.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
  FILE CONTROL.
    SELECT file-name
       ASSIGN TO file-reference identifier
       ORGANIZATION IS RELATIVE
      [ACCESS MODE IS access mode]
      [RELATIVE KEY IS relative-record-number-storage-area].
DATA DIVISION.
  FILE SECTION.
  FD file-name
    [RECORD record-size].
  01 record-name.
   +------------------------------+
   | Contents of records          |
   | 02 data-name ... .           |
   | :                            |
   +------------------------------+
WORKING-STORAGE SECTION.
[01 relative-record-number-storage-area PIC 9(5) BINARY].
PROCEDURE DIVISION.
   OPEN open-mode file-name.
   [MOVE relative-record-number TO relative-record-number-storage-area.]
   [READ file-name.]
   [REWRITE record-name.]
   [DELETE file-name.]
   [WRITE record-name.]
   CLOSE file-name.
END PROGRAM program-name.
```

Defining Relative Files

This section explains file definitions required to use relative files in a COBOL program.

File Name and File-Reference-Identifier

As with a record sequential file, specify a file name and file-reference-identifier for a relative file. For more information about how to specify these items, see “Defining Record Sequential Files.”

File Organization

Specify RELATIVE in the ORGANIZATION clause.

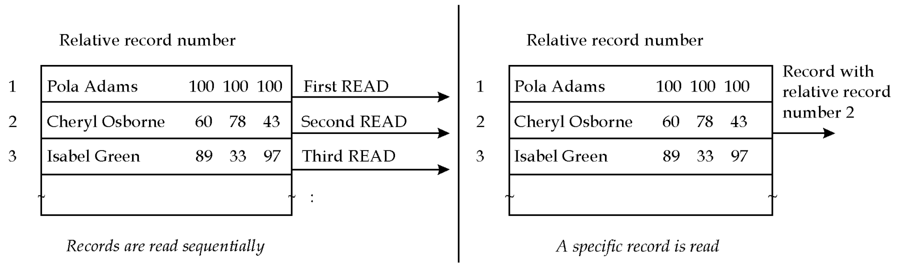
Access Mode

Specify one of the following access modes in the ACCESS MODE clause:

- Sequential access (SEQUENTIAL): Enables records to be processed in ascending order of the relative record numbers from the beginning of the file or a record with a specific relative record number.
- Random access (RANDOM): Enables a record with a specific relative record number to be processed.
- Dynamic access (DYNAMIC): Enables records to be processed in sequential and random access modes.

Referencing records in sequential and random access modes is shown below.

[Sequential access] [Random access]



Relative Record Numbers

Specify the data name where a relative record number is entered in the RELATIVE KEY clause. In sequential access, this clause can be omitted.

The relative record number of the record for this data name is entered when the record is read, and the relative record number of the record to be written is entered when the record is written.

This data name must be defined as an unsigned numeric item in the WORKING-STORAGE section.

Defining Relative File Records

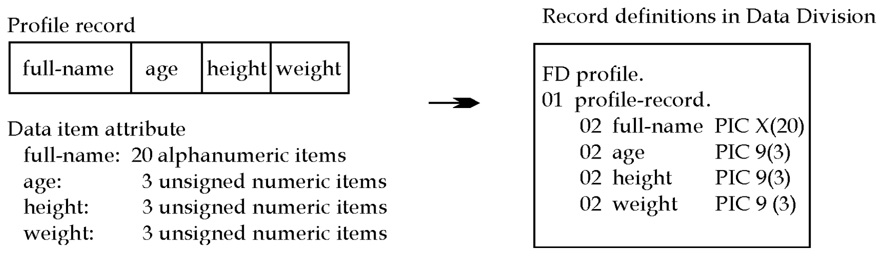
This section explains record formats, lengths, and organization.

Record Formats and Lengths

The explanations of the record formats and lengths of relative files are the same as for record sequential files. See “Defining Record Sequential File Records” for more information.

Record Organization

Define the attribute, position, and size of record data in the record description entries. You do not have to define the area to set the relative record number.



Data item attribute

full-name:	20 alphanumeric items
age:	3 unsigned numeric items
height:	3 unsigned numeric items
weight:	3 unsigned numeric items

```
FD profile.
01  profile-record.
     02  full-name  PIC X(20)
     02  age        PIC 9(3)
     02  height     PIC 9(3)
     02  weight     PIC 9 (3)
```

Processing Relative Files

In relative file processing, creation, extension, insertion, reference, updating, and deletion is done with input-output statements. This section explains the types of input-output statements used in relative file processing, and outlines each type of processing.

Using Input-Output Statements

OPEN, CLOSE, DELETE, READ, START, REWRITE, and WRITE statements are used for input and output in relative file processing.

OPEN and CLOSE Statements

Execute an OPEN statement only once at the beginning of file processing and a CLOSE statement only once at the end of file processing.

The open mode specified in the OPEN statement depends on the kind of file processing. For example, for creation, use the OPEN statement in the OUTPUT mode (OUTPUT specified).

Other Statements

To delete records from a file, use a DELETE statement. To read records in a file, use a READ statement. To indicate a record for which processing is to be started, use a START statement.

To update records in a file, use a REWRITE statement. To write records to a file, use a WRITE statement.

Execution Order of Input-Output Statements

File creation

```
OPEN OUPUT file-name.
Editing records
[Setting relative record-numbers]
WRITE record-name... .
CLOSE file-name.
```

Sequential access

File extension

```
OPEN EXTEND file-name.
Editing records
WRITE record-name... .
CLOSE file-name.
```

File reference

```
OPEN INPUT  file-name.
Editing records
[Setting relative record numbers]
START file-name ... .
READ file-name [NEXT] ... .
CLOSE file-name.
```

File updating

```
OPEN I-O  file-name.
[Setting relative record numbers]
START file-name ... .
READ file-name [NEXT] ... .
Editing records
REWRITE  file-name [NEXT] ... .
CLOSE file-name.
```

File deletion

```
OPEN I-O  file-name.
[Setting relative record numbers]
START file-name ... .
READ file-name [NEXT] ... .
Editing records
DELETE  file-name [NEXT] ... .
CLOSE file-name.
```

Random access

File insertion

```
OPEN I-O file-name.
Editing records
Setting relative record numbers ... .
WRITE record-name ... .
CLOSE file-name.
```

File reference

```
OPEN INPUT  file-name.
Setting relative record numbers
READ file-name... .
CLOSE file-name.
```

File updating

```
OPEN I-O  file-name.
Setting relative record numbers
[READ file-name ... .]
Editing records
REWRITE  record-name [NEXT] ... .
CLOSE file-name.
```

File deletion

```
OPEN I-O  file-name.
Setting relative record numbers
DELETE  file-name ... .
CLOSE file-name.
```

Enter the relative record number for the data-name specified in the RELATIVE KEY clause, for example, MOVE 1 to data-name.

The execution orders of input-output statements for creation, extension, insertion, reference, updating, and deletion are as follows:

Processing Outline

Creating (Sequential, Random, and Dynamic)

To create a relative file, open a file in OUTPUT mode, then write records to the file with a WRITE statement. Records are written with the record length specified in the WRITE statement.

In sequential access, relative record numbers 1, 2, 3, ... are set in the order the records are written.

In random or dynamic access, records are written at the positions specified by the relative record numbers.

If an attempt is made to create a file that already exists, the file is recreated and the original file is lost.

Extending (Sequential)

To extend a relative file, open a file in EXTEND mode, then write records sequentially with a WRITE statement. When writing to files in this mode, records are added to the end of the last record in the file.

The relative record number of the record to be written is incremented by one from the maximum relative record number in the file.

The file can be extended only in sequential access mode.

Referencing (Sequential, Random, and Dynamic)

To refer to records, open the file in INPUT mode, then read records with a READ statement.

In sequential access, specify the start position of the record to be read with a START statement, then read records sequentially from the specified record in order of the relative record numbers.

In random access, the record with the relative record number specified at execution of the READ statement is read.

When OPTIONAL is specified in the SELECT clause in the file control entry, the OPEN statement is successful even if the file does not exist. The at end condition is satisfied with the execution of the first READ statement. For information on the at end condition, see “Input-Output Error Processing.”

If the record with the specified relative record number is not found in random or dynamic access, the invalid key condition is satisfied. For information on the invalid key condition, see “Input-Output Error Processing.”

Updating (Sequential, Random, and Dynamic)

To update records, open the file in I-O mode.

In sequential access, read records with a READ statement, then update them with a REWRITE statement. Executing the REWRITE statement updates the records read by the last READ statement.

In random access, specify the relative record number of the record to be updated, then execute the REWRITE statement.

If the record with the specified relative record number is not found in random or dynamic access, the invalid key condition is satisfied. For information on the invalid key condition, see “Input-Output Error Processing.”

The record length cannot be changed, even in the variable-length record format.

Deleting (Sequential, Random, and Dynamic)

To delete records, open the file in I-O mode.

In sequential access, read records with a READ statement, then delete them with a DELETE statement. Executing the DELETE statement deletes the records read by the last READ statement.

In random access, specify the relative record number of the record to be deleted, then execute the DELETE statement.

If the record with the specified relative record number is not found in random or dynamic access, the invalid key condition is satisfied. For information on the invalid key condition, see “Input-Output Error Processing.”

Inserting (Random and Dynamic)

To insert records, open the file in I-O mode.

Specify the relative record number of the insertion position, then execute a WRITE statement. The record is inserted at the position of the specified relative record number.

If the record with the specified relative record number already exists, the invalid key condition is satisfied. For information about the invalid key condition, see “Input-Output Error Processing.”

Using Indexed Files

This section explains how to define and process files, and define records for indexed files.

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. program-name.
ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
  FILE-CONTROL.
    SELECT file-name
      ASSIGN TO file-reference-identifier
      ORGANIZATION IS INDEXED
     [ACCESS MODE IS access-mode]
      RECORD KEY IS prime-key-name-1 [prime-key-name-n]...
          [WITH DUPLICATES]
   [ [ALTERNATE RECORD KEY IS alternate-record-key-name-1
      [alternate-key-name-n]... [WITH DUPLICATES]] ...].

DATA DIVISION.
 FILE SECTION.
 FD file-name.
       [RECORD record-size].
 01 record-name.

    Contents of records
     02 prime-key-name-1 ... .
    [02 prime-key-name-n ... .]
    [02 alternate-key-name-1 ... .]
    [02 alternate-key-name-n ... .]
    [02 data-other-than-keys ... .]

PROCEDURE DIVISION.
           OPEN open-mode file-name.
           [MOVE prime-key-value TO  prime-key-name-n.]
           [MOVE alternate-key-value TO alternate-key-name-n.]
           [READ file-name.]
           [REWRITE record-name.]
           [DELETE file-name.]
           [START file-name.]
           [WRITE record-name.]
           CLOSE file-name.
END PROGRAM program-name.
```

Defining Indexed Files

This section explains the file definitions required to use indexed files in a COBOL program.

File Name and File-Reference-Identifier

As with a record sequential file, specify a file name and file-reference-identifier for an indexed file. For more information about how to specify these items, see “Defining Record Sequential Files.”

File Organization

Specify INDEXED in the ORGANIZATION clause.

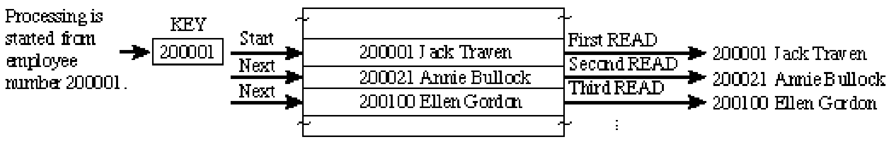
Access Mode

Specify one of the following access modes in the ACCESS MODE clause:

- Sequential access (SEQUENTIAL): Enables records to be processed in ascending key order from the beginning of the file or a record with a specific key.
- Random access (RANDOM): Enables a record with a specific key to be processed.
- Dynamic access (DYNAMIC): Enables records to be processed in sequential and random access modes.

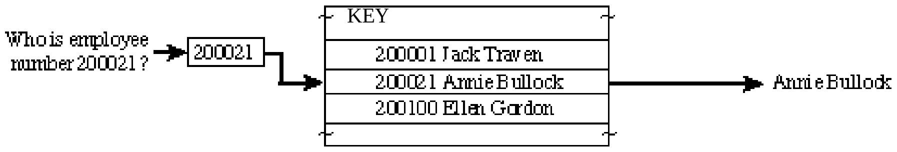
Referencing records in sequential and random access modes is shown in the following examples.

Sequential Access



Data is read sequentially from employee number 200001.

Random Access



Data corresponding to employee number 200021 is fetched.

Primary and Alternate Keys

Keys are classified into primary record keys (primary keys) and alternate record keys (alternate keys). The number of keys, positions in records, and sizes are determined during file creation, and cannot be changed once determined.

Records in the file are in logical ascending order of the primary keys, and specific records can be selected with the primary key values. When defining an indexed file, specify the name of a data item as the primary key in the RECORD KEY clause.

As with the primary key, an alternate key is the information used to select specific records in the file. Specify the name of a data item to use as the alternate key in the ALTERNATE RECORD KEY clause as required.

Multiple data items can be specified in the RECORD KEY and ALTERNATE RECORD KEY clauses. When multiple data items are specified in the RECORD KEY clause, the primary key consists of these data items. Data items specified in the RECORD KEY clause need not be contiguous.

Multiple records can have the same key value (duplicate key value) by specifying DUPLICATES in the RECORD KEY and ALTERNATE RECORD KEY clauses. An error occurs if the key value is duplicated when DUPLICATES is not specified.

Defining Indexed File Records

This section explains record formats, lengths, and organization.

Record Formats and Lengths

The explanations of the record formats and lengths of indexed files are the same as for record sequential files. See “Defining Record Sequential File Records.”

Record Organization

Define the attributes, positions, and sizes of keys and data other than keys in records in the record description entries.

To process an existing file, the number of items, item positions, and sizes to be specified for the primary or alternate keys must be equal to those defined at file creation. The specification order and the number of primary keys must be the same as for the alternate keys.

When writing two or more record description entries for a file, write the primary key data item in only one of these record description entries. The character position and size defining the primary key are applied to other record description entries.

In the variable length records format, the key must be set at the same fixed part (position from the beginning of the record is always the same).

The following is an example of record definitions in the variable length records format:

Primary key	Alternate key	
Employee number	Full-name	Section
6-digit number	20 characters	Up to 32 characters

```
    :
  RECORD KEY IS employee-number
  ALTERNATE RECORD KEY IS full-name.
    :
FD employee-file
  RECORD IS VARYING IN SIZE FROM 27 TO 58 CHARACTERS
                   DEPENDING ON record-length.
    :
01 employee-record.
  02 employee-number  PIC 9(6).
  02 full-name        PIC X(20).
  02 section.
    03 PIC X OCCURS 1 TO 32 TIMES
                  DEPENDING ON section-length.
    :
WORKING-STORAGE SECTION.
01 record-length  PIC 9(3) BINARY.
01 section-length PIC 9(3) BINARY.
```

Processing Indexed Files

Use input-output statements to create, extend, insert, reference, update, and delete indexed files. Some processing may not be used, depending on the access mode.

This section explains the types and input-output statements used in indexed file processing, and outlines each type of processing.

Using Input-Output Statements

OPEN, CLOSE, DELETE, READ, START, REWRITE, and WRITE statements are used to process indexed files.

OPEN and CLOSE Statements

Execute an OPEN statement only once at the beginning of file processing and a CLOSE statement only once at the end of file processing.

The open mode specified in the OPEN statement depends on the kind of file processing. For example, for creation, use the OPEN statement in the OUTPUT mode (OUTPUT specified).

Other Statements

To delete records from a file, use a DELETE statement. To read records in a file, use a READ statement. To indicate a record for which processing is to be started, use a START statement.

To update records in a file, use a REWRITE statement. To write records to a file, use a WRITE statement.

Execution Order of Input-Output Statements

The execution order of input-output statements for creating, extending, referencing, updating, deleting, and inserting are as follows:

Procedure for file creation

```
OPEN OUTPUT file-name.
Editing records
MOVE prime-key-value TO
prime-key-name.
WRITE record-name ... .
CLOSE file-name.
```

Procedure for file reference (random access)

```
OPEN INPUT file-name.
MOVE key-value TO key-name.
READ file-name ... .
CLOSE file-name.
```

Procedure for file updating (random access)

```
OPEN I-O file-name.
MOVE key-value TO key-name.
[READ file-name ... .]
Editing records
REWRITE record-name ... .
CLOSE file-name.
```

Procedure for file deletion (random access)

```
OPEN I-O file-name.
MOVE key-value TO key-name.
DELETE file-name ... .
CLOSE file-name.
```

Procedure for file insertion (random access)

```
OPEN I-O file-name.
Editing records
MOVE key-value TO key-name.
WRITE record-name ... .
CLOSE file-name.
```

Procedure for file extension (sequential access)

```
OPEN EXTEND file-name.
Editing records
WRITE record-name ... .
CLOSE file-name.
```

Procedure for file reference (sequential access)

```
OPEN INPUT file-name.
MOVE key-value TO key-name.
START file-name ... .
READ file-name [NEXT] ... .
CLOSE file-name.
```

Procedure for file updating (sequential access)

```
OPEN I-O file-name.
MOVE key-value TO key-name.
START file-name ... .
READ file-name [NEXT] ... .
Editing records
REWRITE record-name ... .
CLOSE file-name.
```

Procedure for file deletion (sequential access)

```
OPEN I-O file-name.
MOVE key-value TO key-name.
START file-name ... .
READ file-name [NEXT] ... .
DELETE record-name ... .
CLOSE file-name.
```

Processing Outline

Creating (Sequential, Random, and Dynamic)

To create an indexed file, open a file in the OUTPUT mode, and write records to the file with a WRITE statement.

If an attempt is made to create a file that already exists, the file is recreated and the original file is lost.

Before writing a record, the primary key value must be set. In sequential access, records must be written so that the primary keys are in ascending order.

The following tasks are useful for collecting data to create index files:

- Create data with the editor, read the data with the line sequential file function, then write it to an indexed file, as shown in the following example.

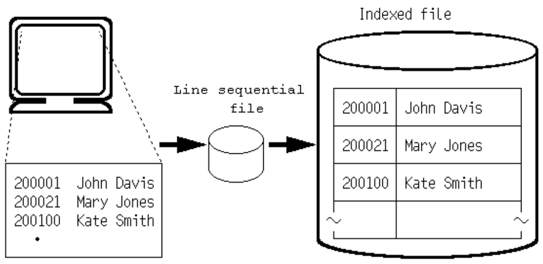


Figure 67. Writing data to an indexed file

- With the screen handling functions (presentation file and screen operation functions), enter data from the screen, then write it to an indexed file, as shown in the following example.

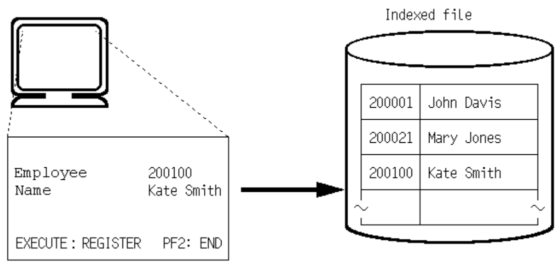


Figure 68. Writing data from the screen to an indexed file

Extending (Sequential)

To extend an indexed file, open the file in EXTEND mode, then write records sequentially. At this time, records are added to the end of the last record in the file.

The file can be extended only in sequential access mode.

To edit the records to be written, the primary key values must be in ascending order.

When DUPLICATES is not specified in the RECORD KEY clause in the file control entry, the primary key value first processed must be greater than the maximum primary key value in the file.

When DUPLICATES is specified, the primary key value first processed must be equal to or greater than the maximum primary key value in the file.

Referencing (Sequential, Random, and Dynamic)

To refer to records, open the file in INPUT mode, then read records with a READ statement.

In sequential access, specify the start position of the record to be read with a START statement. Then, start reading records from the specified position in ascending order of the primary or alternate key values.

In random access, the records to be read are determined by the primary or alternate key values.

When OPTIONAL is specified in the SELECT clause in the file control entry, the OPEN statement is successful even if the file does not exist. The at end condition is satisfied with the execution of the first READ statement.

For information about the at end condition, see “Input-Output Error Processing.”

If the record with the specified key value is not found in RANDOM or DYNAMIC access, the invalid key condition is satisfied. For information on the invalid key condition, see “Input-Output Error Processing.”

A START or READ statement can be executed by specifying multiple keys.

Updating (Sequential, Random, and Dynamic)

To update records, open the file in I-O mode, then rewrite records in the file.

In sequential access, records read with the last READ statement are updated.

In random access, the primary key records with the specified key values are updated.

If the record with the specified key value is not found in RANDOM or DYNAMIC access, the invalid key condition is satisfied. For information on the invalid key condition, see “Input-Output Error Processing.”

The contents of primary keys cannot be changed. The contents of alternate keys, however, can be changed.

Deleting (Sequential, Random, and Dynamic)

To delete records, open the file in I-O mode, then delete records from the file.

In sequential access, records read with the last READ statement are deleted.

In random access, the primary key records with the specified key values are deleted.

If the record with the specified key value is not found in random or dynamic access, the invalid key condition is satisfied. For information about the invalid key condition, see “Input-Output Error Processing.”

Inserting (Random and Dynamic)

To insert records, open the file in I-O mode, then insert records in the file. The record insertion position is determined by the primary key value.

When DUPLICATES is not specified in the RECORD KEY or ALTERNATE RECORD KEY clause in the file control entry, and the record with the specified key value already exists, the invalid key condition is satisfied.

For information about the invalid key condition, see “Input-Output Error Processing.”

Input-Output Error Processing

This section explains input-output error detection methods and execution results if input-output errors occur.

The four input-output error detection methods are:

- AT END specification (detecting occurrence of the at end condition)
- INVALID KEY specification (detecting occurrence of the invalid key condition)
- FILE STATUS clause (detecting input-output errors by checking I-O status)
- Error procedures (detecting input-output errors)

AT END Specification

The AT END condition occurs when all records in a file are read sequentially and no next logical record exists in the file. To detect occurrence of the at end condition, specify AT END in a READ statement. With AT END specified, processing to be done at occurrence of the at end condition can be written.

A coding example of a READ statement with AT END specified is:

```
READ sequential-file AT END
          GO TO at-end-processing
END-READ.
```

With the execution of the next READ statement after the last record in the file is read, the at end condition occurs and the GO TO statement is executed.

INVALID KEY Specification

In input-output processing with an indexed or relative file, an input-output error occurs if no record with a specified key or relative record number exists in the file. This is called an invalid key condition.

To detect occurrence of the invalid key condition, specify INVALID KEY in the READ, WRITE, REWRITE, START, and DELETE statements. With INVALID KEY specified, processing to be done upon the occurrence of an invalid key condition can be written.

A coding example of a READ statement with INVALID KEY specification is:

```
MOVE "Z" TO primary-key.
READ indexed-file INVALID KEY
     GO TO invalid-key-processing
END-READ.
```

If no record with primary key value "Z" is present in the file, the invalid key condition occurs and the GO TO statement is executed.

FILE STATUS Clause

When a FILE STATUS clause is written in the file control entry, the I-O status is posted to the data-name specified in the FILE STATUS clause upon execution of the input-output statement. By writing a statement (IF or EVALUATE statement) to check the contents (I-O status value) of this data name, input-output errors can be detected.

If no I-O status value is checked after the input-output statement, program execution continues even if an input-output error occurs. Subsequent operation is undefined.

For I-O status values to be posted, refer to Appendix B, “I-O Status List.”

A coding example of a FILE STATUS clause is:

```
  SELECT file-1.
      FILE STATUS IS input-output-status
      :
      :
WORKING-STORAGE SECTION.
 01 input-output-status PIC X(2).
      :
      :
  OPEN file-1.
  IF input-output-status NOT = "00"
      THEN GO TO open-error-processing.
```

If a file could not be opened, a value other than "00" is set for the input-output value. The IF statement checks this value, and the GO TO statement is executed.

Error Procedures

You specify error procedures by writing a USE AFTER ERROR/EXCEPTION statement in Declaratives in the PROCEDURE DIVISION.

Writing error procedures executes the processing written in the error procedures if an input-output error occurs. After executing error processing, control is passed to the statement immediately after the input-output statement with which an input-output error occurred. Thus, a statement indicating control of processing of the file where an input-output error occurred must be written immediately after the input-output statement.

Control is not passed to the error procedures in the following cases:

- AT END is specified in the READ statement with which the at end condition occurred

- INVALID KEY is specified in the input-output statement with which the invalid key condition occurred
- An input-output statement is executed before the file is opened (the open mode is specified)

Branch from error procedures to the PROCEDURE DIVISION with the GO TO statement in the following cases:

- An input-output statement is executed for the file in which an input-output error occurred
- Error procedures are re-executed before they are terminated

```
PROCEDURE DIVISION.
  DECLARATIVES.
    error-procedures SECTION.
  USE AFTER ERROR PROCEDURE ON file-1.
       MOVE error-occurrence TO file-status.             (1)
END DECLARATIVES.
  :
  :
 OPEN open-mode file-1.
 IF file-status = error-occurrence                       (2)
  THEN GO TO open-error-processing.
  :
  :
```

If a file could not be opened, error procedures (MOVE statement of (1)) are executed, and control is passed to the statement (IF statement of (2)) immediately after the OPEN statement.

Input-Output Error Execution Results

The execution result when an input-output error occurs depends on whether the AT END specification, INVALID KEY specification, FILE STATUS clause, and error procedures are written. The following table lists execution results when input-output errors occur.

Table 19. Execution results when input-output errors occur

Error type		**With error procedures**		**Without error procedures**	
		With the FILE STATUS clause	Without the FILE STATUS clause	With the FILE STATUS clause	Without the FILE STATUS clause
AT END condition or INVALID KEY condition	Detected by using an executed input-output statement with AT END or INVALID KEY specified	The statement written for the AT END or INVALID KEY specification is executed.			
	Detected by using an executed input-output statement without AT END or INVALID KEY specified	Error procedures are executed immediately after the input-output statement in which an error occurred.		The statement immediately following an input-output statement (in which an error occurred) is executed.	A U-level message is output and the program terminates abnormally.
Other input-output errors				An I-level message is output then the statement immediately following an input-output statement (in which an error occurred) is executed.	A U-level message is output and the program terminates abnormally.

File Processing

This section explains file assignment, file processing results, exclusive control of files, and methods for improving file processing.

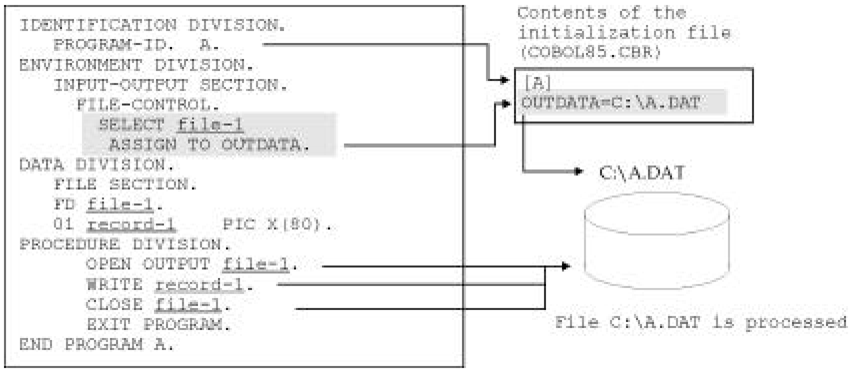
Assigning Files

The method of file input-output processing at program execution is determined by the contents of the ASSIGN clause in the file control entry. The relation between the contents of the ASSIGN clause and files is explained below.

When Specifying a File Identifier in the ASSIGN Clause

Set the name of a file for input-output processing at program execution with a file identifier as the run-time environment information name. For details about how to set run-time environment information, refer to "Setting Run-time Environment Information" in Chapter 5.

The following is an example of setting run-time environment information by using an initialization file.



```
IDENTIFICATION DIVISION.
   PROGRAM-ID.  A.
ENVIRONMENT DIVISION.
   INPUT-OUTPUT SECTION.
     FILE-CONTROL.
       SELECT file-1
        ASSIGN TO OUTDATA.
DATA DIVISION.
   FILE SECTION.
   FD file-1.
   01 record-1     PIC X(80).
PROCEDURE DIVISION.
      OPEN OUTPUT file-1.
      WRITE record-1.
      CLOSE file-1.
      EXIT PROGRAM.
END PROGRAM A.
```

Contents of the initialization file (COBOL85.CBR)

```
[A]
OUTDATA=C:\A.DAT
```

C:\A.DAT
File C:\A.DAT is processed

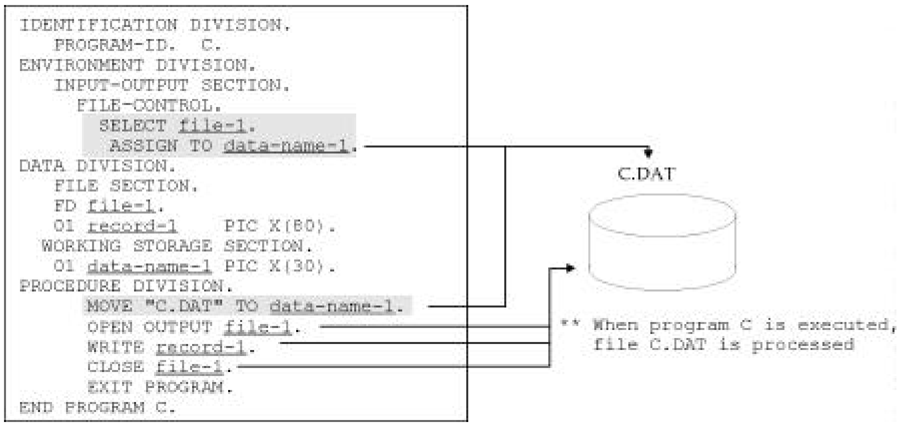
When a lowercase file identifier is specified in the ASSIGN clause:

- A compiler error occurs if a file is compiled with compiler option NOALPHAL specified.
- When setting run-time environment information, use uppercase letters.

If run-time environment information is left blank, a file assignment error occurs.

When Specifying a Data Name in the ASSIGN Clause

Input-output processing is done for a file with the name specified in the data name. For example:



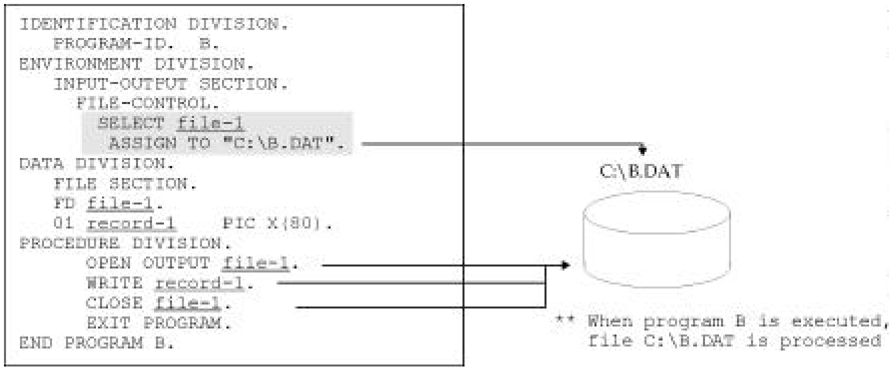
```
IDENTIFICATION DIVISION.
   PROGRAM-ID.  C.
ENVIRONMENT DIVISION.
   INPUT-OUTPUT SECTION.
     FILE-CONTROL.
       SELECT file-1.
        ASSIGN TO data-name-1.
DATA DIVISION.
   FILE SECTION.
   FD file-1.
   01 record-1     PIC X(80).
  WORKING STORAGE SECTION.
   01 data-name-1 PIC X(30).
PROCEDURE DIVISION.
      MOVE "C.DAT" TO data-name-1.
      OPEN OUTPUT file-1.
      WRITE record-1.
      CLOSE file-1.
      EXIT PROGRAM.
END PROGRAM C.
```

When the file name specified in the program is a relative path name, the file having the current directory name prefixed is eligible for input-output processing.

If the data name is left blank, a file assignment error occurs.

When Writing a File-Identifier Literal in the ASSIGN Clause

Input-output processing is done for a file with the name written as a file-identifier literal. For example:

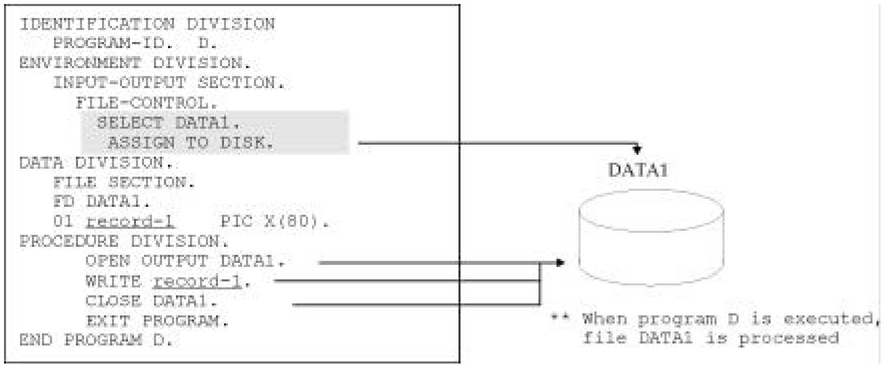


```
IDENTIFICATION DIVISION.
   PROGRAM-ID.  B.
ENVIRONMENT DIVISION.
   INPUT-OUTPUT SECTION.
     FILE-CONTROL.
       SELECT file-1
        ASSIGN TO "C:\B.DAT".
DATA DIVISION.
   FILE SECTION.
   FD file-1.
   01 record-1    PIC X(80).
PROCEDURE DIVISION.
      OPEN OUTPUT file-1.
      WRITE record-1.
      CLOSE file-1.
      EXIT PROGRAM.
END PROGRAM B.
```

When the file name written in the program is a relative path name, the file having the current directory name prefixed is eligible for input-output processing.

When Writing a Character String DISK in the ASSIGN Clause

Input-output processing is done for a file with the name specified in the SELECT clause. For example:



```
IDENTIFICATION DIVISION
   PROGRAM-ID.  D.
ENVIRONMENT DIVISION.
   INPUT-OUTPUT SECTION.
     FILE-CONTROL.
       SELECT DATA1.
        ASSIGN TO DISK.
DATA DIVISION.
   FILE SECTION.
   FD DATA1.
   01 record-1    PIC X(80).
PROCEDURE DIVISION.
      OPEN OUTPUT DATA1.
      WRITE record-1.
      CLOSE DATA1.
      EXIT PROGRAM.
END PROGRAM D.
```

Files under the current directory are eligible for input-output processing.

Exclusive Control of Files

Disable access during file processing by setting the exclusive mode for files or locking records in use. This is called exclusive control of files.

This section explains the relationship between file processing and exclusive control of files.

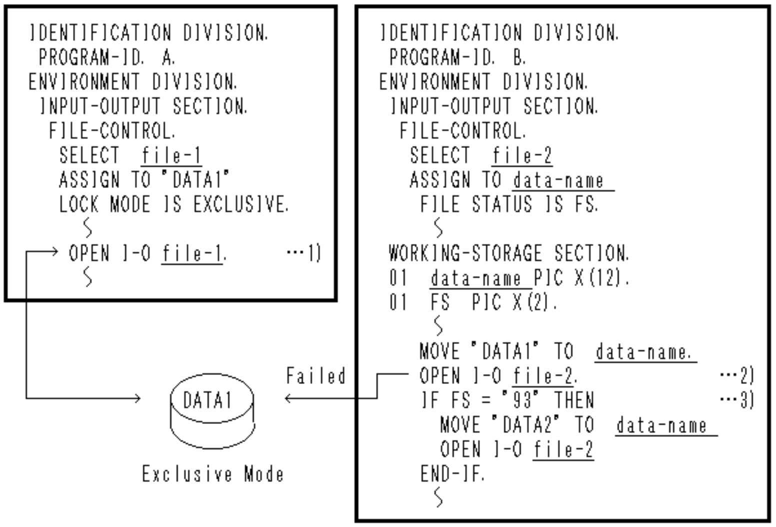
Setting Files in Exclusive Mode

When a file is opened in exclusive mode, other users cannot access the file.

A file is opened in exclusive mode in the following cases:

- An OPEN statement is executed for a file with EXCLUSIVE specified in the LOCK MODE clause in the file control entry.
- An OPEN statement in other than INPUT mode is executed for a file without the LOCK MODE clause specified in the file control entry.
- An OPEN statement with WITH LOCK specified is executed.
- An OPEN statement in the OUTPUT mode is executed.

The following is an example of exclusive control of files:



```
IDENTIFICATION DIVISION.
 PROGRAM-ID. A.
ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
  FILE-CONTROL.
   SELECT  file-1
   ASSIGN TO "DATA1"
   LOCK MODE IS EXCLUSIVE.
     ~
   OPEN I-O file-1.        ...1)
     ~
```

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. B.
ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
  FILE-CONTROL.
   SELECT  file-2
   ASSIGN TO data-name
    FILE STATUS IS FS.
     ~
 WORKING-STORAGE SECTION.
 01  data-name PIC X(12).
 01  FS  PIC X(2).
     ~
    MOVE "DATA1" TO  data-name.
    OPEN I-O file-2.                  ...2)
    IF FS = "93" THEN                 ...3)
      MOVE "DATA2" TO  data-name
      OPEN I-O file-2
    END-IF.
     ~
```

Figure 69. Exclusive control of a file

1. File DATA1 is opened in exclusive mode.
2. If an attempt is made to execute an OPEN statement for file DATA1 that is already in use in exclusive mode by program A, the attempt fails.
3. I-O status value "93" (error caused by exclusive control of files) is set for the data name specified in the FILE STATUS clause.

Locking Records

If records are locked, other users cannot lock them. To lock a record, open the file containing the record in share mode.

A file is opened in share mode in the following cases:

- An OPEN statement without WITH LOCK specified in other than OUTPUT mode is executed for a file with AUTOMATIC or MANUAL specified in the LOCK MODE clause of the file control entry.

- An OPEN statement in the INPUT mode is executed.

Files opened in share mode can be used by other users. If a file is already in use in exclusive mode by another user, however, an OPEN statement fails.

Records in a file opened in share mode are locked with the execution of an input-output statement with exclusive control specified.

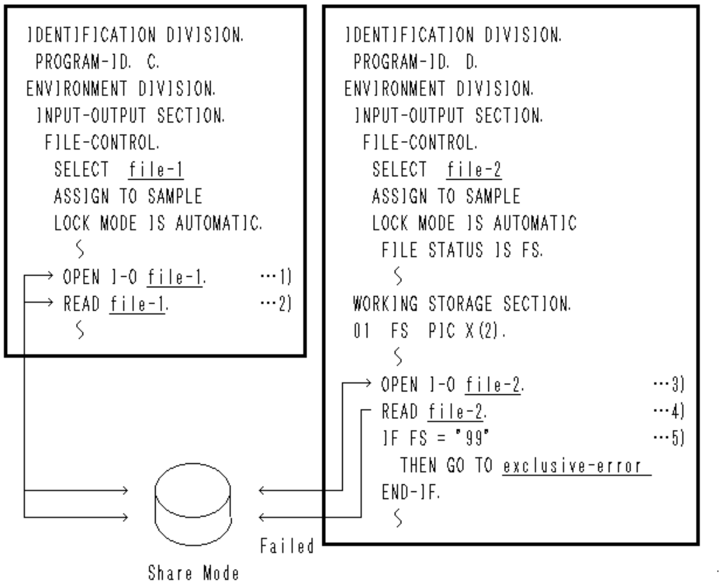
Records are locked in the following cases:

- A file with AUTOMATIC specified in the LOCK MODE clause in the file control entry is opened in I-O mode, then a READ statement without WITH NO LOCK specified is executed.
- A file with MANUAL specified in the LOCK MODE clause in the file control entry is opened in I-O mode, then a READ statement with WITH LOCK specified is executed.

Records are released from lock in the following cases:

- For a file with AUTOMATIC specified in the LOCK MODE clause
  - A READ, REWRITE, WRITE, DELETE, or START statement is executed
  - An UNLOCK statement is executed
  - A CLOSE statement is executed
- For a file with MANUAL specified in the LOCK MODE clause
  - An UNLOCK statement is executed
  - A CLOSE statement is executed

The following is an example of locking records.



```
IDENTIFICATION DIVISION.
 PROGRAM-ID. C.
ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
  FILE-CONTROL.
   SELECT  file-1
   ASSIGN TO SAMPLE
   LOCK MODE IS AUTOMATIC.
     ∫
    OPEN I-O file-1.      …1)
    READ file-1.          …2)
     ∫
```

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. D.
ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
  FILE-CONTROL.
   SELECT  file-2
   ASSIGN TO SAMPLE
   LOCK MODE IS AUTOMATIC
    FILE STATUS IS FS.
     ∫
 WORKING STORAGE SECTION.
 01  FS  PIC X(2).
     ∫
    OPEN I-O file-2.               …3)
    READ file-2.                   …4)
    IF FS = "99"                   …5)
      THEN GO TO exclusive-error
    END-IF.
     ∫
```

Figure 70. Locking records

1. The file is opened in share mode.
2. The first record in the file is locked upon execution of a READ statement.
3. The file is opened in share mode.
4. A READ statement for the locked record fails.
5. I-O status value "99" (error caused by locking records) is set for the data name specified in the FILE STATUS clause.

File Processing Results

File processing generates new files or updates files. This section explains file status when file processing is done.

- File creation: Generates a new file upon execution of an OPEN statement. If a file with the same name already exists, the file is regenerated and the original contents are lost.
- File extension: Extends an existing file upon execution of a WRITE statement. If an attempt is made to extend a file not existing at program execution (optional file), a new file is generated upon execution of an OPEN statement.
- Record reference: Does not change the contents of the file. With an optional file, the at end condition occurs with the first READ statement.
- Record updating, deletion, and insertion: Changes the contents of an existing file with the execution of a REWRITE, DELETE, or WRITE statement. With an optional file, a new file is generated with the execution of an OPEN statement. Since no data exists in this file, however, the at end condition occurs with the first READ statement.

When a program is terminated without executing a CLOSE statement, files are closed unconditionally. If the unconditional close fails, a message is output, and files become unusable even though they are open.

A file can be assigned with the file identifier as a run-time environment information name. If so, an input-output statement is executed for the file assigned with the file identifier, even if the file assignment destination is changed with the environment variable operation function while the file is open.

When an OPEN statement is executed after the file assigned with the file identifier is closed with a CLOSE statement, subsequent input-output statements are executed for the file changed with the environment variable operation function.

Therefore, use the OPEN statement to start the sequence of file processing and the CLOSE statement to terminate the sequence.

If the area is insufficient for file creation, extension, record updating, or insertion, subsequent operation for the file is undefined. If an attempt is made to write records to the file when the area is insufficient, how they are stored in the file is also undefined.

When an indexed file is opened in OUTPUT, I-O, or EXTEND mode, it may become unusable if the program terminates abnormally before it is closed. Make backup copies before executing programs that may terminate abnormally.

Files that became unusable can be recovered with the Recovery command of the COBOL85 FILE UTILITY. A function having the same feature as the above command can be called from an application.

For more information, see “COBOL85 FILE UTILITY” and Appendix L, “Indexed File Recovery.”

COBOL85 FILE UTILITY

This section explains the COBOL85 FILE UTILITY.

The COBOL85 FILE UTILITY allows you to use utility commands to execute file processing that can be used with COBOL85 file systems, without using COBOL applications. In this section, files (record sequential, line sequential, relative, and indexed files) handled by COBOL85 file systems are simply called COBOL files.

With the COBOL85 FILE UTILITY, you can create COBOL files based on data generated with text editors, and manipulate COBOL files and records (such as display, edit, and sort).

Other operations for COBOL files include copying, moving, and deleting files, converting file organization, and reorganizing, recovering, and displaying attributes of indexed files. These utilities can be easily operated by selecting menus in the window.

Using the COBOL85 FILE UTILITY

This section explains how to use the COBOL85 FILE UTILITY.

Setting Up Environments

1. Specify the name of the path containing the COBOL run-time system for environment variable PATH.
2. Specify the directory to generate a temporary work file for environment variable TEMP with a path name beginning with a drive name. When the Recovery command is executed with these utilities, a temporary work file (~UTYnnnn.TMP

(nnnn indicates alphanumeric characters)) of the same size as the file to be processed is generated under the specified directory.

3. Specify the directory to generate a sort work file for environment variable BSORT_TMPDIR with a path name beginning with a drive name. When the Sort command is executed with these utilities, a temporary file (~SRTnnnn.TMP (nnnn indicates alphanumeric characters)) is generated under the specified directory. If environment variable BSORT_TMPDIR is not specified, a sort work file is generated under the directory specified by environment variable TEMP.

Activating the COBOL85 FILE UTILITY

Start the COBOL85 FILE UTILITY as follows:

Windows 95 and Windows NT:

- Select COBFUT32[File Utility] from the P-STAFF Tools menu
- Execute COBFUT32.EXE

Windows 3.1:

- Execute COBFUTY.EXE
- Select an execution command from the P-STAFF Utilities menu.
- Select an execution command from the directory window of File Manager.
- Set up an icon for the command, and activate the command by double-clicking on the icon.
- Select Run from the Program Manager or File Manager menus, then specify the program and parameters to run.

Selecting Commands

Select the command to be executed from the Command pulldown menu on the menu bar. Selecting the command displays a dialog box used to execute each command. For details of each function, see “COBOL85 FILE UTILITY Functions.”

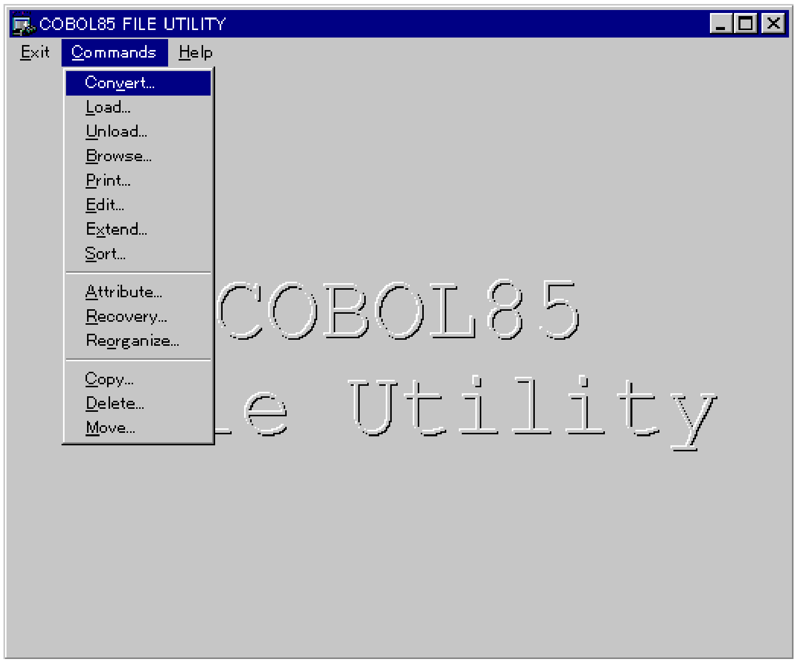


Figure 71. The COBOL85 FILE UTILITY window

The COBOL85 FILE UTILITY contains the following commands:

Convert

Create a new COBOL record sequential file in variable length format with data generated by text editors as input. You can also convert the COBOL file back to a text file.

Load

Convert COBOL file organization to another kind of file organization using a record sequential file in variable length record format as input.

In addition, all variable length records of a sequential file can be added to an arbitrary COBOL file.

Unload

Convert COBOL file organization to variable length records in a sequential file using an arbitrary COBOL file as input.

Browse

Browse the content of records in COBOL files.

Print

Print the records in COBOL files.

Edit

Edit the contents of records from COBOL files.

Extend

Add variable length records in a sequential file to existing arbitrary COBOL files.

Sort

Sort records of an arbitrary COBOL file and place the results in a variable length record sequential file.

Attribute

Display attribute information (record length, record key information etc.) of an index file excluding the record contents.

Recovery

Recover corrupt index files.

Reorganize

Reduce file size by deleting empty blocks in index files.

Copy

Copy a file.

Delete

Delete a file.

Move

Move a file to another location.

Quitting the COBOL85 FILE UTILITY

Select Exit on the menu bar.

As with file processing using COBOL programs, file processing using the COBOL85 FILE UTILITY depends on file organization.

For example, since record and line sequential files are processed in sequential access, records are processed only in a fixed order or cannot be inserted or deleted. See “File Organization Types and Processing.”

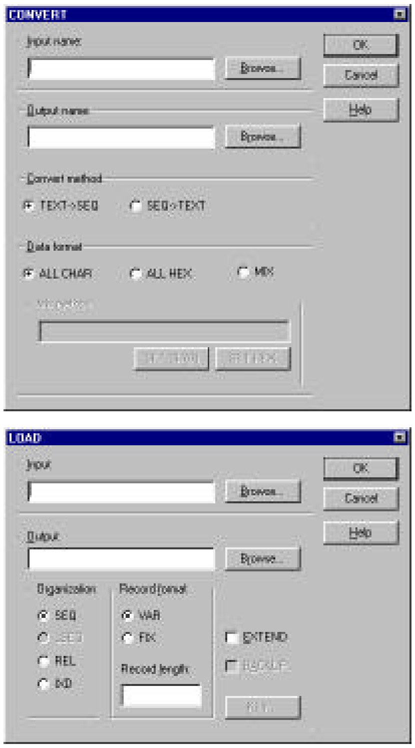
A system error can occur during execution of the COBOL85 FILE UTILITY. If an error occurs, an error code (decimal expression) is displayed in the error message. Refer to the system error code explanation or “System Error Codes” in Appendix F.

COBOL85 FILE UTILITY Functions

The COBOL85 FILE UTILITY provide the functions given below. In quotation marks (") are commands (< > indicates a command name and [ ] indicates its use is optional) to use the functions. For more information about how to operate the dialog box, refer to the online help.

Creating Files

Create a new COBOL file with data generated by text editors as input using the "<Convert> [+ <Load>]" commands.



Convert method: [✓ TEXT->SEQ] SEQ->TEXT

Data format: [✓ ALL CHAR] ALL HEX MIX

Organization: [✓ SEQ] LSEQ REL IXD

Record format: [✓ VAR] FIX

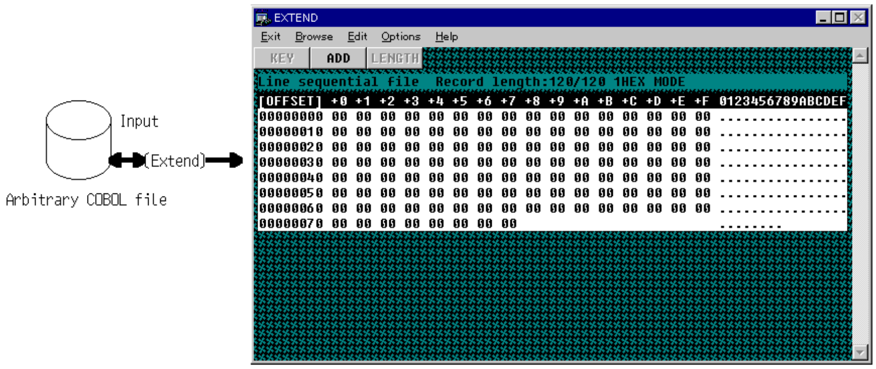
Record length:

Figure 72. Creating a file with the COBOL85 FILE UTILITY

When an error occurs with extended specification, the contents of existing files are lost if the check box of Make BACKUP is not selected.

Adding Records

Add records to existing COBOL files with the "<Extend>" command.



```
Line sequential file  Record length:120/120 1HEX MODE
[OFFSET] +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F 0123456789ABCDEF
00000000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000070 00 00 00 00 00 00 00 00                         ........
```

Figure 73. Adding records with the Extend command

The Extend window contains the following elements:

Exit

Quit (exit) the Extend window.

Browse

Browse the content of records in COBOL files.

Edit

Edit a record.

Option

Determine the method of data input (hexadecimal or character) when adding new records.

Help

Access the online help.

Add button

Add new records to the file.

The following describes other items which are found in this example of an Extend window.

Line sequential file

Illustrates the attributes of the displayed file.

Record length 120/120

Illustrates the record length / maximum record length.

HEX MODE

Illustrates the hexadecimal data input method.

Character string under [OFFSET] and "**;0 +1 ... +E +F**"

The position offset of the data record is illustrated by the hexadecimal. The position of the cursor (offset in the data record) is "0x00000008" which left "00000000" of "+8" of the cursor line and the cursor was added.

Two character strings under "**+0 +1 ... +E +F**"

The data record is displayed by a hexadecimal every one byte.

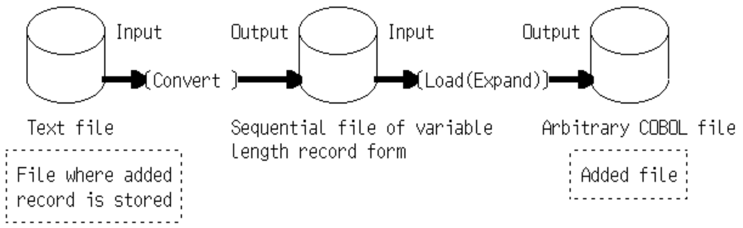
Character string under "**0123456789ABCDEF**"

The data record is displayed by a one-byte character mark.

Note: Characters that cannot be displayed, such as national characters, are displayed with periods (.).This is the same for browsing, editing, and printing records. Refer to the online help for additional details.

Adding records from another file

Use the "[<Convert> +] <Load (extend)> or [<Unload> +] <Load (extend)>" commands to add the contents of other files to existing COBOL files.



or

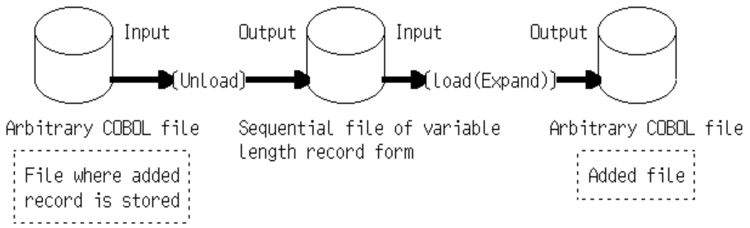
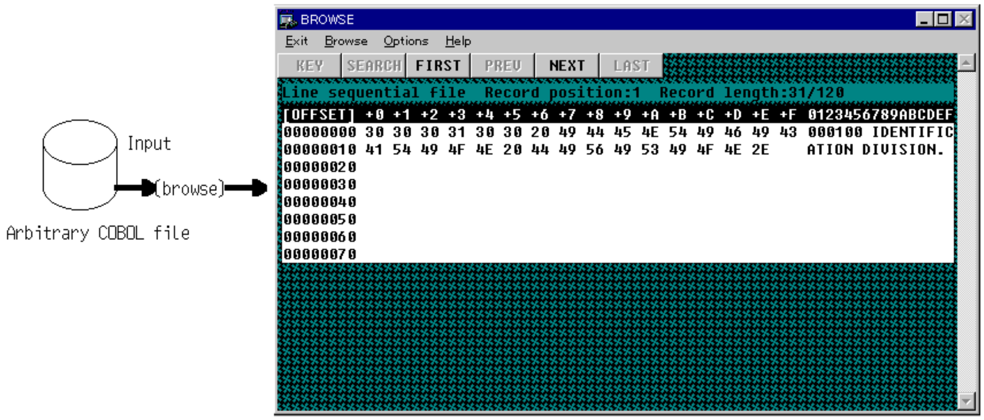


Figure 74. Two methods of adding records from another file

Browsing Records

Browse the contents of records in COBOL files with the "<Browse>" command.



```
Line sequential file  Record position:1  Record length:31/120
[OFFSET] +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F 0123456789ABCDEF
00000000 30 30 30 31 30 30 20 49 44 45 4E 54 49 46 49 43 000100 IDENTIFIC
00000010 41 54 49 4F 4E 20 44 49 56 49 53 49 4F 4E 2E    ATION DIVISION.
00000020
00000030
00000040
00000050
00000060
00000070
```

Figure 75. Browsing files with the Browse command

The Browse window contains the following elements:

Exit

Quit (exit) the Browse window.

Browse

Browse the contents of records in COBOL files.

Options

Determine the method of data input (hexadecimal or character) when adding new records.

Help

Access the online help.

Key button

Change the order of displayed records. The order displayed depends upon which record key is selected.

Search button

Search for a particular record.

First button

Go to the first record in a file.

Prev button

Go to the previous record in a file.

Next button

Go to the next record in a file.

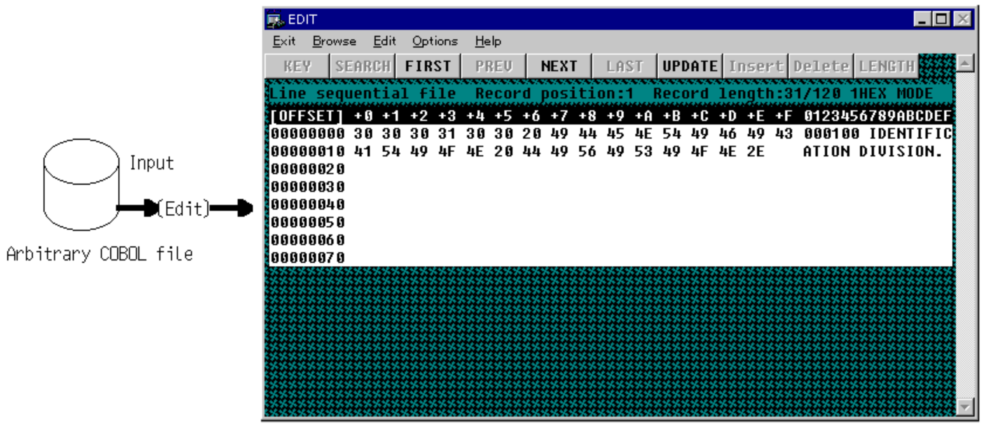
Last button

Go to the last record in the file.

Note: Characters that cannot be displayed, such as national characters, are displayed with periods (.).This is the same for browsing, editing, and printing records. Refer to the online help for additional details.

Editing Records

Update, insert, and delete records within COBOL files using the "<Edit>" command.


Input
Edit
Arbitrary COBOL file
EDIT
Exit Browse Edit Options Help
KEY SEARCH FIRST PREV NEXT LAST UPDATE Insert Delete LENGTH

```
Line sequential file  Record position:1  Record length:31/120 1HEX MODE
[OFFSET] +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F 0123456789ABCDEF
00000000 30 30 30 31 30 30 20 49 44 45 4E 54 49 46 49 43 000100 IDENTIFIC
00000010 41 54 49 4F 4E 20 44 49 56 49 53 49 4F 4E 2E    ATION DIVISION.
00000020
00000030
00000040
00000050
00000060
00000070
```

Figure 76. Updating, inserting or deleting records with the Edit command

The Edit window contains the following elements:

Exit

Quit (exit) the Edit window.

Browse

Browse the content of records in COBOL files.

Edit

Edit a record.

Options

Determine the method of data input (hexadecimal or character) when adding new records.

Help

Access the online help.

Key button

Change the order of displayed records. The order displayed depends upon which record key is selected.

Search button

Search for a particular record.

First button

Go to the first record in a file.

Prev button

Go to the previous record in a file.

Next button

Go to the next record in a file.

Last button

Go to the last record in the file.

Update button

Update a record.

Insert button

Insert a new record.

Delete button

Delete a record.

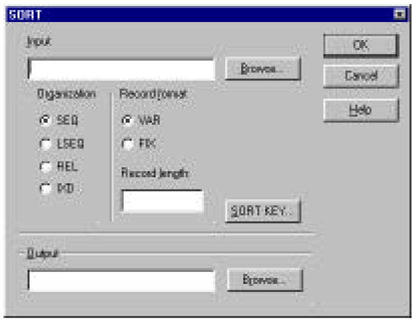
Length button

Change the length of a record.

Note: If the maximum record length of an indexed file is 16 kilobytes or more, the file cannot be opened with the Edit command.

Sorting Records

Sort records in COBOL files with the "<Sort>" command.



Organization: [✓ SEQ] LSEQ REL IXD

Record format: [✓ VAR] FIX

Record length:

Figure 77. Sorting records in COBOL files with the Sort command

A record sequential file with variable length record format is generated.

Manipulating COBOL files

Copy, delete, and move COBOL files with the "<Move>, <Copy>, and <Delete>" commands.

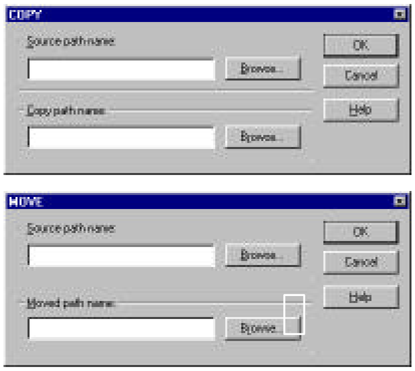
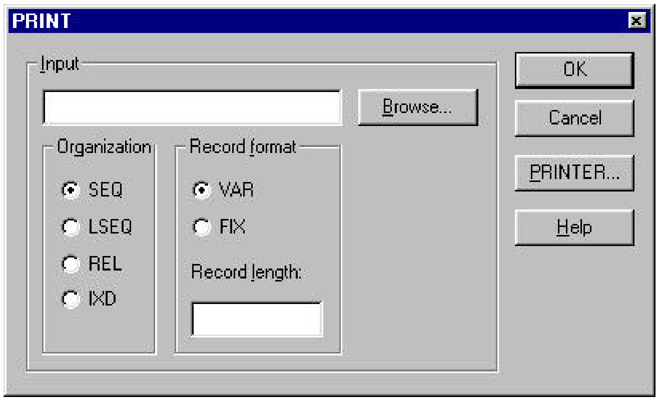


Figure 78. Manipulating COBOL files with the Copy and Move commands

Printing Files

Print the contents of COBOL files with the "<Print>" command.



Organization: [✓ SEQ] LSEQ REL IXD

Record format: [✓ VAR] FIX

Figure 79. Printing COBOL files with the Print command

Converting File Types

Convert COBOL file to another kind of file organization using the "<Load>" or "<Unload>" commands. Use the "<Load>" command to input a record sequential file in variable length record format and convert it to a COBOL file. Use the "<Unload>" command to input a COBOL file and convert it to a record sequential file in variable length record format.

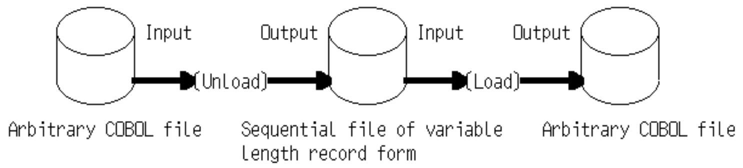
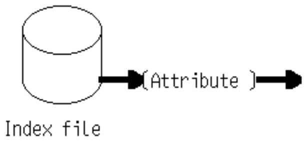


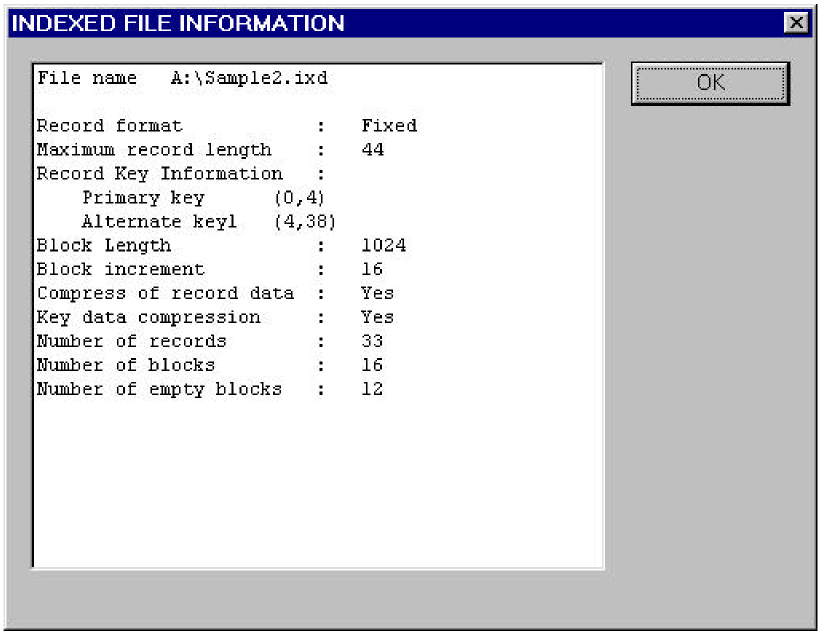
Figure 80. Converting files using the Load and Unload commands

Manipulating Indexed Files

You can execute the following operations for indexed files:

- Display attribute information with the "<Attribute>" command





```
File name   A:\Sample2.ixd

Record format            :   Fixed
Maximum record length    :   44
Record Key Information   :
    Primary key      (0,4)
    Alternate key1   (4,38)
Block Length             :   1024
Block increment          :   16
Compress of record data  :   Yes
Key data compression     :   Yes
Number of records        :   33
Number of blocks         :   16
Number of empty blocks   :   12
```

Figure 81. Displaying attribute information with the Attribute command

The Indexed File Information window contains the following information:

File name

Displays the index file name whose attributes follow.

Record format

Displays the record format (fixed length / variable length).

If the maximum record length is fixed, the maximum record length is displayed.

If the minimum record length is fixed, the minimum record length is not displayed.

If the maximum record length is variable, the maximum record length is displayed.

If the minimum record length is variable, the minimum record length is displayed.

Record key information

Displays key information for the index file. The format of the key information is the same as the format of key information specified when the index file is created by the Load command. Refer to the online help for more details.

Block length

Displays the length of one block in the index file.

Block increment

Displays block increments in the index file.

Compress of record data

Displays whether or not the stored record data is compressed.

Key data compression

Displays whether or not the stored key data is compressed.

Number of records

Displays the number of records in the index file.

Number of blocks

Displays the number of blocks in the index file.

Number of empty blocks

Displays the number of unused blocks in the index file.

- Recover corrupt indexed files with the "<Recovery>" command.

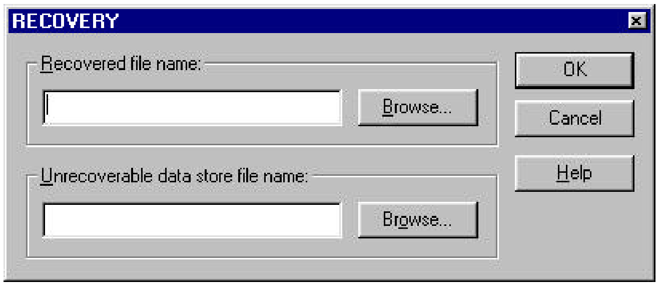


Figure 82. Recovering corrupt files with the Recovery command

- Reorganize (delete empty blocks) indexed files with the "<Reorganize>" command.

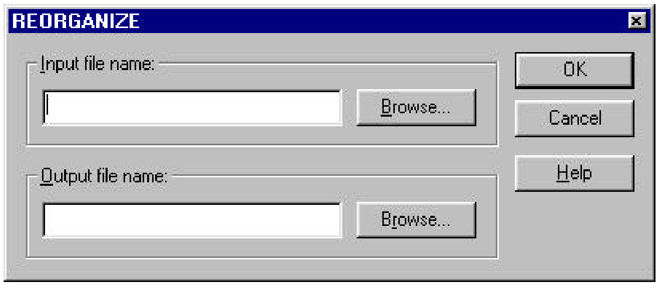


Figure 83. Reorganizing indexed files with the Reorganize command

Chapter 8. Printing

This chapter explains how to print data by line or form, and provides information about types of printing methods, print characters, form overlay patterns, forms control buffers (FCB), and form descriptors.

Types of Printing Methods

Use a print file or presentation file to print data from a COBOL program. This section outlines how to print data using these files, and explains print character types, form overlay patterns, forms control buffers (FCB), and form descriptors. The print function you use depends on the printer being used. Refer to the user manual for each specific printer.

Outline of Printing Methods

There are two types of print files, files with a FORMAT clause and those without a FORMAT clause.

Use a print file without a FORMAT clause to print data in line mode (line by line; Use 1), overlay data line by line with a form overlay pattern, and print data by setting print information with an FCB (Use 2).

By using a print file with a FORMAT clause, data in forms with form descriptors can be printed.

This chapter classifies print and presentation files in the following groups:

1. Print file without a FORMAT clause (Use 1)
2. Print file without a FORMAT clause (Use 2)
3. Print file with a FORMAT clause
4. Presentation file

The following table lists the characteristics, advantages, and uses of printing methods (1) to (4).

Table 20. Characteristics, advantages, and uses of each printing method

File Type		**(1)**	**(2)**	**(3)**	**(4)**
Characteristics	Data can be printed in line mode.	B	B	B	C
	Data can be overlaid with form overlay patterns.	C	B	B	B *1
	Data in forms can be printed with form descriptors.	C	C	B	B
Advantage	Simple program coding.	A	B	B	A
	Simple forms printing.	B	A	A	A
	Existing form descriptors generated by other systems can be used.	C	C	A	A
	Various types of print information can be specified in programs.	C	A	A	B
Use	Output data to files.	A	A	C	C
	Print forms.	B	A	A	A

A : Can be used (suitable).
B : Can be used.
C : Cannot be used.
*1 Can be printed only when an overlay pattern name is specified in form descriptors or is specified in a printer information file. For details, refer to the FORM RTS HELP.

The following table lists related products

Table 21. Related products

File Type	**(1)**	**(2)**	**(3)**	**(4)**
Power FORM	-	-	-	Needed
FORM overlay option	-	Needed	Needed	Needed
FORM RTS	-	-	Needed	Needed
MeFt/NET	-	-	-	Needed

Each printing method is outlined below.

(1) Print File without a FORMAT Clause (Use 1)

By using a print file without a FORMAT clause, data can be sent in line mode to a printer. During output, a logical page size, line feed, and page alignment can be specified.

For details on how to use a print file when printing data in line mode, see “Using Print File 1.”

(2) Print File without a FORMAT Clause (Use 2)

With a print file, a page of output data can be overlaid with a form overlay pattern, or print information can be specified with a FCB. When specifying that a page of output data be overlaid with a form overlay pattern, use a control record. When specifying print information with a FCB, write a FCB control statement in an initialization file.

For details on form overlay patterns, see “Form Overlay Patterns.” For details on FCB, see “Forms Control Buffers (FCB).” For how to use print files, see “Using Print File 2.”

(3) Print File with a FORMAT Clause

For a print file with a FORMAT clause, specify the FORMAT clause in file definitions in a program. By using a print file with a FORMAT clause, data in forms can be printed with partitioned form descriptors. At this time, the attributes of items defined in the form descriptors can be changed by using a special register during program execution.

Forms can be printed with the form overlay patterns described above and FCB. FORM RTS is required, however, to print forms using a print file with a FORMAT clause.

For details on form descriptors, see “Form Descriptors.” For details on print files with form descriptors, see “Using Print Files with Form Descriptors.”

(4) Presentation File

With a presentation file, data in forms defined in the form descriptors can be printed. As with the print file with a FORMAT clause, the attributes of items defined in the form descriptors can be changed by using a special register during program execution.

Before using a presentation file, the file must be defined and a presentation record must be created with a WRITE statement.

When a chart record is output, forms in the format defined in the form descriptors are printed. The attributes of output data defined in the form descriptors can then be changed. For more information about how to use presentation files when printing forms, see “Using Presentation Files (Printing Forms).”

Print Characters

The size, font, form, direction, and space of print characters can be specified in the CHARACTER TYPE clause of a data description entry. MODE-1, MODE-2, MODE-3, and a mnemonic-name, and print mode name can be specified in a CHARACTER TYPE clause.

MODE-1, MODE-2, and MODE-3 designate 12-point, 9-point, and 7-point print characters.

When a mnemonic-name is set, data is printed with the size, font, form, direction, and space of a print character indicated by a function-name. That function-name is related to a mnemonic-name with a function-name clause in the SPECIAL-NAMES paragraph.

When a print mode name is set, data is printed with the size, font, form, direction, and space of a print character defined in a PRINTING MODE clause in the Special-Names paragraph.

For details about how to write a CHARACTER TYPE clause, and how to define a mnemonic-name and print mode name, refer to the “COBOL85 Reference Manual.”

The sizes, fonts, forms, and directions of available print characters, and spaces defined according to the sizes and forms of print characters are explained below.

Available sizes

3.0 to 300.0 points

Character size can be specified in units of 0.1 point. When a raster-type device font (fixed size) is selected, data is printed with a point (character size) of a font mounted on each printer.

If character size specification is omitted and execution environment information @CBR_PrinterANK_Size is specified, the character size will be the default of the executing platform. If no character size is specified, the default is 7.0 points. Refer to “Format of Run-time Environment Information” in Chapter 5.

Specify a font with run-time environment information @PrinterFontName. If the run-time environment information is omitted, CourierNew is normally selected.

If an outline font or a TrueType font is specified for run-time environment information, the print format is a scaleable font and data can be printed with 3.0 to 300.0 points in units of 0.1 point.

Available Fonts

Fonts used are MINCHOU, MINCHOU-HANKAKU, GOTHIC, and GOTHIC-HANKAKU. The default font is GOTHIC.

By using a print file without a FORMAT clause, if the font name is set for run-time environment information @PrinterFontName, data is printed with the character font of the specified font name. If the font name is omitted, CourierNew is used as the default.

For a print file with a FORMAT clause and presentation file, refer to the FORM RTS online help.

Available Print Character Forms

Print character forms are em-size and em-size tall and wide; double size, en-size, and en-size tall and wide; and double size. The default form is em-size.

Orientation of Print Characters

Characters are printed horizontally.

Print Character Spaces

The following table lists spaces determined according to the sizes and forms of print characters. Character spaces of 0.01 to 24.00 cpi are specified in units of 0.01 cpi. The default space is 10.0 cpi.

Table 22. Relationship between print character sizes/forms and character spaces

Unit: cpi

Character Size	Character Form							
	Em-size	En-size	Tall char	En-size tall char	Wide size char	En-size wide char	Double size	En-double size
MODE-1 (12 point)	5	10	5	-	2.5	-	2.5	5
MODE-2 (9 point)	8	16	8	-	4	-	4	8
MODE-3 (7 point)	10	-	10	-	5	-	5	-
A (9 point)	5	10	5	10	2.5	5	2.5	5
B (9 point)	20/3	40/3	20/3	40/3	10/3	20/3	10/3	20/3
X-12P (12 point)	5	10	5	10	2.5	5	2.5	5
X-9P (9 point)	8	16	8	16	4	8	4	8
X-7P (7 point)	10	20	10	20	5	10	5	10
C (9 point)	7.5	15	7.5	15	3.75	7.5	3.75	7.5
D-12P (12 point)	6	12	6	12	3	6	3	6
D-9P (9 point)	6	12	6	12	3	6	3	6

For the meanings of the codes for character sizes, refer to the "COBOL85 Reference Manual."

Arrangement coordinates of print characters

There are two methods for calculating the print coordinates when printing forms using the FORMAT phrase:

(1) Dividing the print resolution character interval with the space between the lines (DPI), and then arranging characters while rounding off the remainder.

(2) Arrange the characters in coordinates and corrected the result for each print inch. This is done when the resolution cannot be divided using the character interval (CPI) and space between lines (DPI).

In method 1 the character stores only with the number of dots rounded by the top and left. If method 1 can't be used, then method 2 can be used as an alternate.

Coordinates for arranging printed characters can be specified with environment variable @CBR_PrintTextPosition when the FORMAT clause is omitted from the file control entry.

Refer to Chapter 5, "Format of Run-time Environment Information" on the method of specifying the arrangement coordinates of print characters with @CBR_PrintTextPosition.

Form Overlay Patterns

A form overlay pattern is used to set fixed sections in forms in advance, such as ruled lines and headers. Forms can be printed by overlaying a page of output data with a form overlay pattern.

A form overlay pattern is generated from a screen image with the FORM overlay option. A form overlay pattern can be shared with multiple programs, and form overlay patterns generated by other systems can also be used.

For more information about how to generate form overlay patterns, refer to the "FORM V1.3 Manual." For printing forms when using form overlay patterns, see "Using Print File 2."

Forms Control Buffers (FCB)

A forms control buffer (FCB) defines the number of lines on one page, line space, and column start position. By using an FCB, the number of lines on one page, line space, and column start position can be changed.

When an FCB name is specified with an I control record or when the default FCB name is specified in the initialization file (@DefaultFCB_Name=FCBxxx), FCB information can be specified externally. Specify the FCB information in the initialization file as follows:

```
FCBxxx=FCB-control-statement
```

xxx is the FCB name specified in the I control record or FCB name (part of xxx) specified for execution environment information @ DefaultFCB_Name=FCBxxx.

For details of the syntax of a FCB control statement, see Appendix K, “FCB Control Statement.”

The default value of the I control record FCB name is:

- 6 lpi x 11 inches (66 lines)
- Channel number : 01, Line number : 4
- Channel number : 02, Line number : 10
- Channel number : 03, Line number : 16
- Channel number : 04, Line number : 22
- Channel number : 05, Line number : 28
- Channel number : 06, Line number : 34
- Channel number : 07, Line number : 40
- Channel number : 08, Line number : 46
- Channel number : 09, Line number : 66
- Channel number : 10, Line number : 52
- Channel number : 11, Line number : 58
- Channel number : 12, Line number : 64

Refer to “Format of Run-time Environment Information” in Chapter 5 for details on specifying the default FCB name.

Refer to “Using Print File 2” for details on specifying I control records.

Form Descriptors

When forms are designed with FORM or Power FORM, form descriptors are generated. COBOL85 enables data items defined in the form descriptors to be included in a program so that forms can be printed with values set in the data items. Form overlay patterns can be included in form descriptors.

FORM RTS is required when printing forms using form descriptors. FORM RTS requires a printer information file. For details of printer information files, refer to the FORM RTS online help.

For information on how to generate form descriptors, refer to the “FORM V1.3 Manual.” For information about printing forms by using form descriptors see “Using Print Files with Form Descriptors,” or “Using Presentation Files (Printing Forms).”

Using Print File 1

This section explains how to print data in line mode using a print file without a FORMAT clause. For a sample of a program using a print file, refer to the “Getting Started with Fujitsu COBOL” guide.

```
IDENTIFICATION    DIVISION.
 PROGRAM-ID.      program-name.
ENVIRONMENT       DIVISION.
 CONFIGURATION    SECTION.
 SPECIAL-NAMES.
  [function-name IS mnemonic-name.]
 INPUT-OUTPUT     SECTION
 FILE-CONTROL.
  SELECT file-name
   ASSIGN TO PRINTER
  [ORGANIZATION IS SEQUENTIAL]
  [FILE STATUS  IS input-output-status].
DATA  DIVISION.
 FILE  SECTION.
 FD  file-name

    [RECORD record-size]
    [LINAGE IS logical-page-configuration-specification].

                                         { MODE-1          }
 01  record-name  [CHARACTER TYPE IS     { MODE-2          } ].
                                         { MODE-3          }
 ┌─Contents of record ─────────┐         { mnemonic-name   }
 │  02  data-name-1... .       │
 │        :                    │
 └─────────────────────────────┘
 WORKING-STORAGE SECTION.
[01  input-output-status  PIC X(2).]
                                         { MODE-1          }
[01  data-name-2  [CHARACTER TYPE IS     { MODE-2          } ].]
                                         { MODE-3          }
PROCEDURE  DIVISION.                     { mnemonic-name   }
    OPEN OUTPUT file-name.
    WRITE record-name [FROM data-name-2] [AFTER ADVANCING ~].
    CLOSE file-name.
 END PROGRAM program-name.
```

Outline

Define a print file in the same manner as a record sequential file, and do the same processing as generating the record sequential file. The following can be specified with a print file without a FORMAT clause:

- Logical page size (LINAGE clause in the file description entry)

- Character size, font, form, direction, and space (CHARACTER TYPE clause in the data description entry)
- Line feed and page alignment (ADVANCING phrase in the WRITE statement)

Program Specifications

This section details program descriptions in a print file using data in line mode for each COBOL division.

ENVIRONMENT DIVISION

In the ENVIRONMENT DIVISION, write the relation between the function-name and mnemonic-name (print characters are indicated in the program) and define a print file.

Relating the Function-Name to the Mnemonic-Name

Relate the function-name indicating the size, font, form, direction, and space of a print character to the mnemonic-name. Specify the mnemonic-name in the CHARACTER TYPE clause when defining data items in records and work data items. For function-name types, refer to the “COBOL85 Reference Manual.”

Defining Print Files

The following table lists information required to specify a file control entry.

Table 23. Information to be specified in a file control entry

	Location	**Information Type**	**Details and Use of Specification**
Required	SELECT clause	File name	Write the name of a file to use in a COBOL program, conforming to the rules of COBOL user-defined words.
	ASSIGN clause	File-reference-identifier	Write PRINTER, local-printer-port-name (LPTn) or serial-printer-port-name (COMn:). When PRINTER is specified, data is output to the system default printer.
Optional	ORGANIZATION clause	Keyword indicating file organization	Write SEQUENTIAL.
	FILE STATUS clause	Data-name	Write the data-name defined as a 2-byte alphanumeric data item in the WORKING-STORAGE or LINKAGE section. The input-output execution result is set for this data-name. For value to be set, see Appendix B, "I-O Status List."

DATA DIVISION

In the DATA DIVISION, define record definitions and the definitions of data-names specified in a file control entry. Write record definitions in file and record description entries. The following table lists information required to write a file description entry.

Table 24. Information to be specified in a file description entry

	Location	Information Type	Details and Use of Specification
Optional	RECORD clause	Record size	Define the size of the printable area.
	LINAGE clause	Logical page configuration	Define the number of lines per page, top and bottom margins, and the footing area starting position. When a data-name is specified in this clause, the data can be changed in the program.

PROCEDURE DIVISION

Execute an input-output statement in the following sequence:

1. OPEN statement with OUTPUT specified: Starts printing.
2. WRITE statement: Outputs data.
3. CLOSE statement: Stops printing.

OPEN and CLOSE Statements

Use an OPEN statement once at the start of printing and a CLOSE statement once at the end of printing.

WRITE Statement

One WRITE statement writes one line of data.

Specify PAGE in the ADVANCING phrase of the WRITE statement for page alignment. When the number of lines is written, the page is advanced by the specified number of lines.

The AFTER ADVANCING and BEFORE ADVANCING phrases are used to specify whether data is output before or after page alignment or a line feed. When the ADVANCING phrase is omitted, AFTER ADVANCING 1 is the default.

Control of print lines with AFTER ADVANCING phrase is shown in the following example.

```
① OPEN  file-1.
② WRITE record-1 AFTER ADVANCING PAGE.
③ WRITE record-2 AFTER ADVANCING 2.
④ WRITE record-3 AFTER ADVANCING PAGE.
⑤ WRITE record-4
⑥ CLOSE file-1.
```

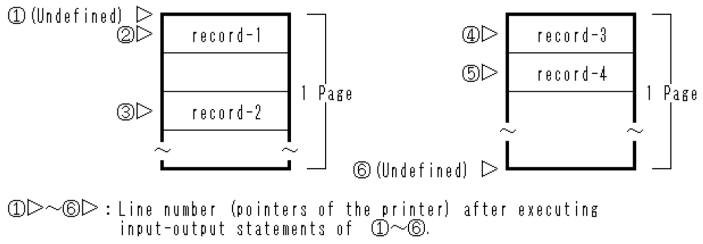

①(Undefined) ▷
②▷
record-1
③▷
record-2
1 Page
④▷
record-3
⑤▷
record-4
⑥(Undefined) ▷
1 Page
①▷~⑥▷ : Line number (pointers of the printer) after executing input-output statements of ①~⑥.

Figure 85. Print lines with AFTER ADVANCING

When writing WRITE A FROM B, define the CHARACTER TYPE clause for B but not for A. If the CHARACTER TYPE clause is defined for both A and B, only the specification of B is valid.

Program Compilation and Linkage

There are no required compiler and linker options.

Program Execution

First, assign a printer. Assigning a printer depends on the contents of descriptions in the ASSIGN clause in the file control entry.

This section explains how to specify a printer in a program or initialization file, and gives the relation between the contents of descriptions in the ASSIGN clause and the output destination by using examples.

If the destination of a print file is other than a printer, contents are not assured.

Specifying a Printer

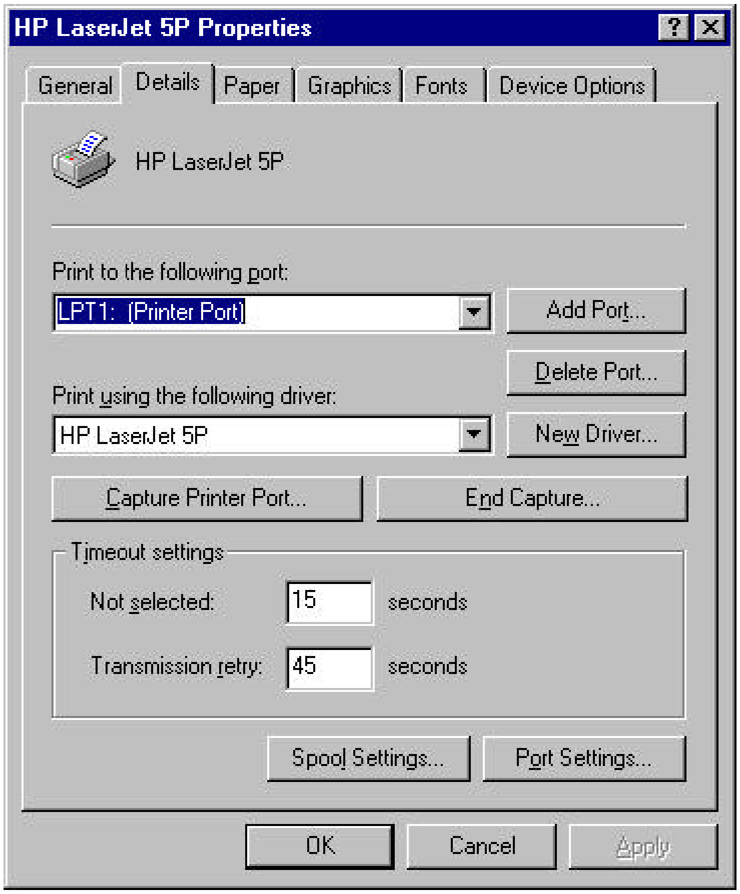
A printer can be specified by the following methods:

- Printer name
- Local printer port name ("LPTn:") or Serial port name ("COMn: "). Where "n" is a number from 1-9.

Windows 95

Use the information in the Details tab of the Printer Properties dialog box to obtain the printer name. To access the Details tab, click on the Start button and then select Settings→Printers. Highlight a printer and select Properties from the File menu. Click on the Details tab.

The Printer Properties dialog box (with the Details tab selected) is shown below.


HP LaserJet 5P Properties

General | Details | Paper | Graphics | Fonts | Device Options

HP LaserJet 5P

Print to the following port: LPT1: (Printer Port)

Print using the following driver: HP LaserJet 5P

Timeout settings

Not selected: 15 seconds

Transmission retry: 45 seconds

Figure 86. The Details tab in the Printer Properties dialog box

Printer name

Specify a locally connected printer in the following manner:

PRTNAME:HP LaserJet 5P

In this example, HP LaserJet 5P is the specified printer.

Specify a printer connected to a network in the following manner:

PRTNAME:\\NMSVR20\Canon LBP-A404E

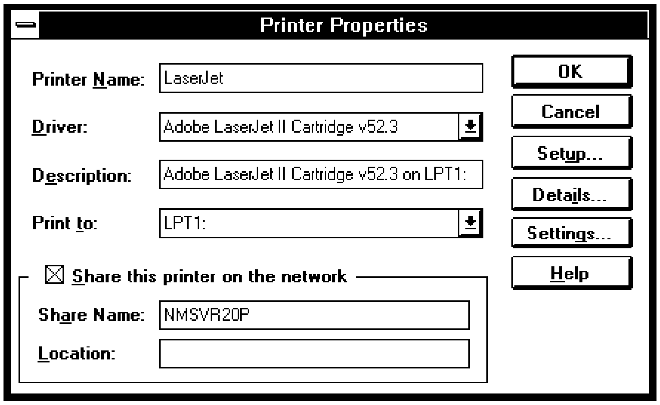
Local printer port name / Serial port name

Specify a local printer port name or serial port name in the following manner:

Example : "LPT1:"

Windows NT

Use the information in the Printer Properties dialog to obtain the printer name. Activate the Print Manager and select Properties from the Printer menu to display the Printer Properties dialog box. The Printer Properties dialog box is shown below:


Printer Properties

Printer Name: LaserJet

Driver: Adobe LaserJet II Cartridge v52.3

Description: Adobe LaserJet II Cartridge v52.3 on LPT1:

Print to: LPT1:

☒ Share this printer on the network

Share Name: NMSVR20P

Location:

Figure 87. The Printer Properties dialog box

Printer name

Specify a locally connected printer in the following manner:

PRTNAME:LaserJet

In this example, LaserJet is the specified printer.

Specify a printer connected to a network in the following manner:

PRTNAME:\\NMSVR20P\LaserJet

Local printer port name / Serial port name

Specify a local printer port name or serial port name in the following manner:

Example : "LPT1: "

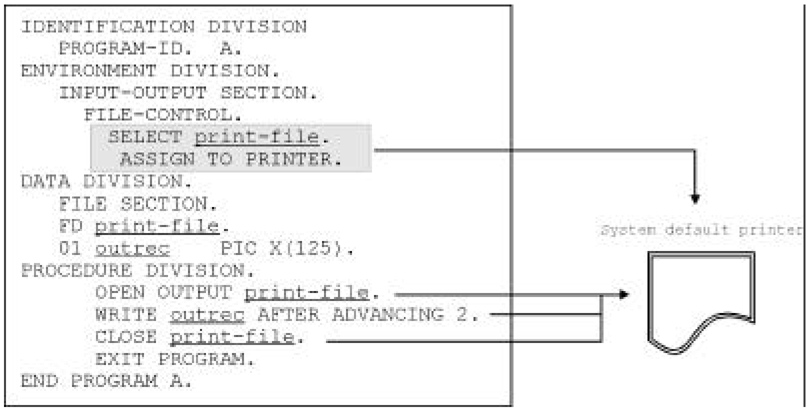
Windows 3.1

- Specify the name of the port where the printer is connected.
- Specify "LPTn: " when the port where the printer is connected is a local printer port. Specify "COMn:" when connected to a serial port. "n" designates 1 to 9.

Examples of Programs

When PRINTER is Specified in the ASSIGN Clause

Data is sent to the system default printer (*1).



```
IDENTIFICATION DIVISION
   PROGRAM-ID.  A.
ENVIRONMENT DIVISION.
   INPUT-OUTPUT SECTION.
     FILE-CONTROL.
       SELECT print-file.
        ASSIGN TO PRINTER.
DATA DIVISION.
   FILE SECTION.
   FD print-file.
   01 outrec    PIC X(125).
PROCEDURE DIVISION.
      OPEN OUTPUT print-file.
      WRITE outrec AFTER ADVANCING 2.
      CLOSE print-file.
      EXIT PROGRAM.
END PROGRAM A.
```

Figure 88. Data flow when sent to the default printer

*1 System default printer:

Windows 95

"Set as Default" is checked for the printer.

Windows NT

Printer is set as default using the Print Manager.

Windows 3.1

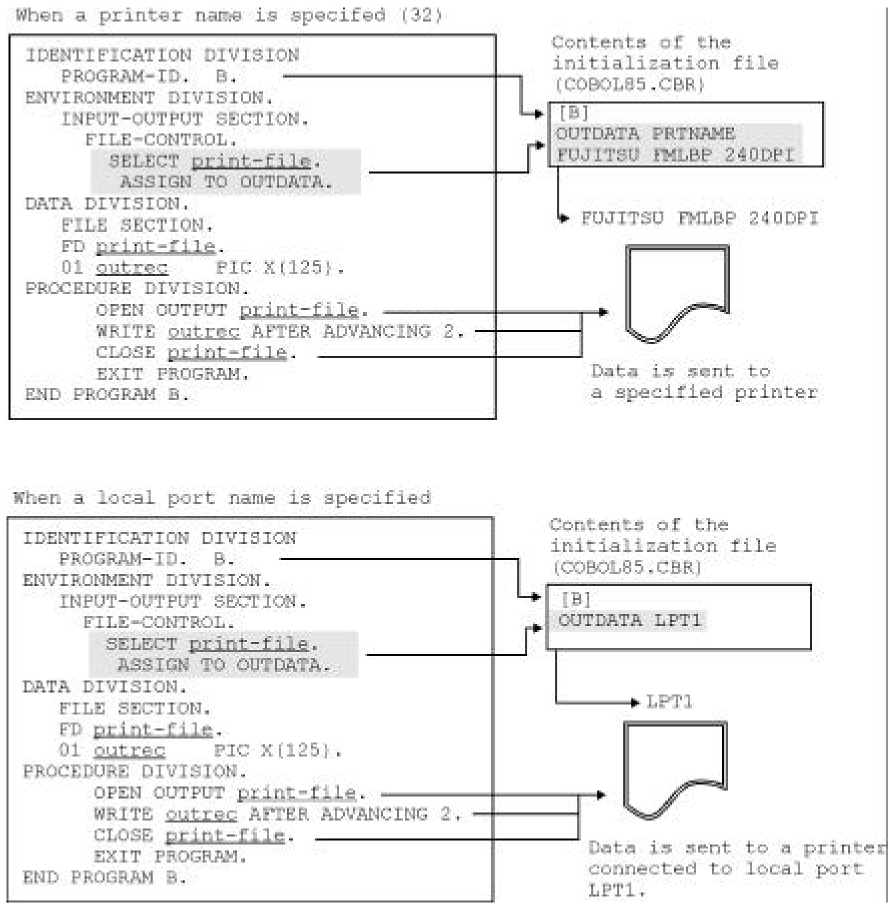
Printer is set as default using the control panel.

To send data to a printer connected to a specific port, follow the method explained below.

When a File-Identifier is Specified in the ASSIGN Clause

Define the name of the output destination printer, local printer port name, or serial port name with the file-identifier as the run-time environment information name. For details on setting run-time environment information, refer to “Setting Run-time Environment Information” in Chapter 5.

A file assignment error occurs if no printer is assigned to the file-identifier. An example of when the initialization file is used is shown in the following figure.


When a printer name is specifed (32)

```
IDENTIFICATION DIVISION
   PROGRAM-ID.  B.
ENVIRONMENT DIVISION.
   INPUT-OUTPUT SECTION.
     FILE-CONTROL.
       SELECT print-file.
        ASSIGN TO OUTDATA.
DATA DIVISION.
   FILE SECTION.
   FD print-file.
   01 outrec    PIC X(125).
PROCEDURE DIVISION.
      OPEN OUTPUT print-file.
      WRITE outrec AFTER ADVANCING 2.
      CLOSE print-file.
      EXIT PROGRAM.
END PROGRAM B.
```

Contents of the initialization file (COBOL85.CBR)

```
[B]
OUTDATA PRTNAME
FUJITSU FMLBP 240DPI
```

When a local port name is specified

```
IDENTIFICATION DIVISION
   PROGRAM-ID.  B.
ENVIRONMENT DIVISION.
   INPUT-OUTPUT SECTION.
     FILE-CONTROL.
       SELECT print-file.
        ASSIGN TO OUTDATA.
DATA DIVISION.
   FILE SECTION.
   FD print-file.
   01 outrec    PIC X(125).
PROCEDURE DIVISION.
      OPEN OUTPUT print-file.
      WRITE outrec AFTER ADVANCING 2.
      CLOSE print-file.
      EXIT PROGRAM.
END PROGRAM B.
```

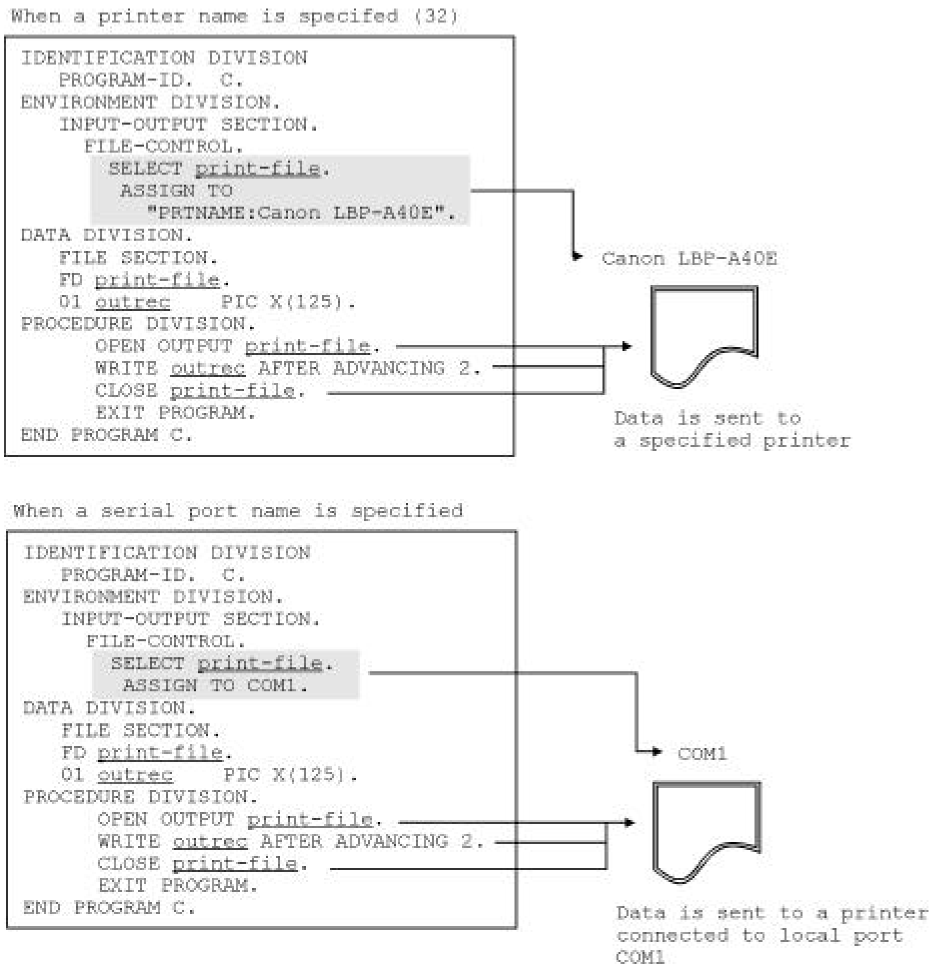
Contents of the initialization file (COBOL85.CBR)

```
[B]
OUTDATA LPT1
```

Figure 89. Data sent to a printer specified in the initialization file

When a File-Identifier Literal is Specified in the ASSIGN Clause

Data is sent to a printer specified in the file-identifier literal or to a printer connected to a local or serial port specified in the file-identifier literal.


When a printer name is specifed (32)
Canon LBP-A40E
Data is sent to
a specified printer
When a serial port name is specified
COM1
Data is sent to a printer
connected to local port
COM1

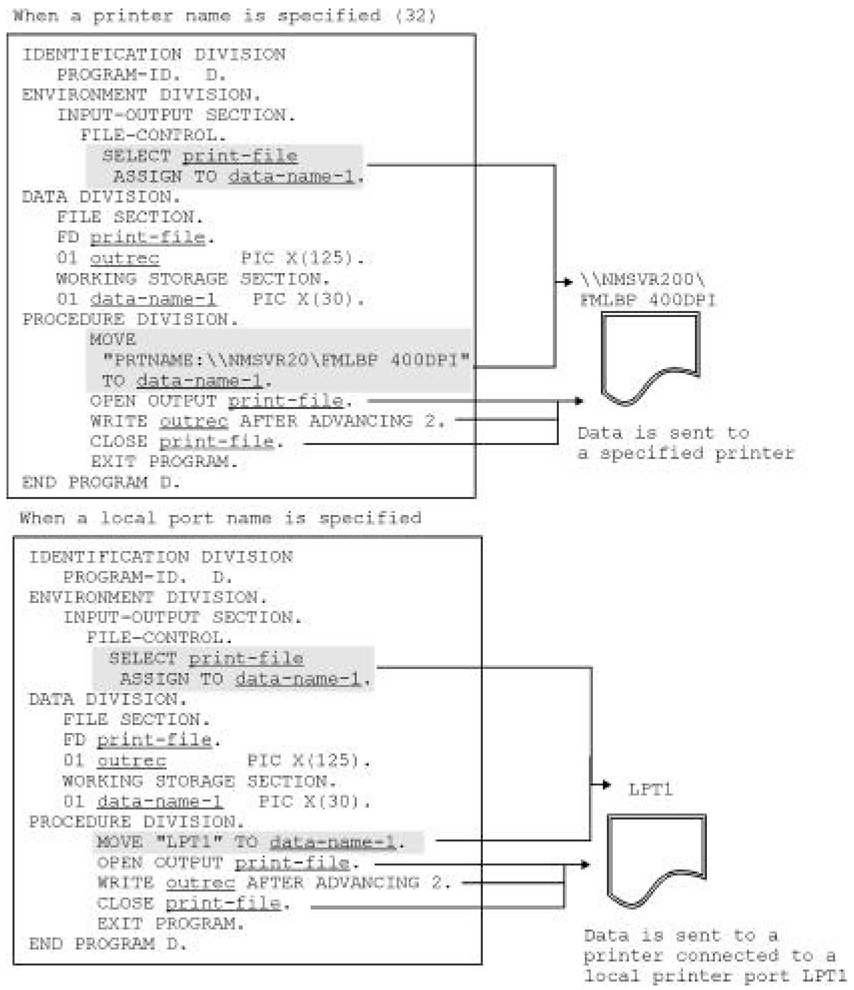
```
IDENTIFICATION DIVISION
   PROGRAM-ID.  C.
ENVIRONMENT DIVISION.
   INPUT-OUTPUT SECTION.
     FILE-CONTROL.
       SELECT print-file.
        ASSIGN TO
          "PRTNAME:Canon LBP-A40E".
DATA DIVISION.
   FILE SECTION.
   FD print-file.
   01 outrec    PIC X(125).
PROCEDURE DIVISION.
      OPEN OUTPUT print-file.
      WRITE outrec AFTER ADVANCING 2.
      CLOSE print-file.
      EXIT PROGRAM.
END PROGRAM C.
```

```
IDENTIFICATION DIVISION
   PROGRAM-ID.  C.
ENVIRONMENT DIVISION.
   INPUT-OUTPUT SECTION.
     FILE-CONTROL.
       SELECT print-file.
        ASSIGN TO COM1.
DATA DIVISION.
   FILE SECTION.
   FD print-file.
   01 outrec    PIC X(125).
PROCEDURE DIVISION.
      OPEN OUTPUT print-file.
      WRITE outrec AFTER ADVANCING 2.
      CLOSE print-file.
      EXIT PROGRAM.
END PROGRAM C.
```

Figure 90. Data sent to a printer specified in the file-identifier literal

When a Data-Name is Specified in the ASSIGN Clause

Data is sent to a printer specified in the data-name or to a printer connected to a local or serial port specified in the data-name. A file assignment error occurs if the data-name is left blank.


When a printer name is specified (32)
\\NMSVR200\ FMLBP 400DPI
Data is sent to a specified printer
When a local port name is specified
LPT1
Data is sent to a printer connected to a local printer port LPT1

```
IDENTIFICATION DIVISION
   PROGRAM-ID.  D.
ENVIRONMENT DIVISION.
   INPUT-OUTPUT SECTION.
     FILE-CONTROL.
       SELECT print-file
        ASSIGN TO data-name-1.
DATA DIVISION.
   FILE SECTION.
   FD print-file.
   01 outrec        PIC X(125).
   WORKING STORAGE SECTION.
   01 data-name-1    PIC X(30).
PROCEDURE DIVISION.
      MOVE
       "PRTNAME:\\NMSVR20\FMLBP 400DPI"
       TO data-name-1.
      OPEN OUTPUT print-file.
      WRITE outrec AFTER ADVANCING 2.
      CLOSE print-file.
      EXIT PROGRAM.
END PROGRAM D.
```

```
IDENTIFICATION DIVISION
   PROGRAM-ID.  D.
ENVIRONMENT DIVISION.
   INPUT-OUTPUT SECTION.
     FILE-CONTROL.
       SELECT print-file
        ASSIGN TO data-name-1.
DATA DIVISION.
   FILE SECTION.
   FD print-file.
   01 outrec        PIC X(125).
   WORKING STORAGE SECTION.
   01 data-name-1    PIC X(30).
PROCEDURE DIVISION.
      MOVE "LPT1" TO data-name-1.
      OPEN OUTPUT print-file.
      WRITE outrec AFTER ADVANCING 2.
      CLOSE print-file.
      EXIT PROGRAM.
END PROGRAM D.
```

Figure 91. Data sent to a printer specified in the data-name in the ASSIGN clause

Using Print File 2

This section explains how to use a form overlay pattern and FCB in a print file without a FORMAT clause.

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. program-name.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
  [function-name IS mnemonic-name-1]
   CTL          IS mnemonic-name-2.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
  SELECT file-name
   ASSIGN TO PRINTER
  [ORGANIZATION IS SEQUENTIAL]
  [FILE STATUS IS input-output-status].
DATA DIVISION.
 FILE SECTION.
 FD file-name
   [RECORD record-size-specification]
   [LINAGE IS logical-page-configuration-specification].

                                            { MODE-1                 }
 01  line-record-name  [CHARACTER TYPE IS   { MODE-2                 } ].
                                            { MODE-3                 }
 ┌─ Contents of the line record ─┐          { mnemonic-name-1        }
 │ 02  data-name-1... .           │
 │         :                      │
 └────────────────────────────────┘

01  control-record-name.

 ┌─Contents of the control record─┐
 │ 02  data-name-2 ... .           │
 │         :                       │
 └─────────────────────────────────┘
 WORKING-STORAGE SECTION.
 [01 input-output-status  PIC X(2).]
                                              { MODE-1              }
 [01 data-name-3        [CHARACTER TYPE IS    { MODE-2              } ].]
                                              { MODE-3              }
 PROCEDURE DIVISION.                          { mnemonic-name-1     }
   OPEN OUTPUT  file-name.
   WRITE  control-record-name ADVANCING mnemonic-name-2.
   WRITE  line-record-name    AFTER ADVANCING PAGE.
  [WRITE  line-record-name  [FROM data-name-3] [ AFTER ADVANCING ~].]
   CLOSE  file-name.
 END PROGRAM program-name.
```

Outline

Use a control record to employ a form overlay pattern with a print file.

As with ordinary data, a control record is created with a WRITE statement. If a control record is created with a form overlay pattern name specified, the data and form overlay pattern are overlaid on the subsequent page.

The size, font, form, direction, and space of a print character are defined with a COBOL program and form overlay pattern. For character settings, see “Print Characters,” and also refer to the “FORM V1.3 Manual.”

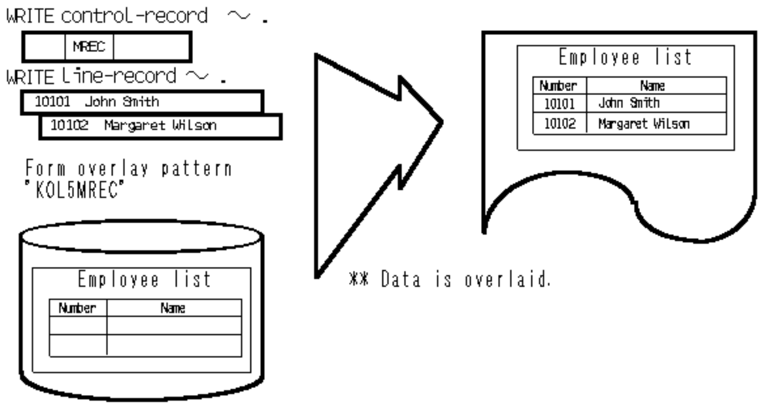


Figure 92. Defining print characters

Program Specifications

This section details program descriptions in COBOL divisions when forms are printed with form overlay patterns.

ENVIRONMENT DIVISION

In the ENVIRONMENT DIVISION, write the relation between the function-name and mnemonic-name, and define a print file.

Relating the Function-Name to the Mnemonic-Name

To enter a control record to the function-name CTL, relate the mnemonic-name. Specify this mnemonic-name in the WRITE statement when creating the control record.

When specifying a print character with a CHARACTER TYPE clause, relate the function-name indicating the size, font, form, direction, and space of a print character to the mnemonic-name. For function-name types, refer to the "COBOL85 Reference Manual."

Defining Print Files

Define a print file in a file control entry. For details of descriptions in a file control entry, see Table 23, "Information to be specified in a file control entry."

DATA DIVISION

In DATA DIVISION, define the record definitions and the definitions of data-names used in the ENVIRONMENT DIVISION.

Defining Records

Define a record in file and record description entries. For details of descriptions in a file description entry, see Table 24, "Information to be specified in the file description entry." Define the following records in the record description entry:

Line Records

Define a record to print data edited in a program. Multiple line records can be written.

The contents of one line record are printed sequentially from the left margin of the printable area. Specify the size of a line record so that it is written within the printable area.

The size of a print character can be specified in the CHARACTER TYPE clause in the data description entry. For contents that can be specified in the CHARACTER TYPE clause, see "Print Characters."

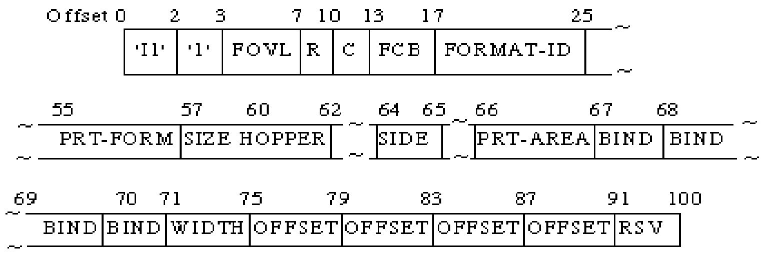
Control Records

Two types of control records are used for printing, I and S control records. The format of each control record is shown below.

For detailed explanations of the record areas and areas not explained in this manual, refer to the "COBOL85 Reference Manual."

As printer functions vary by model, refer to the users guide for each printer specified.

I Control Record



01	I-CONTROL-REC.		
	03 REC-ID	PIC X(2) VALUE "I1".	
	03 M	PIC X(1) VALUE "1".	
	03 FOVL	PIC X(4).	⇒ FOVL
	03 R	PIC 9(3).	⇒ R
	03 C	PIC 9(3).	⇒ C
	03 FCB	PIC X(4).	⇒ FCB
	03 FORMAT-ID	PIC X(8).	⇒ FORMAT-ID
	03	PIC X(30).	
	03 PRT-FORM	PIC X(2).	⇒ PRT-FORM
	03 FSIZE	PIC X(3).	⇒ SIZE
	03 HOPPER	PIC X(2).	⇒HOPPER
	03	PIC X(2).	
	03 SIDE	PIC X.	⇒ SIDE
	03	PIC X.	
	03 PRT-AREA	PIC X.	⇒ PRT-AREA
	03 BIND		⇒ BIND
	04	PIC X OCCURS 4 TIMES.	
	03 WIDTH	PIC 9(4).	⇒WIDTH
	03 OFFSET.		
	04	PIC 9(4) OCCURS 4 TIMES.	⇒ OFFSET
	03	PIC X(9) VALUE SPACE.	⇒ RSV

FOVL

Specify the name of the form overlay pattern to use. If an overlay group is specified, a single overlay is performed by using the first overlay pattern in the group.

R

Specify the number of times (0 to 255) that the form overlay pattern is printed. With this system, the same value as the number of copies is assumed.

C

Specify the number of copies (0 to 255) for each page.

FCB

Specify the applicable FCB name. Cannot be specified at the same time as FORMAT-ID.

FORMAT-ID

Specify the applicable name of the screen and form descriptors to use. Cannot be specified at the same time as FCB.

PRT-FORM

Specify a print format. The following values can be specified:

- P (portrait mode)
- L (landscape mode)
- LP (line printer mode)
- PZ (reduced printing portrait mode),
- LZ (reduced printing landscape mode)

PZ and LZ reduction specifications, however, are ignored. P and L specifications are used instead.

SIZE

Specify a forms size. The following sizes can be specified:
A3, A4, A5, B4, B5, and LTR.

You can dynamically specify a paper size to be used at run time. In the I control record, the FSIZE field is set to a user-defined string of three characters or less. The user-defined string replaces "XXX" in the environment variable @PRN_FormName_xxx which is called at run time.

Environment variable @PRN_FormName_xxx is set equal to a string that defines the paper size. The values for the paper size strings can be found in the Windows system printer defaults.

Refer to the method of specifying execution environment information in “Format of Run-time Environment Information” in Chapter 5.

Note: The user-defined string specifies a blank for this field when paper sizes are specified by using the print file with a FORMAT clause and specifies an actual paper size with the printer information file. For additional details, refer to the FORM RTS online help.

HOPPER

Specify a hopper to be used to feed forms. The following values can be specified:

- P1 (hopper 1)
- P2 (hopper 2)
- S (sub-hopper)
- P (arbitrary hopper)

The hopper cannot be changed by an application using the I control record because the system automatically selects a hopper according to the form size specified. (32)

SIDE

Specify a print side. The following values can be specified:

- F (single-sided printing)
- B (double-sided printing)

Single-sided printing is output, however, even when B is specified.

PRT-AREA

Specify whether the unprintable area is set (L) or not set (N). This specification has no effect, however, if the form descriptors are not used.

BIND

Specify the binding direction when multiple pages are outputted sequentially.

WIDTH

Specify the binding width from 0 through 9999 (unit: 1/1440 inch).

OFFSET

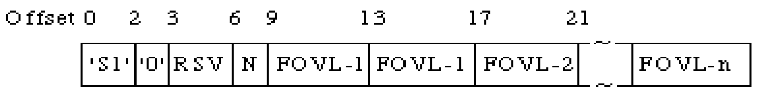
Specify the offset from 0 through 9999 (unit: 1/1440 inch).

RSV

Area used by the system. Enter a blank.

S Control Record

[S control record]



```
01  S-CONTROL-REC.
    03 REC-ID            PIC  X(2)  VALUE "S1".
    03 M                 PIC  X(1)  VALUE "0".
    03                   PIC  X(3).             ⇨  RSV
    03 N                 PIC  9(3).             ⇨  N
    03 FOVL-n            PIC  X(4).             ⇨  FOVL-n
      :
```

RSV	: Area used by the system. Enter a blank.
N	: Set the number of form overlay pattern names to be specified.
FOVL-n	: Set a form overlay pattern name. With this system, only the form overlay pattern name specified at the beginning is valid.

PROCEDURE DIVISION

Execute an input-output statement in the following sequence:

1. OPEN statement with OUTPUT specified: Starts printing.

2. WRITE statement: Outputs data.
3. CLOSE statement: Stops printing.

OPEN and CLOSE Statements

Use an OPEN statement once at the start of printing and a CLOSE statement once at the end of printing.

WRITE Statement

Use a WRITE statement when creating a control or line record.

A line record is written in the same manner as when using a WRITE statement to create data in line mode. See “Program Specifications.”

To write a control record, write the mnemonic-name related to the function-name CTL in the ADVANCING phrase.

To overlay data created by a line record with a form overlay pattern, write a control record with a form overlay pattern name specified.

To not overlay data with a form overlay pattern, write a control record with a blank entered as the form overlay pattern name.

After a control record is written, the format of the output page is set according to the contents of the control record. When a control record is written, however, no line record can be written to the current page. Thus, to write a line record immediately after a control record, the AFTER ADVANCING PAGE phrase must be specified.

```
      :
      :
    FILE SECTION.
    FD file-1.
     :
     :
    01 control-record.
     :
     :
      02 FOVL         PIC X(4).
     :
     :
    MOVE  "MREC"     TO FOVL.
    WRITE  control-record AFTER ADVANCING mnemonic-
           name.                                 (1)
    MOVE  SPACE      TO line-record.
    WRITE  line-record    AFTER ADVANCING PAGE.  (2)
    MOVE  101234     TO employee-number.
    MOVE  "Jack London" TO name.
    WRITE  line-record    AFTER ADVANCING 3.     (3)
    MOVE  105678     TO employee-number.
    MOVE  "Anne Miller" TO name.
    WRITE line-record    AFTER ADVANCING 2.      (4)
     :
     :
```

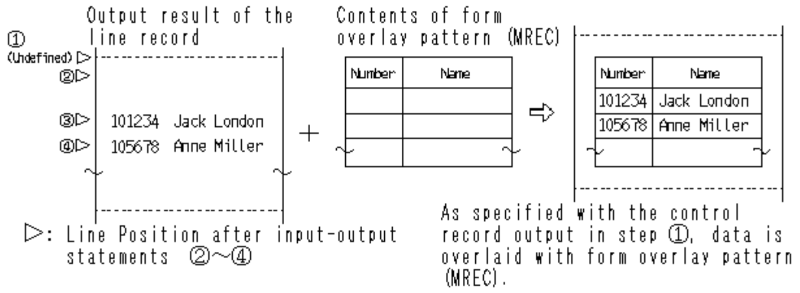

Output result of the line record
①
(Undefined) ▷
②▷
③▷
④▷
101234 Jack London
105678 Anne Miller
+
Contents of form overlay pattern (MREC)
Number
Name
⇨
Number
Name
101234 Jack London
105678 Anne Miller
▷: Line Position after input-output statements ②~④
As specified with the control record output in step ①, data is overlaid with form overlay pattern (MREC).

Figure 93. Flow of control with a form overlay pattern

Program Compilation and Linkage

There are no required compiler and linker options.

Program Execution

This section explains how to execute a program using a form overlay pattern and FCB.

Programs Using Form Overlay Patterns

When a form overlay pattern is used with a print file, the following settings are required:

- Specify the form overlay pattern storage directory for run-time environment information FOVLDIR. If the FOVLDIR setting is omitted, the form overlay pattern is not printed.
- If the extent of the form overlay pattern storage file is other than OVD, specify an extent for run-time environment information OVD_SUFFIX. If the file has no extent, specify "None".
- If the first four characters of the name of the form overlay pattern storage file is other than KOL5, specify the first four characters of the file name for run-time environment information FOVLTYP.

For details about how to set run-time environment information FOVLDIR, FOVLTYPE, and OVD_SUFFIX, refer to “Format of Run-time Environment Information” in Chapter 5. An example of writing an initialization file is shown below.

Contents of an initialization file when executing program A using a form overlay pattern

```
[A]
FOVLDIR=C:\FOVLDIR                  (1)
OVD_SUFFIX=                         (2)
FOVLTYPE=FOVL                       (3)
```

(1) Set the directory (C:\FOVLDIR) where the form overlay pattern is stored for run-time environment information FOVLDIR.

(2) The extent of the form overlay pattern storage file is OVD.

(3) The first four characters of the form overlay pattern storage file are KOL6.

Outputting Form Overlay Patterns

For printers supporting form overlay patterns:

The software overlay function provided by the Windows 95, Windows NT , and Windows 3.1 graphic device interface (GDI) is supported, so a form overlay pattern can be sent to printers having a Windows 95, Windows NT or Windows 3.1 printer driver.

For Windows 3.1 (16), use a COBOL85 run-time system information file.

COBOL85 Run-time System Information File (COB85RTS.CBR) (16)

A COBOL85 run-time system information file keeps information common to all the COBOL85 application run-time environments within one system. Use a text editor to change the contents of a COBOL85 run-time system information file.

The PRINTERSEQUENCE section is entered in a COBOL85 run-time system information file. The PRINTERSEQUENCE section defines a printer name and a mode setting for the printer (hardware setting).

The specification format of the PRINTERSEQUENCE section is shown below.

```
 [PRINTERSEQUENCE]
printer-name=[mode-type]
printer-name=[mode-type]
  (1)           (2)
```

[ ]: Optional

(1) Printer name: Specify the printer name displayed on the Windows Control Panel.

(2) Mode type: Specify the mode set for the printer (hardware setting). The mode type is indicated by a blank character or a uppercase alphabetic character. COBOL85 supports the following printer modes (orders):

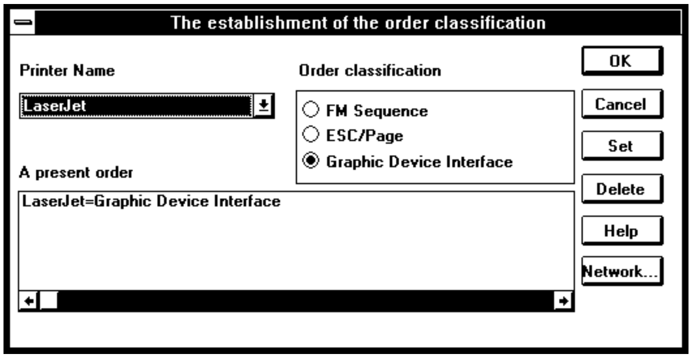
Mode (order)	**Mode Type**
FM	(For blank or default)
FM (image printing) (*1)	F
ESC / Page	E
LIPS III	L
Graphic device interface (GDI) (*2)	G

For the modes (orders) supported by the printer, refer to the manual included with the printer. If the printer name is omitted, FM is assumed as the mode (order).

*1 To use a form overlay pattern utilizing image printing with a printer supporting the FM sequence, specify F for the mode type.

*2 The result of a form overlay pattern output in the graphic device interface (GDI) mode depends on the printer specification. Print a form overlay pattern in FM mode using a printer supporting the FM sequence. This improves performance and printing quality, compared to when the graphic device interface (GDI) mode is used.

For information about how to use the tool, refer to the online help.



Printer Name: LaserJet

Order classification: ( ) FM Sequence ( ) ESC/Page [✓ Graphic Device Interface]

A present order: LaserJet=Graphic Device Interface

Figure 94. The establishment of the order of classification dialog box

When printing with a form overlay pattern is done in other than the graphic device interface (GDI) mode, a blank sheet of paper may be ejected, depending on the printer. In this case, if the printer can control "Print Without Data" on the printer operation panel, reset it to "Print Without Data".

Programs Using FCB

When FCB is used with a print file, write an FCB control statement in the initialization file.

For the specification format of an FCB control statement, see “Forms Control Buffers (FCB).” For details of an FCB control statement, refer to Appendix K, “FCB Control Statement.”

An error occurs if an FCB name is specified in the I control record, but no applicable FCB control statement is found in the initialization file.

An example of an FCB control statement is shown below.

Contents of the initialization file when executing program A with FCB1 set as the FCB name:

```
[A]
FCBFCB1=LPI((6,1),(12,4),(6,1),(12,2),(6,1))
```

The output format of one page set by a FCB control statement is shown below:

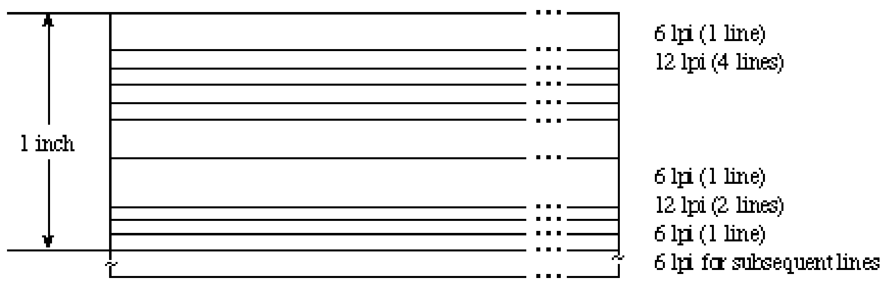


Figure 95. Format of an FCB control statement

Using Print Files with Form Descriptors

This section explains how to use partitioned form descriptors by using a print file with a FORMAT clause.

```
IDENTIFICATION    DIVISION.
 PROGRAM-ID.      program-name.
ENVIRONMENT      DIVISION.
 CONFIGURATION    SECTION.
 SPECIAL-NAMES.
  [function-name IS mnemonic-name.]
 INPUT-OUTPUT     SECTION.
 FILE-CONTROL.
  SELECT file-name
   ASSIGN TO file-reference-identifier
  [ORGANIZATION IS SEQUENTIAL]
   FORMAT IS form-descriptor-name-notification-area
   GROUP  IS item-group-name-notification-area
  [FILE STATUS  IS input-output-status-1 input-output-status-2].
DATA DIVISION.
FILE SECTION.
 FD  file-name
   [RECORD record-size]
   CONTROL RECORD IS control-record-name.
                                            ⎧ MODE-1        ⎫
 01  line-record-name  [CHARACTER TYPE IS   ⎨ MODE-2        ⎬ ].
┌─ Contents of line record ─┐               ⎪ MODE-3        ⎪
│  02  data-name-1... .     │               ⎩ mnemonic-name ⎭
│          :                │
└───────────────────────────┘
 01  control-record-name.
┌─ Contents of control record ─┐
│  02  data-name-2... .        │
│          :                   │
└──────────────────────────────┘
   COPY form-descriptor-name OF XMDLIB.

┌--- Expanding a COPY statement ---┐
¦01  chart-record-name.            ¦
¦   02  data-name-3... .           ¦
└----------------------------------┘
 WORKING-STORAGE   SECTION.
 01  form-descriptor-name-notification-area  PIC X(8).
 01  item-group-name-notification-area       PIC X(8).
[01  input-output-status-1                  PIC X(2).]
[01  input-output-status-2                  PIC X(4).]
                                          ⎧ MODE-1        ⎫
[01  data-name-4  [CHARACTER TYPE IS      ⎨ MODE-2        ⎬ ].]
                                          ⎪ MODE-3        ⎪
PROCEDURE        DIVISION.                ⎩ mnemonic-name ⎭
   OPEN OUTPUT file-name.
   MOVE form-descriptor-name TO form-descriptor-name-notification-area.
   MOVE item-group-name TO item-group-name-notification-area.
   WRITE chart-record-name [AFTER ADVANCING ...  ].
   WRITE line-record-name [FROM data-name-4] AFTER ADVANCING PAGE.
   WRITE control-record-name.
   CLOSE file-name.
 END PROGRAM program-name.
```

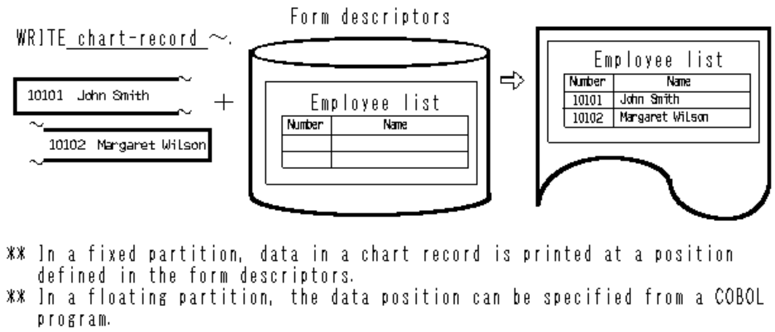
Outline

Use a chart record to print forms using partitioned form descriptors.

Define a partition (item group) designated in the form descriptors in a chart record. You do not have to write the definition statement of a chart record, as it can be fetched from the form descriptors with a COBOL COPY statement.

As with ordinary data, create a chart record with a WRITE statement. The size, font, form, direction, and space of print characters can be indicated in a COBOL program and form descriptors.

For character settings, see “Print Characters,” and the “FORM V1.3 Manual.”



** In a fixed partition, data in a chart record is printed at a position defined in the form descriptors.
** In a floating partition, the data position can be specified from a COBOL program.

Figure 96. Flow of control with form descriptors

Program Specifications

This section explains how to write programs that use form descriptors in a print file with a FORMAT clause.

ENVIRONMENT DIVISION

In the ENVIRONMENT DIVISION, write the relation between the function-name and mnemonic-name (print characters are indicated in the program) and define a print file.

Relating the Function-Name to the Mnemonic-Name

To specify a print character with a CHARACTER TYPE clause, relate the function-name indicating the size, form, direction, and space of a print character to the mnemonic-name. For function-name types and type styles, refer to the “COBOL85 Reference Manual.”

Defining Print Files

Define a print file in a file control entry. The following table lists information required to write a file control entry.

Table 25. Information to be specified in a file control entry

	Location	**Information Type**	**Details and Use of Specification**
Required	SELECT clause	File name	Write the name of a file to use in a COBOL program, conforming to the rules of COBOL user-defined words.
	ASSIGN clause	File-reference-identifier	Write a file-identifier, file-identifier literal, or data-name. Use a file reference code to assign a printer information file to be used by Form RTS at execution time.
	FORMAT clause	Data-name	Specify a data-name defined as an 8-byte alphanumeric data item in the WORKING-STORAGE or LINKAGE section. Use this data-name to set the form descriptor name.
	GROUP clause	Data-name	Specify a data-name defined as an 8-byte alphanumeric data item in the WORKING-STORAGE or LINKAGE section. Use this data-name to set the name of the item group defined in the form descriptor.

Table 25. Information to be specified in a file control entry (cont.)

	Location	**Information Type**	**Details and Use of Specification**
Optional	FILE STATUS clause	Data-name	Write the data-name defined as a 2-byte alphanumeric data item in the WORKING-STORAGE or LINKAGE section. The input-output execution result is set for this data-name. For value to be set, see Appendix B, “I-O Status List.” For detail information, specify a 4-bytes alphanumeric data item.

How you assign a printer information file at execution time depends on whether a file-identifier, file-identifier literal, or data-name is specified for a file-reference-identifier.

What you specify for a file-reference-identifier depends on when the name of the printer information file is determined. Specify a file-identifier literal if the name of the printer information file is determined during COBOL program generation and not changed afterwards. Specify a file-identifier if the name of the printer information file is undetermined during COBOL program generation or it is to be determined every program execution time. Specify a data-name to determine the name of the printer information file in a program.

DATA DIVISION

In the DATA DIVISION, define record definitions and the definitions of data used in the ENVIRONMENT DIVISION.

Define a record in file and record description entries. The following table lists information required to write a file description entry.

Table 26. Information to be specified in a file description entry

	Location	**Information Type**	**Details and Use of Specification**
Optional	RECORD clause	Record size	Define the size of the printable area.
	CONTROL RECORD clause	Control record name	Specify a control record name.

In a record description entry, line, control, and chart records can be defined. For information about how to define and use line and control records, see “Program Specifications.”

A record description statement defined in a chart record can be fetched from form descriptors with a COPY statement with XMDLIB specified as a library-name at compile time. For details on the record description statement to be expanded, see “Program Specifications.”

PROCEDURE DIVISION

Execute an input-output statement in the following sequence:

1. OPEN statement with OUTPUT specified: Starts printing.
2. WRITE statement: Writes data.
3. CLOSE statement: Stops printing.

OPEN and CLOSE Statements

Use an OPEN statement once at the start of printing and a CLOSE statement once at the end of printing.

WRITE Statement

In a WRITE statement, the line record, control record, and chart record can be output. The output of these records makes page 1 either a fixed form page or an irregular form page. In a fixed form page, the layout and chart record are defined by form descriptors and the line record can be printed. The irregular form page is not defined by form descriptors, therefore only the line record can be printed because it acts in the same manner as the print FORMAT phrase none file.

When the chart record and the line record exist together on a fixed form page, the form descriptors defining the chart record should be set in the CONTROL RECORD phrase of the file description paragraph.

When the control record has a blank immediately after the execution of an OPEN statement and the form descriptor name is output, the page becomes an irregular form page. In order to become a fixed form page, the form descriptors defining the chart record should be set in the FORMAT phrase of the file control description paragraph.

The fixed or irregular form of these pages is maintained on subsequent pages as long as a WRITE statement changing the form of the page is not executed.

Pages can be changed if new form descriptors are added. In this instance, AFTER ADVANCING is specified for a WRITE statement immediately after the output of the control record. See “Program Specifications” for the ADVANCING specification of a WRITE statement.

When executing a WRITE statement without the AFTER ADVANCING PAGE phrase, the first print line is not validated. In this case, printing is started from the first printable line of the printer.

Using Special Registers

Attributes of data items defined by form descriptors can be changed by using special registers.

There are three special registers that can be used.

EDIT-MODE

To specify whether or not output processing should occur.

EDIT-OPTION

To specify underlines and overstrikes.

EDIT-COLOR

To specify color.

These special registers are modified by the data name defined by the form descriptors. For example, to edit the color of data name A, use "EDIT-COLOR OF A". Refer to “Special Registers Used with Presentation File Module” in Appendix E for the values set in special usage of each register.

Note: A special register cannot be used by form descriptors where the item control part none is specified. Additionally, form descriptors where the item control part none is specified and the item control part *is* specified cannot be used together in a single program.

Program Compilation and Linkage

Compilation

Select the compiler option FORMLIB, and specify the directory name of the form descriptors storage file.

Linkage

There are no libraries that need to be linked.

Program Execution

To execute a program that uses form descriptors with a print file, FORM RTS requires a printer information file. For details about how to generate a printer information file, see “Generating Printer Information Files.”

The following environments must be set to execute a program that uses form descriptors with a print file:

- Add the directory containing FORM RTS to environment variable PATH.
- Generate a printer information file to be used by FORM RTS, and assign the file according to the contents of the ASSIGN clause. For information about how to assign a file, refer to “Assigning Files” in Chapter 7, because it is the same as assigning ordinary files. For details on a printer information file and how to generate it, refer to the FORM RTS online help.

When a printer information file is specified with a relative path name, it is retrieved in the following sequence:

1. Directory set for environment variable MEFTDIR
2. Current directory

The following is an example showing form descriptors used with a print file.

Contents of the ASSIGN clause in a COBOL program:

```
ASSIGN TO PRTFILE
```

Contents of the initialization file(COBOL85.CBR):

```
  :
PRTFILE=C:\DIR\MEFPRC
  :
```

Assign a printer information file to the file-identifier specified in the ASSIGN clause of a COBOL program.

To use a form overlay pattern in form descriptors, set the extents of the path and file names of the directory where the form overlay pattern is stored for the printer information file. In this case, the extents set in run-time environment information FOVLDIR and OVD_SUFFIX have no effect.

Using Presentation Files (Printing Forms)

This section explains how to print forms using a presentation file. For information about operation environments, refer to “Operation Environments” in Chapter 9, because they are the same as when executing screen input-output with a presentation file.

```
IDENTIFICATION DIVISION.
  PROGRAM-ID.  program-name.
ENVIRONMENT DIVISION.
  INPUT-OUTPUT SECTION.
  FILE CONTROL.
   SELECT file-name
   ASSIGN TO GS-file-reference-code
   SYMBOLIC DESTINATION IS "PRT"
   FORMAT IS form-descriptor-name-notification-area
   GROUP IS item-group-name-notification-area
   [PROCESSING MODE IS processing-type-notification-area]
   [UNIT CONTROL IS special-control-information-notification-area]
   [FILE STATUS IS input-output-status-1 input-output-status-2].
DATA DIVISION.
  FILE SECTION.
  FD file-name.
   COPY form-descriptor-name OF XMDLIB.

   Expanding a COPY statement
  01 chart-record-name.
    02 data-name-1 ... .

  WORKING STORAGE SECTION.
    01 form-descriptor-name-notification area        PIC X(8).
    01 item-group-name-notification-area             PIC X(8).
   [01 processing-type-notification-area             PIC X(2).]
   [01 special-control-information-notification-area PIC X(6).]
   [01 input-output-status-1                         PIC X(2).]
   [01 input-output-status-2                         PIC X(4).]
PROCEDURE DIVISION.
  OPEN OUTPUT file-name.
 [MOVE output-specification TO EDIT-MODE OF data-name.]
 [MOVE emphasis-specification TO EDIT-OPTION OF data-name.]
 [MOVE color TO EDIT-COLOR OF data-name.]
  MOVE form-descriptor-name TO form-descriptor-name-notification-area.
  MOVE item-group-name TO item-group-name-notification-area.
  WRITE chart-record-name.
  CLOSE file-name.
END PROGRAM program-name.
```

Outline

The presentation file module prints forms in the forms format defined with Power FORM (form descriptors). Form descriptors are generated with a screen image.

Data items defined in form descriptors can be included in a COBOL program at compile time by using a COBOL COPY statement. Thus, you do not have to write data item definitions for printing forms in a COBOL program.

The attributes of output data defined in form descriptors can be changed during execution of a program by using a COBOL special register.

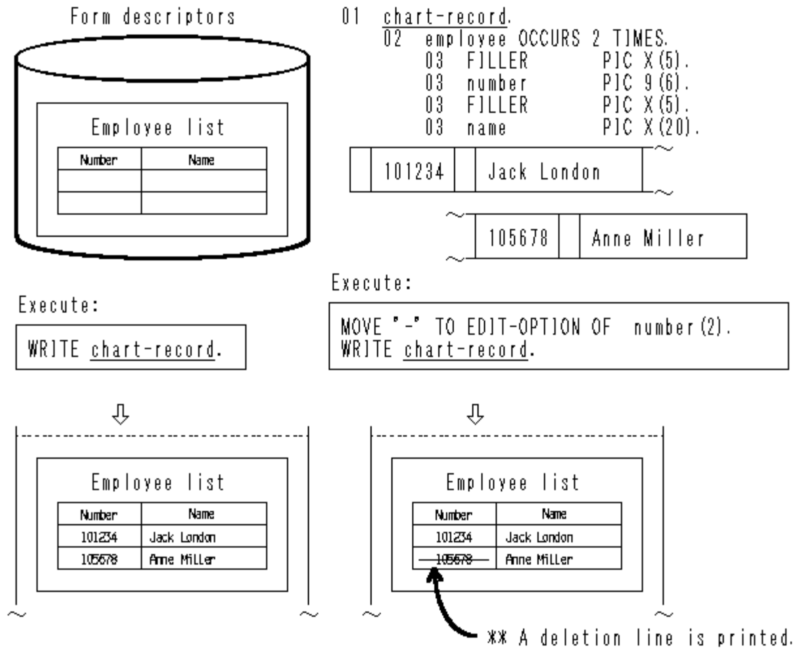


Figure 97. Using a COBOL special register to change output data

Work Procedures

To print forms with the presentation file module, form descriptors, COBOL source programs, and printer information files are required.

Generate form descriptors and COBOL source programs before compile time, and print information files before execution.

The following is the standard order of work procedures for printing forms with the presentation file module:

1. Generate form descriptors with FORM or Power FORM.
2. Generate COBOL source programs by using a text editor.
3. Compile and link COBOL source programs to generate executable programs.
4. Generate printer information files with a text editor.
5. Execute the executable programs.

Generating Form Descriptors

This section describes how to generate form descriptors used by the presentation file module.

For detailed FORM functions and how to use FORM, refer to the “FORM V1.3 Manual.”

The following table lists information to be set to generate form descriptors.

Table 27. Information to be set to generate form descriptors

Information Type		**Details and Use of Specification**
Required	File name	Specify the name of a file in which form descriptor is stored.
	Definition size	Specify the forms size with the numbers of lines and columns.
	Descriptors format	Specify the free format.
	Data item	Specify the data item to set print data. This item name is used as a data-name when writing a COBOL program.
	Item group	Put one or more item to be printed at a single run of print processing into one item group. This item group name is used when writing a COBOL program.
Optional	Item control field	Specify a 5-byte item control field if the contents of form descriptors need to be changed with a special register in a COBOL program.

The item control field is information appended to data items defined in form descriptors, and is classified into three input-output options:

- Sharable (3 bytes)
- Non-sharable (5 bytes)
- None

When a special register in a COBOL program is used, 5 bytes must be specified.

Program Specifications

This section explains how to write programs when printing forms with the presentation file module for each COBOL division.

ENVIRONMENT DIVISION

ENVIRONMENT DIVISION defines a presentation file. In the presentation file, as with defining an ordinary file, write a file control entry in the FILE-CONTROL paragraph of the INPUT-OUTPUT section.

The following table lists the contents to be written in the file control entry. These information values can be determined regardless of the contents of form descriptors generated with FORM or Power FORM.

Table 28. Information to be specified in a file control entry

	Location	**Information Type**	**Details and Use of Specification**
Required	SELECT clause	File name	Write the name of a file to use in a COBOL program, conforming to the rules of COBOL user-defined words.
	ASSIGN clause	File-reference-identifier	Specify this item in the format of "GS- file-identifier". This file-identifier is the environment variable to set the path name of the printer information file used by the connection product at execution.
	FORMAT clause	Data-name	Specify a data-name defined as an 8-byte alphanumeric data item in the WORKING-STORAGE or LINKAGE section. For this data-name, specify the name of form descriptors during forms printing.
	GROUP clause	Data-name	Specify a data-name defined as an 8-byte alphanumeric data item in the WORKING-STORAGE or LINKAGE section. For this data-name, specify the item group name to be output during forms printing.
	SYMBOLIC DESTINATION clause	Specification of the output destination	Specify "PRT".

Table 28. Information to be specified in a file control entry (cont.)

	Location	**Information Type**	**Details and Use of Specification**
Optional	FILE STATUS clause	Data-name	Write the data-name defined as a 2-byte alphanumeric data item in the WORKING-STORAGE or LINKAGE section. The input-output execution result is set for this data-name. For value to be set, see Appendix B, “I-O Status List.” For detail information, specify a 4-byte alphanumeric data item.
	PROCESSING MODE clause	Data-name	Specify the data-name defined as a 2-byte alphanumeric data item in the WORKING-STORAGE or LINKAGE section. For this data-name, set in input-output processing type during forms input-output. See Table 29.
	UNIT CONTROL clause	Data-name	Specify the data-name defined as a 6-byte alphanumeric data item in the WORKING-STORAGE or LINKAGE section. For this data-name, set the control information of input-output processing at printing. See Table 29.

Table 29. Input-output processing types and specification values

Processing Type	**Value**	**Control Information**	
Printer control	"CT"	Form feed	"PAGE"
Partition output	"PW"	Output after feeding nnn lines	"Annn"
		Feed nnn lines after output	"Bnnn"
		Output at line number nnn	"Pnnn"
Line movement output	"FW"	Move backward by nnn lines	"Annn"
		Move forward by nnn lines	"Snnn"

DATA DIVISION

In the DATA DIVISION, write chart record definitions and the definitions of data-names specified in the file control entry.

A record description statement defined in a chart record can be fetched from form descriptors with a COPY statement with XMDLIB specified as a library-name. For details of the record description statement to be expanded, see “Program Specifications.”

PROCEDURE DIVISION

Like ordinary file processing, use input-output statements for printing forms. Execute input-output statements in the following sequence:

1. OPEN statement with I-O specified: Starts printing.
2. WRITE statement: Outputs data.
3. CLOSE statement: Stops printing.

OPEN and CLOSE Statements

Use an OPEN statement once at the start of printing and a CLOSE statement once at the end of printing.

WRITE Statement

One WRITE statement prints one form.

The name of form descriptors used for printing must be set for the data-name specified in the FORMAT clause before executing a WRITE statement.

With a WRITE statement, data items in an item group set for the data-name specified in a GROUP clause are eligible for printing. By setting a value in a special register before executing a WRITE statement, the data item attributes can be changed. For details about how to use a special register, see “Program Specifications.”

Program Compilation and Linkage

Compiling

Select compiler option FORMLIB, and specify the path name to the directory of the file containing form descriptors.

Linking

There are no libraries that need to be linked.

Generating Printer Information Files

This section describes how to print forms with the presentation file module. For details on printer information files and how to generate them, refer to the FORM RTS online help.

The following table lists information to be entered in a printer information file.

Table 30. Information to be entered in a printer information file

Information Type	Details and Use of Specification
PRTDRV	Specify the device name of the output printer device.
PRTDEV	Specify the printer model name of the output printer device.
MEDDIR	Specify the path name to the directory containing form descriptors.
MEDSUF	Specify the extension of form descriptor files. When the specification of the extension is omitted, the default value of FORM RTS is used.

Program Execution

The following environments must be set to execute a program that prints forms with the presentation file module:

- When using FORM RTS:
  - Add the directory containing FORM RTS to environment variable PATH.
  - Set the name of the printer information file with the file-identifier as an environment variable name.

    When a printer information file is specified with a relative path name, it is retrieved in the following sequence:

1. Directory set for environment variable MEFTDIR
2. Current directory

- When using MeFt/NET (32):
  - Add the directory containing MeFt/NET-SV to environment variable PATH.
  - Set the printer information file name with the file-identifier as an environment variable name. The actual printer information file must be prepared for Windows 3.1.
  - Specify an environment variable indicating that MeFt/NET is used as a connector. There are two specification methods:

- Set "MEFTNET" for the connector name with the file-identifier as an environment variable name.
- Set "MEFTNET" for environment variable @CBR_PSFILE_PRT.

For details on how to set MEFTNET, refer to “Format of Run-time Environment Information” in Chapter 5.

Examples of how to print forms with the presentation file module are shown below.

Contents of the ASSIGN clause in a COBOL program:

```
ASSIGN TO GS-PRTFILE
```

Contents of the initialization file (COBOL85.CBR):

(When using FORM RTS)

```
  :
PRTFILE=MEFPRC
  :
```

(When using MeFt/NET (32))

```
  :
PRTFILE=MEFPRC,MEFTNET
  :
```

Assign a printer information file to the file-identifier specified in the ASSIGN clause of a COBOL program. In the example using FORM RTS, the printer information file is retrieved from the directory set for environment variable MEFTNETDIR because the file is assigned with a relative path specification.

Chapter 9. Input-Output Using Screens

This chapter explains how to display screens and enter data from the displayed screens. Chapter 9 also describes how to use presentation files, work procedures, and the screen handling function.

Types of Input-Output Using Screens

A COBOL program allows you to display a screen where you can enter data. This is referred to as screen input-output. The following two types of screen input-output functions are provided:

- Presentation file function
- Screen handling function

The following table lists the features and uses of the presentation file and screen handling functions.

Table 31. The features and uses of the presentation file and screen handling functions

Feature and Use		**Presentation File Function**	**Screen Handling Function**
Feature	Screen design	Designs a screen based on screen image (using FORM).	Designs a screen as a collection of lines.
	Number of screens	Defines the number of presentation files in a program.	One
	Changing screen attributes during program execution	Enabled	Enabled
	Program specification contents	- Presentation file definition - Presentation record definition - OPEN statement - READ statement - WRITE statement - CLOSE statement	- Screen definition - ACCEPT statement - DISPLAY statement
	Related products	FORM (screen definition) FORM RTS (LOCAL Input-Output processing) MeFt / NET (REMOTE Input-Output processing)	None
Use		Used for screen input-output with complex screens (for example, form, slip)	Used for screen input-output simple screens

Using Presentation Files (Screen Input-Output)

This section outlines how the presentation file module is used for screen input-output, explains how to establish operation environments, and describes how to generate screen descriptors and COBOL source programs. For sample programs of screen input-output using the presentation file module, refer to the "Getting Started with Fujitsu COBOL" guide.

Outline

The presentation file module performs screen input-output using screens (screen descriptors) defined with FORM. Screen descriptors are generated with a screen image with FORM.

Data items for input-output processing defined in the screen descriptors can be included in a COBOL program at compile time with a COBOL COPY statement. Therefore, you do not specify data item definitions for input-output processing in a COBOL program.

The attributes of data defined in the screen descriptors can be changed during execution of a program with a COBOL special register.

Operation Environments

To use the presentation file (screen input-output) module, screen descriptors generated with FORM are required. The following figure shows how an executable file is generated.

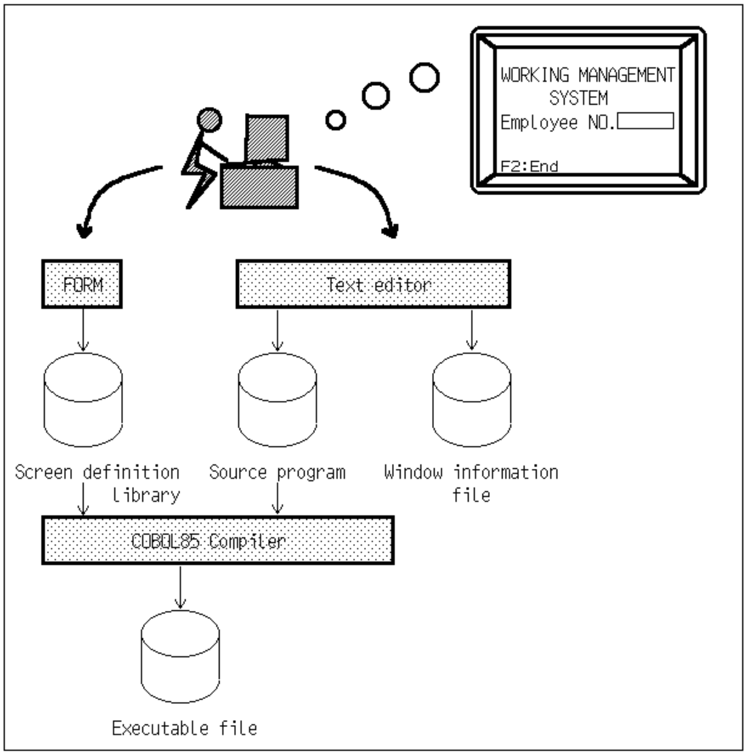


Figure 98. Generating a program using the presentation file module

To use the presentation file function, both the screen descriptors and FORM RTS are required.

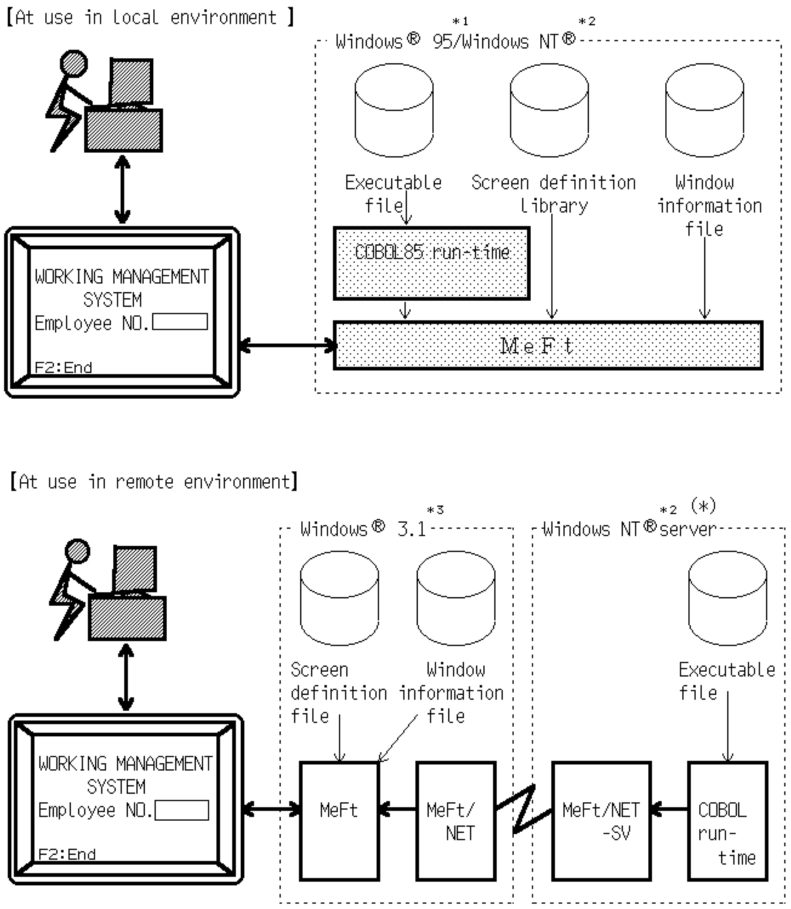


Figure 99. Operating a program using the presentation file module

*1 Besides Windows NT, the function is available with UXP / DS, Sun and HP-UX. For UXP / DS, FORM RTS is also required

Work Procedures

To perform screen input-output with the presentation file module, screen descriptors, COBOL source programs, and window information files are required. The screen descriptors and COBOL source programs must be generated before compilation, and window information files must be generated before execution.

The following is the standard work procedure of screen input-output with the presentation file module:

1. Generate screen descriptors with FORM.
2. Generate COBOL source programs with a text editor.
3. Compile and link COBOL source programs to generate executable programs.
4. Generate window information files with a text editor.
5. Execute the programs.

Generating Screen Descriptors

This section describes how to generate screen descriptors used by the presentation file module. For detailed FORM functions and how to use FORM, refer to the “FORM V1.3 Manual.”

The following table lists information to be entered for screen descriptors.

Table 32. Information to be entered for screen descriptors

Information Type		**Details and Use of Specification**
Required	File name	Specify the name of a file in which screen descriptor is stored.
	Definition size	Specify the screen size with the numbers of lines and columns.
	Descriptors format	Specify the free format.
	Data item	Specifies the area for screen input-output. The item name specified here is used as a data-name in the COBOL program specifications.
	Item group	Groups one or more items (displayed or entered once per input-output group. The group name specified here is used in the COBOL program specifications.
Optional	Item control field	Specify a 5-byte item control field if the contents of screen descriptors need to be changed with a special register in a COBOL program.
	Attention information	Specifies to decide an input key in the COBOL program.

The item control field is information appended to data items defined in screen descriptors, and is classified into three options:

- Sharable (3 bytes)
- Non-sharable (5 bytes)
- None (for input and output)

Program Specification

This section explains how to use the presentation file module for each COBOL division.

ENVIRONMENT DIVISION

The presentation file is defined in the ENVIRONMENT DIVISION.

To define the presentation file, specify a file control entry in the FILE-CONTROL paragraph of the INPUT-OUTPUT section. This is the same as when you define an ordinary file.

The following table lists the contents to be written in the file control entry. These values can be determined regardless of the definition contents of screen descriptors generated with FORM.

Table 33. Information to be specified in a file control entry

	Location	**Information Type**	**Details and Use of Specification**
Required	SELECT clause	File name	Specifies the presentation file name to be used in the COBOL program This file name must conform to the COBOL user-defined word rules.
	ASSIGN clause	File-reference identifier	Specify this item in the format of "GS- file-identifier". This file-identifier is the environment variable to set the path name of the printer information file used by FORM RTS at execution.
	FORMAT clause	Data-name	Specify a data-name defined as an 8-byte alphanumeric data item in the WORKING-STORAGE or LINKAGE section. For this data-name, specify the name of screen descriptors during screen input-output.
	GROUP clause	Data-name	Specify a data-name defined as an 8-byte alphanumeric data item in the WORKING-STORAGE or LINKAGE section. For this, specify the item group name to be input or output.

Table 33. Information to be specified in a file control entry (cont.)

	Location	**Information Type**	**Details and Use of Specification**
Optional	SYMBOLIC DESTINATION clause	output destination	Specify "DSP." (This clause can be omitted, since "DSP" is the default value of this clause.)
	FILE STATUS clause	Data-name	Specify a data-name defined as a 2-byte alphanumeric data item in the WORKING-STORAGE or LINKAGE section. The execution results of input-output processing are set in this data name. For value to be set, see Appendix B, “I-O Status List.” For detail information, specify a 4-byte alphanumeric data item.
	PROCESSING MODE clause	Data-name	Specify a data-name defined as a 2-byte alphanumeric data item in the WORKING-STORAGE or LINKAGE section. The processing class of screen input- output processing is set in this data-name. See Table 34.
	SELECTED FUNCTION clause	Data-name	Specify a data-name defined as a 4-byte alphanumeric data item in the WORKING-STORAGE or LINKAGE section. The attention information returned upon READ statement completion is set in this data-name. See Table 35.
	UNIT CONTROL clause	Data-name	Specify a data-name defined as a 6-byte alphanumeric data item in the WORKING-STORAGE or LINKAGE section. For this data-name, specify the unit control information during screen input-output. See Table 34.

Table 34. Input-output processing types and specification values

	Processing Mode		**Control Information**	
Input	Ordinary entry	Blank	None	
	Non-erasing entry	"NE"		
	Entry after alarm	"AL"		
	Full screen erasing entry	"CL"		
	Change notification entry	"NI"		
	Change notification entry after alarm	"AI"		
Output	Normal output	Blank	None	
	No erasing output	"CL"		
	Selection of menu item disability	"PF"	Menu item	Attention information
	Selection of menu item	"PN"		
	Terminal control output	"CT"	Start preentry	"BSTR"
			End preentry	"BSTP"
			Clear preentry buffer	"BCLR"
			Buzzer type: High tone	"BZ1"
			Buzzer type: Low tone	"BZ2"

Table 35. Attention information values and attention types

Attention Information Value	**Attention Type**
Customized values (*1)	Value specified by user at screen definition
"C000"	Clear key
"E000"	Execution key (*2)
"E000"	Data full key
"E000"	Item escape key

*1 Includes such items as function keys and menu items.

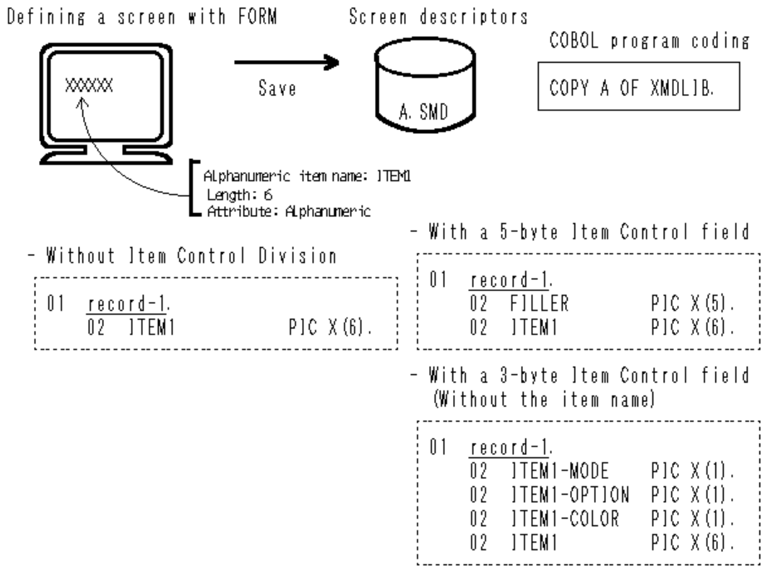
*2 This key is assigned as the for window information files. For details, refer to the FORM RTS online help.

DATA DIVISION

In DATA DIVISION, presentation record definitions and data items specified in the file control entry are defined.

A record description statement defined in a presentation record can be fetched from screen descriptors with a COPY statement with XMDLIB specified as a library-name.

The details of the record description statement to be expanded are explained below.



- Without Item Control Division

```
01  record-1.
    02  ITEM1            PIC X(6).
```

- With a 5-byte Item Control field

```
01  record-1.
    02  FILLER         PIC X(5).
    02  ITEM1          PIC X(6).
```

- With a 3-byte Item Control field (Without the item name)

```
01  record-1.
    02  ITEM1-MODE     PIC X(1).
    02  ITEM1-OPTION   PIC X(1).
    02  ITEM1-COLOR    PIC X(1).
    02  ITEM1          PIC X(6).
```

Figure 100. Expanding a record description statement

PROCEDURE DIVISION

The input-output statements are used for screen input-output processing the same as for ordinary file processing.

Input-output processing is executed in the following order:

1. OPEN statement with I-O phrase specified: Starts screen input-output processing.
2. READ and WRITE statements: Screen input-output processing.
3. CLOSE statement: Stops screen input-output processing.

OPEN and CLOSE Statements

Use an OPEN statement once at the start of screen input-output processing and a CLOSE statement once at the end of screen input-output processing.

READ or WRITE Statements

To display a screen, use a WRITE statement with a presentation record specified.

To read data from a screen, use a READ statement with a presentation record specified.

Before executing a WRITE statement, the screen descriptors name must be set for a data-name specified in a FORMAT clause, and an item group name be set for a data-name specified in a GROUP clause.

Data items in an item group set for a data-name specified in a GROUP clause are eligible for input-output.

By setting values in special registers before executing a WRITE or READ statement, the attributes of data items in presentation records can be changed. For values to be set for special registers, refer to “Special Registers Used with the Presentation File Module” in Appendix E.

If an error is found in input data while executing a READ statement, and a re-input request order has been defined in screen descriptors, screen display and input editing to enable re-input are repeated until the error is cleared.

When no error is found or no re-input request order has been defined, the input edit result is posted to a COBOL program. In addition, attention information is posted to a data-name specified in a SELECTED FUNCTION clause.

After executing a READ statement, the input result is returned to special register EDIT-STATUS. For the values to be set, refer to “Special Registers Used with the Presentation File Module” in Appendix E.

How to Use Special Registers

The attributes of data items defined with screen descriptors can be changed with special registers. Multiple screen descriptors with control fields that have different sizes cannot coexist. The five types of special registers are as follows:

- EDIT-MODE: Specify whether items are eligible for output processing.
- EDIT-OPTION: Specify emphasis, underline, and reverse display.
- EDIT-COLOR: Specify a color.
- EDIT-STATUS: Specify whether items are eligible for input processing. In addition, input results are posted.
- EDIT-CURSOR: Specify a cursor position.

Use these special registers with modification of data-names defined with screen descriptors. For example, write EDIT-COLOR OF A for the color of data-name A.

For the values to be set for special registers, refer to “Special Registers Used for the Presentation File Module” in Appendix E.

With 3-byte screen and form descriptors where item control fields are shared in input and output processing, EDIT-MODE, EDIT-STATUS, EDIT-CURSOR, and EDIT-OPTION use the same storage area for items defined in the screen and form descriptors. For example, EDIT-MODE OF A and EDIT-STATUS OF A, and EDIT-CURSOR OF A and EDIT-OPTION OF A both use the same storage area.

Program Compilation and Linkage

Compilation

Select compiler option FORMLIB, and specify the path name to the directory containing the screen descriptors file.

Linkage

No libraries need to be linked.

Generating Window Information Files

This section describes how to generate a window information file to perform screen input-output processing with the presentation file module. For details on window information files and how to generate them, refer to the FORM RTS online help. The following table lists information to be set in a window information file.

Table 36. Information to be set in a window information file

	Information Type	**Details and Use of Specification**
Required	MEDDIR	Specify the path name to the directory containing screen descriptors.
Optional	MEDSUF	Specify the extension of the file containing screen descriptors. When the specification of the extension is omitted, the default value of FORM RTS is used.
	MEDCNT	Specify the number of registrations of screen descriptors. The default value is 10.

Program Execution

The following environments must be set to execute a program that performs screen input-output processing using the presentation file module:

- When using FORM RTS:
  - Add the directory containing FORM RTS to environment variable PATH.
  - Set the name of the window information file with the file-identifier as an environment variable name.

  When a window information file is specified with a relative path name, it is retrieved in the following sequence:

1. Directory set for environment variable FORM RTSDIR
2. Current directory

- When using MeFt/NET (32):
  - Add the directory containing MeFt/NET-SV to environment variable PATH.
  - Set the name of the window information file with the file-identifier as an environment variable name. The window information file must be prepared for Windows 3.1. The name of the window information file must be the same as the file-identifier defined in the program.
  - Specify an environment variable indicating that MeFt/NET is used as a connector. There are the following two specification methods:
- Set "MEFTNET" for the connector name with the file-identifier as an environment variable name.

- Set "MEFTNET" for environment variable @CBR_PSFILE_PRT.

For details on how to set MEFTNET, see “Environment Variables” in Chapter 5.

The following example shows how to perform screen input-output with the presentation file module.

Contents of the ASSIGN clause in a COBOL program:

```
ASSIGN TO GS-DSPFILE
```

Contents of the initialization file:

(When using FORM RTS)

```
  :
  :
DSPFILE=C:\DIR1\MEFWRC
  :
  :
```

(When using MeFt / NET (32))

```
  :
  :
DSPFILE=MEFWRC,MEFTNET
  :
  :
```

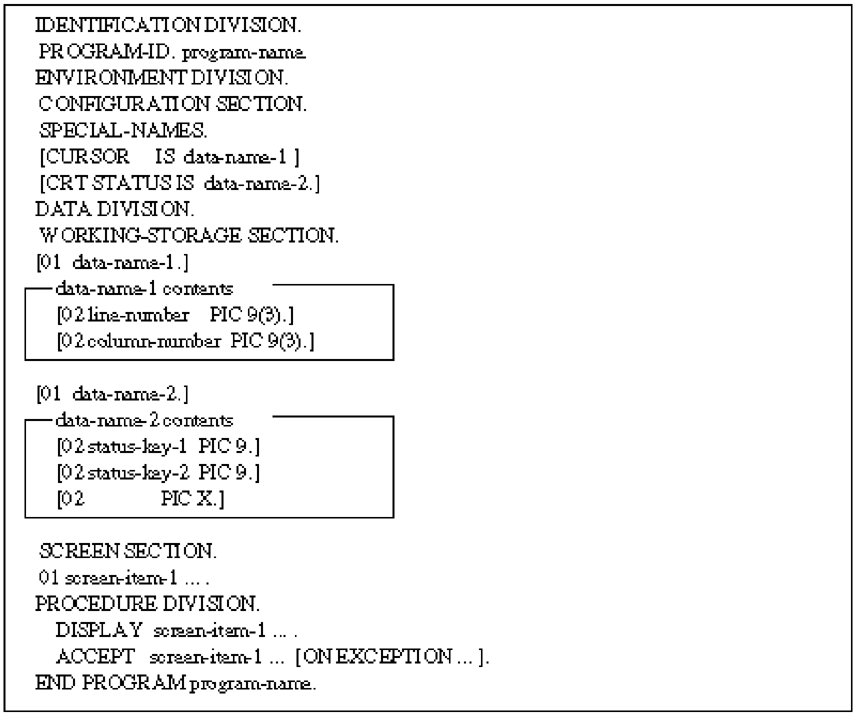
Assign a window information file to the file-identifier specified in the ASSIGN clause of a COBOL program.

Display File Input Interruption

When MeFt is called from other applications, the system waits for input and the display can be interrupted. (This is true only when MeFt is used). Refer to “COBOL-Supported Subroutine” in Appendix N for details.

Using the Screen Handling Function

This section explains how to perform screen input-output using the screen handling function. For sample programs using the screen handling function, refer to the “Getting Started with Fujitsu COBOL” guide.

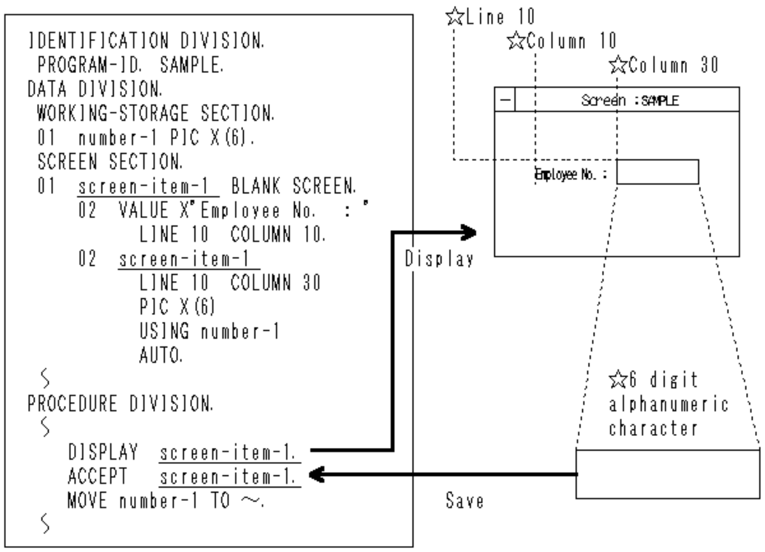


```
IDENTIFICATION DIVISION.
 PROGRAM-ID. program-name.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
 [CURSOR    IS data-name-1 ]
 [CRT STATUS IS  data-name-2.]
DATA DIVISION.
 WORKING-STORAGE SECTION.
[01  data-name-1.]
  data-name-1 contents
   [02 line-number    PIC 9(3).]
   [02 column-number  PIC 9(3).]

[01  data-name-2.]
  data-name-2 contents
   [02 status-key-1  PIC 9.]
   [02 status-key-2  PIC 9.]
   [02               PIC X.]

 SCREEN SECTION.
 01 screen-item-1 ... .
PROCEDURE DIVISION.
   DISPLAY  screen-item-1 ... .
   ACCEPT   screen-item-1 ...  [ON EXCEPTION ... ].
END PROGRAM program-name.
```

Outline

The screen handling function displays a screen with a DISPLAY statement and inputs data from a screen with an ACCEPT statement.

The screen layout is defined in a screen data description entry in the SCREEN section in the DATA DIVISION. A screen item defined in the SCREEN section is arranged on the screen with the line and column numbers.



```
IDENTIFICATION DIVISION.
 PROGRAM-ID. SAMPLE.
DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  number-1 PIC X(6).
 SCREEN SECTION.
 01  screen-item-1  BLANK SCREEN.
     02  VALUE X"Employee No.   : "
           LINE 10  COLUMN 10.
     02  screen-item-1
           LINE 10  COLUMN 30
           PIC X(6)
           USING number-1
           AUTO.
  ∫
PROCEDURE DIVISION.
  ∫
    DISPLAY  screen-item-1.
    ACCEPT   screen-item-1.
    MOVE number-1 TO ～.
  ∫
```

Figure 101. Screen item defined in the SCREEN section

Screen Windows

One screen window to perform screen input-output with the screen handling function is generated per run unit. Screen windows are generated by the first ACCEPT or DISPLAY statement executed, and closed when the run unit is terminated.

Although the size of the logical screen of this window is normally 24 lines and 80 columns, it can be changed with run-time environment information @ScrnSize.

You can specify whether to close this window automatically or close after message confirmation with run-time environment information @WinCloseMsg.

The following table lists window attributes that can be changed at execution time. For details about how to specify run-time environment information, see “Setting Run-time Environment Information” in Chapter 5.

Table 37. Changing window attributes for screen handling

Attribute	**Run-time Environment Information**	**Setting Value**	**Meaning**
Automatic window closing	@WinCloseMsg (*1)	ON	Closes the window after displaying a message.
		OFF	Closes the window without displaying a message.
Window size	@ScrnSize	(m,n)	Specifies the size of a logical screen with the number of columns (m) and lines (n).

*1 This run-time environment information is also valid for console windows used with the ACCEPT/DISPLAY function.

User-Defined Function Keys

The screen handling function invalidates the input of specific function keys and corresponding processing in programs to specified function keys.

To use a function key when executing a program, define the function key with the run-time environment variable TERMINATOR. Data input from the screen cannot be ended by using a function key invalidated by the run-time environment variable TERMINATOR.

In a program, information on the input function key is set for data items specified in the CRT STATUS clause in the SPECIAL-NAMES paragraph. For details about how to set the run-time environment variable TERMINATOR, refer to “Setting Run-time Environment Information” in Chapter 5.

Program Specification

This section explains how to write programs using the screen handling function for each COBOL division.

ENVIRONMENT DIVISION

The following information can be entered in the SPECIAL-NAMES paragraph:

- Data items for setting or receiving cursor positions in the CURSOR clause.
- Data items for receiving screen input-output status in the CRT STATUS clause. The following table lists values set for these data items.

Table 38. Screen input-output status values

Status Key 1 (1st Character)	**Status Key 2 (2nd Character)**	**Meaning**
"0"	"0"	A termination key was entered by the operator.
	"1"	The last item was entered.
"1"	0x00-0xff	A user-defined function key was entered. (A function key number is set for status key2.) (*1)
"2"	0x00-0xff	A system-defined function key was entered. (A function key number is set for status key2.) (*2)
"9"	0x00	No input item was found. (Error)

*1 A user-defined function key is a function key defined using the run-time environment variable TERMINATOR.
*2 A system-defined function key is a function key not defined using the run-time environment variable TERMINATOR.

DATA DIVISION

The SCREEN section is written at the end of the DATA DIVISION.

The SCREEN section has literal, input, output, update, and input-output items. These items are classified based on coding of the screen data description entry.

The following table lists relationships between screen item attributes and clauses that can be specified in the screen data description entry. For information about how to code the screen data description entry, refer to the “COBOL85 Reference Manual.”

Data items defined in the BASED-STORAGE section cannot be specified in the SCREEN section.

Table 39. COBOL clauses specified for screen items

Purpose		**COBOL Clause**	**Screen Item Attribute**				*1		*2
			L	I	O	U	IO	G	E
Emphasis	Display in high intensity	HIGHLIGHT clause	o	o	o	o	o	x	o
	Display in low intensity	LOWLIGHT clause	o	o	o	o	o	x	o
	Display with blinking	BLINK clause	Δ	Δ	Δ	Δ	Δ	x	Δ
	Display with underline	UNDERLINE clause	o	Δ	o	o	o	x	o
Color	Specify background color	BACKGROUND-COLOR clause	o	o	o	o	o	o	o
	Specify foreground color	FOREGROUND-COLOR clause	o	o	o	o	o	o	o
	Reverse background and foreground colors	REVERSE-VIDEO clause	o	o	o	o	o	x	o
Sound	Sound an audio tone	BELL clause	o	Δ	o	o	o	x	o
Expression format	Display a blank for zero	BLANK WHEN ZERO clause	x	Δ	o	o	o	x	o
	Specify justification	JUSTIFIED clause	x	o	o	o	o	x	o
	Specify operation sign position	SIGN clause	x	o	o	o	o	o	o

Table 39. COBOL clauses specified for screen items (cont.)

Purpose	**COBOL Clause**	**Screen Item Attribute**					*1		*2
Display method	Specify full-screen erasure	BLANK SCREEN clause	o	Δ	o	o	o	o	o
	Specify partial screen erasure	ERASE EOS clause	o	Δ	o	o	o	x	o
	Specify full-line erasure	BLANK LINE clause	o	Δ	o	o	o	x	o
	Specify partial line erasure	ERASE EOL clause	o	Δ	o	o	o	x	o
	Specify non-display status	SECURE clause	x	o	x	x	x	o	o
Position	Specify column number	COLUMN NUMBER clause	o	o	o	o	o	x	o
	Specify line number	LINE NUMBER clause	o	o	o	o	o	x	o
Input	Specify input mode	FULL clause	x	o	Δ	o	o	o	o
	Specify input mode	REQUIRED clause	x	o	Δ	o	o	o	o
Others	Automatic cursor skip	AUTO clause	x	o	Δ	o	o	o	o
	Specify general characteristics	PICTURE clause	x	o	o	o	o	x	o
	Specify expression format	USAGE clause	x	o	o	o	o	o	o
	Specify literal item	VALUE clause	o	x	x	x	x	x	o

o : Can be specified
Δ : Can be specified, but not validated
x : Cannot be specified
*1 L: Literal, I: Input, O: Output, U: Update, IO: Input-output
*2 Define G (Group) as a group item, and E (Elementary) as an elementary item.

PROCEDURE DIVISION

To display a screen, use a DISPLAY statement with a screen item defined.

After executing a DISPLAY statement, data can be input from a displayed screen. After input, by referring the values of screen input-output status set for data-names specified in the CRT STATUS clause in the SPECIAL-NAMES paragraph, appropriate processing can be selected.

Program Compilation and Linkage

No compiler options or additional libraries are required.

Program Execution

To define a function key, set the run-time environment variable TERMINATOR as shown in the following example:

```
TERMINATOR=PF1-PF4,!PF5-PF12,PF13-PF16
```

PF1 to PF4 and PF13 to PF16 are valid, but PF5 to PF12 are invalid.

To change a window attribute, define the run-time environment information listed in Table 37, “Changing window attributes for screen handling.” Then, execute the program.

The ACCEPT/DISPLAY statement and screen handling function operate windows with an ACCEPT or DISPLAY statement, but use different windows.

With this system, function keys PF10 and PF17 to PF24 are invalid, regardless of the specification of the run-time environment variable TERMINATOR.

Windows used with the screen handling function cannot be minimized.

The size of a logical screen must not be smaller than the sizes of data items defined in the screen items.

Input items are always underlined.

An error occurs when a program is executed if a logical screen set for run-time environment information @ScrnSize is (number of columns + 1) $*$ number of lines > 16,250.

Chapter 10. Calling Subprograms (Inter-Program Communication)

This chapter explains how to call programs from other programs, or inter-program communication. Chapter 10 contains an outline of calling relationships, including relation forms and linkages rules, and describes how COBOL programs call COBOL and other language programs, such as C and Visual Basic.

Outline of Calling Relationships

This section outlines program calling relationships.

Calling Relationship Forms

COBOL programs can call other programs or can be called from other programs, as shown in the following figure (1), even if the other programs are coded in other languages.

COBOL programs, however, cannot be called recursively (2), and they cannot call themselves (3).

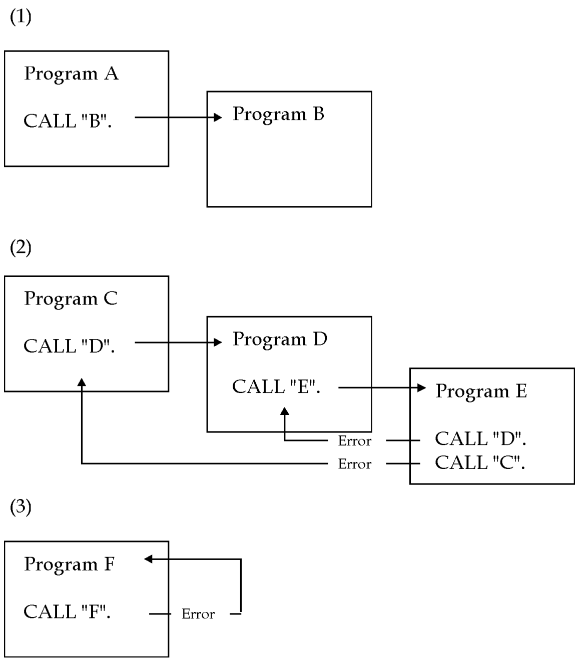


Figure 102. Calling relationship forms

Differences Among Linkage Rules

COBOL85 supports the following linkage rules:

1. COBOL linkage rules
2. C linkage rules
3. Pascal linkage rules (16)
4. STDCALL linkage rules (32)

Items 2-4 are applicable to _cdecl, _pascal, and _stdcall supported by Microsoft C and Visual C++ compilers.

Table 40. Differences between linkage rules

Linkage Rules	**Stacking and Fetching Parameters**	**Restoring Stack Pointer (SP)**	**Calling Name (*1) and External Reference Name Rules**
COBOL linkage rules	Parameters are stacked from right to left, and fetched from left to right	The calling program restores the SP	The calling name is the external reference name without modification
C linkage rules			The calling name with "_" prefixed is the external reference name
Pascal linkage rules (16)	Parameters are stacked from left to right, and fetched from right to left	The called program restores the SP	The calling name is the external reference name without modification
STDCALL linkage rules (32)	Parameters are stacked from right to left, and fetched from left to right		The calling name with "_" prefixed and @###(*2) suffixed is the external reference name

*1 When writing a calling name in COBOL, specify case-sensitivity and specify compiler option NOALPHAL.
*2 ### represents a decimal number that indicates the number of bytes of the parameter.

Linkage Rules and Supporting Compilers

The following table lists the correspondence between the linkage rules supported by COBOL, and the compilers supporting the linkage rules.

Table 41. Linkage rules and supporting compilers

Linkage Rules	**Specification in a CALL Statement, ENTRY Statement, or PROCEDURE DIVISION**	**(16)**	**(32)**
COBOL linkage rules	None	o	o
C linkage rules	WITH C LINKAGE	o	o
Pascal linkage rules	WITH PASCAL LINKAGE	o	x(*2)
STDCALL linkage rules	WITH STDCALL LINKAGE	x(*1)	o

*1 A compile time W error is output, and Pascal linkage rules are assumed.
*2 A compile time W error is output, and STDCALL linkage rules are assumed.

Calling COBOL Programs from COBOL Programs

This section explains how to call other COBOL programs (subprograms) from COBOL programs (calling programs).

Calling Method

To call programs from COBOL programs, use the CALL statement. In the CALL statement, write a program name with a literal, or specify a data-name and set the name of a called program in this data-name.

Called programs can be changed at program execution by specifying the data-name in the CALL statement. Calling programs by specifying data names in the CALL statement makes the program structure dynamic between calling programs and subprograms.

For details of program structure, refer to “Linkage Types and Program Structure” in Chapter 4.

Secondary Entry Points

An entry point to call programs can be set in the procedures in COBOL programs. The start point of a program procedure is a primary entry point, and an entry point set in the middle of a procedure is a secondary entry point.

Executing the CALL statement with a program name specified executes a subprogram from the primary entry point.

To execute a subprogram from a secondary entry point, specify the name of the secondary entry point in the CALL statement the same way you specify the program name.

To set a secondary entry point in a COBOL program, write an ENTRY statement. When a program is sequentially executed, the ENTRY statement is skipped. An ENTRY statement cannot be written in internal programs.

Returning Control and Exiting Programs

To return control from subprograms to calling programs, execute the EXIT PROGRAM statement. When the EXIT PROGRAM statement is executed, control returns immediately after the CALL statement executed by the calling program.

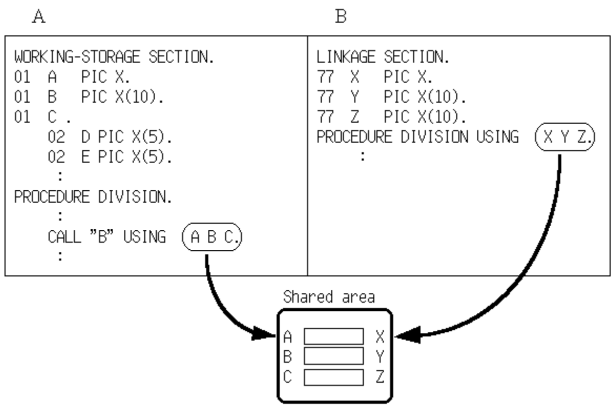
To quit execution of all COBOL programs, execute the STOP RUN statement. When the STOP RUN statement is executed, control returns to the calling source of the COBOL main program.

Passing Parameters

Parameters can be passed between calling programs and subprograms.

In a calling program, write data items defined in the FILE, WORKING-STORAGE, or LINKAGE sections in the USING phrase of the CALL statement. In a subprogram, write data-names to receive parameters in the USING phrase of the PROCEDURE DIVISION header or ENTRY statement.

The order of data-names entered in the USING phrase of the CALL statement of the calling program corresponds to that of data names written in the USING phrase of the called subprogram. Data names need not be the same between the calling program and subprogram. The attribute, length, and number of corresponding data items, however, should be the same.



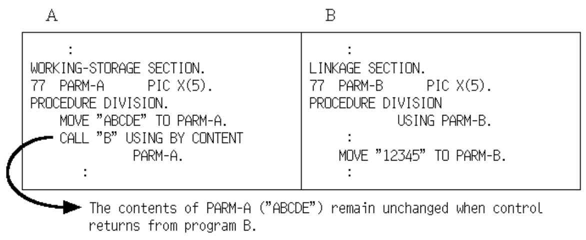
```
WORKING-STORAGE SECTION.
01  A   PIC X.
01  B   PIC X(10).
01  C .
    02  D PIC X(5).
    02  E PIC X(5).
     :
PROCEDURE DIVISION.
     :
    CALL "B" USING  A B C.
     :
```

```
LINKAGE SECTION.
77  X   PIC X.
77  Y   PIC X(10).
77  Z   PIC X(10).
PROCEDURE DIVISION USING  X Y Z.
      :
```

Figure 103. Order of data-names between calling and called programs

When the contents of parameters of the calling program should not be changed with the execution of the subprogram, write BY

CONTENT data-name in the USING phrase of the CALL statement.



```
    :
WORKING-STORAGE SECTION.
77  PARM-A      PIC X(5).
PROCEDURE DIVISION.
    MOVE "ABCDE" TO PARM-A.
    CALL "B" USING BY CONTENT
              PARM-A.
      :
```

```
    :
LINKAGE SECTION.
77  PARM-B      PIC X(5).
PROCEDURE DIVISION
            USING PARM-B.
    :
   MOVE "12345" TO PARM-B.
    :
```

The contents of PARM-A ("ABCDE") remain unchanged when control returns from program B.

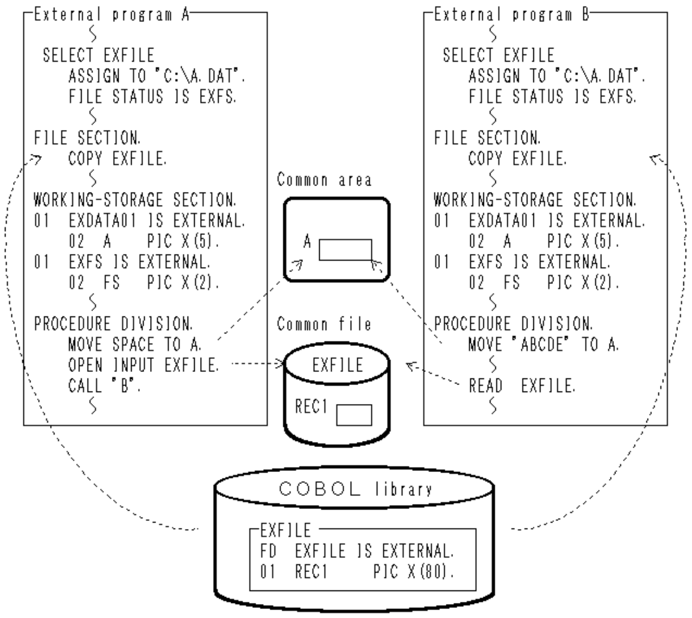
Figure 104. BY CONTENT data-name in the USING phrase of the CALL statement

Sharing Data

By specifying the EXTERNAL clause in the data or file description entry, the data area can be shared among multiple external programs. Specifying the EXTERNAL clause gives the data or file the external attribute. Data having the external attribute is external data, and a file having the external attribute is an external file.

Note: The EXTERNAL clause cannot be specified for the presentation file without OUTPUT mode.

Program maintainability can be improved by generating the definitions of external data or files as a COBOL library, then including the data in programs with the COPY statement.



```
External program A
        ⟆
 SELECT EXFILE
    ASSIGN TO "C:\A.DAT".
    FILE STATUS IS EXFS.
        ⟆
FILE SECTION.
    COPY EXFILE.
        ⟆
WORKING-STORAGE SECTION.
01  EXDATA01 IS EXTERNAL.
    02  A    PIC X(5).
01  EXFS IS EXTERNAL.
    02  FS   PIC X(2).
        ⟆
PROCEDURE DIVISION.
    MOVE SPACE TO A.
    OPEN INPUT EXFILE.
    CALL "B".
        ⟆
```

```
External program B
        ⟆
 SELECT EXFILE
    ASSIGN TO "C:\A.DAT".
    FILE STATUS IS EXFS.
        ⟆
FILE SECTION.
    COPY EXFILE.
        ⟆
WORKING-STORAGE SECTION.
01  EXDATA01 IS EXTERNAL.
    02  A    PIC X(5).
01  EXFS IS EXTERNAL.
    02  FS   PIC X(2).
        ⟆
PROCEDURE DIVISION.
    MOVE "ABCDE" TO A.
        ⟆
    READ  EXFILE.
        ⟆
```

```
Common area
  A

Common file
  EXFILE
  REC1
```

```
COBOL library
EXFILE
FD  EXFILE IS EXTERNAL.
01  REC1      PIC X(80).
```

Figure 105. Improving program maintainability by generating data and file definitions

Return Codes

When control returns to calling programs from subprograms, return code values can be passed with special register PROGRAM-STATUS (or RETURN-CODE). Special register PROGRAM-STATUS is implicitly declared as PIC S9(9) COMP-5, and you do not have to define it in the program.

When a subprogram writes a value to special register PROGRAM-STATUS, the value is written to special register PROGRAM-STATUS of the calling program.

Internal Programs

COBOL programs are classified into external and internal programs based on the program structure. The outermost program not included in other programs is an external program. Programs directly or indirectly included in an external program are internal programs.

An external program (A) can call its internal program (A1). The internal program (A1) can call another external program (B) or its internal program (A11). However, internal programs (C1 and D1) cannot call outside programs (C and D2) other than common programs.

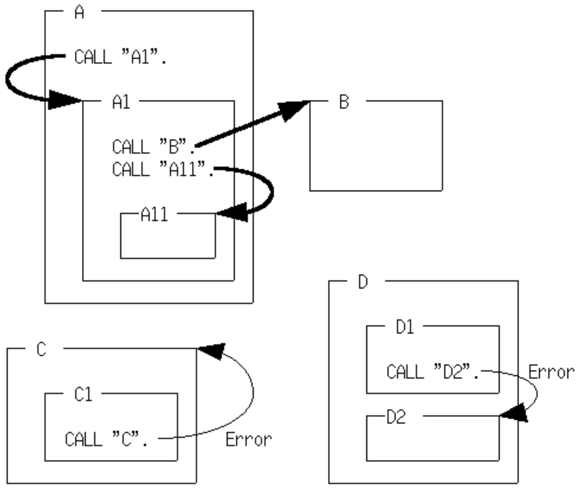
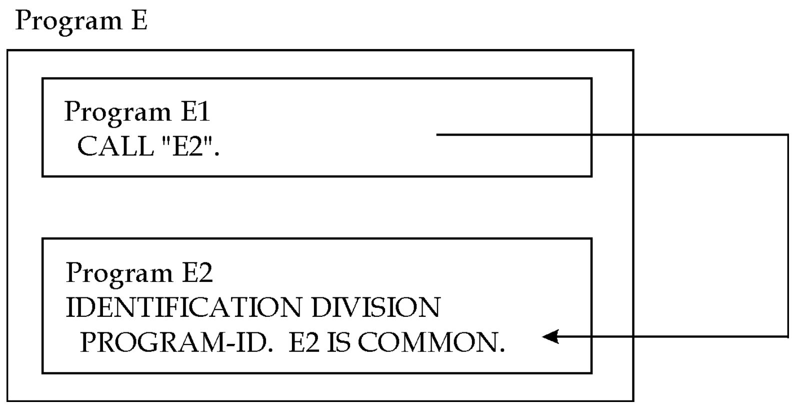


Figure 106. Calling sequence

Common Programs

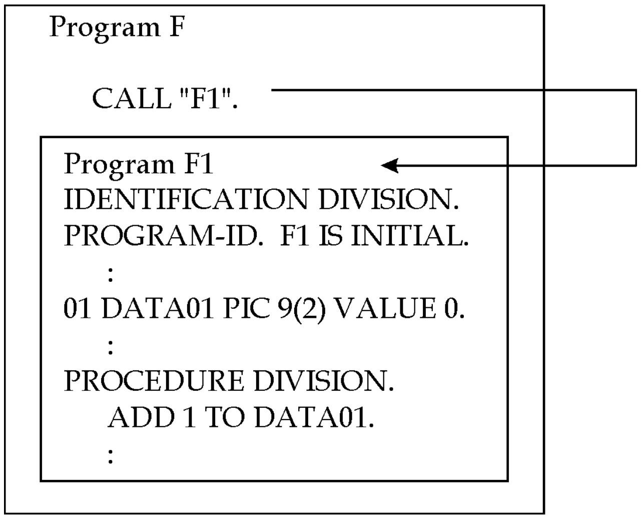
To call an internal program from an outside internal program, specify a COMMON clause in the PROGRAM-ID paragraph of the called internal program.

The program with COMMON specified is a common program, and it can be called from internal programs not including the program with COMMON specified.



Initial Programs

To place a program in the initial state whenever it is called, specify an INITIAL clause in the PROGRAM-ID paragraph. This program is an initial program. When the initial program is called, the program is always placed in the initial state.



When program F1 is called, the value of DATA01 is always 0.

Valid Scope of Names

To use data items defined by external programs in the internal programs, specify the GLOBAL clause in the data description entry. Normally, names are valid only within the same program, but data items with the GLOBAL clause specified can be used by internal programs.

External programs cannot use data items with GLOBAL clause specified in the internal programs.

Notes

If compiler option ALPHAL is valid in both the calling program and subprogram, literal of the program name is treated as follows:

- Calling program: A program name specified with a literal in the CALL statement is always treated as uppercase.

- Subprogram: A program name written in the PROGRAM-ID paragraph is always treated as uppercase.

When compiling the calling program and subprogram, specify the same compiler option for each, ALPHAL or NOALPHAL. **Note**: Specify compiler option NOALPHAL when using lowercase letters for a program name.

When compiling the main program, compiler option MAIN must be specified. When compiling the subprogram, compiler option NOMAIN must be specified.

Linking C Programs

This section explains how to call C programs (functions) from COBOL programs, and how to call COBOL programs from C programs (functions). In this section, C programs (functions) are simply called C programs.

Calling C Programs from COBOL Programs

This section explains how to call C programs from COBOL programs.

Calling Method

To call C programs from COBOL programs, specify function names in the CALL statement of COBOL. When the return statement is executed in the called C program, control returns immediately after the CALL statement of COBOL.

Passing Parameters

To pass parameters from COBOL programs to C programs, specify data-names in the USING clause of the CALL statement.

The parameters to be passed to the C program are the area addresses or the values of the data names. Specify parameters in the USING clause of the CALL statement.

The relation between the description of the USING clause and the contents of parameters is explained below.

When BY REFERENCE Data-Name is Specified: Area Address

The actual argument value that the COBOL program passes to the C program is the area address of the specified data-name. Declare a pointer having a data type corresponding to the attribute of the parameter to be passed as a dummy argument in the C program. For correspondence between COBOL and C data types, see Table 42.

When BY CONTENT Data-Name (or Literal) is Specified: Area Address

The actual argument value that the COBOL program passes to the C program is the address of the area containing the value of the specified data-name. Declare a pointer having the data type corresponding to the attribute of the parameter to be passed as a dummy argument in the C program.

Changing the contents of the area pointed to by the actual argument in the C program does not change contents of the data name of the COBOL program.

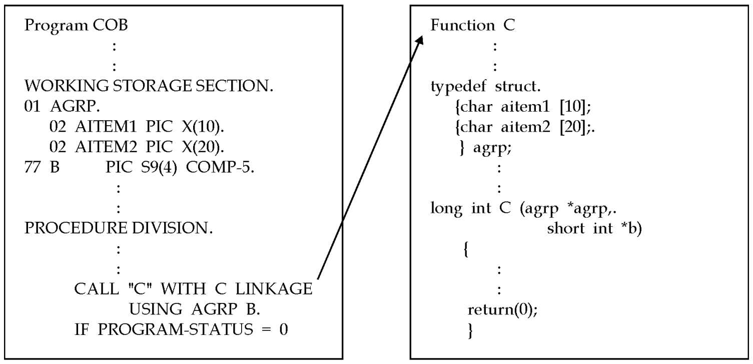
When BY VALUE Data-Name is Specified: Contents of the Area

The actual argument value that the COBOL program passes to the C program is the contents of the specified data-name. Changing the contents of the actual argument in the C program does not change contents of the data name of the COBOL program.

Return Codes (Function Values)

Use the PROGRAM-STATUS special register to receive return codes (function values) from C programs. C function values must be of the long int type.

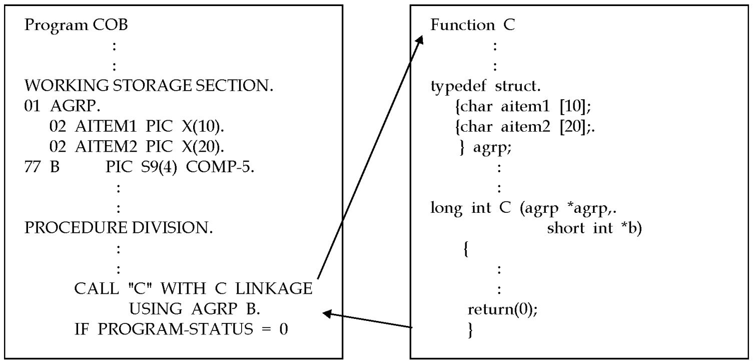
C Call Rules



```
Program COB
              :
              :
WORKING STORAGE SECTION.
01  AGRP.
    02  AITEM1  PIC  X(10).
    02  AITEM2  PIC  X(20).
77  B         PIC  S9(4)  COMP-5.
               :
               :
PROCEDURE DIVISION.
               :
               :
        CALL  "C"  WITH  C  LINKAGE
                 USING  AGRP  B.
        IF  PROGRAM-STATUS  =  0
```

```
Function  C
          :
          :
typedef  struct.
    {char  aitem1  [10];
    {char  aitem2  [20];.
     }  agrp;
           :
           :
long  int  C  (agrp  *agrp,.
                   short  int  *b)
      {
           :
           :
       return(0);
       }
```

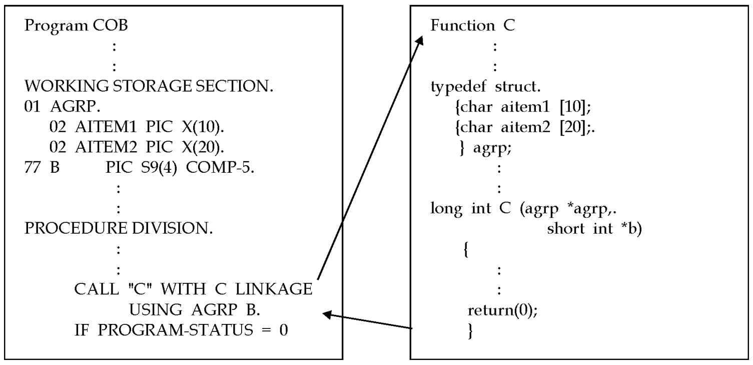
Pascal Call Rules (16)



```
Program COB
              :
              :
WORKING STORAGE SECTION.
01  AGRP.
    02  AITEM1  PIC  X(10).
    02  AITEM2  PIC  X(20).
77  B         PIC  S9(4)  COMP-5.
               :
               :
PROCEDURE DIVISION.
               :
               :
        CALL  "C"  WITH  C  LINKAGE
                 USING  AGRP  B.
        IF  PROGRAM-STATUS  =  0
```

```
Function  C
          :
          :
typedef  struct.
    {char  aitem1  [10];
    {char  aitem2  [20];.
     }  agrp;
           :
           :
long  int  C  (agrp  *agrp,.
                   short  int  *b)
      {
           :
           :
       return(0);
       }
```

STDCALL Call Rules (32)



```
Program COB
              :
              :
WORKING STORAGE SECTION.
01  AGRP.
    02  AITEM1  PIC  X(10).
    02  AITEM2  PIC  X(20).
77  B         PIC  S9(4)  COMP-5.
               :
               :
PROCEDURE DIVISION.
               :
               :
        CALL  "C"  WITH  C  LINKAGE
                 USING  AGRP  B.
        IF  PROGRAM-STATUS  =  0
```

```
Function  C
          :
          :
typedef  struct.
    {char  aitem1  [10];
    {char  aitem2  [20];.
     }  agrp;
           :
           :
long  int  C  (agrp  *agrp,.
                   short  int  *b)
      {
           :
           :
       return(0);
       }
```

When a C program of the short int type is called, C function values written to the PROGRAM-STATUS special register are referred to as follows:

```
        :
  01 return-value-storage-area PIC  S9(9) COMP-5.
  01 REDEFINES return-value-storage-area.
          02  function-value   PIC  S9(4) COMP-5.
          02                   PIC   9(4) COMP-5.
        :
     CALL  "Cprog" USING PARAM1 PARAM2.
     MOVE PROGRAM-STATUS TO return-value-storage-area.
            IF  function-value =  0  THEN  ...
        :
```

The attribute of the PROGRAM-STATUS special register is applicable to the C long int type. After a C program of the short int type is called, therefore, the values of special register PROGRAM-STATUS, if referred to as they are, cannot be referred to as correct function values.

Calling COBOL Programs from C Programs

This section explains how to call COBOL programs from C programs.

Calling Method

To call COBOL programs from C programs, specify COBOL program names in the C function calling format. When the EXIT PROGRAM statement is executed in the called COBOL program, control returns immediately after the C program function call.

When the main program (C program) calls a COBOL program, call JMPCINT2 before calling the first COBOL program, and call JMPCINT3 after calling the last COBOL program. JMPCINT2 is a subroutine that initializes COBOL programs. JMPCINT3 is a subroutine that terminates COBOL programs.

Calling COBOL programs without calling JMPCINT2 and JMPCINT3 may degrade performance because COBOL program run-time environments are initialized and terminated every time COBOL programs are called.

Passing Parameters

To pass arguments from C programs to COBOL programs, specify actual arguments with C function calls. The values of actual arguments that can be passed from C programs to COBOL programs must be storage addresses.

With COBOL programs, by specifying data-names in the USING clause of the PROCEDURE DIVISION header or the ENTRY statement, the contents of the area at the address specified for an actual argument are received.

A CONST type specifier can be designated in the declaration or definition of a variable at the address specified by an actual argument. When this happens, do not change the contents of the area at the address specified by the actual argument.

Return Codes (Function Values)

Values set to special register PROGRAM-STATUS are passed to C programs as function values. C programs receive the function values as the long int type.

Correspondence of Data Types

The combination of the attributes of data passed between COBOL and C programs are optional. The following table lists the basic correspondence of data items between COBOL and C programs.

For the COBOL internal format, refer to the “COBOL85 Reference Manual.” For the C internal expression format, refer to C language manuals.

Table 42. Correspondence between COBOL data items and C data types

COBOL Data Item	**C Data Type**	**COBOL Coding Example**	**C Declaration Example**	**Size**

Alphabetic or alphanumeric	char, char array
	type or struct (structure type)
		77 A PIC X.
		
		01 B PIC X(20).
			char A;
			
			char B[20];
				1 byte
				
				20 bytes

External decimal (*1)	char array type, or struct (structure type)	77 C PIC S9(5) SIGN IS LEADING SEPARATE	char C[6];	6 bytes

Binary (*2)	short int
	
	long int
		01 D PIC S9(4) COMP-5
		
		77 E PIC S9(9) COMP-5
			short int D;
			
			long int E;
				2 bytes
				
				4 bytes

Group item (*3)	char, char array type, or struct (structure type)	01 FGRP 02 F1 PIC S9(4) COMP-5. 02 F2 PIC X(4).	Struct { short int F1; char F2[4];} FGRP;	6 bytes

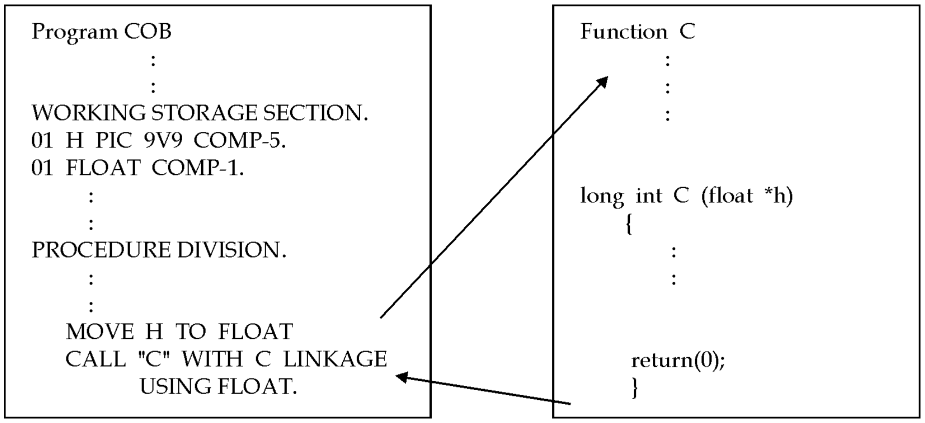
Internal floating-point
(single-precision)
double-precision)
	float
	
	double
		01 G COMP-1.
		
		01 H COMP-2.
			float G;
			
			float H;
				4 bytes
				
				8 bytes

*1 The internal format of COBOL external decimal items are character strings consisting of characters indicating signs and numeric characters. In the C program, therefore, these are treated as character data but not as numeric data. To handle external decimal items as numeric data in the C program, the data type must be converted in the C program.

*2 Binary items are applicable to short int or long int of the C program depending on the number of digits as follows:

- Up to 4 digits: short int
- 5 to 9 digits : long int

A binary item with a fraction part is converted to a floating-point item, then passed.



```
Program COB
            :
            :
WORKING STORAGE SECTION.
01  H  PIC  9V9  COMP-5.
01  FLOAT  COMP-1.
       :
       :
PROCEDURE DIVISION.
       :
       :
    MOVE  H  TO  FLOAT
    CALL  "C"  WITH  C  LINKAGE
             USING FLOAT.
```

```
Function  C
         :
         :
         :


long  int  C  (float  *h)
     {
          :
          :


      return(0);
      }
```

*3 When declaring group items as structure, note the storage boundary of variables included in the structure. To align the data item storage boundary with COBOL, specify the SYNCHRONIZED clause in the data description entry. For details, refer to the “COBOL85 Reference Manual.” For C variable storage boundary alignment, refer to C language manuals.

Compiling Programs

When calling a C program with the CALL statement with a literal specified, and if the program is compiled with compiler option ALPHAL specified, the desired program may not be called since the program name is always treated as being uppercase. Specify compiler option NOALPHAL when compiling programs.

For example:

```
CALL "abc".
```

When compiler option NOALPHAL is specified, program "abc" is called upon execution of the CALL statement.

When compiler option ALPHAL is valid, program "ABC" is called upon execution of the CALL statement.

```
MOVE  "abc" TO A.
CALL A.
```

Program "abc" is called regardless of whether or not compiler option ALPHAL or NOALPHAL is specified.

A COBOL program compiled with compiler option ALPHAL specified can be called from a C program. If so, the desired program may not be called since the program name specified in the PROGRAM-ID paragraph is always treated as being uppercase. Specify compiler option NOALPHAL when compiling programs.

For example:

```
PROGRAM-ID. abc.
```

When compiler option NOALPHAL is specified, the program name is "abc".

When compiler option ALPHAL is valid, the program name is "ABC".

When compiling a C program to be linked to a COBOL program, an option with which the DS and SS registers are discriminated and the memory model is large (for example, Microsoft C compiler option /ALw) must be specified. (16)

Since COBOL85 supports multiple data segments (you can write the Data Division of 64 K bytes or more), DS and SS registers do not always have the same address. The operation, therefore, may be undefined when a C program operating on the assumption that DS and SS registers have the same address is called. C run-time libraries have many functions that process DS and SS on the assumption that they are the same.

In addition, suppressing stack checks and adding Windows prologue and epilogue codes to all FAR functions (using Microsoft C compiler option /Gws) in C programs executed under the Windows environment must be specified.

When compiling a C program to be linked to a COBOL program, an option (/Gs) to suppress stack checks and an option (/G3) to optimize the program for 80386 processor. For details, refer to the Visual C++ on-line Help. (32)

Because data names used in C and COBOL are the same, the underscore (_) can be used for the data name.

Linking Programs

Library F1BCOWEP.LIB provided by the COBOL85 run-time system includes the following functions (16):

- WEP
- LIBENTRY
- LIBMAIN

When linking C programs that call COBOL programs (16), the file specification order in the LINK command depends on whether executable files are generated by linking C programs that call COBOL programs or dynamic link libraries (DLLs) are generated.

Examples of linking C programs that call COBOL programs are shown below. To link C programs that call COBOL programs, link option /NOD must be specified.

(1) Generating executable files from C programs that call COBOL programs:

```
LINK /NOD Cprog.OBJ, Cprog.EXE, Cprog.MAP, Cimplib
LIBW, Cprog.DEF
```

CPROG.OBJ

C program object file name.

CPROG.EXE

Name of the executable file to be generated.

CPROG.MAP

Map file name (optional).

CIMPLIB

Import library of C run-time library. Refer to a C manual. With Microsoft C, LLIBCEW is applicable to this import library.

LIBW(.LIB)

Windows function import library name.

CPROG.DEF

C program module definition file name.

F1BCCIMP.LIB is required to call JMPCINT2 or JMPCINT3.

(2) Generating executable files from COBOL programs and C programs that call COBOL programs:

```
LINK /NOD COB.OBJ Cprog.OBJ, COB.EXE, COB.MAP, F1BCCIMP
F1BCARMV.LIB
Cimplib LIBW, COB.DEF
```

COB.OBJ

COBOL program object file name.

COB.EXE

Name of the executable file to be generated.

COB.MAP

Map file name (optional).

F1BCCIMP(.LIB) F1BCARMV.LIB

COBOL run-time system import library name.

CIMPLIB

Import library of C run-time library.

LIBW(.LIB)

Windows function import library name.

COB.DEF

COBOL program module definition file name.

To generate an executable file by linking a COBOL program to a C program, the import library of the run-time library (COBOL run-time system) of the program having the WinMain function must be specified first.

For example, when a COBOL program has the WinMain function (when compiled with compiler option MAIN specified), specify the import library of the COBOL run-time system prior to that of the C run-time library. Doing so prevents the startup routine provided by the import library of the C run-time library from being linked to the COBOL program.

(3) Generating DLLs from C programs that call COBOL programs:

```
LINK /NOD Cprog.OBJ LIBENTRY.OBJ, Cprog.DLL, Cprog.MAP, Cimplib
LIBW, Cprog.DEF
```

CPROG.OBJ

C program object file name.

LIBENTRY.OBJ

Name of the initialization function to be dynamically linked.

CPROG.DLL

Name of the DLL to be generated.

CPROG.MAP

Map file name (optional).

CIMPLIB

Import library of C run-time library. Refer to C manuals. With Microsoft C, LDLLCEW is applicable to this import library.

LIBW(.LIB)

Windows function import library name.

CPROG.DEF

C program module definition file name.

(4) Generating DLLs from COBOL programs and C programs that call COBOL programs:

```
LINK /NOD COB.OBJ Cprog.OBJ F1BCOWEP.OBJ, COB.DLL, COB.MAP,
F1BCCIMP
F1BCARMV Cimplib LIBW,COB.DEF
```

COB.OBJ

COBOL program object file name.

CPROG.OBJ

C program object file name.

F1BCOWEP.LIB

Name of the initialization function to be dynamically linked.

COB.DLL

Name of the DLL to be generated.

COB.MAP

Map file name (optional).

F1BCCIMP(.LIB) F1BCARMV.LIB

COBOL run-time system import library name.

CIMPLIB

Import library of C run-time library.

LIBW(.LIB)

Windows function import library name.

COB.DEF

COBOL program module definition file name.

To generate the DLL, except for the following cases, link F1BCOWEP.LIB, which contains the DLL initialization function (LibMain) and the termination function (WEP):

- The C program provides DLL initialization and termination functions.
- Microsoft C V7.0 or later is used.

To link F1BCOWEP.LIB, specify it in the object field on the command line.

When linking C programs that call COBOL programs (32), the file specification order in the LINK command depends on whether executable files are generated by linking C programs that call COBOL programs or dynamic link libraries (DLLs) are generated.

Examples of linking C programs that call COBOL programs are shown below.

(1) Generating executable files from C programs that call COBOL programs:

```
LINK Cprog.OBJ Cimplib /OUT:Cprog.EXE
```

CPROG.OBJ

C program object file name.

CPROG.EXE

Name of the executable file to be generated.

CIMPLIB

Import library of C run-time library. Refer to C manuals.

F3BICIMP.LIB is required to call JMPCINT2 or JMPCINT3.

(2) Generating executable files from COBOL programs and C programs that call the COBOL programs:

```
LINK COB.OBJ Cprog.OBJ F3BICIMP.LIB COB.EXP LIBC.LIB KERNEL32.LIB
USER32.LIB
/OUT:COB.EXE
```

COB.OBJ

COBOL program object file name.

COB.EXE

Name of the executable file to be generated.

F3BICIMP.LIB

COBOL run-time system import library.

COB.EXP

COBOL program export file generated by the LIB command.

LIBC.LIB KERNEL32.LIB USER32.LIB

C run-time library import library name required by COBOL

To generate an executable file by linking a COBOL program to a C program, the import library of the run-time library (COBOL run-time system) of the program having the WinMain function must be specified first.

For example, when a COBOL program has the WinMain function (when compiled with compiler option MAIN specified), specify the import library of the COBOL run-time system prior to that of the C run-time library. Doing so prevents the startup routine provided by the import library of the C run-time library from being linked to the COBOL program.

(3) Generating DLLs from C programs that call COBOL programs:

(Generating a C program export file):

```
LIB /DEF:Cprog.DEF /OUT:Cprog.LIB /MACHINE:IX86
```

(Generating a COBOL program import library):

```
LIB /DEF:COB.DEF /OUT:COB.LIB /MACHINE:IX86
```

(Generating a DLL):

```
LINK Cprog.OBJ Cprog.EXP COB.LIB Cimplib /DLL
/OUT:Cprog.DLL
```

CPROG.DEF

C program module definition file name .

CPROG.LIB

C program import library name.

COB.DEF

COBOL program module definition file name.

COB.LIB

COBOL program import library name.

CPROG.OBJ

C program object file name.

CPROG.DLL

Name of the DLL to be generated.

CPROG.EXP

C program export file.

CIMPLIB

Import-library of C run-time library. Refer to C manuals.

(4) Generating DLLs from COBOL programs and C programs that call the COBOL programs:

(Generating an export file):

```
 LIB /DEF:COB.DEF /OUT:COB.LIB /MACHINE:IX86
```

(Generating a DLL):

```
LINK COB.OBJ F3BICIMP.LIB COB.EXP LIBC.LIB KERNEL32.LIB
USER32.LIB
/DLL /OUT:COB.DLL
```

COB.DEF

COBOL program module definition file name.

COB.LIB

COBOL program import library name.

COB.OBJ

COBOL program object file name.

F3BICIMP.LIB

COBOL run-time system import library.

COB.EXP

COBOL program export file generated by the LIB command.

LIBC.LIB KERNEL32.LIB USER32.LIB

C run-time library import library name required by COBOL.

COB.DLL

Name of the DLL to be generated.

Executing Programs

When calling COBOL programs from C programs, no COBOL run-time options can be specified for arguments of functions that call COBOL programs. The run-time options are treated the same as other arguments and are not handled as COBOL run-time options even if specified.

Do not use the exit function to unconditionally terminate C programs called from COBOL programs.

Do not use the STOP RUN statement to terminate COBOL programs called from C programs with JMPCINT2.

Unlike C character strings, no null characters are inserted automatically at the end of COBOL character strings.

Chapter 11. Using ACCEPT and DISPLAY Statements

This chapter offers tips on using the ACCEPT and DISPLAY statements, including I-O destination types and specification methods, using console windows, message boxes, and program using files, and entering current date and time. Additionally, Chapter 11 describes fetching command line arguments and environment variable handling.

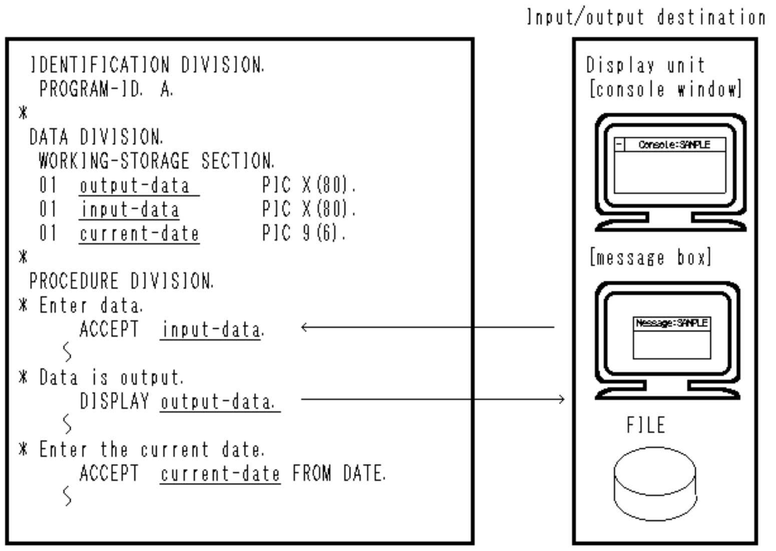
ACCEPT/DISPLAY Function

This section explains how the ACCEPT/DISPLAY function inputs and outputs data with ACCEPT and DISPLAY statements. For a sample program using the ACCEPT/DISPLAY function, refer to the “Getting Started with Fujitsu COBOL” guide.

Outline

With the ACCEPT/DISPLAY function, data is input and output with console windows, message boxes, and files. Additionally, the current date and time can be read from the system.

Use the ACCEPT statement to input data, and use the DISPLAY statement to output data.



```
 IDENTIFICATION DIVISION.
  PROGRAM-ID. A.
*
 DATA DIVISION.
  WORKING-STORAGE SECTION.
  01  output-data       PIC X(80).
  01  input-data        PIC X(80).
  01  current-date      PIC 9(6).
*
 PROCEDURE DIVISION.
* Enter data.
        ACCEPT  input-data.
* Data is output.
        DISPLAY output-data.
* Enter the current date.
        ACCEPT  current-date FROM DATE.
```

Figure 107. The ACCEPT/DISPLAY function

Input/Output Destination Types and Specification Methods

The input/output destination of data depends on:

- ACCEPT statement FROM specification
- DISPLAY statement UPON specification
- Compiler option specifications
- Setting of run-time environment information

The following table lists the relationship between these specifications and input/output destinations.

Table 41. Input/output destinations of the ACCEPT/DISPLAY function

	FROM or UPON Specification	**Compile Option to be Specified**	**Run-time Environment Information Setting**	**Input/Output Destination**
1)	None or function-name SYSIN / SYSOUT (*1)	None SSIN (run-time environment information name) SSOUT (run-time environment information name)	Enter a file name for the run-time environment information name	Display unit (console window) File (*2)
2)	Function-name SYSERR		Enter a blank for @MessOutFile Enter a file name for @MessOutFile	Display unit (message box) File
3)	Function-name CONSOLE(*1)			Display unit (console window)

*1 1) and 3) cannot both be used simultaneously in the program through its complete execution. The direction specified in the first ACCEPT or DISPLAY statement executed in the program becomes effective.
*2 When SYSIN and SYSOUT are specified for compiler options SSIN and SSOUT as environment variable names, the input source and output destinations are system standard input and output.

Reading/Writing Data with Console Windows

This section explains how to write, compile, link, and execute programs, and provides an example of the simplest coding using console windows.

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. program-name.
DATA DIVISION.
 WORKING-STORAGE SECTION.
  01  data-name ... .
PROCEDURE DIVISION.
     ACCEPT   data-name.
     DISPLAY  data-name.
     DISPLAY "nonnumeric-literal".
     EXIT PROGRAM.
END PROGRAM program-name.
```

Console Windows

In a console window, data is read from or displayed on a display unit with the ACCEPT/DISPLAY function. One console window is provided for each COBOL program run unit.

A console window is generated by the ACCEPT or DISPLAY statement first executed in the program run unit, and is closed upon normal termination of the run unit.

The attributes of console windows can be changed with the COBOL initialization file (COBOL85.CBR). The following table lists the attributes that can be changed and the specification methods.

Table 42. Changing the attributes of console windows

Attribute	**Run-time Environment Information**	**Setting Value**	**Meaning**
Automatic window closing	@WinCloseMsg	ON	Closes the window after displaying a message
	(*1)	OFF	Closes the window without displaying a message
Window size	@CnslWinSize	(m,n)	Specifies the size of a window with the number of lines (m) and the number of columns (n)
Number of lines for data to be retained	@CnslBufLine	Number of lines	Data for the number of specified lines is retained and can be browsed by vertically scrolling the window

*1 This run-time environment information is also valid for windows used with the screen operation function.

Program Specifications

This section explains the program descriptions for each COBOL division.

ENVIRONMENT DIVISION

No specifications are required.

DATA DIVISION

In the DATA DIVISION, define data items to store input data and items to set output data. Define the data items with one of the following attributes:

- Group item
- Alphabetic item

- Alphanumeric item
- Binary item
- Internal decimal item
- External decimal item
- Alphanumeric edited item
- Numeric edited item
- National item
- National edited item

National item and national edited item cannot be used in an ACCEPT statement.

PROCEDURE DIVISION

Use an ACCEPT statement to input data from a console window. Input data is stored in a data-name specified in an ACCEPT statement for the length (80 alphanumeric characters if defined as 01 input-data PIC X(80)) defined for the data-name.

If the length of input data is less than that of data to be stored, the input request is repeated.

For character data, the input request is repeated until the entered data satisfies the specified length.

For numeric data, the input request is repeated until the Enter or Return key is pressed.

Use a DISPLAY statement to output data to a console window.

When a data-name is specified in a DISPLAY statement, data stored in the data-name is output.

When a nonnumeric literal is specified in a DISPLAY statement, a specified character string is output.

Program Compilation and Linkage

Do not specify compiler options SSIN and SSOUT.

Program Execution

Execute programs like ordinary programs.

Data input is requested on a console window when an ACCEPT statement in a program is executed. Enter data as required.

Input data is set for the data item for the length (80 bytes) specified in an ACCEPT statement. If the length of input data is less than that of the data item, the input request is repeated.

For character data, the input request is repeated until the entered data satisfies the specified length.

For numeric data, the input request is repeated until the Enter or Return key is pressed.

When a DISPLAY statement in the program is executed, data is output to the console window.

To associate data at one entry with one ACCEPT statement, use function-name CONSOLE. When function-name CONSOLE is used, space is filled if the length of input data is less than that of the data item.

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. A.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
   CONSOLE IS CONS.
DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  data-1   PIC X(80).
PROCEDURE DIVISION.
    ACCEPT  data-1  FROM CONS.
    DISPLAY data-1  UPON CONS.
    EXIT PROGRAM.
END PROGRAM A.
```

Writing Messages to Message Boxes

This section explains how to write messages to message boxes.

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. program-name.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
   SYSERR IS mnemonic-name.
DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  data-name ....
PROCEDURE DIVISION.
    DISPLAY data-name UPON mnemonic-name.
    EXIT PROGRAM.
END PROGRAM program-name.
```

Message Boxes

Normally, COBOL program run-time messages are displayed in message boxes. With a COBOL program, messages other than run-time messages can also be displayed in the message boxes. Executing a DISPLAY statement opens a message box, and clicking on the OK button closes it.

Program Specifications

This section explains program descriptions for each COBOL division.

ENVIRONMENT DIVISION

In the ENVIRONMENT DIVISION, associate a mnemonic-name with function-name SYSERR.

DATA DIVISION

In the DATA DIVISION, define data items to set output data. Define these data items with one of the following attributes:

- Group item
- Alphabetic item
- Alphanumeric item
- External decimal item
- Alphanumeric edited item
- Numeric edited item
- National item
- National edited item

PROCEDURE DIVISION

To send a message to a message box, use a DISPLAY statement where a mnemonic-name associated with function-name SYSERR is specified in the UPON clause.

If a data-name is specified in a DISPLAY statement, data stored in the specified data-name is output.

If a nonnumeric literal is specified, the specified character string is output. The maximum length of the message that can be written at one time is the length of the data that can be displayed within the message box.

Program Compilation and Linkage

No specific compiler and linker options are required.

Program Execution

Execute programs as ordinary programs.

Programs Using Files

This section explains how to write, compile, link, and execute programs, and provides an example of the simplest coding for file processing using the ACCEPT/DISPLAY function.

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. program-name.
DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  data-name  ...
PROCEDURE DIVISION.
    ACCEPT  data-name.
    DISPLAY data-name.
    EXIT PROGRAM.
END PROGRAM program-name.
```

Program Specifications

This section explains program descriptions for each COBOL division.

ENVIRONMENT DIVISION

No specifications are required.

DATA DIVISION

In the DATA DIVISION, define data items to store input data and items to set output data. Define these data items with one of the following attributes:

- Group item
- Alphabetic item
- Alphanumeric item
- External decimal item
- Alphanumeric edited item
- Numeric edited item
- National item
- National edited item

National item and national edited item cannot be used in an ACCEPT statement.

PROCEDURE DIVISION

At program execution, an input file is opened with the first ACCEPT statement, and an output file is opened with the first DISPLAY statement of the program. With subsequent ACCEPT and DISPLAY statements, data is read or output only.

The input and output files are closed upon termination of program execution.

The input file is opened in input mode and used in share mode. Records are not locked during read.

The output file is opened in output mode and used in exclusive mode.

Line feed characters are not handled as data.

After a file is opened (after executing the first ACCEPT and DISPLAY statements), the input/output destination cannot be changed with the environment variable operation function.

Inputting Data

Use an ACCEPT statement to input data from a file.

Data at the byte immediately before the line feed character is handled as one record.

Input data is read by record. Input data is stored in a data-name specified in an ACCEPT statement for the length (80 alphanumeric characters if defined as 01 input-data PIC X(80)) defined for the data-name. If the length of input data is less than that of data to be stored, the next record is read and linked to the previously read data.

In this case, line feed characters are not treated as data. Records are read until the entered data satisfies the specified length.

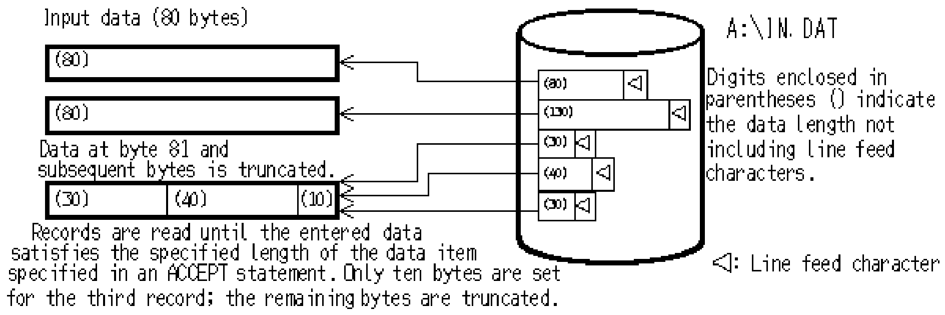

Input data (80 bytes)
(80)
(80)
Data at byte 81 and subsequent bytes is truncated.
(30) (40) (10)
Records are read until the entered data satisfies the specified length of the data item specified in an ACCEPT statement. Only ten bytes are set for the third record; the remaining bytes are truncated.
A:\IN.DAT
(80)
(130)
(30)
(40)
(30)
Digits enclosed in parentheses () indicate the data length not including line feed characters.
◁: Line feed character

Figure 108. Reading records

Outputting Data

Use a DISPLAY statement to send data to a file.

When a data-name is specified in a DISPLAY statement, the contents stored in the data-name is output.

If a nonnumeric literal is specified in a DISPLAY statement, the specified character string is output. With one DISPLAY statement, the length of the line feed character plus output data is the length of data to be output.

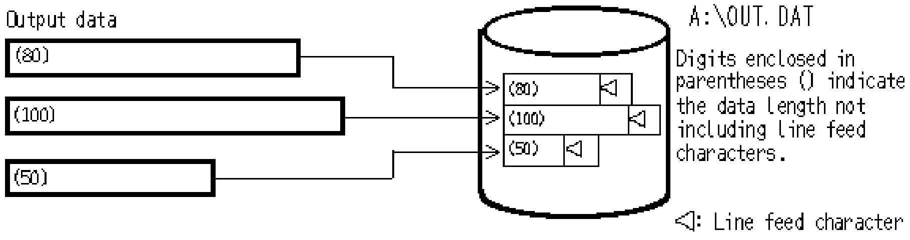

Output data
(80)
(100)
(50)
A:\OUT.DAT
(80)
(100)
(50)
Digits enclosed in parentheses () indicate the data length not including line feed characters.
◁: Line feed character

Figure 109. Using DISPLAY to output data

Program Compilation and Linkage

To enter data from a file with ACCEPT, specify compiler option SSIN. To send data to a file with DISPLAY, specify compiler option SSOUT.

For example:

```
SSIN(INDATA),SSOUT(OUTDATA)
```

When SYSIN is specified to compiler option SSIN, a console window is the data input destination of an ACCEPT statement, regardless of the specification of run-time environment information SYSIN.

When SYSOUT is specified to compiler option SSOUT, a console window is the data output destination of a DISPLAY statement, regardless of the specification of run-time environment information SYSOUT.

Program Execution

Specify the names of files used for input-output in the run-time environment information specified for compiler options SSIN and SSOUT.

For example:

```
INDATA=A:\IN.DAT
OUTDATA=A:\OUT.DAT
```

If a specified file already exists at the output destination, the file is recreated (previous data is deleted).

Entering Current Date and Time

This section explains how to write, compile, link, and execute programs for entering the current date and time by using system clocks with the ACCEPT / DISPLAY function.

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. program-name.
DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  date-1        PIC 9(6).
 01  day-1         PIC 9(5).
 01  day-of-week-1 PIC 9(1).
 01  time-1        PIC 9(8).
PROCEDURE DIVISION.
    ACCEPT  date-1        FROM DATE.
    ACCEPT  day-1         FROM DAY.
    ACCEPT  day-of-week-1 FROM DAY-OF-WEEK.
    ACCEPT  time-1        FROM TIME.
    EXIT PROGRAM.
END PROGRAM program-name.
```

Programs Descriptions

This section explains program descriptions for each COBOL division.

ENVIRONMENT DIVISION

No specifications are required.

DATA DIVISION

In the DATA DIVISION, define the data items required to store input data.

PROCEDURE DIVISION

To input the current date and time, use an ACCEPT statement in which DATE, DAY, DAY-OF-WEEK, or TIME are written for the FROM specification.

Program Compilation and Linkage

No specific compiler and linker options are required.

Program Execution

Execute programs as ordinary programs.

When an ACCEPT statement in a program is executed, the current date and time are set for the data-name specified in an ACCEPT statement.

For example: 12-23-1991 (Mon.) 14:15:45.00

Table 43. Entering current date and time

Coding of FROM Specification	Contents Set for the Data-Name
FROM DATE	|9|1|1|2|2|3|
FROM DAY	|9|1|3|5|7|
FROM DAY-OF-WEEK	|1|
FROM TIME	|1|4|1|5|4|5|0|0|

Fetching Command Line Arguments

This section explains how to refer to the counts and values of arguments specified for commands that call programs.

```
IDENTIFICATION DIVISION.
 PROGRAM-ID.  program-name.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
    ARGUMENT-NUMBER IS mnemonic-name-1.
    ARGUMENT-VALUE  IS mnemonic-name-2.
DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  data-name-1 ... .
 01  data-name-2 ... .
 01  data-name-3 ... .
PROCEDURE DIVISION.
    ACCEPT  data-name-1     FROM mnemonic-name-1.
   [DISPLAY numeric literal UPON mnemonic-name-1.]
   [DISPLAY data-name-2     UPON mnemonic-name-1.]
    ACCEPT  data-name-3     FROM mnemonic-name-2
                                 [ON EXCEPTION ... ].
END PROGRAM  program-name.
```

Outline

During program execution, the values of environment variables can be referred to and updated.

To obtain the number of arguments, use an ACCEPT statement for which a mnemonic-name corresponding to function-name ARGUMENT-NUMBER is specified.

To refer to an argument value, use a DISPLAY statement for which a mnemonic-name corresponding to function-name ARGUMENT-NUMBER is specified and an ACCEPT statement for which a mnemonic-name corresponding to function-name ARGUMENT-VALUE is specified.

A character string delimited with spaces or quotation marks (") is counted as one argument.

The following is an example of an input command:

```
SAMPLE  M.HORIUCHI  901234  SHIZUOKA
```

Program Specifications

This section explains program descriptions for each COBOL division when command line argument handling function are used.

ENVIRONMENT DIVISION

Associate the following function-names with mnemonic-names:

- ARGUMENT-NUMBER
- ARGUMENT-VALUE

```
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
    ARGUMENT-NUMBER IS mnemonic-name-1.
    ARGUMENT-VALUE  IS mnemonic-name-2.
```

DATA DIVISION

Define data items to deliver values.

Table 44. Counts and values of arguments

Contents	**Attribute**
Number of arguments	Unsigned integer item
Argument position (not required if specified with a literal)	Unsigned integer item
Argument value	Fixed-length group item or alphanumeric data item

The following is a definition example of data items:

```
DATADIVISION.
 WORKING-STORAGE SECTION.
 01  number-of-arguments  PIC 9(2)  BINARY.
 01  argument-position    PIC 9(2)  BINARY.
 01  argument-value.
     02  argument-value  PIC X(10)  OCCURS 1 TO 10 TIMES
                       DEPENDING ON number-of-arguments.
```

PROCEDURE DIVISION

To obtain the number of arguments, use an ACCEPT statement where a mnemonic-name corresponding to function-name ARGUMENT-NUMBER is specified.

To refer to an argument value, first, specify the argument position with DISPLAY statement (1) corresponding to function-name ARGUMENT-NUMBER. Then, fetch the value with an ACCEPT statement (2) corresponding to function-name ARGUMENT-VALUE.

If the position of a non-existing argument is specified (for example, 4 is specified although there are only three arguments), an exception condition occurs and the statement (3) specified for ON EXCEPTION is executed.

At the start of program execution when a DISPLAY statement is not executed, the argument position is 1.

For each subsequent execution of the ACCEPT statement, the argument position is moved to the next argument.

For argument positioning, 0 to 99 can be specified. 0 is the command itself.

```
DISPLAY  5   UPON mnemonic-name-1.                      ...  (1)
ACCEPT argument-value(5) FROM mnemonic-name-2           ...  (2)
 ON EXCEPTION MOVE 5 TO error-number GO TO ERROR PROCESS   ...  (3)
END-ACCEPT.
```

If an argument value is referred to without executing a DISPLAY statement for positioning, the argument position is 1 at the start of program execution. For each subsequent execution of the ACCEPT statement, the argument position is moved to the next argument.

The lengths of argument values cannot be obtained.

The rules for the COBOL MOVE statement are applied to the setting of data items for the number of arguments and argument values.

```
    :
01  number-1   PIC  9.
01  argument-1 PIC  X(10).
    :
    ACCEPT  number-1 FROM argument-number.   ...  (1)
    ACCEPT argument-1 FROM argument-value.   ...  (2)
```

If statement (1) is executed when the number of arguments specified in the command is 10, the contents of "number-1" is 0.

If statement (2) is executed when the value of an argument to be fetched is "ABCDE", the content of "argument-1" is as follows:

A	B	C	D	E	Blank	Blank	Blank	Blank

If statement (2) is executed when the value of an argument to be fetched is "ABCDE12345FGHIJ", the content of "argument-1" is as follows:

A	B	C	D	E	1	2	3	4	5

Program Compilation and Linkage

No specific compiler and linker options are required.

Program Execution

Execute programs as ordinary programs.

This can be used only with a COBOL program activated by the system.

With a COBOL program called from another COBOL program, the value of an argument to be referred to is that of an argument specified on a command line activated from the system.

Environment Variable Handling Function

This section explains how to refer to and update the values of environment variables. Environment variables explained in this section indicate run-time environment information set at program execution.

```
IDENTIFICATION DIVISION.
 PROGRAM-ID.  program-name.
ENVIRONMENT DIVISION
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
    ENVIRONMENT-NAME   IS mnemonic-name-1.
    ENVIRONMENT-VALUE  IS mnemonic-name-2.
DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  data-name-1 ... .
 01  data-name-2 ... .
PROCEDURE DIVISION.
    DISPLAY { "nonnumeric-literal" }
            {                      }  UPON mnemonic-name-1.
            {  data-name-1         }

    ACCEPT data-name-2
          FROM  mnemonic-name-2 [ON EXCEPTION ...].

    DISPLAY { "nonnumeric-literal" }
            {                      }  UPON mnemonic-name-1.
            {  data-name-1         }

    DISPLAY data-name-2
          UPON  mnemonic-name-2 [ON EXCEPTION ...].

END PROGRAM  program-name.
```

Outline

During program execution, the values of environment variables can be referred to and updated.

To refer to the value of an environment variable, use a DISPLAY statement where a mnemonic-name corresponding to function-name ENVIRONMENT-NAME is specified and an ACCEPT statement for which a mnemonic-name corresponding to function-name ENVIRONMENT-VALUE is specified.

To update the value of an environment variable, use a DISPLAY statement where a mnemonic-name corresponding to function-name ENVIRONMENT-NAME is specified and a DISPLAY statement where a mnemonic-name corresponding to function-name ENVIRONMENT-VALUE is specified.

Programs Specifications

This section explains program descriptions for each COBOL division when using the environment variables handling function.

ENVIRONMENT DIVISION

Associate the following function-names with mnemonic-names:

- ENVIRONMENT-NAME
- ENVIRONMENT-VALUE

DATA DIVISION

Define data items to deliver values.

Table 45. Attributes of environment variables

Contents	**Attribute**
Name of an environment variable (not required if specified with a literal)	Fixed-length group item or alphanumeric data item
Value of an environment variable (not required if specified with a literal)	Fixed-length group item or alphanumeric data item

PROCEDURE DIVISION

To refer to the value of an environment variable, first, specify the environment variable name to be referred to with a DISPLAY statement (1) where a mnemonic-name corresponding to function-name ENVIRONMENT-NAME is specified. Then, refer to the value of the environment variable with an ACCEPT statement (2) where a mnemonic-name corresponding to function-name ENVIRONMENT-VALUE is specified.

If the name of the environment variable to be referred to has not been specified or the name of a non-existing environment variable has been specified, an exception condition occurs. In this case, a statement (3) specified for ON EXCEPTION is executed.

To update the value of an environment variable, first, specify the environment variable name to be updated with a DISPLAY statement (4) where a mnemonic-name corresponding to function-name ENVIRONMENT-NAME is specified. Then, update the value of the environment variable with a DISPLAY statement (5) where a mnemonic-name corresponding to function-name ENVIRONMENT-VALUE is specified.

If the name of the environment variable to be updated has not been specified or the area to set the value of the environment variable cannot be assigned, an exception condition occurs. In this case, a statement (6) specified for ON EXCEPTION is executed.

```
DISPLAY "TMP1" UPON environment-variable-name   ... (1)
ACCEPT value-of-TMP1
       FROM environment-variable-value          ... (2)
           ON EXCEPTION                         ... (3)
           MOVE  occurrence-of-error TO ...
END-ACCEPT.
   :
DISPLAY "TMP2" UPON environment-variable-name.  ... (4)
DISPLAY value-of-TMP2
       UPON   environment-variable-value        ... (5)
           ON  EXCEPTION                        ... (6)
           MOVE  occurrence-of-error TO ...
END-DISPLAY.
```

The lengths of environment variables cannot be obtained.

Program Compilation and Linkage

No specific compiler and linker options are required.

Program Execution

Execute programs as ordinary programs.

The value of an environment variable changed during program execution is valid only in the process being executed by the program, but is invalid for programs after the process is terminated.

Chapter 12. Using SORT/MERGE Statements (Sort-Merge Function)

Sort rearranges the records in a file according to a certain sequence, while merge integrates the records in multiple files into one file. This chapter describes the sort/merge function, looking at the types of sort and merge processing.

Outline of Sort and Merge Processing

This section outlines sort and merge processing.

Sort

Sort means that records in a file are rearranged in ascending or descending order using record information as a sort key. The records are rearranged according to the attribute of the program key item.

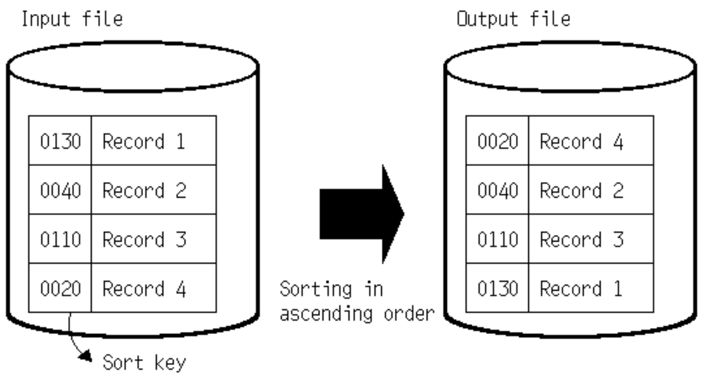


Figure 110. Sorting records

Merge

Merge means that records in multiple files with records sorted in ascending or descending order are integrated into one file.

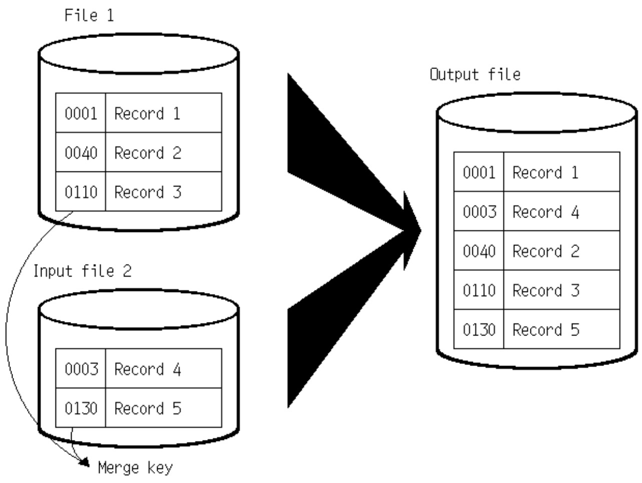


Figure 111. Merging records

Using Sort

This section explains the types of sort processing and how to write, compile, and link a program that uses sort processing.

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. program-name.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
   SELECT  sort-file-1  ASSIGN TO SORTWK01.
  [SELECT input-file-1 ASSIGN TO file-reference-identifier-1 ... .]
  [SELECT output-file-1 ASSIGN TO file-reference-identifier-2 ... .]
DATA DIVISION.
 FILE SECTION.
   SD  sort-file-1
     [RECORD record-size].
   01  sort-record-1.
     ┌──────────────────────────┐
     │ Contents of sort record  │
     │ 02  sort-key-1 ... .     │
     │ 02  data1 ... .          │
     └──────────────────────────┘
    FD  input-file-1 ... .
    01  input-record-1.
     ┌──────────────────────────┐
     │ Contents of input record │
     │ 02  data2 ... .          │
     └──────────────────────────┘
    FD output-file-1 ... .
    01  output-record-1.
     ┌───────────────────────────┐
     │ Contents of output record │
     │ 02  data3                 │
     └───────────────────────────┘
PROCEDURE DIVISION.
 SORT sort-file-1
      ⎧ ASCENDING  ⎫
ON    ⎨            ⎬ KEY sort-key-1
      ⎩ DESCENDING ⎭
      ⎧ USING input-file-1                      ⎫
      ⎨                                         ⎬
      ⎩ INPUT PROCEDURE IS input-procedure-1    ⎭
      ⎧ GIVING output-file-1                    ⎫
      ⎨                                         ⎬
      ⎩ OUTPUT PROCEDURE IS output-procedure-1  ⎭ .
 input-procedure-1 SECTION.
     OPEN INPUT input-file-1.
   input-start.
     READ input-file-1 AT END GO TO input-end.
     MOVE input-record-1 TO sort-record-1.
     RELEASE sort-record-1.
     GO TO input-start.
   input-end.
     CLOSE input-file-1.
 output-procedure-1 SECTION.
     OPEN OUTPUT output-file-1.
 output-start.
     RETURN sort-file-1 AT END GO TO output-end END-RETURN.
     MOVE sort-record-1 TO output-record-1.
     WRITE output-record-1.
     GO TO output-start.
 output-end.
     CLOSE output-file-1.
END PROGRAM program-name.
```

Types of Sort Processing

The following four types of sort processing are possible:

1. All records in the input file are sorted in ascending or descending order, then are written to the output file: (Input) file (Output) file
2. All records in the input file are sorted in ascending or descending order, then are handled as output data: (Input) file (Output) records
3. Specific records or data items are sorted in ascending or descending order, then are written to the output file: (Input) record (Output) file
4. Specific records or data items are sorted in ascending or descending order, and are handled as output data: (Input) record (Output) record

When sorting records in the input file (file to sort) without changing their contents, sort processing type 1 or 2 is normally used. When an input file is not used or the contents of a record is changed, sort processing type 3 or 4 is used.

When writing sorted records to the output file without processing the contents of each record, sort processing type 1 or 3 is used. When the output file is not used or the contents of each record is changed, sort processing type 2 or 4 is used.

Program Specifications

This section explains the contents of a program that sorts records for each COBOL division.

ENVIRONMENT DIVISION

The following files must be defined.

- Sort-merge file

  A work file for sort processing must be defined. The ASSIGN clause is assumed as a comment; up to 8 alphanumeric characters must be specified in it.

- Input file

  Define the same way as for ordinary file processing, if required.

- Output file

  Define the same way as for ordinary file processing, if required.

When performing merge processing within the same program, define only one sort-merge file.

DATA DIVISION

Define the records of files defined in the ENVIRONMENT DIVISION.

PROCEDURE DIVISION

The SORT statement is used for sort processing. The contents of the SORT statement differ depending on what is used for input-output:

- For file input, "USING input-file-name" must be specified.
- For record input, "INPUT PROCEDURE input-procedure-name" must be specified.

- For file output, "GIVING output-file-name" must be specified.
- For record output, "OUTPUT PROCEDURE output-procedure-name" must be specified.

The input procedure specified in INPUT PROCEDURE can pass records to be sorted one-by-one with the RELEASE statement.

The output procedure specified in OUTPUT PROCEDURE can receive sorted records one-by-one with the RETURN statement.

Multiple sort keys can be specified.

When all records are sorted, the sort result is set in special register SORT-STATUS. Special register SORT-STATUS need not be defined in the COBOL program since it is automatically generated.

By checking the SORT-STATUS value after the SORT statement is executed, COBOL program execution can continue even if sort processing terminates abnormally. Setting 16 in SORT-STATUS in the input or output procedure specified by the SORT statement terminates sort processing.

The following table lists the valid values for special register SORT-STATUS and their meanings.

Table 46. SORT-STATUS values and their meanings

Value	**Meaning**
0	Normal termination
16	Abnormal termination

Any input and output files used must be closed during SORT statement execution.

Program Compilation and Linkage

Compiler option EQUALS must be specified as required.

When more than one record has the same sort key value in sort processing, EQUALS guarantees that the records are written in the same order as they are read.

Note: Using this compiler option degrades performance.

Program Execution

Execute the program that uses sort as follows:

1. Set environment variable BSORT_TMPDIR.

A work file called the sort-merge file is required for sort processing. The sort-merge file is temporarily created in the directory specified in environment variable BSORT_TMPDIR.

When the directory is not specified in the environment variable, it is temporarily created in the directory specified in environment variable TEMP. These environment variables must be set in advance.

2. Assign input and output files

When input and output files are defined using file-identifiers, use these identifiers as environment variables to set the names of input and output files.

3. Execute the program

Note: When PowerBSORT is installed, PowerBSORT will be used although the sort/merge process is handled by the COBOL85 run-time system.

Using Merge

This section explains the types of merge processing and how to write, compile, link, and execute a program that uses merge processing.

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. program-name.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
   SELECT  merge-file-1  ASSIGN TO SORTWK01.
  [SELECT input-file-1 ASSIGN TO file-reference-identifier-1 ... .]
  [SELECT output-file-1 ASSIGN TO file-reference-identifier-2 ... .]
DATA DIVISION.
 FILE SECTION.
   SD  merge-file-1
     [RECORD record-size].
   01  merge-record-1.
    +---------------------------+
    | Contents of merge-record  |
    | 02  merge-key-1 ... .     |
    | 02  data1 ... .           |
    +---------------------------+
    FD  input-file-1 ... .
    01  input-record-1.
    +---------------------------+
    |  Contents of input record |
    |  02  data2 ... .          |
    +---------------------------+
    FD output-file-1 ... .
    01  output-record-1.
    +---------------------------+
    |  Contents of output record|
    |  02  data3                |
    +---------------------------+
PROCEDURE DIVISION.
 MERGE merge-file-1
    { ASCENDING  }
 ON {            }  KEY merge-key-1
    { DESCENDING }
    { USING input-file-1                        }
    {                                           }
    { INPUT PROCEDURE IS input-procedure-1      }
    { GIVING output-file-1                      }
    {                                           }
    { OUTPUT PROCEDURE IS output-procedure-1    } .
 input-procedure-1 SECTION.
    OPEN INPUT input-file-1.
  input-start.
    READ input-file-1 AT END GO TO input-end.
    MOVE input-record-1 TO sort-record-1.
    RELEASE sort-record-1.
    GO TO input-start.
  input-end.
    CLOSE input-file-1.
 output-procedure-1 SECTION.
    OPEN OUTPUT output-file-1.
 output-start.
    RETURN sort-file-1 AT END GO TO output-end END-RETURN.
    MOVE sort-record-1 TO output-record-1.
    WRITE output-record-1.
    GO TO output-start.
 output-end.
    CLOSE output-file-1.
END PROGRAM program-name.
```

Types of Merge Processing

The following two types of merge processing are possible:

1. Write all records in multiple files already sorted in ascending or descending order to the output file in ascending or descending order: (Input) file (Output) file
2. Processing all records in multiple files already sorted in ascending or descending order as output data in ascending or descending order: (Input) file (Output) records

When merged records are written to the output file without changing the record contents, merge processing type 1 is normally used. When an output file is not used or record contents are changed, merge processing type 2 is used.

Program Specifications

This section explains the contents of a program that uses merge for each COBOL division.

ENVIRONMENT DIVISION

The following files must be defined:

- Sort-merge file

  A work file for merge processing must be defined. The ASSIGN clause is assumed as a comment; up to 8 alphanumeric characters must be specified in it.

- Input file

  All files to be merged must be defined.

- Output file

  Define the same way as for ordinary file processing, if required.

When sort is performed within the same program, only one sort-merge file must be defined.

DATA DIVISION

Define the records of files defined in the ENVIRONMENT DIVISION.

PROCEDURE DIVISION

The MERGE statement is used for merge processing. The contents of the MERGE statement differ depending on whether file or record output is to be used for merge processing:

- For file output, "GIVING output-file-name" must be specified.
- For record output, "OUTPUT PROCEDURE output-procedure-name" must be specified.

The output procedure specified in OUTPUT PROCEDURE can receive merged records one-by-one by using the RETURN statement.

Multiple merge keys can be specified.

When all records are merged, the merge result is set in special register SORT-STATUS.

Unlike general data, special register SORT-STATUS need not be defined in the COBOL program since it is automatically generated. By checking the SORT-STATUS value after MERGE statement execution, COBOL program execution can continue even if merge processing terminates abnormally.

Setting 16 in SORT-STATUS in the output procedure specified by the MERGE statement terminates merge processing. The following table lists the possible values for special register SORT-STATUS and their meanings.

Table 47. SORT-STATUS values and their meanings

Value	**Meaning**
0	Normal termination
16	Abnormal termination

Any input and output files used must be closed during MERGE statement execution.

Program Compilation and Linkage

No specific compiler or linkage options are required.

Program Execution

1. Set environment variable BSORT_TMPDIR.

A work file called the sort-merge file is required for merge processing. The sort-merge file is temporarily created in the directory specified in environment variable BSORT_TMPDIR.

When the directory is not specified in the environment variable, it is temporarily created in the directory specified in environment variable TEMP. These environment variables must be set in advance.

2. Assign input and output files.

When input and output files are defined using file-identifiers, use these identifiers as environment variables to set the names of input and output files.

3. Execute the program.

Chapter 13. System Program Description (SD) Functions

This chapter describes the system program description (SD) functions used to code a system program. Included in this chapter are explanations of how to use pointers, the ADDR and LENG functions, and the PERFORM statement without an “at end” condition.

Types of System Program Description Functions

COBOL85 has the following functions that are not included in the COBOL standards. These functions are called the system program description (SD) functions in COBOL85. The following SD functions are useful for the description of a system program:

- Pointer
- ADDR function and LENG function
- PERFORM statement without an “at end” condition

The following section outlines and explains the features of each function.

Pointer

An area with a specific address can be referenced and updated using a pointer.

For example, when a parameter identifying an area address is used to call a COBOL program from a program written in another language, the area contents at that address can be referenced or updated with a pointer in the COBOL program.

ADDR Function and LENG Function

The ADDR function can obtain the address of a data item defined by COBOL.

The LENG function can obtain the lengths of a data item and literal defined by COBOL in bytes.

For example, an area address or length can be passed as a parameter to call a program written in another language from a COBOL program.

PERFORM Statement without an “at end” Condition

In COBOL85, the PERFORM statement can be written without setting an “at end” condition. For example, repeat processing can be terminated by determining the processing result during repeat processing.

Using Pointers

This section explains how to use pointers.

```
IDENTIFICATION DIVISION.
 PROGRAM-ID.  program-name.
DATA DIVISION.
 BASED-STORAGE SECTION.
 01  data-name-1 [BASED ON pointer-1].
 77  data-name-2 ... [BASED ON pointer-2].
 WORKING-STORAGE SECTION.
 01  pointer-1  POINTER.
 LINKAGE SECTION.
 01  pointer-2  POINTER.
PROCEDURE DIVISION USING  pointer-2.
 MOVE [pointer-1 =>] data-name-1 ... .
 IF   [pointer-2 =>] data-name-2 ... .
END PROGRAM  program-name.
```

Outline

A pointer is used to reference an area having a specific address. The following data items are required to use a pointer:

- Data item defined in BASED-STORAGE section (1)
- Data item whose attribute is pointer data item (2)

The pointer is normally used with the pointer qualifier (=>). (1) is pointed to by (2). This is called pointer qualification, as shown in the following example:

```
(2) => (1)
```

In this case, the contents of (1) are those of the area whose address is set in (2).

Program Specifications

This section explains the contents of a program that uses pointers for each COBOL division.

ENVIRONMENT DIVISION

No specifications are required.

DATA DIVISION

The data-names where addresses are specified for reference or update must be defined in the BASED-STORAGE section. The data-names for storing addresses (with the attribute of pointer data item (POINTER)) must also be defined in the FILE section, WORKING-STORAGE section, BASED-STORAGE section, and LINKAGE section.

Defining Data-Names in BASED-STORAGE Section

A data-name can be defined in the BASED-STORAGE section by using a data description entry the same way as defining ordinary data.

An actual area is not secured for a data-name defined in the BASED-STORAGE section during program execution. Thus, when referencing a data item defined in BASED-STORAGE, specify the address of the area to reference.

When the BASED ON clause is specified in a data description entry, the data-name can be used without pointer qualification since a pointer is implicitly given by the data-name specified in the BASED ON clause. When using a data-name where the BASED ON clause is not specified, pointer qualification is required.

PROCEDURE DIVISION

A data-name with a pointer given can be specified in such statements as the MOVE and IF statements just like an ordinary data-name.

Program Compilation and Linkage

No specific compiler and linkage options are required.

Program Execution

No specific environment settings are required.

Using the ADDR and LENG Functions

This section explains how to use the ADDR and LENG functions.

```
IDENTIFICATION DIVISION.
 PROGRAM-ID.  program-name.
DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  data-name-1 ... .
 01  pointer-1  POINTER.
 01  data-name-2.
 02  ... [ OCCURS  ...  DEPENDING ON ...].
 01  data-name-3  PIC 9(4) BINARY.
PROCEDURE DIVISION.
 MOVE FUNCTION  ADDR(data-name-1) TO pointer-1.
 MOVE FUNCTION  LENG(data-name-2) TO data-name-3.
END PROGRAM  program-name.
```

Outline

The ADDR function returns the address of a data item as a function value. The LENG function returns the size of a data item or literal in bytes.

Program Specifications

This section explains the contents for each COBOL division of a program that uses the ADDR and LENG functions.

ENVIRONMENT DIVISION

No specifications are required.

DATA DIVISION

The data-names for storing functions values returned by the ADDR and LENG functions must be defined.

The attribute of a function value of the ADDR function is a pointer data item.

The attribute of a function value of the LENG function is a numeric data item.

PROCEDURE DIVISION

The ADDR and LENG functions can be specified in such statements as the MOVE and IF statements, the same as with ordinary data-names.

Program Compilation and Linkage

No specific compiler and linkage options are required.

Program Execution

No specific environment settings are required.

Using the PERFORM Statement without an “at end” Condition

This section explains how to use the PERFORM statement without an “at end” condition.

```
IDENTIFICATION DIVISION.
 PROGRAM-ID.  program-name.
PROCEDURE DIVISION.
 PERFORM WITH NO LIMIT
 IF  ...
 EXIT PERFORM
 END-IF
 :
 END-PERFORM.
END PROGRAM  program-name.
```

Outline

When using a conventional PERFORM statement to determine the “at end” condition during repeat processing, complex program specifications are required. Using the PERFORM statement without an “at end” condition simplifies program specifications.

Program Specifications

This section explains the contents of each division of a COBOL program that uses the PERFORM statement without an “at end” condition.

ENVIRONMENT DIVISION

No specifications are required.

DATA DIVISION

No specifications are required.

PROCEDURE DIVISION

WITH NO LIMIT must be specified for the PERFORM statement to eliminate an “at end” condition.

For repeat processing executed by the PERFORM statement, a statement determining the “at end” condition and a statement to exit repeat processing must be specified.

If a statement for exiting repeat processing is not specified, processing continues infinitely (in an infinite loop).

Program Compilation and Linkage

No specific compiler and linkage options are required.

Program Execution

No specific environment settings are required.

Chapter 14. Communication Functions

This chapter explains the presentation file module (asynchronous message communication / synchronous communication programs) and simplified inter-application communication functions provided for message communication from COBOL programs.

Communication Types

This section explains the features and usage of the three functions provided for message communication from COBOL programs:

- Presentation file module (asynchronous message communication)
- Presentation file module (synchronous communication programs)
- Simplified inter-application communication function.

Table 48. Differences between the presentation file module and simplified inter-application communication function

Feature	**Presentation File**		**Simplified Inter-application**
Communication function	Asynchronous message communication	Synchronous communication programs	Asynchronous message communication
Communication mode	[Client] Windows 3.1 Windows NT [Server] (*A) Windows NT DS90 K series	[Cooperate] Windows 95 Windows NT DS90 HP (*B) S family GS-series SURE system (*B)	[Client] Windows 95 Windows 3.1 Windows NT DS90 HP [Server] (*A) Windows 95 Windows NT DS90 HP

Table 48. Differences between the presentation file module and simplified inter-application communication function (cont.)

Feature	**Presentation File**		**Simplified Inter-application**
Program definition	- Presentation file description - Communication record description - OPEN statement - READ statement - WRITE statement - CLOSE statement		- Communication record description - Call function of simple application communication apparatus ability by CALL statement
Required product(s)	RDB/7000 Server for Windows NT BS*NET (*C)	IDCM	None
Use	Use for message communication	When decentralized development between mutual systems by communication of real time is done	At message communication with COBOL unit

(*A) The application can be started in the server system.
(*B) Cannot cooperate with the COBOL application.
(*C) Cannot cooperate with a Windows client.

Using Presentation File Module (Asynchronous Message Communication)

This section explains how to use the presentation file module for asynchronous message communication, and describes the operating environment and creation of COBOL source programs.

```
IDENTIFICATION DIVISION.
 PROGRAM-ID.  program-name.
ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
    SELECT file-name
      ASSIGN TO GS-file-identifier
      SYMBOLIC DESTINATION IS "ACM"
      DESTINATION-1 IS logical-destination-name-notification-area
     [PROCESSING MODE IS processing-type-notification-area]
     [PROCESSING TIME IS monitoring-time-notification-area]
     [MESSAGE CLASS IS priority-notification-area]
     [FILE STATUS IS input-output-status-1 input-output-status-2].
DATA DIVISION.
 FILE SECTION.
 FD file-name
     [RECORD IS VARYING IN SIZE DEPENDING ON record-size].
 01 communication-record-name ... .
 WORKING-STORAGE SECTION.
 01 logical-destination-name-notification-area  PIC X(8).
[01 processing-type-notification-area           PIC X(2).]
[01 monitoring-time-notification-area           PIC 9(4).]
[01 priority-notification-area                  PIC 9.]
[01 input-output-status-1                       PIC X(2).]
[01 input-output-status-2                       PIC X(4).]
[01 record-size                                 PIC 9(5).]
PROCEDURE DIVISION.
    OPEN I-O file-name.
    MOVE logical-destination-name
          TO logical-destination-name-notification-area.
   [MOVE processing-type TO processing-type-notification-area.]
   [MOVE monitoring-time TO monitoring-time-notification-area.]
   [MOVE priority TO priority-notification-area.]
    WRITE communication-record-name.
    READ  file-name.
    CLOSE file-name.
END PROGRAM program-name.
```

Outline

The presentation file module performs asynchronous message communication using connected product communication functions.

The following figure outlines data transfer through the presentation file module.

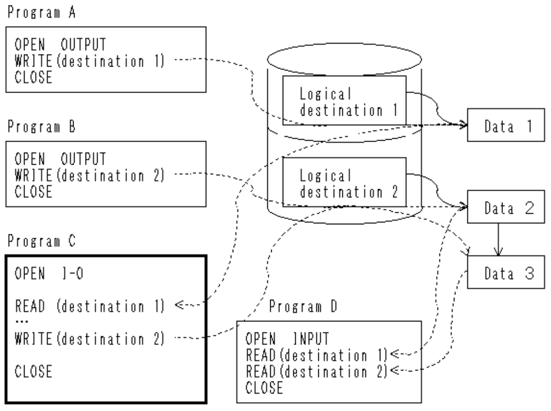


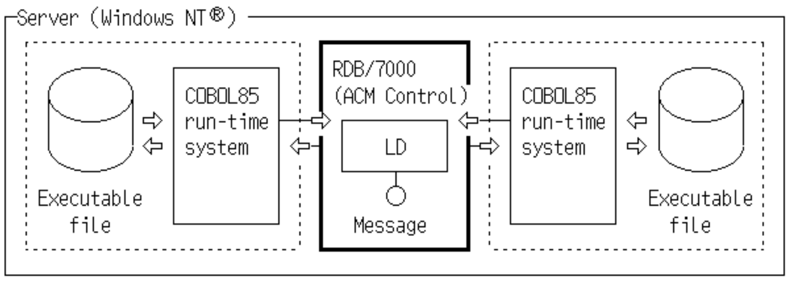
Figure 112. Outline of data transfer using the presentation file module (asynchronous message communication)

Operating Environment

To use the presentation file module, use connected products to enable communication functions.

This section gives an example using RDB/7000 servers for Windows NT. The following figure is a diagram of the operating environment using the presentation file module with the servers.

With Server (Windows NT)



With Client-Server (Windows NT - Windows NT)

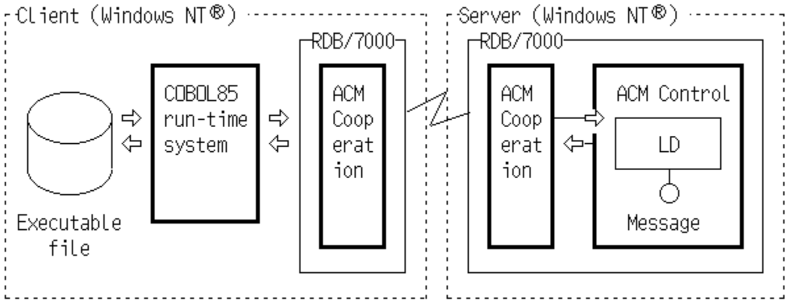


Figure 113. Operating environment for using the presentation file module (asynchronous message communication)

Program Specifications

This section explains the contents of a program using the presentation file module (asynchronous message communication).

ENVIRONMENT DIVISION

Define a presentation file in the same manner as you define a general file. Write a file control entry in the file control paragraph of the input-output section. The following table lists the information to be specified in the file control entry.

Table 49. Information to be specified in the file control entry

	Location	Information Type	Details and Use of Specification
Required	SELECT clause	File name	Specify the file name used in the COBOL program. The file name must comply with the COBOL user-defined words rules.
	ASSIGN clause	File-reference-identifier	Specify in the "GS-file-identifier" format. This file identifier is used as run-time environment information for specifying the connected product name at run-time.
	SYMBOLIC-DESTINATION clause	Destination type	Specify "ACM".
	DESTINATION clause	Data-name	Specify a data-name defined as an 8-byte alphanumeric data item in the WORKING-STORAGE section or LINKAGE section. The logical destination name of input-output processing must be specified for this data-name when communication starts.
Optional	FILE STATUS clause	Data-name	Specify a data-name defined as a 2-byte alphanumeric data item in the WORKING-STORAGE section or LINKAGE section. The execution results of input-output processing must be specified for this data-name. See Appendix B, “I-O Status List” for the value to be specified. Specify a 4-byte alphanumeric data item for detailed information.

Table 49. Information to be specified in the file control entry (cont.)

	Location	**Information Type**	**Details and Use of Specification**
Optional	MESSAGE CLASS clause	Data-name	Specify a data-name defined as a 1-byte numeric item in the WORKING-STORAGE section or LINKAGE section. Input-output processing priority (1-9) must be specified for this name when communication starts. A value of 1 specifies the highest priority. If the default or 0 is specified, the system assumes the lowest level of specified logical destination.
	PROCESSING MODE clause	Data-name	Specify a data-name defined as a 2-byte alphanumeric data item in the WORKING-STORAGE section or LINKAGE section. The type of input-output processing must be specified for this data-name when communication starts. See Table 50.
	PROCESSING TIME clause	Data-name	Specify a data-name defined as a 4-byte numeric item in the WORKING-STORAGE section or LINKAGE section. Input-output processing monitoring time must be specified in seconds for this data-name when communication starts. If the default or 0 is specified, the system assumes infinite wait time.

Table 50. Processing types and values to be specified

	Processing Mode		Control Information
Input	Queuing not specified	"NW"	If there is no data at the logical destination, the system posts "0A" as the input-output status.
	Queuing specified	"WT" Blank	If there is not data at the logical destination, the system follows the monitoring time specification and waits until data is written.
Output	Queuing not specified	"NW"	If the maximum amount of data is written to the logical destination, the system cancels write processing and posts "9G" as the input-output status.
	Queuing specified	"WT" Blank	If the maximum amount of data is written to the logical destination, the system follows the monitoring time specification and waits until data has been read.
	Forced execution specified	"NE"	If the maximum amount of data has been written to the logical destination, the system writes data exceeding the maximum amount and posts "0B" as the input-output status.

DATA DIVISION

Write the record definition to be used for communication and the data definition specified in the file control entry.

PROCEDURE DIVISION

Use input-output statements for communication processing in the same manner as for general file processing:

- OPEN statement : Starts message send-receive processing.
- READ and WRITE statements: Send and receive messages.
- CLOSE statement: Quits message send-receive processing.

OPEN and CLOSE Statements

Execute an OPEN statement to start message send-receive processing. Execute a CLOSE statement to quit message send-receive processing.

READ and WRITE Statements

To send a message, use the WRITE statement where the communication record is specified. To receive a message, use the READ statement where the presentation file is specified.

Before executing the READ or WRITE statement, specify the logical destination name for the data name specified in the DESTINATION-1 clause. Messages are sent to and received from the destination of the specified logical destination name.

Program Compilation and Linkage

No specific compiler or linkage options are required.

Program Execution

This section shows the environment setup for executing a program that sends and receives messages by using the presentation file module.

Specify the name of the connected product to use with the file identifier as the environment variable name.

For example, to specify the ASSIGN clause of the COBOL program:

```
ASSIGN TO GS-ACMFILE
```

The initial file contents are:

```
[program-name]
     :
ACMFILE=,ACM
     :
```

Specify "ACM" (connected product name) for the file identifier specified in the ASSIGN clause of the COBOL file.

Specify a comma (,) before "ACM".

If the connected product is not specified for the file identifier, the connected product specified in the environment variable @CBR_PSFILE_ACM is used.

```
@CBR_PSFILE_ACM=ACM
```

For details, refer to “Environment Variables” in Chapter 5.

To use ACM communication from a COBOL program using the presentation file module, set up the ACM environment before executing the COBOL program.

Using ACM Communication

ACM communication is used through the presentation file module. ACM communication can also be used from COBOL programs by using the CALL statement to call an ACM-supported COBOL subroutine.

Using Presentation File Module (Synchronous Communication Programs)

This section explains how to use the presentation file module for synchronous communication programs, describes the operating environment and the creation of COBOL source programs, and provides instructions on how to communicate between programs.

```
IDENTIFICATION DIVISION.
  PROGRAM-ID.  program-name.
 ENVIRONMENT DIVISION.
  INPUT-OUTPUT SECTION.
  FILE-CONTROL.
    SELECT file-name
     ASSIGN TO GS-file-identifier
     SYMBOLIC DESTINATION IS "APL"
    [SESSION CONTROL IS conversation-information-notification-region
    [MESSAGE MODE IS message-kind-notification-region
    [DESTINATION-1 IS logical-destination-1
    [DESTINATION-2 IS logical-destination-2
    [MESSAGE OWNER IS transmission-right-information-notification-region]
    [MESSAGE CODE IS reason-code-notification-region
    [PROCESSING CONTROL IS extended-message-control-information-region]
    [FILE STATUS IS input-output-status-1 input-output-status-2].
 DATA DIVISION.
  FILE SECTION.
  FD file-name
      [RECORD IS VARYING IN SIZE DEPENDING ON record-size].
WORKING-STORAGE SECTION.
 [01 conversation-information-notification-region PIC X.]
 [01 message-kind-notification-region             PIC X.]
 [01 logical-destination-notification-region-1    PIC X(8).]
 [01 logical-destination-notification-region-2    PIC X(8).]
 [01 transmission-right-information-notification-region PIC ~.]
 [01 reason-code-notification-region              PIC ~.]
 [01 input-output-status-1                        PIC X(2).]
 [01 input-output-status-2                        PIC X(4).]
 [01 size-of-record                               PIC 9(5).]
PROCEDURE DIVISION.
     OPEN I-O file-name.
     [MOVE conversation-information
           TO conversation-information-notification-region.
     [MOVE message-kind TO message-kind-notification-region.]
     [MOVE logical-destination-1
```

```
             TO logical-destination-notification-region-1.]
       [MOVE logical-destination-2
             TO logical-destination-notification-region-2.]
       [MOVE transmission-right-information
             TO transmission-right-information-notification-region.]
       [MOVE reason-code TO reason-code-notification-region.]
       WRITE communication-record-name.
       READ  file-name.
       CLOSE file-name.
   END PROGRAM program-name.
```

Outline

The presentation file module performs synchronous communication programs using the synchronous communication programs of IDCM.

The following figure outlines data transfer through the use of IDCM.

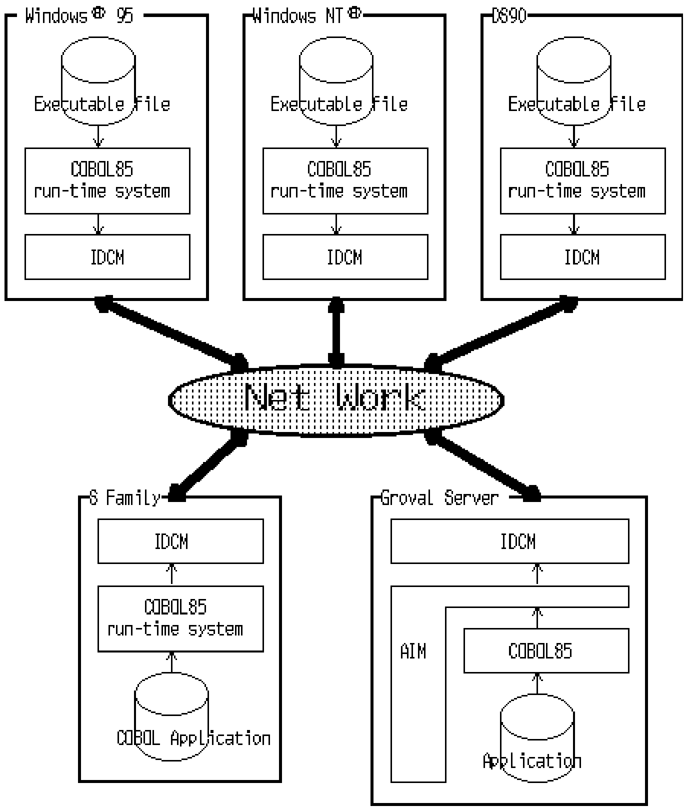


Figure 114. Outline of data transfer using the presentation file module (synchronous communication programs)

Note: Refer to the “IDCM User’s Guide” for the types of networks that can be connected.

Program Specifications

This section explains the contents of a program using the presentation file module (synchronous communication programs).

ENVIRONMENT DIVISION

Define a presentation file in the same manner as you define a general file. Write a file control entry in the file control paragraph of the input-output section.

The following table lists the information to be specified in the file control entry.

Table 51. Information to be specified in the file control entry

	Location	**Information Type**	**Details and Use of Specification**
Required	SELECT clause	File name	Specify the file name used in COBOL program. The file name must comply with the COBOL user-defined words rules.
	ASSIGN clause	File-reference- identifier	Specify in the "GS-file identifier" format. This file identifier is used as run-time environment information for specifying the connected product name at run-time.
	SYMBOLIC-DESTINATION clause	Destination type	Specify "APL".

Table 51. Information to be specified in the file control entry (cont.)

	Location	**Information Type**	**Details and Use of Specification**
Optional	FILE STATUS clause	Data name	Specify a data-name defined as a 2-byte alphanumeric data item in the WORKING-STORAGE section or LINKAGE section. The execution results of input-output processing must be specified for this data-name. See Appendix B, "I-O Status List" for the value to be specified. Specify a 4-byte alphanumeric data item for detailed information.
	SESSION CONTROL clause	Data name	Specify a data-name defined as a 1-byte numeric item in the WORKING-STORAGE section or LINKAGE section. Input-output processing priority (1-9) must be specified for this name when communication starts. A value of 1 specifies the highest priority. If the default or 0 is specified, the system assumes the lowest level of specified logical destination.
	MESSAGE MODE clause	Data name	Specify a data-name defined as a 1-byte numeric item in the WORKING-STORAGE section or LINKAGE section. Input-output processing priority (1-9) must be specified for this name when communication starts. A value of 1 specifies the highest priority. If the default or 0 is specified, the system assumes the lowest level of specified logical destination.

Table 51. Information to be specified in the file control entry (cont.)

Optional	DESTINATION -1 clause	Data-name	Specify a data-name defined as an 8-byte alphanumeric data item in the WORKING-STORAGE section or LINKAGE section. The logical destination name of input-output processing must be specified for this data-name when communication starts.
	DESTINATION -2 clause	Data-name	Specify a data-name defined as an 8-byte alphanumeric data item in the WORKING-STORAGE section or LINKAGE section. The logical destination name of input-output processing must be specified for this data-name when communication starts.
	MESSAGE OWNER clause	Data-name	Specify a data-name defined as a 1-byte numeric item in the WORKING-STORAGE section or LINKAGE section. Input-output processing priority (1-9) must be specified for this name when communication starts. A value of 1 specifies the highest priority. If the default or 0 is specified, the system assumes the lowest level of specified logical destination.
	MESSAGE CODE clause	Data-name	Specify a data-name defined as a 1-32767 digit alphanumeric character in the WORKING-STORAGE section or LINKAGE section. This sets the reason codes that will be given in the event of force-ended communication.
	PROCESSING CONTROL clause	Data-name	Specify a data-name defined as an alphanumeric character of the digit provided by IDCM in the WORKING-STORAGE section or LINKAGE section.

Note: Refer to the online help in the IDCM software development kit for more details on the settings for the optional clauses.

DATA DIVISION

Write the record definition to be used for communication and the data definition specified in the file control entry.

PROCEDURE DIVISION

Use input-output statements for communication processing in the same manner as for general file processing:

- OPEN statement : Starts message send-receive processing.
- READ and WRITE statements: Send and receive messages.
- CLOSE statement: Quits message send-receive processing.

OPEN and CLOSE Statements

Execute an OPEN statement to start message send-receive processing. Execute a CLOSE statement to quit message send-receive processing.

READ and WRITE Statements

To send a message, use the WRITE statement where the communication record is specified. To receive a message, use the READ statement where the presentation file is specified.

Before executing the READ or WRITE statement, specify the logical destination name for the data name specified in the DESTINATION-1 clause. Messages are sent to and received from the destination of the specified logical destination name.

Program Compilation and Linkage

No specific compiler or linkage options are required.

Program Execution

The communication environment of IDCM should be straightened from the execution of the COBOL program when the program which communicates between programs which use the display file function is executed. Refer to the “IDCM User’s Guide” and the online help in the IDCM software development kit for additional details.

Using Simplified Inter-application Communication

This section explains how to use simplified inter-application communication. For more information about how to use simplified inter-application communication, refer to the sample programs in the “Getting Started with Fujitsu COBOL” guide.

Outline

Simplified inter-application communication is used to transfer messages between programs.

Messages are transferred between programs through the logical destination of a communication system process (called a server) that controls messages.

It generates logical destinations for storing messages in the server.

A user send-receive program is called a client. Before starting the client, connect a client logical destination (local logical destination) to a server logical destination (remote logical destination).

Consequently, if read-write processing is specified for the client logical destination, data is read from or written to the connected server logical destination.

A server can be operated under Windows 95 and Windows NT, and a client can be operated under Windows 95, Windows 3.1 and Windows NT.

The following figure outlines simplified inter-application communication.

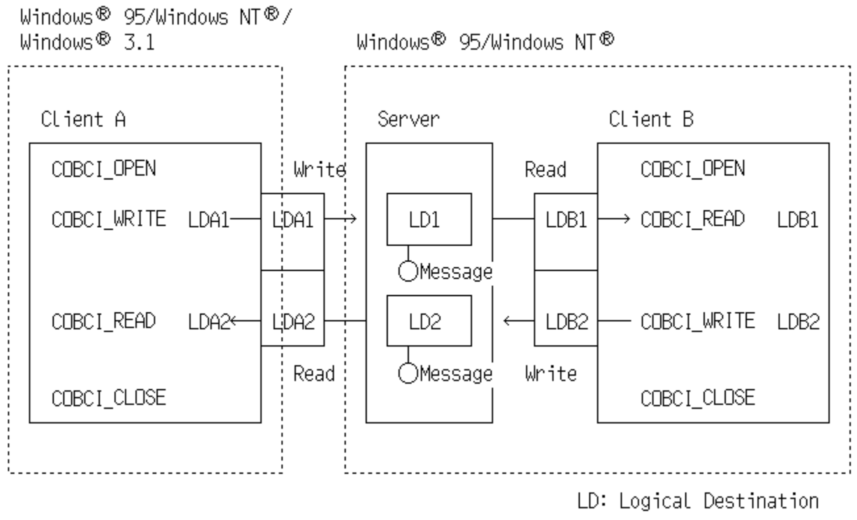


Figure 115. Outline of simplified inter-application communication

Operating Procedures

This section explains procedures for server and client operations used to execute simplified inter-application communication.

The summary of procedures for server operation is:

1. Set up the operating environment
2. Start the server (*)
3. Generate logical destinations (*)
4. Change the logical destination mode (*)
5. Collect logs (messages are received or sent by clients), display logical destination information
6. Quit the server

(*) These steps can be performed automatically by using the definition file.

The details of the above steps are given below:

1. Simplified inter-application communication enables communication using the TCP/IP protocol with the Windows socket API. To use simplified inter-application communication, set up an environment where the Windows socket API and TCP/IP protocol can be used.

- host file setup

  The host name and IP address of the machine where the server operates must be defined in the host files of the machines where clients operate.

- services file setup

  A service name and port number must be defined in the services files of the machines where the server and clients operate.

  Any number can be defined as a port number unless the number is defined in another entry. A port number must be consistent in local area networks (LANs) that use the simplified inter-application communication function. For example:

  Service name: cobci

  Port number/protocol: 20000/tcp

  Using Windows 95:

  After the TCP/IP protocol is installed, the hosts file and the services file must be created using an editor. These files are stored in the directory where Windows 95 is installed. **Note**: the sample files (hosts.sam, services.sam) are also stored in this directory.

Using Windows NT:

If the TCP/IP protocol is installed, the hosts files and services files are created in directory system32/drivers/etc under the directory where Windows NT is installed.

2. to 4. Execute COBCISRV.EXE to start the server.

For screen operation and setup, see “Communication System Environment Setup Dialog Box” and “Simplified Inter-application Communication Server Window.”

The logical destinations have no permission for reading or writing messages immediately after they are generated. Change the logical destination mode to give read/write permission to these destinations. The clients can then be started.

Only one server can be started on a machine.

Note: Steps 2 to 4 can be omitted by using a definition file. See “Starting by Using the Server Definition” for more details.

5. Logs can be collected, and logical destination information can be displayed during client execution. For screen operation and setup, see “Simplified Inter-application Communication Server Window.”

6. to 9. Switch the logical destination mode back to the original mode. The logical destinations no longer have permission for reading and writing messages. Delete the messages and logical destinations, then quit the server.

The procedure for client operation:

1. Set up the operating environment

   Define the information in the hosts files and services files. See step 1 of the procedure for server operation.

2. Create a logical destination definition file

Start the logical destination definition file creation utility ((16): COBCIU16.EXE, (32): COBCIU32.EXE) to create a logical destination definition file. For the screen operation and setup, see “Logical Destination Definition File Creation Utility.”

3. Start the program

   A client requires information regarding the logical destination definition file at run time. Therefore, specify @CBR_CIINF=definition-file-name in the initial file for COBOL execution or in the Run-time Environment Setup window. Windows 95 and Windows NT are designed so that the definition file name can be directly specified in the environment variable.

The definition file name can be specified with the relative path from the directory containing the executable client file.

Server Operation Windows

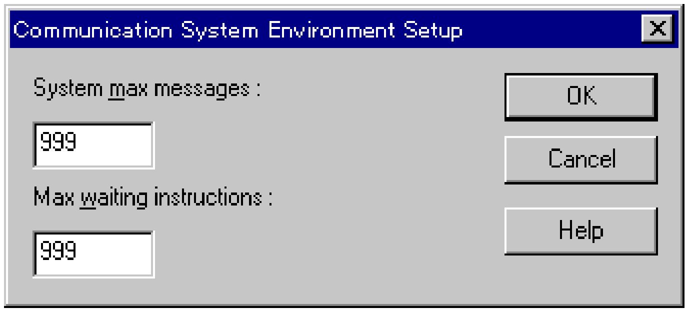
This section explains procedures for operating server windows:

- Communication System Environment Setup dialog box
- Simplified Inter-application Communication Server window

See “Log Collection” for the server error display format and response to server errors.

Communication System Environment Setup Dialog Box

Start the server, specify the maximum number of system messages and maximum waiting instructions in the Communication System Environment Setup dialog box, then click on the OK button.



System max messages :

999

Max waiting instructions :

999

Figure 116. The Communication System Environment Setup dialog box

The server can be used. The Simplified Inter-application Communication Server window is displayed.

- System max messages (0 to 999999999)

  Specify the maximum number of messages that can be stored in the server.

  If the default or 0 is specified, the system uses the maximum value. For the value to be specified, see “Estimating Memory.”

- Max waiting instructions (0 to 999999999)

  Specify the maximum number of read and write instructions from clients that can be queued in the server.

  If the default or 0 is specified, the system uses the maximum value. For the value to be specified, see “Estimating Memory.”

Simplified Inter-application Communication Server Window

The following can be done in the Simplified Inter-application Communication Server window:

- Logical destination operation (create, delete, change mode, delete message, quit server)
- Display information
- Logging

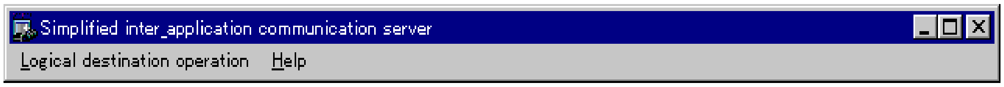
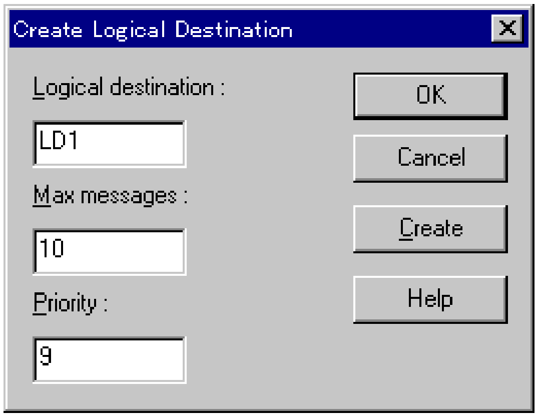

Simplified inter_application communication server
Logical destination operation
Help

Figure 117. Simplified inter-application communication server window

Creating a Logical Destination

To create a logical destination, select Create logical destination from the Logical destination operation menu, specify the logical destination name, the maximum number of messages, and priority in the Create Logical Destination dialog box, then click on the OK button.



Logical destination : LD1
Max messages : 10
Priority : 9

Figure 118. The Create Logical Destination dialog box

A logical destination is generated.

The Create Logical Destination dialog box contains the following elements:

Logical destination edit box

Specify the name of the logical destination to be generated in the server.

The logical destination name can be up to eight alphanumeric characters. The logical destination name for an opponent logical destination is specified in the Client Definition dialog box.

Max messages (0 to 999999999)

Specify the maximum number of messages that can be stored at the logical destination to be generated.

If the default or 0 is specified, the system uses the value specified for "System max messages" in the Communication System Environment Setup dialog box.

Priority (3 to 9)

Specify the number of priority levels in which messages can be stored at the logical destination to be generated.

If the default or a value from 0 to 2 is specified, the system assumes 9.

Deleting a Logical Destination

To delete a logical destination, select Delete logical destination from the Logical destination operation menu, select the logical destination to be deleted in the Delete Logical Destination dialog box, then click on the OK button.

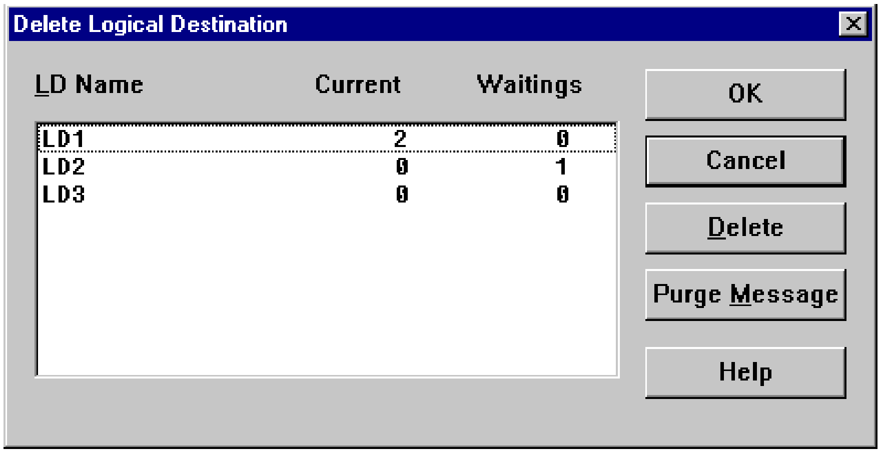
The following are the requirements for deleting a logical destination:

- The logical destination contains no messages

- Inactive is specified for the read and write attributes

If Purge Message button is clicked, the message stored in the selected logical destination can be deleted.

Other logical destinations can be deleted continuously by pressing the Delete button.



LD Name	Current	Waitings
LD1	2	0
LD2	0	1
LD3	0	0

Figure 119. The Delete Logical Destination dialog box

The Delete Logical Destination dialog box contains the following elements:

LD name

Displays the logical destination name.

Current

Displays the number of the messages stored in a logical destination.

Waitings

Displays the number of the instructions waiting in a logical destination.

OK button

Click to delete selected logical destinations and close the Delete Logical Destination dialog box.

Cancel button

Click to cancel any deletions selected and return to the state before the Delete Logical Destination dialog box was opened.

Delete button

Click to continuously delete other logical destinations.

Purge Message button

Click to delete the messages stored in the selected logical destination.

Help button

Click to access the online help.

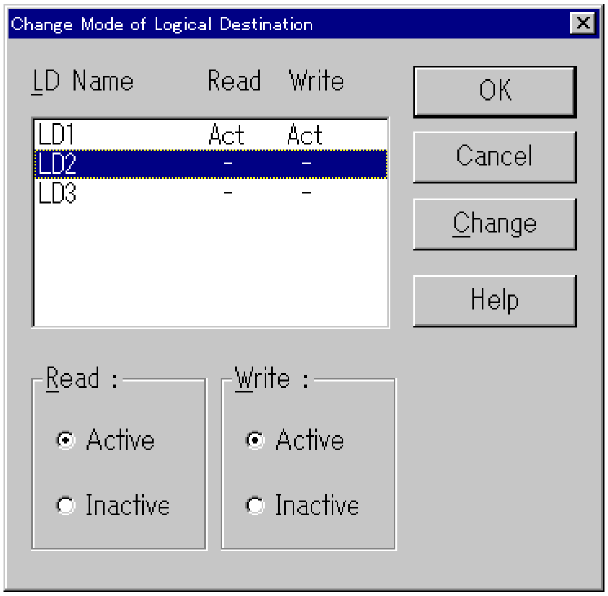
Changing a Logical Destination Mode

To change the mode of a logical destination, select Change mode from the Logical Destination Operation menu, then change the read and write attributes of the logical destination in the Change Mode of Logical Destination dialog box.

Select a logical destination name from the list box, select either "Active" or "Inactive" for the read and write attributes, then click on the OK button.

The logical destination attributes are changed.

You can change the mode of another logical destinations by clicking the Change button.



LD Name	Read	Write
LD1	Act	Act
LD2	-	-
LD3	-	-

Read : (● Active) ○ Inactive

Write : (● Active) ○ Inactive

Figure 120. The Change Mode of Logical Destination dialog box

The Change Mode of Logical Destination dialog box contains the following elements:

LD name

Displays the logical destination name.

Read (status line)

Displays the active (Act) or inactive (-) read status of a logical destination.

Write (status line)

Displays the active (Act) or inactive (-) write status of a logical destination.

Read (radio buttons)

To read messages from the logical destination, select the "Active" radio button. If "Inactive" is clicked, messages cannot be read from the logical destination.

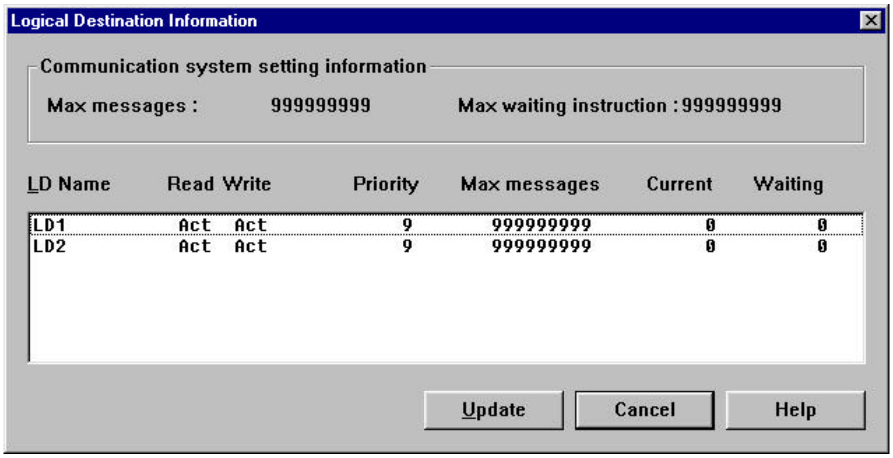
Write (radio buttons)

To write messages to the logical destination, select the "Active" radio button. If "Inactive" is clicked, messages cannot be written to the logical destination.

Displaying Information

To display information, select Display information from the Logical Destination operation menu. The Logical Destination Information dialog box is displayed.

To update information, click on the Update button.



Communication system setting information

Max messages : 999999999   Max waiting instruction : 999999999

LD Name	Read	Write	Priority	Max messages	Current	Waiting
LD1	Act	Act	9	999999999	0	0
LD2	Act	Act	9	999999999	0	0

Figure 121. The Logical Destination Information dialog box

The Logical Destination Information dialog box contains the following elements:

Communication system set information

- Max messages

Displays the maximum number of messages that can be stored in the server.

- Max waiting instruction

Displays the maximum number of read and write instructions that can be queued in the server.

LD name

Displays the names of generated logical destinations.

Read

Displays the read attribute of each logical destination. "Act" indicates that read is enabled; "-" indicates that read is disabled.

Write

Displays the write attribute of each logical destination. "Act" indicates that write is enabled; "-" indicates that write is disabled.

Priority

Displays the number of priority levels where messages can be stored at each logical destination.

Max messages

Displays the maximum number of messages that can be stored at each logical destination.

Current

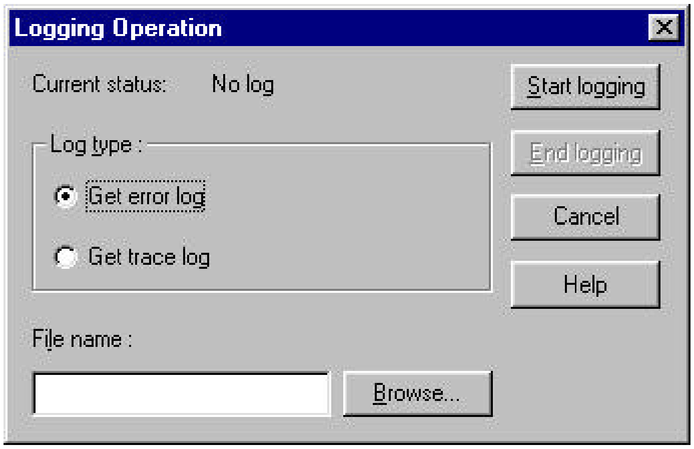
Displays the current number of messages that are stored at each logical destination.

Waiting

Displays the number of read and write instructions that are queued at each logical destination.

Log Collection

To collect logs, select Operate from the Logging menu. The Logging Operation dialog box is displayed.



Current status: No log

Log type :

[✓ Get error log]

Get trace log

File name :

Figure 122. The Logging Operation dialog box

Acquire the log according to the following procedure:

1. Specify the name of the file for log collection.
2. Select either "Get error log" or "Get trace log".
3. Click on the Start logging button.

Log collection starts.

To quit log collection, click on the End logging button.

The name of the file for log collection can be specified with an absolute or relative path. The path from the current server directory serves as the relative path.

If the specified file does not exist, a new file is created. If the specified file exists, log information is appended to the file.

Logs are classified into error information and trace information. Select "Get error log" to collect server error information in the file. Select "Get trace log" to write server error information, processing requests from clients, and processing results in the file.

Error information should be collected during server operation so that server errors can be ascertained.

Use the trace information when you debug the client because there is much output information.

The following shows the log output format:

```
< (1) >  [ (2) ]  ID: (3)  Client: (4)  EXE: (5)   (6)
```

(1) Collection date and time
(year / month / day / hours:minutes:seconds)

(2) Event

Table 52. Log Collection Explanation of Events

Event	**Explanation**
OPEN req.	The client requested connection.
CLOSE req.	The client requested disconnection.
READ req.	The client requested read processing.
WRITE req.	The client requested write processing.
OPEN res.	Connection processing for the client ended.
CLOSE res.	Disconnection processing for the client ended.
READ res.	Read processing ended.
WRITE res.	Write processing ended.
Communication error	An error occurred during communication with the client.
API error	A Windows function error occurs.
Internal conflict	Internal conflict occurred.

(3) Client identification number

(4) Client IP address

(5) Name of the client execution module (12 bytes). Excess bytes are truncated.

(6) Additional information

Table 53. Log Collection Additional Information

Event	Additional Information
OPEN req.	----
CLOSE req.	----
READ req.	Parameters of the COBCI_READ function (server-logical-destination-name, buffer-length, processing-type, and monitoring-time)
WRITE req.	Parameters of the COBCI_WRITE function (server-logical-destination-name, message-length, priority, processing-type, and monitoring-time)
OPEN res.	Error code (rtn: displayed in decimal) (*1) detail code (detail: displayed in decimal) (*1)
CLOSE res.	Error code (rtn: displayed in decimal) (*1) detail code (detail: displayed in decimal) (*1)
READ res.	Error code (rtn: displayed in decimal) (*1) detail code (detail: displayed in decimal) (*1)
WRITE res.	Error code (rtn: displayed in decimal) (*1) detail code (detail: displayed in decimal) (*1)
Communication error	Windows Sockets function name, error code (err: displayed in decimal) (*1) error content
API error	Windows function name, error code (err: displayed in decimal) (*2) error content
Internal conflict	Error content

*1 See "Error Codes."
*2 See the Windows function error codes.

Only communication errors, API errors, and internal conflicts are output as error information.

The information from (3) to (6) is not always output.

The following table lists the responses to errors.

Table 54. Error responses

Event	**Response**
Communication error	Take action according to the corresponding error code (See the Windows Sockets function error codes.) If the error value is 0, a client communication error has probably occurred. See "Error Codes."
API error	A probable cause of the error is insufficient memory or disk resources. Review the operating environment.
Internal conflict	Contact a Fujitsu systems engineer (SE).

Quitting a Server

To quit or exit a server, select End server from the Logical destination operation menu.

All messages stored in a logical destination will be deleted when End server is selected and the acquisition of the log is stopped.

Starting by Using a Server Definition File

The inter-application communication server can be started by creating a server definition file which reads information on each logical destination.

Creating a Server Definition File

Use an editor to create a server definition file.

Note: If you change information on logical destinations (create new logical destinations, change log file names, etc.) do not update this file after starting the inter-application communication server.

Specifying a Server Definition File

If the execution environment information @CBR_CI_SRVINF is not specified when the COBCISRV.EXE is started, the

COBSVINF.INI file in the directory where COBCISRV.EXE is stored will be used.

If the execution environment information @CBR_CI_SRVINF is specified, it will then be used.

Use the AUTOEXEC.BAT file in Windows 95 to set the execution environment information and use the System in the Control Panel in Windows NT.

The full path name and the relative path name can be used for the file name specified for execution environment information @CBR_CI_SRVINF. When the relative path name is used, it becomes the relative path from the current directory when the Inter-application Communication Server window is started. If the specified file does not exist, the Communication System Environment Set dialog box is displayed.

Server Definition File Description

[SRVINF] section

Necessary information is specified by the Communication System Environment Set dialog box and the Log Acquisition Operation dialog box. The start and end methods for the inter-application communication server window can also be specified.

1. SysMaxMsg

```
SysMaxMsg=n
```

Specify the maximum number of system storage messages set in the Communication System Environment Set dialog box. The default is "999999999".

2. SysMaxWait

```
SysMaxWait=n
```

Specify the number of system waiting instructions set in the Communication System Environment Set dialog box. The default is "999999999".

3. StartMode

```
StartMode= { WINDOW | ICON }
```

Specify how to display the screen on server startup. You can have a window (WINDOW) or an icon (ICON) appear at startup.

4. IconMenu

```
IconMenu= { SYSDEF | CLOSEONLY }
```

Specify the type of system menu that starts with the icon.

SYSDEF: The default system menu is displayed.

CLOSEONLY: Only "Close" is displayed in the system menu. In this instance, you cannot change from an icon to a window. Only when "ICON" is specified in "StartMode" can you change from an icon to a window.

5. EndMode

```
EndMode= { MANUAL | AUTO }
```

Specify the action of the Inter-application Communication Server window when Windows is shut down.

MANUAL: Displays a message which states that the Inter-application Communication Server window is still running after ending Windows. After the Inter-application Communication Server window is ended, Windows should be shut down again.

AUTO: The Inter-application Communication Server window ends with Windows shutdown. No message is displayed.

6. Log

```
Log= { YES | NO }
```

Specify whether or not to acquire the log. Specify either "YES" or "NO".

7. LogType

```
LogType= { ERROR | TRACE }
```

Specify the log type set in the Log Acquisition Operation dialog box.

ERROR: Specify the error log.

TRACE: Specify the trace log.

Only when "YES" is specified for "Log", will this item will become effective.

8. LogFileName

```
LogFileName=XXXXXXXX.XXX
```

Specify the log file name set in the Log Acquisition Operation dialog box.

The full path name and the relative path name can be used for the log file name. When the relative path name is used, it becomes a relative path from the current directory.

The default is "COBCISRV.LOG".

Only when "YES" is specified for "Log", will this item will become effective.

[LDINF] section

Necessary information is specified by the Logical Destination Creation dialog box and the Logical Destination Mode Change dialog box.

Specify this section in the following manner:

```
logical destination name= number of maximum storage message,
number of maximum priority level, logical destination mode
```

If information regarding the number of maximum storage messages is omitted, "," should still be specified.

1. Logical destination name

   Specify the logical destination name set in the Logical Destination Creation dialog box.

   The logical destination name can be up to 8 alphanumeric characters in length. There is no default.

2. Number of maximum storage messages

   Specify the number of maximum storage messages set in the Logical Destination Creation dialog box.

   The default is the maximum number of system storage messages (SysMaxMsg).

3. Number of maximum priority level

   Specify the maximum priority level number set in the Logical Destination Creation dialog box.

   The default is "9".

4. Logical destination mode

   Specify each logical destination mode set in the Logical Destination Mode Change dialog box. When the server is started, each logical destination will be set in the mode according to its specification.

   RDWR : Read / write is acceptable.

   READ : Read is acceptable; write is not.

   WRITE : Read is not acceptable; write is acceptable.

   The default is "RDWR".

Note: A line in the server definition file which starts with a semicolon (;) is interpreted as a comment.

The following is an example of a server definition file.

```
[SRVINF]
; Number of System maximum storage message
SysMaxMsg=100
; Number of System waiting instruction
SysMaxWait=50
; Screen display when server starts
StartMode=ICON
; System menu when starts with icon
IconMenu=
; Movement when Windows is ended
EndMode=AUTO
; Presence of log acquisition
Log=YES
; Log kind
LogType=TRACE
; Log file name
LogFileName=LOGDATA.TXT

[LDINF]
; Logical destination name= number of maximum storage
; message, number of maximum priority level, logical destination
; mode
LD1=100, 9, READ
LD2= , ,WRITE
LD3=500, ,
LD4= , ,
```

Client Operation Window

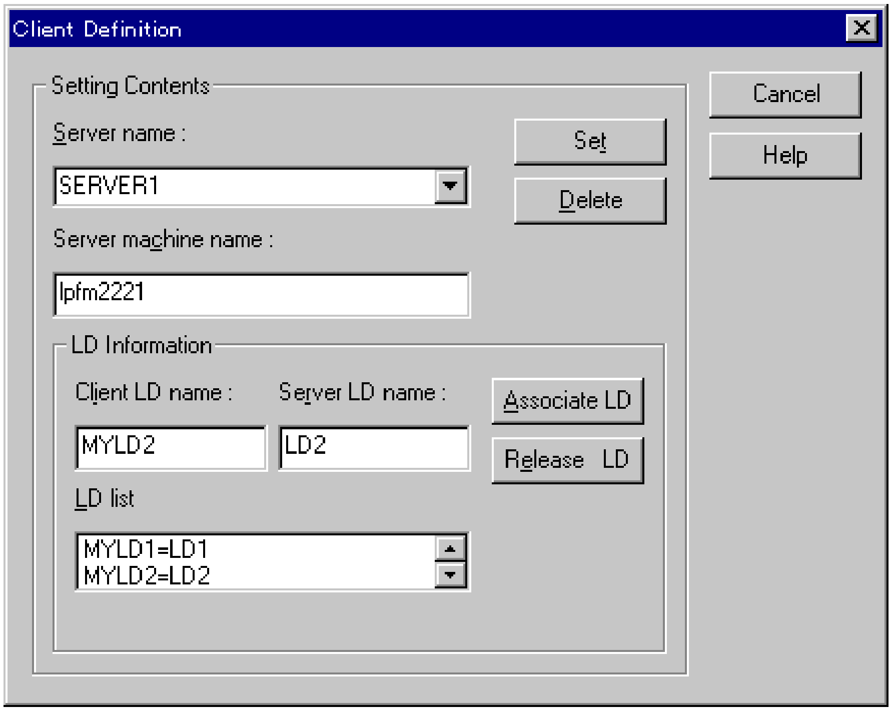
This section explains the procedure for operating the logical destination definition file creation utility in a client window.

Logical Destination Definition File Creation Utility

Start the logical destination definition file creation utility ((16): COBCIU16.EXE, (32): COBCIU32.EXE), specify the name of the definition file to be created in the Select Definition File dialog box, then click on the OK button.

If the specified file does not exist, a new file is created. If an existing file is specified, information is appended to the file.

The Client Definition dialog box is displayed.



Server name : SERVER1

Server machine name : lpfm2221

Client LD name : MYLD2

Server LD name : LD2

LD list

MYLD1=LD1
MYLD2=LD2

Figure 123. The Client Definition dialog box

The Client Definition dialog box contains the following elements:

Server name

Specify a logical name for the server that sends messages to or receives messages from a client. You can specify any server name. The server name can be up to 15 alphanumeric characters in length.

Server machine name

Specify the host name of the machine at which the server is operating. The server machine name can be up to 15 alphanumeric characters in length.

Client LD name

Specify the logical destination name of the client. The client logical destination name can be up to eight alphanumeric characters in length.

Server LD name

Specify the logical destination name of the server to be connected to the client logical destination. The server logical destination name can be up to eight alphanumeric characters in length.

Specify the server name, then specify the server machine name, client logical destination name, and server logical destination name. If information (server machine name, client logical destination name, or server logical destination name) is already defined for the specified server name, the information is displayed.

Only one server machine name can be specified for a server name. Client logical destination names and server logical destination names, however, can be specified for a server name.

Verify that the selected server name is correct, then register a new server name by clicking on the Associate LD button.

If a logical destination and its opponent logical destination are both applied to an already registered server name, use the Delete button in the Logical Destination Information group box to select the correct logical destination.

To delete all the information (server machine name, client logical destination name, and server logical destination name) of the specified server, click on the Delete button in the Setting Contents group box.

If the specified file does not exist, a new file is created. If an existing file is specified, information is appended to the file.

The following is an example of creating a logical destination definition file with this utility:

```
[SERVERNAME_DEF_LIST]
SERVER1=SERVERNAME          }
SERVER2=SERVERNAME          }    (1)
SERVER3=SERVERNAME          }

[SERVER1]                        (2)
@HOSTNM=SRV1                     (3)

[SERVER1.LD]                     (4)
MYLD1=LD1                   }
MYLD2=LD2                   }    (5)
MYLD3=LD3                   }

[SERVER2]
    :
[SERVER2.LD]
    :
[SERVER3]
    :
[SERVER3.LD]
    :
```

(1) server-name=SERVERNAME

(2) [server-name]

(3) @HOSTNM=server-machine

(4) [server-name.LD]

(5) client-LD-name=server-LD-name

Estimating Memory

This section explains how to estimate memory for the server when simplified inter-application communication is used.

- Memory for messages stored at a logical destination

  A message written to a logical destination is managed with an additional control area of 160 bytes. The memory required for a message is the message length + 160 bytes.

- Memory for queuing instructions

  A queuing read instruction is managed in a control area of 160 bytes. The memory required for a queuing read instruction is 160 bytes.

As with a message stored at a logical destination, a queuing write instruction is managed with an additional control area of 160 bytes. The memory required for a queuing write instruction is the message length + 160 bytes.

The maximum memory required for the server depends on the following values specified in the system:

- System max messages (Maximum number of messages)
- Max waiting instructions (Maximum number of instructions)

Determine the maximum memory required for the server based on the following formula:

```
Memory required for server =  M * (160 + ML) + WR * 160 + WW *
(160 + ML)
```

where the following conditions are satisfied:

```
MM =< M
MW =< WR + WW
```

- M: Number of messages stored at the logical destination
- ML: Message length (bytes)
- WR: Number of queuing read instructions
- WW: Number of queuing write instructions
- MM: Maximum number of messages specified in the system (System max messages)
- MW: Maximum number of instructions specified in the system (Max waiting instructions)

Notes

1. If there is a forced ending of a client application:

   If the power supply to the client terminal is cut off before the client application writes back to the server, the waiting instruction remains in the server.

   If there is a forced ending of the client application by issuing "Close", the waiting instruction corresponding to the client application is automatically deleted in the server.

2. Messages are remaining when the Inter-application Communication Server window ends:

   When the Inter-application Communication Server window ends, all message on the server will be deleted.

Simplified Inter-application Communication Functions

This section explains the functions used in simplified inter-application communication.

Clients use the COBCI_OPEN, COBCI_CLOSE, COBCI_READ, and COBCI_WRITE simplified inter-application communication functions with the CALL statement to transfer messages between them.

[Calling sequence]

- COBCI_OPEN (Connection to the server)
- COBCI_READ / COBCI_WRITE (Message reading from the server or writing to the server)
- COBCI_CLOSE (Disconnection from the server)

An example of these functions used in an application is shown below.

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. Program-name.
ENVIRONMENT DIVISION.
DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  STATUS-RE.
      05  ERROR                        PIC S9(9)  COMP-5.
      05  DETAIL                       PIC S9(9)  COMP-5.
 01  SERVERNAME-REC       PIC X(15).
 01  SERVERHD-REC             PIC S9(9)  COMP-5.
 01 UNUSED-REC                  PIC S9(9)  COMP-5  VALUE 0.
 01 LDNAME-REC                  PIC X(8).
 01 BUFFER-REC                   PIC X(n).
 01 RECEIVETYPE-REC.
      05  BUFFERLEN              PIC S9(9)  COMP-5.
      05  MSGLEN                     PIC S9(9)  COMP-5.
      05  MSGTYPE                  PIC S9(9)  COMP-5.
      05  WAITTIME                 PIC S9(9)  COMP-5.
      05  UNUSED                     PIC X(16)  VALUE  LOW-VALUE.
 01 SENDTYPE-REC.
      05  MSGLEN                     PIC S9(9)  COMP-5.
      05  PRIORITY                   PIC S9(9)  COMP-5.
      05  MSGTYPE                  PIC S9(9)  COMP-5.
      05  WAITTIME                 PIC S9(9)  COMP-5.
      05  UNUSED                     PIC S9(9)  COMP-5.
PROCEDURE DIVISION.
           :
[Connect with server]
 CALL  "COBCI_OPEN"    USING  BY  REFERENCE  STATUS-REC
                              BY  REFERENCE  SERVERNAME-REC
                              BY  REFERENCE  SERERHD-REC
                              BY  VALUE      UNUSED-REC.
           :
[Read message]
 CALL  "COBCI_READ"    USING  BY  REFERENCE  STATUS-REC
                              BY  VALUE      SERVERHD-REC
                              BY  REFERENCE  LDNAME^REC
                              BY  REFERENCE  RECEIVETYPE-REC
                              BY  REFERENCE  BUFFER-REC
                              BY  VALUE      UNUSED-REC.
           :
[Write message]
 CALL  "COBCI_WRITE"   USING  BY REFERENCE   STATUS-REC
                              BY  VALUE      SERVERHD-REC
                              BY  REFERENCE  LDNAME-REC
                              BY  REFERENCE  SENDTYPE-REC
                              BY  REFERENCE  BUFFER-REC
                              BY  VALUE      UNUSED-REC.
           :
[Disconnect from server]
```

```
 CALL  "COBCI_CLOSE"   USING BY  REFERENCE  STATUS-REC
                             BY  VALUE      SERVERHD-REC
                             BY  VALUE      UNUSED-REC.
           :
END PROGRAM  Program-name.
```

Figure 124. An example of simplified inter-application communication functions

COBCI_OPEN

Establishes a connection between a client and server. The server is identified by the specified server name.

The server identifier is returned when the client and server are connected normally.

Calling Format

```
       :
DATA DIVISION.
01  STATUS-REC.
    05  ERRORCD  PIC S9(9) COMP-5.
    05  DETAIL   PIC S9(9) COMP-5.
01  SERVERNAME-REC  PIC  X(15).
01  SERVERHD-REC    PIC  S9(9) COMP-5.
01  UNUSED-REC      PIC  S9(9) COMP-5  VALUE 0.
PROCEDURE DIVISION.
    CALL  "COBCI_OPEN" WITH C LINKAGE
                        USING  BY REFERENCE  STATUS-REC
                               BY REFERENCE  SERVERNAME-REC
                               BY REFERENCE  SERVERHD-REC
                               BY VALUE      UNUSED-REC.
```

Parameters

STATUS-REC

An error code is returned to ERRORCD; a detail code is returned to DETAIL.

SERVERNAME-REC

The client specifies the name of the server to be connected. The server name must be defined in the logical destination definition file. If the server name is shorter than 15 bytes, the remaining bytes must be padded with spaces.

SERVERHD-REC

The server identifier is returned.

UNUSED-REC

Zero must be set for this area, which is reserved for future expansion.

Return Value

If the client is connected with the server normally, the system returns 0 to special register PROGRAM-STATUS. Otherwise, the system returns -1 to special register PROGRAM-STATUS.

COBCI_CLOSE

Disconnects a client from a server. The server is identified by the server identifier returned at calling of the COBCI_OPEN function.

Calling Format

```
      :
DATA DIVISION.
01  STATUS-REC.
    05  ERRORCD  PIC S9(9) COMP-5.
    05  DETAIL   PIC S9(9) COMP-5.
01  SERVERHD-REC    PIC  S9(9) COMP-5.
01  UNUSED-REC      PIC  S9(9) COMP-5  VALUE 0.
PROCEDURE DIVISION.
    CALL  "COBCI_CLOSE" WITH C LINKAGE
            USING  BY REFERENCE  STATUS-REC
                   BY VALUE      SERVERHD-REC
                   BY VALUE      UNUSED-REC.
      :
```

Parameters

STATUS-REC

An error code is returned to ERRORCD; a detail code is returned to DETAIL.

SERVERHD-REC

Specify the server identifier returned at calling of the COBCI_OPEN function.

UNUSED-REC

Zero must be set for this area, which is reserved for future expansion.

Return Value

If the client is disconnected from the server normally, the system returns 0 to special register PROGRAM-STATUS. Otherwise, the system returns -1 to special register PROGRAM-STATUS.

COBCI_READ

Reads messages from the server logical destination. The message in the highest-priority level is first read. Messages in the same priority level are read in the order they were written.

If there is no message at the logical destination, a method of queuing processing can be specified depending on the processing type specification.

Calling Format

```
      :
DATA DIVISION.
01  STATUS-REC.
    05  ERRORCD   PIC S9(9) COMP-5.
    05  DETAIL    PIC S9(9) COMP-5.
01  SERVERHD-REC  PIC S9(9) COMP-5.
01  LDNAME-REC    PIC X(8).
01  RECEIVETYPE-REC.
    05  BUFFERLEN PIC S9(9) COMP-5.
    05  MSGLEN    PIC S9(9) COMP-5.
    05  MSGTYPE   PIC S9(9) COMP-5.
    05  WAITTIME  PIC S9(9) COMP-5.
    05  UNUSED    PIC X(16) VALUE LOW-VALUE.
01  BUFFER-REC    PIC X(n).
01  UNUSED-REC    PIC S9(9) COMP-5 VALUE 0.
PROCEDURE DIVISION.
    CALL  "COBCI_READ" WITH C LINKAGE
                USING  BY REFERENCE  STATUS-REC
                       BY VALUE      SERVERHD-REC
                       BY REFERENCE  LDNAME-REC
                       BY REFERENCE  RECEIVETYPE-REC
                       BY REFERENCE  BUFFER-REC
                       BY VALUE      UNUSED-REC.
      :
```

Parameters

STATUS-REC

An error code is returned to ERRORCD; a detail code is returned to DETAIL.

SERVERHD-REC

Specify the server identifier returned at calling of the COBCI_OPEN function.

LDNAME-REC

Specify the name of the client logical destination from where messages are to be read. If the logical destination name is shorter than eight bytes, the remaining bytes must be padded with spaces. The logical destination name is specified in the Client Definition dialog box.

RECEIVETYPE-REC

For BUFFERLEN, specify the length of the record area for receiving messages.

Up to 32,000 bytes can be specified for BUFFERLEN.

The length of a message read from the logical destination is returned as MSGLEN.

Specify a processing type for MSGTYPE. The following text explains processing types:

- Queuing not specified

  Specify 0 for MSGTYPE. Consequently, if the system cannot read a message because no message was found at the logical destination, the system returns control immediately. The system returns an error code indicating that there is no message to read.

- Queuing specified

Specify 1 for MSGTYPE. Consequently, the system waits until messages (up to the maximum number of instructions specified in the system) have been written to the logical destination. Monitoring time can specified for WAITTIME at the same time. The specified monitoring time must be in the range from 0 to 999999999 (seconds). If 0 is specified, infinite wait time is assumed. **Note**: Again, this depends on the maximum number of instructions specified in the system.

If messages are not written to the logical destination within the specified time, the system returns the error code.

UNUSED is an area reserved for future expansion. Zero must be set for the area.

BUFFER-REC

Specify the record area for receiving messages from the logical destination. Messages are returned here. Data exceeding the length returned as MSGLEN are not guaranteed.

If a read message exceeds the length specified for BUFFERLEN, the excess data is truncated.

UNUSED-REC

Zero must be set for this area, which is reserved for future expansion.

Return Value

If a message is read normally, the system returns 0 to special register PROGRAM-STATUS. Otherwise, the system returns -1 to special register PROGRAM-STATUS.

The WM_TIMER message is used for checking time-out of monitoring time. There may be a difference between the elapsed time and the specified monitoring time.

COBCI_WRITE

Writes messages to the server logical destination. Priority levels can be specified for the messages to be written. If the logical destination has no free space, a method of queuing processing can be specified depending on the processing-type specification.

Calling Format

```
      :
DATA DIVISION.
01  STATUS-REC.
    05  ERRORCD   PIC S9(9) COMP-5.
    05  DETAIL    PIC S9(9) COMP-5.
01  SERVERHD-REC  PIC S9(9) COMP-5.
01  LDNAME-REC    PIC X(8).
01  SENDTYPE-REC.
    05  MSGLEN    PIC S9(9) COMP-5.
    05  PRIORITY  PIC S9(9) COMP-5.
    05  MSGTYPE   PIC S9(9) COMP-5.
    05  WAITTIME  PIC S9(9) COMP-5.
    05  UNUSED    PIC X(16) VALUE LOW-VALUE.
01  BUFFER-REC    PIC X(n).
01  UNUSED-REC    PIC S9(9) COMP-5 VALUE 0.
PROCEDURE DIVISION.
    CALL  "COBCI_WRITE" WITH C LINKAGE
                 USING  BY REFERENCE  STATUS-REC
                        BY VALUE      SERVERHD-REC
                        BY REFERENCE  LDNAME-REC
                        BY REFERENCE  SENDTYPE-REC
                        BY REFERENCE  BUFFER-REC
                        BY VALUE      UNUSED-REC.
      :
```

Parameters

STATUS-REC

An error code is returned to ERRORCD; a detail code is returned to DETAIL.

SERVERHD-REC

Specify the server identifier returned at calling of the COBCI_OPEN function.

LDNAME-REC

Specify the name of the client logical destination where messages are to be written. If the logical destination name is shorter than eight bytes, the remaining bytes must be padded with spaces.

SENDTYPE-REC

For MSGLEN, specify the length of the messages to be written to the logical destination. Up to 32,000 bytes can be specified for MSGLEN.

For PRIORITY, specify the priority levels of the messages to be written to the server logical destination. The message in the highest-priority level is first read. Messages in the same priority level are read in the order they were written.

A priority level can be specified in the range from 1 (highest) to 9. If the specified priority level does not exist at the server logical destination, the message is stored in the priority level nearest to the one specified. A negative value cannot be specified.

Specify a processing type for MSGTYPE. The following text explains processing types:

- Queuing not specified

  Specify 0 for MSGTYPE. Consequently, if a message cannot be written because the logical destination has no free space, the system immediately returns control. The system returns an error code indicating no free space.

- Queuing specified

  Specify 1 for MSGTYPE. Consequently, the system waits until messages (up to the maximum number of instructions specified in the system) have been written to the logical destination. Monitoring time can be specified for WAITTIME at the same time. The specified monitoring time must be in the range from 0 to 999999999 (seconds). If 0 is specified, infinite wait time is assumed. **Note**: Again, this depends on the maximum number of instructions specified in the system.

  If the logical destination fails to make free space within the specified time, the system returns the error code.

- Forced execution specified

  Specify 2 for MSGTYPE. Consequently, more messages than the maximum number of messages specified for the logical destination can be written.

  In this case, the system returns an error code indicating that more messages than the maximum number have been written. However, messages exceeding the maximum number of messages specified in the system (System max messages) cannot be written.

UNUSED is an area reserved for future expansion. Zero must be set for the area.

BUFFER-REC

Specify the record area containing the messages to be written to the logical destination.

UNUSED-REC

Zero must be set for this area, which is reserved for future expansion.

Return Value

If a message is written normally, the system returns 0 to special register PROGRAM-STATUS. Otherwise, the system returns -1 to special register PROGRAM-STATUS.

The WM_TIMER message is used for checking time-out of monitoring time. There may be a difference between the elapsed time and the specified monitoring time.

Error Codes

This section explains the error codes of functions used with simplified inter-application communication.

If a function quits normally, the system returns 0 to the special register PROGRAM-STATUS. The system also returns 0 as the error code (STATUS-REC ERROR value) and detail code (STATUS-REC DETAIL value).

If a function quits with an error, the system returns -1 to special register PROGRAM-STATUS. In this case, the system returns non-zero values as the error code and detail code. For error codes and detail codes, see Table 55.

Abbreviations of the functions in Table 55:

O: COBCI_OPEN function

C: COBCI_CLOSE function

R: COBCI_READ function

W: COBCI_WRITE function

o: Related

Table 55. Error codes of simplified inter-application communication functions

Error Code	Detail Code	Explanation	Response	O	C	R	W
0	0	Normal termination	----	o	o	o	o
1	257	The message was read normally, but the message was longer than the buffer. The excess data was truncated.	----			o	
2	513	No message was found to read. (Queuing not specified)	----			o	
3	769	Messages exceeding the maximum number of messages specified for the logical destination were stored. (Forced execution specified)	----			o	
4	1025	A message could not be read because no message was found at the logical destination when the specified monitoring time elapsed (Queuing specified)	----			o	
	1026	A message could not be written because the maximum number of messages specified for the logical destination or the system was reached when monitoring time elapsed (Queuing specified)	Delete unprocessed messages, then re-execute.				o

Table 55. Error codes of simplified inter-application communication functions (cont.)

Error Code	Detail Code	Explanation	Response	O	C	R	W
6	1537	A message could not be read because the logical destination was not in the read-enabled state.	Specify Active for the read attribute of the server of the logical destination.			o	
	1538	A message could not be written because the logical destination was not in the write-enabled state.	Specify Active for the write attribute of the server of the logical destination.				o
	1539	An attempt was made to read a message or write a message to a logical destination that had not been generated.	Generate a logical destination in the server.			o	o
7	1793	A message could not be written because the maximum number of messages specified in the logical destination was reached. (Queuing specified)	Delete unprocessed messages, then re-execute.				o
8	2049	Either a message could not be written because the maximum number of messages specified in the system was reached, or a message could not be read or written because the maximum number of instructions specified in the system was reached.	Delete unprocessed messages, then re-execute.			o	o
10	2561	A connection request was issued for an already connected server.	Check whether a connection request was issued repeatedly for the same server.	o			

Table 55. Error codes of simplified inter-application communication functions (cont.)

Error Code	Detail Code	Explanation	Response	O	C	R	W
11	2833	Logical destination name error.	Check whether the specified logical destination name is correct.			o	o
	2834	Processing type error.	Check whether the specified processing type is correct.			o	o
	2835	Message or buffer length error.	Check whether the specified message and buffer lengths are correct.			o	o
	2836	Priority level error.	Check whether the specified priority level is correct.				o
	2837	Monitoring time error.	Check whether the specified monitoring time is correct.			o	o
	2838	Server name error.	Check whether the specified server name is correct.	o			
	2839	Server identifier error.	Check whether the server is connected and if the specified server identifier is correct.		o	o	o
12	3105	The logical destination definition file is not specified correctly.	Check whether the logical definition file is correctly specified in @CBR_CIINF.	o			
	3016	The logical destination definition file contains incorrect specification.	Check whether the data in the logical destination definition file is correct and whether the server machine name (host name specified in @HOSTNM) is defined in the hosts file correctly.	o		o	o

Table 55. Error codes of simplified inter-application communication functions (cont.)

Error Code	Detail Code	Explanation	Response	O	C	R	W
13	XXXX	An error occurred during communication between systems (Detail code XXXX: Windows Sockets function error code. See Table 56).	If this error occurs, the client is disconnected from the server. To resume communication, restart operation from connection (COBCI_OPEN) with the server.	o	o	o	o
14	3633	Memory became insufficient.	Review the operating environment.	o	o	o	o
15	3841	The COBOL environment is not established (16).	Check whether simplified inter-application communication is used via a COBOL program.	o			
	3842	The client was disconnected from the server.	To resume communication, restart operation from connection (COBCI_OPEN) with the server.	o	o	o	o
	3843	The socket in use is not supported in Windows Sockets V1.1.	Use a socket supported in Windows Socket V1.1.	o			
	3844	The client was forcibly ended.	----	o	o	o	o
	4080 or higher	An internal conflict was detected.	Contact a FUJITSU systems engineer (SE).	o	o	o	o

Table 56. Windows Sockets function error codes

Description in WINSOCK.H	Error Code	Explanation
WSAEMFILE	10024	A new socket identifier cannot be generated because all permitted socket identifiers are in use. Quit unnecessary communication connections, then re-execute processing.
WSAENOTSOCK	10038	Data other than socket identifiers is specified in the socket function.
WSAEPROTONOSUPPORT	10043	The specified protocol is not supported. Check whether the TCP/IP operating environment is set up correctly.
WSAESOCKTNOSUPPORT	10044	The specified socket type is not supported by this address family. Check whether the TCP/IP operating environment is set up correctly.
WSAEAFNOSUPPORT	10047	The specified address family is not supported. Check whether the TCP/IP operating environment is set up correctly.
WSAEADDRINUSE	10048	The address for connecting the server is in use.
WASEADDRNOTAVAIL	10049	The client could not be connected with the server. Check whether the remote machine name specified in the client definition file is defined in the hosts file correctly.
WSAENETDOWN	10050	The network subsystem failed. Activate the network.
WSAENETUNREACH	10051	The host where the client was operating could not be connected with the host where the server was operating. Check whether the network is defined correctly.
WSAENETRESET	10052	The network failed. Reactivate the network.
WSAECONNABORTED	10053	Connection was rejected because of a time-out or other error, or processing ended abnormally. Check whether the client and server are operating normally. This error may be caused by interference during communication or by a LAN card error.

Table 56. Windows Sockets function error codes (cont.)

Description in WINSOCK.H	Error Code	Explanation
WSAECONNRESET	10054	The client or server was reset.
WSAENOBUFS	10055	The communication buffer space is insufficient. Increase the communication buffer space or reduce communication traffic.
WSAEISCONN	1056	The client is already connected with the server.
WSAENOTCONN	10057	The client it not connected with the server.
WSAESHUTDOWN	10058	The client was disconnected from the server.
WSAETIMEDOUT	10060	A time-out was detected because the client was not connected with the server within the specified time. Check whether the server is active and whether the machine where the server is operating is connected to the LAN.
WSAECONNREFUSED	10061	Connection with the server was rejected. Check whether the server is active.
WSAEDISCON	10101	The client or server was disconnected.
WSANO_DATA	11004	Service name ‘cobci’ is not defined in the services file. Define the service name ‘cobci’ in the services file.

Chapter 15. Database (SQL)

This chapter covers remote database access (ODBC) and explains how to write embedded SQL in a COBOL program and access a database with the ODBC driver.

The database (SQL) function accesses a database in a server from a PC client using embedded SQL. Embedded SQL is a database manipulation language written in a COBOL source program. The database function enables distributed development and development of a variety of application types.

ODBC (Open DataBase Connectivity), proposed by Microsoft Corp., is an application program interface for database access.

ODBC Outline

This section outlines database access with an ODBC driver from a COBOL application. For an example about using the ODBC driver, refer to sample program SAMPLE11, supplied with the database (SQL) function.

The ODBC driver enables access to one or more databases.

The following figure outlines database access using the ODBC driver.

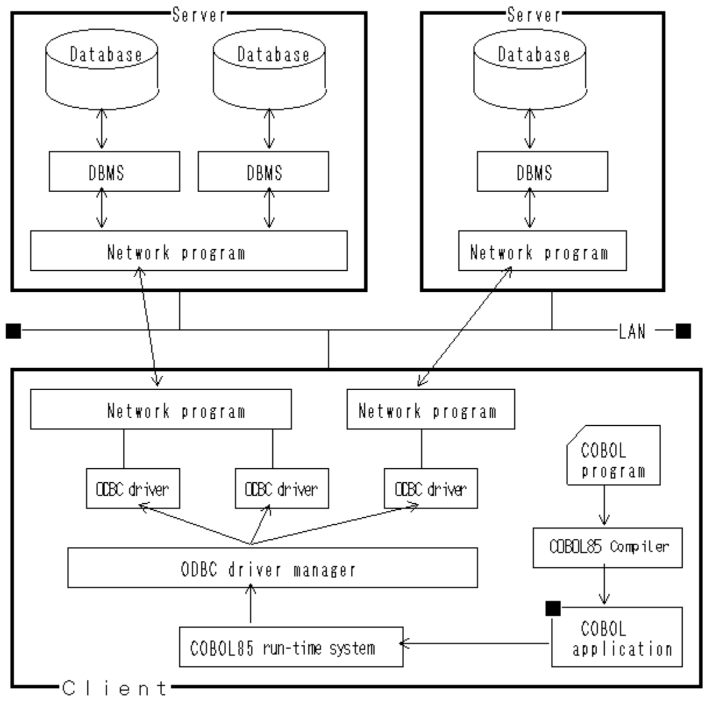


Figure 125. Outline of database access using the ODBC driver

Configuration of a COBOL Program with SQL

The following shows the configuration of a COBOL program with SQL:

```
IDENTIFICATION  DIVISION.
          :
ENVIRONMENT DIVISION.                                  (1)
          :
DATA DIVISION.
          :
WORKING-STORAGE SECTION.
          :
    EXEC SQL BEGIN DECLARE SECTION END-EXEC.           (2)
01  SQLSTATE    PIC X(5).
          :
    EXEC SQL END DECLARE SECTION END-EXEC.
          :
PROCEDURE DIVISION.
          :
    EXEC SQL CONNECT ... END-EXEC.                     (3)
          :
    EXEC SQL DECLARE CUR1 ... END-EXEC.                (4)
          :
    EXEC SQL OPEN CUR1 END-EXEC.                       (5)
          :
    EXEC SQL FETCH CUR1 ... END-EXEC.
          :
    EXEC SQL CLOSE CUR1 END-EXEC.                      (6)
          :
    EXEC SQL ROLLBACK WORK END-EXEC.
          :
    EXEC SQL DISCONNECT ... END-EXEC.                  (7)
          :
    STOP RUN.
```

Figure 126. Configuration of a COBOL program

(1) No SQL-specific description.

(2) Declare SQLSTATE in the DECLARE section. Define a host variable if necessary.

(3) Connect to the server.

(4) Declare the cursor, if using one.

(5) Open the cursor.

(6) Close the cursor.

(7) Disconnect from the server.

Write SQL in a COBOL program as shown in the previous figure. Each SQL statement must begin with EXEC SQL (SQL prefix) and end with END-EXEC (SQL terminator). The database is processed using the SQL statements written in the PROCEDURE DIVISION.

Operations Using Embedded SQL

The following operations can be executed with embedded SQL statements:

- Connection

  A client connects to a server to access a database. The connection enables execution of SQL statements for accessing a server database from the client. To connect a client and server, use the CONNECT statement. See “Connecting to a Server.”

- Selecting a connection

  To select connections, use the SET CONNECTION statement. See “Selecting a Connection.”

- Manipulating data

  To manipulate data, use the SELECT, INSERT, UPDATE, and DELETE statements. If the cursor is defined, the FETCH statement can be used to fetch data. These statements can be executed dynamically. See “Manipulating Data.”

- For the correspondence between data handled in COBOL85 and ODBC, see “Correspondence Between ODBC-Handled Data and COBOL85-Handled Data.”

- Transaction processing

  Consistency of database data manipulation is ensured within transaction units. A transaction starts when the first SQL statement is executed, and terminates when the COMMIT or ROLLBACK statement is executed.

- Disconnection

  To disconnect a program from the server, use the DISCONNECT statement.

  Before executing the DISCONNECT statement, terminate the transaction. See "Disconnecting from a Server."

The following sections provide examples of the above operations:

Connection

This section explains methods for connecting, disconnecting, and selecting a server (connection).

Connecting to a Server

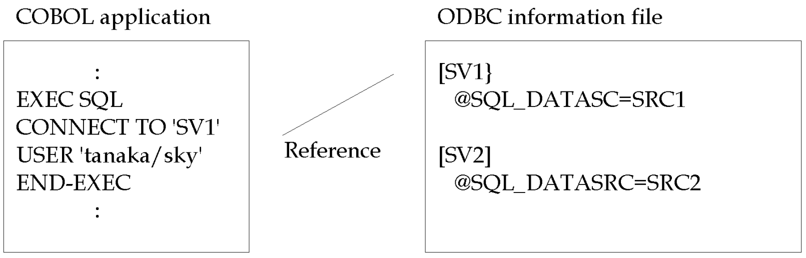
To connect a client to a server, use the CONNECT statement. The following steps should be used for Client/Server connection:

1. Define server information.
2. Connect to the server by either of the following methods:
   - Specify the server name
   - Specify DEFAULT

Connecting by Specifying the Server Name

Before executing the program, define the server information in the ODBC information file. For details on the information to be defined and how to define the information, see “Executing the Program.”

Specify the server name in the CONNECT statement, and execute the statement. This will result in the ODBC information file being searched for the specified server information. If a section is found having the same name, the server information is referenced in definition order to establish a connection to the server.



```
          :
EXEC SQL
CONNECT TO 'SV1'
USER 'tanaka/sky'
END-EXEC
          :
```

```
[SV1}
  @SQL_DATASC=SRC1

[SV2]
  @SQL_DATASRC=SRC2
```

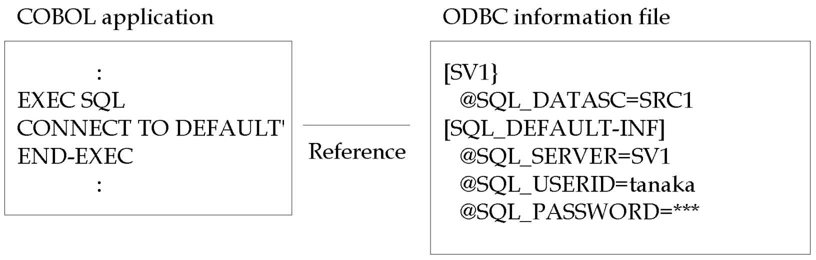
The server information in the ODBC information file is referenced at execution of the CONNECT statement when connecting to the server.

Connecting by Specifying DEFAULT

Before executing the program, define the default connection information in the ODBC information file. For details definition information and how to define the information, see “Executing the Program.”

Specify DEFAULT in the CONNECT statement and execute the statement. This will result in the default connection information file for ODBC to be referenced. The server information is defined in the default connection information. The ODBC information file is searched for the server information. If the server information is found, it is referenced when connecting to the server.

The default connection information is referenced at execution of the CONNECT statement.



```
        :
EXEC SQL
CONNECT TO DEFAULT'
END-EXEC
        :
```

```
[SV1}
  @SQL_DATASC=SRC1
[SQL_DEFAULT-INF]
  @SQL_SERVER=SV1
  @SQL_USERID=tanaka
  @SQL_PASSWORD=***
```

The SV1 information defined in the default connection information is referenced to connect the server.

Disconnecting from a Server

To disconnect a client from a server, use the DISCONNECT statement.

Specify the connection to be terminated in the DISCONNECT statement. The connection name, DEFAULT, CURRENT, or ALL can be specified in the DISCONNECT statement. For the DISCONNECT statement specification, see Figure 127.

Selecting a Connection

To select the available connections, use the SET CONNECTION statement.

More than one server can be connected by executing the CONNECT statement for each database to be connected. If an application connects more than one server, a server for which SQL statements are to be executed must be defined.

If more than one CONNECT statement is executed, the server connected with the last CONNECT statement will be used as the current server (current connection).

SQL statements are executed for the current connection. To execute SQL statements for another connection, use the SET CONNECTION statement to select the appropriate connection.

The following figure is a COBOL program that illustrates the connection to multiple servers. The following example shows the selection of a connection, connecting and disconnecting of these multiple servers. The example has a default connection, and connection to servers SV1, SV2, SV3, and SV4 as defined in the ODBC information file.

```
       :
EXEC SQL CONNECT TO DEFAULT END-EXEC.            (1)
EXEC SQL
  CONNECT TO 'SV1' AS 'CNN1' USER 'tanaka/sky'  (2)
END-EXEC.
EXEC SQL
  CONNECT TO 'SV2' AS 'CNN2  USER 'tanaka/sky'  (3)
END-EXEC.
EXEC SQL
  CONNECT TO 'SV3' AS 'CNN3' USER 'tanaka/sky'  (4)
END-EXEC.
EXEC SQL
  CONNECT TO 'SV4' AS 'CNN4' USER 'tanaka/sky'  (5)
END-EXEC.
EXEC SQL DISCONNECT 'CNN4' END-EXEC.            (6)
EXEC SQL SET CONNECTION 'CNN1' END-EXEC.        (7)
     :
EXEC SQL ROLLBACK WORK END-EXEC.                (8)
EXEC SQL DISCONNECT CURRENT END-EXEC.           (9)
EXEC SQL SET CONNECTION DEFAULT END-EXEC.       (10)
     :
EXEC SQL COMMIT WORK END-EXEC.                  (11)
EXEC SQL DISCONNECT DEFAULT END-EXEC.           (12)
EXEC SQL SET CONNECTION 'CNN2' END-EXEC.
     :
EXEC SQL ROLLBACK WORK END-EXEC.
EXEC SQL DISCONNECT ALL END-EXEC.               (13)
```

Figure 127. Example showing the connection of multiple servers, selecting from them, and disconnecting them

(1) The default connection is enabled.

(2) The server SV1 is connected. This connection name is CNN1.

(3) The server SV2 is connected. This connection name is CNN2.

(4) The server SV3 is connected. This connection name is CNN3.

(5) The server SV4 is connected. This connection name is CNN4. The last-connected CNN4 is the current connection.

(6) The connection name CNN4 is disconnected.

(7) The connection name CNN1 is selected. CNN1 becomes the current connection.

(8) Any changes made in the CNN1 are canceled.

(9) The current connection is disconnected. CNN1 is now the current connection in the example, so CNN1 is disconnected.

(10) Default connection is selected. The default connection becomes the current connection.

(11) Any changes made in the default connection are saved.

(12) The default connection is disconnected.

(13) All available connections are disconnected.

Up to 16 servers can be connected using the CONNECT statement at the same time. The number can vary depending on the ODBC driver and the related environments.

Manipulating Data

This section explains the following data manipulations:

- Retrieving data
- Updating data
- Deleting data
- Inserting data
- Using dynamic SQL
- Using variable length character strings
- Operating the cursor with more than one connection

Also see “Correspondence Between ODBC-Handled Data and COBOL85-Handled Data.”

Sample Database

The following three sample database tables are used in the program examples which follow:

STOCK table: Shows product numbers (GNO), product names (GOODS), quantity in stock (QOH), and warehouse numbers (WHNO).

ORDERS table: Shows company numbers (COMPANYNO), product trade numbers (GOODSNO), purchase prices (PRICE), and order quantities (OOH).

COMPANY table: Shows company numbers (CNO), company names (NAME), telephone numbers (PHONE), and address (ADDRESS).

STOCK table

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

ORDERS table

COMPANY NO.	GOODSNO	PRICE	OOH
61	123	48000	60
61	124	64000	40
61	138	6400	180
61	140	9000	80
61	215	240000	10
61	240	80000	20
62	110	37500	120
62	226	112500	20
62	351	375	800
63	111	57400	80
63	200	123000	60
63	201	164000	50
63	212	205000	30
63	215	246000	10
71	140	7800	50
71	351	390	600
72	137	3500	120
72	140	7000	70
72	215	210000	10
72	226	105000	20
72	243	84000	10
72	351	350	1000
73	141	16000	60
73	380	2400	250
73	390	2400	150
74	110	39000	120
74	111	54000	120
74	226	117000	20
74	227	140400	10
74	351	390	700

COMPANY table

CNO	NAME	PHONE	ADDRESS
61	ADAM LTD.	731-1111	SANTA CLARA CA USA
62	IDEA INC.	423-222	LONDON W.C.2 ENGLAND
63	MOON CO.	143-3333	NEW YORK NY USA
71	RIVER CO.	344-1212	PARIS FRANCE
72	DRAGON CO.	373-7777	SAN FRANCISCO CA USA
73	BIG INC.	391-0808	DALLAS TX USA
74	FIRST CO.	255-9944	SYDNEY AUSTRALIA

Retrieving Data

This section explains methods for retrieving data from a database.

Retrieving Data from All Table Rows

The following figure shows a COBOL program used to retrieve data from all rows of a database table.

```
        :
    EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01  PRODUCT-NUMBER      PIC S9(4) COMP-5.           }
01  PRODUCT-NAME        PIC X(20).                  }
01  QUANTITY-IN-STOCK   PIC S9(9) COMP-5.           }         (1)
01  WAREHOUSE-NUMBER    PIC S9(4) COMP-5.           }
01  SQLSTATE            PIC X(5).                   }
    EXEC SQL END DECLARE SECTION END-EXEC.
PROCEDURE DIVISION.
    EXEC SQL WHENEVER NOT FOUND GO TO:P-END END-EXEC.         (2)
    EXEC SQL
      DECLARE CUR1 CURSOR FOR SELECT * FROM STOCK             (3)
    END-EXEC.
P-START.
    EXEC SQL CONNECT TO DEFAULT END-EXEC.                     (4)
    EXEC SQL OPEN CUR1 END-EXEC.                              (5)
P-LOOP.
    EXEC SQL
      FETCH CUR1 INTO :PRODUCT-NUMBER,                        (6)
                      :PRODUCT-NAME,
                      :QUANTITY-IN-STOCK,
                      :WAREHOUSE-NUMBER
    END-EXEC.
        :
    GO TO P-LOOP.
P-END.
    EXEC SQL CLOSE CUR1 END-EXEC.                             (7)
    EXEC SQL ROLLBACK WORK END-EXEC.                          (8)
    EXEC SQL DISCONNECT DEFAULT END-EXEC.                     (9)
    STOP RUN.
```

Figure 128. Retrieving data from all rows

(1) Specifies an embedded SQL DECLARE section in the WORKING-STORAGE section, and defines the data input areas (all columns of the STOCK table) as host variables. Refer to the “COBOL85 Reference Manual” for the host variable declaration rules.

(2) Operation for an exception event can be specified by specifying the embedded SQL exception declaration. The example specifies NOT FOUND as a condition. Therefore, the exception declaration is effective if "no data" is shown as the SQLSTATE value. The example specifies executing P-END procedure when there is no row fetched by the FETCH statement in (6). Operation for an exception event can also be specified by checking SQLSTATE with a COBOL IF statement.

(3) Declares a cursor to define the cursor name for referring to the STOCK table. The example neither selects any specific columns from the STOCK table nor specifies any search conditions. Thus, a table derived from the query expression is identical to the original STOCK table.

(4) Executes the CONNECT statement to connect the server.

(5) Executes the OPEN statement to enable the specified cursor.

(6) Executes the FETCH statement to fetch data row by row from the table, and sets the values of each column into the corresponding host variable area.

(7) Executes the CLOSE statement to disable the specified cursor.

(8) Executes the ROLLBACK statement to terminate the transaction.

(9) Executes the DISCONNECT statement to disconnect the server.

If an asterisk (*) is specified in the select list of the query expression, the columns are selected in the order specified when the table was defined.

Retrieving Data with Conditions Specified

The following illustrates retrieving data for only rows that meet specified conditions. In the example, the condition of the quantity in stock (QOH) is tested to be less than 51. It retrieves the data (product number, product name, and quantity in stock) of products that meet the condition.

```
        :
    EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01  PRODUCT-NUMBER      PIC S9(4) COMP-5.          }
01  PRODUCT-NAME        PIC X(20).                 }         (1)
01  QUANTITY-IN-STOCK   PIC S9(9) COMP-5.          }
01  SQLSTATE            PIC X(5).                  }
    EXEC SQL END DECLARE SECTION END-EXEC.
PROCEDURE DIVISION.
    EXEC SQL WHENEVER NOT FOUND GO TO :P-END END-
                EXEC.
    EXEC SQL
      DECLARE CUR2 CURSOR FOR                                (2)
      SELECT GNO, GOODS, QOH FROM STOCK
        WHERE QOH < 51
    END-EXEC.
P-START.
    EXEC SQL CONNECT TO DEFAULT END-EXEC.                    (3)
    EXEC SQL OPEN CUR2 END-EXEC.                             (4)
P-LOOP.
    EXEC SQL
      FETCH CUR2 INTO :PRODUCT-NUMBER,                       (5)
                      :PRODUCT-NAME,
                      :QUANTITY-IN-STOCK
    END-EXEC.
        :
    GO TO P-LOOP.
P-END.
    EXEC SQL CLOSE CUR2 END-EXEC.                            (6)
    EXEC SQL ROLLBACK WORK END-EXEC.                         (7)
    EXEC SQL DISCONNECT DEFAULT END-EXEC.                    (8)
    STOP RUN.
```

Figure 129. Retrieving data with conditions specified

(1) The embedded SQL DECLARE section defines all host variables to be specified in embedded SQL statements.

(2) The table is never referred to by declaring a cursor. The table is referred to when the OPEN statement in (4) is executed.

(3) Executes the CONNECT statement to connect the server.

(4) Executes the OPEN statement to create a virtual table including rows that meet the condition specified during cursor declaration in (2). The virtual table is created and rows showing 50 or less in column QOH are extracted from the STOCK table, then the data of columns GNO, GOODS, and QOH are retrieved from the extracted rows. Generally, the row order of a virtual table is undefined.

(5) Executes the FETCH statement to fetch data row by row from the beginning of the virtual table created in (4), then writes the values of each column to the corresponding host variable area. To retrieve the values of a column in ascending or descending order, specify the ORDER BY clause at the end of the query expression in (2).

(6) Executes the CLOSE statement to disable the virtual table created in (4). The virtual table can no longer be referenced by executing SQL statements unless the OPEN statement is executed again.

(7) Executes the ROLLBACK statement to terminate the transaction.

(8) Executes the DISCONNECT statement to disconnect the server.

Retrieving Data from a Single Row

The following are program examples for retrieving the data of individual or multiple rows.

Use the SELECT statement if retrieving the data for only one row. No cursor is used in this case, and cursor declaration and processing using the OPEN, FETCH, and CLOSE statements is unnecessary.

The following is the SELECT statement used to retrieve the product name where the quantity in stock for the product is 200 as noted in the STOCK table:

```
EXEC SQL
    SELECT GOODS, QOH INTO :PRODUCT-NAME,:QUANTITY-IN-STOCK
        FROM  STOCK
    WHERE GNO = 200
END-EXEC.
```

If a set function is specified as the select list value expression, the SELECT statement obtains the radix of the table (the number of rows) and the maximum, minimum, average, and summation (total) of the specified value expression.

The following is the SELECT statement used to retrieve the maximum, minimum, and average of the purchase prices in the ORDERS table:

```
EXEC SQL
 SELECT MAX(PRICE), MIN(PRICE), AVG(PRICE)
 INTO :MAX-VALUE, :MIN-VALUE, :AVG-VALUE FROM ORDERS
END-EXEC.
```

The results are set in the host variables MAX-VALUE, MIN-VALUE, and AVG-VALUE.

Retrieving Data from Related Tables

Retrieving Data from a Table Created by Relating Different Tables

Data can be retrieved from a table created by relating different tables. The tables are related according to their column values.

The following example relates three tables of the sample database in order to retrieve data. The example specifies TELEVISION as a target product name, and retrieves the names of companies which deal with the specified product and the order quantity of each company.

The STOCK and ORDERS tables are related according to the product numbers (column GNO) and product trade numbers (column GOODSNO).

The ORDERS and COMPANY tables are related according to the company numbers (column COMPANYNO) and company numbers (column CNO).

The following shows the query expression:

```
SELECT NAME, OOH
  FROM STOCK, ORDERS, COMPANY
  WHERE GOODS = 'TELEVISION' AND
        GNO = GOODSNO        AND
        COMPANYNO = CNO
```

Rows that meet the search conditions are derived from the table created by relating three tables:

GNO	...	WHNO	COMPANYNO	...	OOH	CNO	...	ADDRESS
110	...	2	61	..	60	61	..	SANTA CLARA CA USA
110	...	2	61	.	60	62	.	LONDON W.C.2
110	...	2	61	..	60	63	..	ENGLAND
				.			.	FIFTH AVENUE NY USA
				..			..	
				.			.	
	~	~	~	~	~	~	~	~
110	...	2	61	..	40	61	..	SANTA CLARA CA USA
110	...	2	61	.	40	62	.	LONDON W.C.2
110	...	2	61	..	40	63	..	ENGLAND
				.			.	FIFTH AVENUE NY USA
				..			..	
				.			.	
	~	~	~	~	~	~	~	~
111	...	2	61	..	60	61	..	SANTA CLARA CA USA
111	...	2	61	.	60	62	.	LONDON W.C.2
111	...	2	61	..	60	63	..	ENGLAND
				.			.	FIFTH AVENUE NY USA
				..			..	
				.			.	
	~	~	~	~	~	~	~	~
390	...	3	74	..	700	72	..	SAN FRANCISCO, CA
390	...	3	74	.	700	73	.	USA
390	...	3	74	..	700	74	..	DALLAS, TX USA
				.			.	SYDNEY, AUSTRALIA
							..	
							.	

Yielding the results of the query expression:

NAME	**OOH**
IDEA INC.	120
MOON CO.	80
MOON CO.	30
FIRST CO.	120
FIRST CO.	120

Retrieving Data from a Table Where Rows are Related

Data can be retrieved from a table where rows are related in the same manner as when different tables are related.

The following example retrieves the names of products that are in the warehouse where televisions (TELEVISION) are stored. Two different aliases (correlation names) are given to the STOCK table. They are treated as if they were different tables.

This COBOL program relates table rows and retrieves the product names in the warehouse which stores televisions (TELEVISION).

```
       :
     EXEC SQL BEGIN DECLARE SECTION END-EXEC.
 01  PRODUCT-NAME   PIC X(20).
 01  SQLSTATE       PIC X(5).
     EXEC SQL END DECLARE SECTION END-EXEC.
 PROCEDURE DIVISION.
     EXEC SQL WHENEVER NOT FOUND GO TO :P-END END-EXEC.
     EXEC SQL
       DECLARE CUR4  CURSOR FOR
         SELECT DISTINCT X2.GOODS
         FROM STOCK X1,STOCK X2                             (1)
         WHERE X1.GOODS = 'TELEVISION' AND
               X1.WHNO  = X2.WHNO
     END-EXEC.
 P-START.
     EXEC SQL CONNECT TO DEFAULT END-EXEC.
     EXEC SQL OPEN CUR4 END-EXEC.
 P-LOOP.
     EXEC SQL
       FETCH CUR4 INTO :PRODUCT-NAME
     END-EXEC.
       :
     GO TO P-LOOP.
 P-END.
     EXEC SQL CLOSE CUR4 END-EXEC.
     EXEC SQL ROLLBACK WORK END-EXEC.
     EXEC SQL DISCONNECT DEFAULT END-EXEC.
     STOP RUN.
```

Figure 130. Retrieving data from a table in which rows are related

(1) Specifies correlation names (X1 and X2) for the STOCK table in the FROM clause of the cursor declaration statement. X1

and X2 are treated as if they are different tables. The example specifies search conditions in the WHERE clause. The column name is qualified with the correlation names. The rows that result show products that are in the warehouse storing televisions (TELEVISION).

Updating Data

Use the UPDATE statement to update table data. The following example specifies the UPDATE statement to decrement each quantity in stock (column QOH) of the STOCK table by 10%:

```
EXEC SQL
  UPDATE STOCK SET QOH = QOH * 0.9
END-EXEC.
```

This UPDATE statement multiplies each value in column QOH of the STOCK table by 0.9, and replaces the original value with the result.

To update values in only rows that meet a specified condition, specify a search condition in the WHERE clause of the UPDATE statement.

The following example changes the above UPDATE statement to decrement only the number of televisions in stock by 10%:

```
EXEC SQL
  UPDATE STOCK SET QOH = QOH * 0.9
  WHERE GOODS = 'TELEVISION'
END-EXEC.
```

The previous method cannot update data in a table created by relating multiple tables.

Deleting Data

Use the DELETE statement to delete data. The following example specifies the DELETE statement to delete all rows showing CASSETTE DECK from the STOCK table:

```
EXEC SQL
  DELETE FROM STOCK WHERE GOODS = 'CASSETTE DECK'
END-EXEC.
```

The previous method cannot delete data from a table created by relating multiple tables.

Inserting Data

Use the INSERT statement to insert data. Select either of the following methods to insert data:

- Inserting the data in only one row
- Inserting the data in a set of rows extracted from another table based on search conditions

Inserting a Single Row

The following example specifies the INSERT statement to add a row showing product number 301 to the STOCK table. The product name is WASHER, the quantity in stock is 50, and the warehouse number is 1:

```
EXEC SQL
  INSERT INTO STOCK (GNO,GOODS, QOH, WHNO)
    VALUES (301, 'WASHER', 50, 1)
END-EXEC.
```

Inserting Multiple Rows from Another Table

The following example assumes that the database of the STOCK table contains another stock table (table name: SUBSTOCK, column names and attributes: same as those of the STOCK table).

The example specifies the INSERT statement to insert rows showing product name MICROWAVE OVEN in the SUBSTOCK table into the STOCK table.

The example also sets the warehouse numbers of all new rows to 2:

```
EXEC SQL
  INSERT INTO STOCK (GNO, GOODS, QOH, WHNO)
    SELECT GNO, GOODS, QOH, 2 FROM SUBSTOCK
    WHERE GOODS = 'MICROWAVE OVEN'
END-EXEC.
```

Using Dynamic SQL

To generate SQL statements during program execution and to execute the statements, use dynamic SQL.

Determining Search Conditions Dynamically

The previous examples use host variables to set search conditions at execution. The following explains a method of directly determining search conditions at execution.

The following figure is a COBOL program for determining search conditions at execution through dynamic cursor declaration. It is the query expression for cursor definition:

```
SELECT GNO, GOODS, QOH FROM STOCK
   WHERE GOODS = 'REFRIGERATOR' AND QOH < 10
```

The ACCEPT statement is used to read the query expression at execution.

```
       :
    EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01  PRODUCT-NUMBER     PIC S9(4) COMP-5.
01  PRODUCT-NAME       PIC X(20).
01  QUANTITY-IN-STOCK  PIC S9(9) COMP-5.
01  STMVAR             PIC X(254).                          (1)
01  SQLSTATE           PIC X(5).
    EXEC SQL END DECLARE SECTION END-EXEC.
PROCEDURE DIVISION.
    EXEC SQL WHENEVER NOT FOUND GO TO :P-END END-EXEC.
    EXEC SQL
      DECLARE CUR8 CURSOR FOR STMIDT                        (2)
    END-EXEC.
    ACCEPT STMVAR FROM CONSOLE.                             (3)
P-START.
    EXEC SQL CONNECT TO DEFAULT END-EXEC.
    EXEC SQL PREPARE STMIDT FROM :STMVAR END-EXEC.          (4)
    EXEC SQL OPEN CUR8 END-EXEC.                            (5)
 P-LOOP.
    EXEC SQL
      FETCH CUR8 INTO :PRODUCT-NUMBER,                      (6)
                      :PRODUCT-NAME,
                      :QUANTITY-IN-STOCK
    END-EXEC.
       :
    GO TO P-LOOP.
P-END.
    EXEC SQL CLOSE CUR8 END-EXEC.                           (7)
   EXEC SQL ROLLBACK WORK END-EXEC.
   EXEC SQL DISCONNECT DEFAULT END-EXEC.
   STOP RUN.
```

Figure 131. Specifying search conditions at execution

(1) The SQL statement variable STMVAR is referred to when the PREPARE statement in (4) is executed.

(2) The SQL statement identifier STMIDT is corresponding to the SQL statement variable STMVAR when the PREPARE statement in (4) is executed.

(3) Executes the ACCEPT statement to read the query expression, and sets the read data into the SQL statement variable STMVAR.

(4) Executes the PREPARE statement corresponding to the statement (dynamic SELECT statement in the example) and set in the SQL statement variable STMVAR of the SQL statement identifier STMIDT.

(5) Executes the dynamic OPEN statement to extract, from the STOCK table, rows showing a value less than 10 as the number of refrigerators in stock, then creates a table including the data of columns GNO (product number), GOODS (product name), and QOH (quantity in stock) that is retrieved from the extracted rows.

(6) Executes the dynamic FETCH statement to fetch data row by row from the table, and sets the values of each column into the corresponding host variable area.

(7) Executes the dynamic CLOSE statement to disable the specified cursor and the table corresponding to the cursor.

Determining SQL Statements Dynamically

Dynamic SQL not only determines search conditions in a query expression through cursor declaration, but dynamically determines the SQL statements to be executed there. The following explains methods of determining the SQL statements using the EXECUTE statement.

The following figure is a COBOL program used to dynamically execute SQL statements input with the ACCEPT statement. The example inputs the UPDATE statement at execution:

```
UPDATE STOCK SET QOH = 0 WHERE GOODS = 'TELEVISION'
```

```
         :
     EXEC SQL BEGIN DECLARE SECTION END-EXEC.
 01  STMVAR      PIC X(254).
 01  SQLSTATE    PIC X(5).
     EXEC SQL END DECLARE SECTION END-EXEC.
 PROCEDURE DIVISION.
     ACCEPT STMVAR FROM CONSOLE.                    (1)
     EXEC SQL CONNECT TO DEFAULT END-EXEC.
     EXEC SQL PREPARE STMIDT FROM :STMVAR END-EXEC. (2)
     EXEC SQL EXECUTE STMIDT END-EXEC.              (3)
        :
     EXEC SQL ROLLBACK WORK END-EXEC.
     EXEC SQL DISCONNECT DEFAULT END-EXEC.
     STOP RUN.
```

Figure 132. Dynamically determining SQL statements (1)

(1) Executes the ACCEPT statement to read the UPDATE statement, and sets the read data into the SQL statement variable STMVAR.

(2) Executes the PREPARE statement corresponding to the statement (the UPDATE statement in the example) set in the SQL statement variable STMVAR of the SQL statement identifier STMIDT.

(3) The EXECUTE statement executes the prepared statement that is associated with the specified SQL statement identifier.

If parameter specification is not required for dynamically determining SQL statements, the EXECUTE IMMEDIATE statement can be used instead of the EXECUTE statement.

The following COBOL program illustrates the use of the EXECUTE IMMEDIATE statement to perform the same processing as shown in previous figure. The UPDATE statement used is specified at execution.

```
              :
      EXEC SQL BEGIN DECLARE SECTION END-EXEC.
```

```
 01  STMVAR      PIC X(254).
  01  SQLSTATE    PIC X(5).
      EXEC SQL END DECLARE SECTION END-EXEC.
  PROCEDURE DIVISION.
      ACCEPT STMVAR FROM CONSOLE.                   (1)
      EXEC SQL CONNECT TO DEFAULT END-EXEC.
      EXEC SQL EXECUTE IMMEDIATE :STMVAR END-EXEC.   (2)
         :
      EXEC SQL ROLLBACK WORK END-EXEC.
      EXEC SQL DISCONNECT DEFAULT END-EXEC.
      STOP RUN.
```

Figure 133. Dynamically determining SQL statements (2)

(1) Executes the ACCEPT statement to read the UPDATE statement, then writes the read data to SQL statement variable STMVAR.

(2) The EXECUTE IMMEDIATE statement directly executes the SQL statements written to the SQL statement variable.

Specifying Dynamic Parameters

The following a COBOL program which illustrated the specification of dynamic parameters and retrieving data from the STOCK table. The example processes the SELECT statement at execution:

```
SELECT GNO, GOODS FROM STOCK WHERE WHNO = ?
```

```
        :
     EXEC SQL BEGIN DECLARE SECTION END-EXEC.
 01  PRODUCT-NUMBER      PIC S9(4) COMP-5.
 01  PRODUCT-NAME        PIC X(20).
 01  QUANTITY-IN-STOCK   PIC S9(9) COMP-5.
 01  WAREHOUSE-NUMBER    PIC S9(4) COMP-5.
 01  STMVAR              PIC X(254).
 01  SQLSTATE            PIC X(5).
 01  SQLMSG              PIC X(254).
     EXEC SQL END DECLARE SECTION END-EXEC.
 PROCEDURE DIVISION.
     EXEC SQL WHENEVER NOT FOUND GO TO :P-END END-EXEC.
     EXEC SQL
       DECLARE CUR11 CURSOR FOR STMIDT              (1)
     END-EXEC.
     ACCEPT STMVAR FROM CONSOLE.                    (2)
 P-START.
     EXEC SQL CONNECT TO DEFAULT END-EXEC.
     EXEC SQL PREPARE STMIDT FROM :STMVAR END-EXEC. (3)
     ACCEPT WAREHOUSE-NUMBER FROM CONSOLE.          (4)
     EXEC SQL OPEN CUR11 USING:WAREHOUSE-NUMBER
       END-EXEC.                                    (5)

  P-LOOP.
     EXEC SQL
       FETCH CUR11 INTO :PRODUCT-NUMBER,:
                        PRODUCT-NAME                (6)
     END-EXEC.
        :
     GO TO P-LOOP.
 P-END.
     EXEC SQL CLOSE CUR11 END-EXEC.                 (7)
     EXEC SQL ROLLBACK WORK END-EXEC.
     EXEC SQL DISCONNECT DEFAULT END-EXEC.
     STOP RUN.
```

Figure 134. Specifying dynamic parameters

(1) Defines CUR11 through dynamic cursor declaration.

(2) Executes the ACCEPT statement to read the dynamic SELECT statement.

(3) Executes the PREPARE statement to correspond the statement set in the SQL statement variable STMVAR to the SQL statement identifier STMIDT.

(4) Reads the search condition values corresponding to dynamic parameters.

(5) Executes the dynamic OPEN statement to extract rows matching the search conditions from the table. The values specified in the USING clause are referred to as the values of the dynamic parameters in the prepared statement. The values specified in the USING clause and the dynamic parameters in the prepared statement are associated in the order they appear.

(6) Executes the dynamic FETCH statement to fetch data row by row from the table, and then writes the values of each column to the host variables specified in the INTO clause.

(7) Executes the dynamic CLOSE statement to disable the specified cursor and the table corresponding to the cursor.

Using Variable Length Character Strings

This section explains how to use variable length character strings as the host variables of a COBOL program.

To operate variable length character string data, the length of the character string is needed. Host variables of a variable length character string type are defined as the following items:

- Signed binary item for storing the character string length information.

- Group item (an alphanumeric item or national item) for storing character strings.

The following figure illustrates retrieving an address from the COMPANY table. The host variable used as the search condition and the host variable used for storing variable length character string data.

```
       :
    EXEC SQL BEGIN DECLARE SECTION END.
01  COMPANY-NAME            PIC X(20).
01  TELEPHONE-NUMBER.                               }
 49 TELEPHONE-NUMBER-LENGTH  PIC S9(4) COMP-5.       }
 49 TELEPHONE-NUMBER-STRING  PIC X(20).              }       (1)
01  ADDRESS.                                        }
 49 ADDRESS-LENGTH           PIC S9(9) COMP-5.       }
 49 ADDRESS-STRING           PIC X(30).              }
01  SQLSTATE                 PIC X(5).
    EXEC SQL END DECLARE SECTION END-EXEC.
PROCEDURE DIVISION.
    DISPLAY "Retrieves company according to telephone
             number."
    DISPLAY "Input telephone number. ->" WITH NO ADVANCING.
    ACCEPT TELEPHONE-NUMBER-STRING FROM CONSOLE.            (2)
    INSPECT TELEPHONE-NUMBER-STRING                         (3)
      TALLYING TELEPHONE-NUMBER-LENGTH
      FOR CHARACTERS BEFORE SPACE.
    EXEC SQL CONNECT TO DEFAULT END-EXEC.
    EXEC SQL
      SELECT NAME,ADDRESS INTO :COMPANY-NAME,
                       :ADDRESS                             (4)
             FROM COMPANY
             WHERE PHONE = :TELEPHONE-NUMBER
    END-EXEC.
       :
    EXEC SQL ROLLBACK WORK END-EXEC.
    EXEC SQL DISCONNECT DEFAULT END-EXEC.
    STOP RUN.
```

Figure 135. Operating variable length character string data

(1) Declares variable length character strings as host variables.

(2) Executes the ACCEPT statement to read the value of the host variable used as the search condition, and sets the value as TELEPHONE-NUMBER-STRING.

(3) Sets the length of the read value as TELEPHONE-NUMBER-LENGTH.

(4) Executes the SELECT statement (single row) to retrieve the row meeting the search condition from column ADDRESS. The length and value of the retrieved character string are set into the host variable "ADDRESS".

If the data of a host variable used as a search condition is a character string type or a national character string type, define the length of the host variable to be equal to or shorter than the length of the character string.

Operating the Cursor with More than One Connection

This section explains how to operate a cursor with more than one connection. The following COBOL program is used for operating the cursor in the following conditions:

- SV1 and SV2 exist as servers
- Both servers contain a table under the same name and in the same format.

```
       :
    EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01  PRODUCT-NUMBER      PIC S9(4) COMP-5.
01  PRODUCT-NAME        PIC X(20).
01  QUANTITY-IN-STOCK   PIC S9(9) COMP-5.
01  WAREHOUSE-NUMBER    PIC S9(4) COMP-5.
01  SQLSTATE            PIC X(5).
    EXEC SQL END DECLARE SECTION END-EXEC.
01  NEXTFLAG            PIC X(4) VALUE SPACE.
PROCEDURE DIVISION.
    EXEC SQL DECLARE CUR9 CURSOR FOR                         (1)
               SELECT * FROM STOCK
    END-EXEC.
    EXEC SQL  WHENEVER NOT FOUND
              GO TO :P-NEXT END-EXEC.                        (2)
P-START.
    EXEC SQL
      CONNECT TO 'SV1' AS 'CNN1' USER 'summer/w43'           (3)
    END-EXEC.
    EXEC SQL
      CONNECT TO 'SV2' AS 'CNN2' USER 'tanaka/sky'           (4)
    END-EXEC.
P-CNN2-1.
    EXEC SQL OPEN CUR9 END-EXEC.                             (5)
    GO TO P-LOOP.
P-NEXT.
    IF NEXTFLAG = "NEXT" THEN
      GO TO P-CNN1-2
    END-IF.
```

Figure 136. Operating a cursor with more than one connection

```
 P-CNN1-1.
    EXEC SQL SET CONNECTION 'CNN1' END-EXEC.                 (7)
    EXEC SQL
      INSERT INTO STOCK                                      (8)
        VALUES(:PRODUCT-NUMBER, :PRODUCT-NAME,
               :QUANTITY-IN-STOCK, :WAREHOUSE-NUMBER)
    END-EXEC.
    EXEC SQL OPEN CUR9 END-EXEC.                             (9)
    MOVE "NEXT" TO NEXTFLAG.
    GO TO P-LOOP.
P-CNN1-2.
    EXEC SQL CLOSE CUR9 END-EXEC.                            (10)
    EXEC SQL ROLLBACK WORK END-EXEC.
    EXEC SQL DISCONNECT 'CNN1' END-EXEC.
P-CNN2-2.
    EXEC SQL SET CONNECTION 'CNN2' END-EXEC.                 (11)
    EXEC SQL CLOSE CUR9 END-EXEC.                            (12)
    EXEC SQL ROLLBACK WORK END-EXEC.
    EXEC SQL DISCONNECT CURRENT END-EXEC.
P-END.
    STOP RUN.
P-LOOP.
    EXEC SQL
      FETCH CUR9                                             (6)
        INTO :PRODUCT-NUMBER, :PRODUCT-NAME,
             :QUANTITY-IN-STOCK, :WAREHOUSE-NUMBER
    END-EXEC.
       :
   GO TO P-LOOP.
```

Figure 136. Operating a cursor with more than one connection (cont.)

(1) Defines the cursor for retrieving data from the table.

(2) Specifies branching to the procedure name P-NEXT if there is no row to be retrieved when the embedded SQL exception declaration is specified.

(3) The server SV1 is connected. This connection name is CNN1.

(4) The server SV2 is connected. This connection name is CNN2. CNN2 is the current connection.

(5) Executes the OPEN statement in the server with CNN2 to enable the cursor CUR9.

(6) Executes the FETCH statement to fetch data row by row from the table, then writes the values of each column to the corresponding host variable area.

(7) Select the current connection to CNN1.

(8) Executes the INSERT statement to insert a row into the STOCK table of the server with CNN1.

(9) Executes the OPEN statement in the server with CNN1 to enable the cursor CUR9. The cursor CUR9 is treated as another cursor different from the cursor that was opened in the server with CNN2.

(10) Executes the CLOSE statement in the server with CNN1 to disable the cursor CUR9.

(11) Select the current connection to CNN2. Note that the cursor CUR9 is still enabled in the server with CNN2.

(12) Executes the CLOSE statement in the server with CNN2 to disable the cursor CUR9.

Compiling the Program

Specify a database access method so that a COBOL program accessing a database using embedded SQL via an ODBC driver can be compiled with the COBOL85 compiler.

To enable case-sensitivity for host variable names, specify the NOALPHAL compiler option before compiling the program.

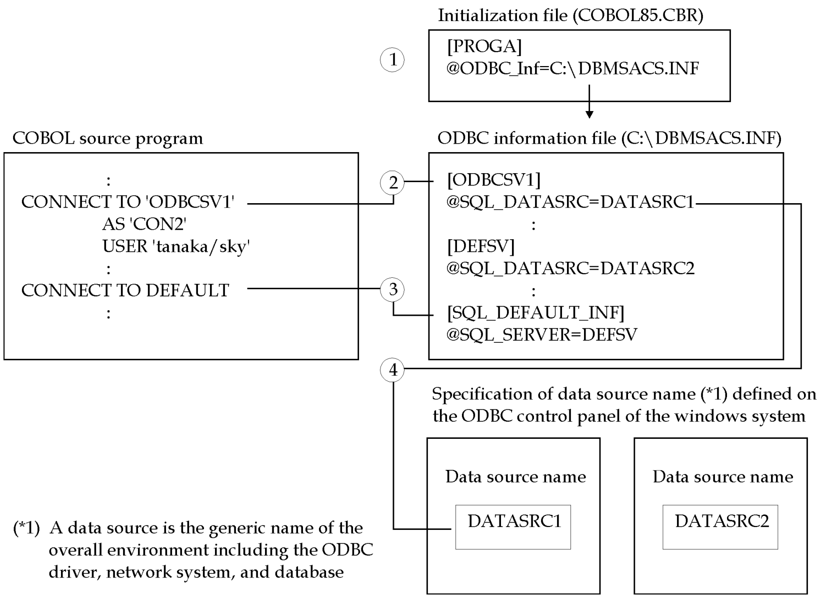
Note: Embedded SQL keywords cannot be used as user-defined names.

Executing the Program

This section explains how to construct the program execution environment, and how to use the ODBC Information Setup Tool.

Constructing the Program Execution Environment

An initialization file (COBOL85.CBR) where run-time environment information is specified, and the ODBC information file are required for program execution. Specify information in the files so that the files are related as follows:



(*1) A data source is the generic name of the overall environment including the ODBC driver, network system, and database

- Specify the ODBC information file name (C:\DBMSACS.INF).

- If a server name is specified in the CONNECT statement, specify the server name in the ODBC information file.
- If DEFAULT is specified in the CONNECT statement, specify a fixed character string indicating the definition of default connection information in the ODBC information file.
- To define the data source name of each server specified in the ODBC information file, specify the data source name defined in the Windows system ODBC control panel.

Setting Runtime Environment Information

To select the ODBC environment as client-server linkage software, specify the information shown below in the Runtime Environment Information and is reflected in the initialization file (COBOL85.CBR). The file can also be edited.

@ODBC_Inf (specification of the ODBC information file name)

```
@ODBC_Inf=C:\DBMSACS.INF
```

Specify the file name that the COBOL85 run-time system will refer when using ODBC. See “Creating an ODBC Information File” for more information.

Creating an ODBC Information File

An ODBC information file mainly contains information for connecting a client and server. Use the CONNECT statement to specify the connection.

Use the ODBC Information Setup Tool to create an ODBC information file. See “Using the ODBC Information Setup Tool” for more information.

The contents of an ODBC information file is classified into server information and default connection information.

Defining Server Information

The following table defines server information.

Table 57. How to define server information

Information Name [Server Name]	**Definition [Server Name]**	**Remarks [Connect Sentence server name or default connection information]**
@SQL_DATASCR	Data source name	Specify the data source name defined (added) on the Windows system ODBC control panel.
@SQL_ACCESS_MODE	Access mode READ_ONLY READ_WRITE	Specify an access mode for the data source. The default is READ_ONLY. To enable the read only mode, specify READ_ONLY. To enable the read-write mode, specify READ_WRITE. Operation in the specified mode can vary READ_WRITE based on the capabilities of the ODBC driver.
@SQL_COMMIT_MODE	Commit mode MANUAL AUTO	Specify a commit mode (operation of transaction) for the data source. The default is MANUAL. If MANUAL is specified, SQL operation is determined by specifying the COMMIT or ROLLBACK statement in the COBOL source program. If AUTO is specified, operation is determined each time an SQL statement is executed, regardless of the specification in the COBOL source program. If AUTO is specified, SQL statement processing is reflected into the database at execution of the SQL statement. As a result, the database cannot be restored to the original status with a ROLLBACK statement. Manual should be specified to avoid this problem.

@SQL_QUERY_TIMEOUT(32)	Timeout(seconds)	Specify the number of seconds to wait for a SQL statement to execute before returning to the application. The timeout range must be between 0 and 4294967285 (seconds). The default is 0 meaning that there is not timeout limit. Operation of the timeout can vary depending on the ODBC driver. If errors occur for timeout the specification should be removed.
SQL_CONCURRENCY(32)	Cursor concurrency mode READ_ONLY LOCK ROWVER VALUES	Specify the cursor concurrency. Concurrency is the ability of more than one user to use the same data at the same time. The default is READ_ONLY. Refer to the next table for more information. Operation for the cursor concurrency mode can vary depending on the ODBC driver.

NOTE: The ODBC Information Setup Tool cannot be used to set @SQL_QUERY_TIMEOUT and @SQL_CONCURRENCY. This information must be manually placed in the COBOL85.CBR file using a text editor.

Table 58. Operation for the cursor concurrency mode of @SQL_CONCURRENCY (32)

Mode	Operation		
READ_ONLY	FETCH statement is allowed. No position UPDATE and DELETE statements are allowed.		
LOCK	FETCH, position UPDATE and position DELETE statements are allowed.	NO CONCURRENCY	When processing FETCH, position UPDATE and position DELETE statements, the table is locked and operations from sever or other clients wait until operation is complete.
ROWVER		CONCURRENCY	FETCH statement using same data is allowed. When processing position UPDATE or DELETE statements, and a row is changed, the transaction containing the update or delete operation fails. If the row has not changed, the table is locked until the operation is complete. To determine if a row has changed, the rows and versions should be compared.
VALUE			Operation is the same as ROWVER. To determine if a row has changed, the data values should be compared.

Note: Operation for the cursor concurrency mode of @SQL_CONCURRENCY depends on the data source. (32)

Defining Default Connection Information

If DEFAULT is specified in the CONNECT statement, connection is established.

A user ID and password defined in default connection information are used for establishing connection if server name is specified in the CONNECT statement, but the user ID and password are omitted.

The following table defines default connection information.

Table 59. How to define default connection information

Information Name	Definition	Remarks
[SQL_DEFAULT_INF]	Fixed character string	Specify the fixed character string (section name) indicating the start of definition of default connection information.
@SQL_SERVER	Server name	Specify the name of a server where default connection is to be established. This server name is used for retrieving the definition information of each server and for establishing connection for the data source for each server. The definition information must be specified.
@SQL_USERID	User ID	Specify the user ID for operating the data source of the server.
@SQL_PASSWORD	Password	Specify the password for operating the data source of the server. Use the ODBC Information Setup Tool to encrypt the password.

Note: Specify both user IDs and passwords. Neither can be omitted.

Using the ODBC Information Setup Tool

The ODBC information setup tool allows you to set information in the ODBC information file in order to access remote databases via ODBC.

The following functions are provided in ODBC information setup tool.

- Select an ODBC information file
- Set server information
- Set default connection information

To use the ODBC information setup tool, follow the steps below (Refer to the online help for additional details).

1. Start the ODBC information setup tool.
   Use the SQLODBCS.EXE to start the ODBC information setup tool.
2. Select an ODBC information file.
   Specify an ODBC information file to set up connection information. If a file does not exist, create one with a text editor.
3. Setup the server information.
   Use the CONNECT statement or select default connection information to set the server name.
4. Setup the default connection information.
   Use the CONNECT statement with the DEFAULT specification to set default connection information (server name, user ID, and password).

Maximum Length of Information Specified in the ODBC Information File

The following table shows the maximum length of information specified in an ODBC information file.

Table 60. ODBC information file specifications

Information Type	**Maximum Length**	**Remarks**
User ID	32 bytes	The maximum length depends on the specification of the data source used for establishing the connection. Refer to the ODBC manual for more information about the ODBC driver environment.
Password	32 bytes	
Server name	32 bytes	-------------------
Data source name	32 bytes	-------------------

Use the ODBC Information Setup Tool to set up a password. The password must be encrypted. Do not edit the password with an editor.

Preparing Linkage Software and the Hardware Environment

Prepare the linkage software and hardware environment for a COBOL application program to access a server database through ODBC using the following procedures:

Setting Up the ODBC Environment

- Install ODBC in the Windows system. If the ODBC environment is installed in the Windows system for the first time, use the ODBC setup supported with the ODBC driver.
- ODBC is added to the Windows control panel. Install the ODBC driver, then create a data source. Start the ODBC

control panel (usually in a Windows control panel group), then set it up. The ODBC driver is installed and the data source is defined.

- Prepare the ODBC driver environment. An ODBC driver is designed to assume the environment where it operates. For information on the ODBC driver environment, refer to the manual and help information of the ODBC driver. The help information can be referenced by starting the ODBC control panel.

Confirming a Connection with the Data Source

A database usually supports programs for a client to operate a server database. Before executing a COBOL application program using ODBC, use the programs to check whether the client and server are connected normally.

Execute the COBOL application program after completing the setup of the ODBC control panel, with the runtime environment information which is placed in initialization file (COBOL85.CBR), and the ODBC information file.

Embedded SQL Keyword List

This section gives a keyword list for embedded SQL.

[A]
ABSOLUTE
ADA
ADD
ALL
ALLOCATE
ALTER
AND
ANY
ARE
AS
ASC
ASSERTION
AT
AUTHORIZATION
AVG
[B]
BEGIN
BETWEEN
BIND
BIT
BIT_LENGTH
BY
[C]
CASCADE
CASCADED
CASE
CAST
CATALOG
CHAR
CHAR_LENGTH
CHARACTER
CHARACTER_LENGTH
CHARACTER_SET_CATALOG
CHARACTER_SET_NAME
CHARACTER_SET_SCHEMA
CHECK
CLOSE
COALESCE
COBOL
COLLATE
COLLATION
COLLATION_CATALOG
COLLATION_NAME
COLLATION_SCHEMA
COLUMN
COMMIT
CONNECT
CONNECTION
CONSTRAINT
CONSTRAINTS
CONTINUE
CONVERT
CORRESPONDING COUNT
CREATE
CURRENT
CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP_ CURSOR
[D]
DATA
DATE
DATETIME_INTERVAL_CODE
DATETIME_INTERVAL_PRECISION
DAY
DEALLOCATE
DEC
DECIMAL
DECLARE
DEFAULT
DEFERRABLE
DEFERRED
DELETE
DESC
DESCRIBE
DESCRIPTOR
DIAGNOSTICS
DICTIONARY

DISCONNECT
DISPLACEMENT
DISTINCT
DOMAIN
DOUBLE
DROP
[E]
ELSE
END
END-EXEC
ESCAPE
EXCEPT
EXCEPTION
EXEC
EXECUTE
EXISTS
EXTERNAL
EXTRACT
[F]
FALSE
FETCH
FIRST
FLOAT
FOR
FOREIGN
FORTRAN
FOUND
FROM
FULL
[G]
GET
GLOBAL
GO
GOTO
GRANT
GROUP
[H]
HAVING
HOUR
[I]
IDENTITY
IGNORE
IMMEDIATE
IN
INCLUDE
INDEX
INDICATOR
INITIALLY
INNER
INPUT
INSENSITIVE
INSERT
INTEGER
INTERSECT
INTERVAL
INTO
IS
ISOLATION
[J]
JOIN
[K]
KEY
[L]
LANGUAGE
LAST
LEFT
LENGTH
LEVEL
LIKE
LIST
LOCAL
LOWER
[M]
MATCH
MAX
MIN
MINUTE
MODULE
MONTH
MUMPS
[N]
NAME
NAMES
NATIONAL
NCHAR
NEXT
NONE
NOT
NULL
NULLABLE
NULLIF
NUMERIC
[O]
OCTET_LENGTH
OF
OFF
ON
ONLY

OPEN
OPTION
OR
ORDER
OUTER
OUTPUT
OVERLAPS

[P]
PARTIAL
PASCAL
PLI
POSITION
PRECISION
PREPARE
PRESERVE
PREVIOUS
PRIMARY
PRIOR
PRIVILEGES
PROCEDURE
PUBLIC
[R]
RELATIVE
RESTRICT
REVOKE
RIGHT
ROLLBACK
ROWS
[S]
SCALE
SCHEMA
SCROLL
SECOND
SECTION
SELECT
SEQUENCE
SET
SIZE
SMALLINT
SOME
SQL
SQLCA
SQLCODE
SQLERROR
SQLSTATE
SQLWARNING
START
SUBSTRING
SUM
SYSTEM
[T]
TABLE
TEMPORARY
THEN
TIME
TIMESTAMP
TIMEZONE_HOUR
TIMEZONE_MINUTE
TO
TRANSACTION
TRANSLATE
TRANSLATION
TRUE
TYPE
[U]
UNION
UNIQUE
UNKNOWN
UPDATE
UPPER
USAGE
USER
USING
[V]
VALUE
VALUES
VARCHAR
VARIABLES
VARYING
VIEW
[W]
WHEN
WHENEVER
WHERE
WITH
WORK
[Y]
YEAR

Correspondence Between ODBC-Handled Data and COBOL85-Handled Data

COBOL85 handles the ODBC data corresponding definitions listed in the following table. To understand how an ODBC driver handles ODBC SQL data, refer to ODBC driver users guide and online help.

Data contents can only be assured when data correspondence as defined in the following table is used.

Table 61. Correspondence between ODBC and COBOL85 data handling

Arithmetic Data:

ODBC SQL Data Type		**COBOL85 Description**
Binary	SQL_SMALLINT (SMALLINT)	PIC S9(4) BINARY or PIC S9(4) COMP-5
	SQL_INTEGER (INTEGER)	PICS9(9) BINARY or PIC S9(9) COMP-5
Decimal	SQ:_DECIMAL (DECIMAL)	PIC S(9p) PACKED-DECIMAL or PIC S9(p)V9(q) PACKED-DECIMAL 1 =< p =<15, 1=< q, p+q =<18

	SQL_NUMERIC (NUMERIC)	{PIC S9(p) }
		{PIC S9(p)V9(q)}
		[SIGN IS
		{LEADING SEPARATE CHARACTER}
		{TRAILING }
		]
		1 =< p =<15, 1=< q, p+q =<18

Internal floating point	SQL_REAL (REAL)	COMP-1
	SQL_DOUBLE (FLOAT)	COMP-2

Character Data:

ODBC SQL Data Type		COBOL85 Description
Fixed length	SQL_CHAR (CHAR)	PIC X(n) 1=< n =< 254 PIC N(n) 1=< n =<127 (*1)
Variable	SQL_VARCHAR (VCHAR)	01 data-name-1. 49 data-name-2 PIC S9(m) COMP-5. 49 data-name-3 PIC X(n). m = 4 or 9 1 =< n =< 254 (*2) ——— 01 data-name-1 49 data-name-2 PIC S9(m) BINARY. 49 data-name-3 PIC X(n). or 01 data-name-1. (*1) 49 data-name-2 PIC S9(m) BINARY. 49 data-name-3 PIC N(n). m = 4 or 9 1 =< n =< 254 (*2)

*1: Depends on the database that supports national character data and the database-related products.
*2: Specify the number of characters for the variable length of character data.

SQLSTATE, SQLCODE, and SQLMSG

This section describes information which is shown in the notification area of the COBOL, ODBC driver manager, ODBC driver, or DBMS if an SQL statement is executed using ODBC.

The following table explains information posted as SQLSTATE, SQLCODE, and SQLMSG.

Table 62. SQLSTATE, SQLCODE, and SQLMSG information

Information	Value Posted	Explanation	Programmer Response
SQLSTATE	000000	Normal termination	________________________
	5 alphanumeric characters	Error or warning during execution of an SQL statement	CHECK SQLCODE and SQLMSG, and take action if error is detected. For SQLSTATE information, Refer to the ODBC SDK manual.
SQLCODE	0	Normal termination	________________________
	Negative integer other than 0	Error or warning during execution of an SQL statement	Check the cause of the error according to the ODBC environment (such as ODBC or DBMS) manual, and take corrective action on the error.
SQLMSG	Blank (no output)	Normal termination	________________________
	Message (character string)	The message explains an error or warning posted during execution of an SQL statement	Check the cause of the error according to the displayed message, and take corrective action on the error.

The following describes errors which might be detected by COBOL during execution of an embedded SQL statement. The information is posted as SQLSTATE, SQLCODE, and SQLMSG.

Table 63. SQLSTATE, SQLCODE, and SQLMSG error information

SQLSTATE	SQLCODE	SQLMSG	Programmer response
99999	-999999999	The connection has been established with the same connection name	A connection name must not be specified for more than one connection. Specify a unique name for each connection.
9999A	-999999990	The number of connections exceeds the maximum.	The number of connections exceeds the maximum specified in the COBOL system. Decrease the number of connections. The maximum number of connections can vary depending on the ODBC environment. Refer to the ODBC environment manual and take action.
9999B	-999999800	The specified connection does not exist.	SQL statements cannot be executed because the connection is not active. Check the SQL statement sequence in the program, and take corrective action on the error.
999SA	-999999700	The cursor is not opened.	The cursor is not ready. Check the sequence of SQL statements using the cursor, and take corrective action for the error.
999SB	-999999600	The prepared statement is not prepared.	Check the sequence of dynamic SQL statements, and take corrective action for the error.
?????	-999999992	Invalid process occurred.	A system error occurred.

Notes on Using the ODBC Driver

This section explains usage notes for ODBC drivers. Pay special attention to this section, since it covers important usage information.

Notes on SQL Statement Syntax

DATA DIVISION

The correspondence between ODBC data types and host variable data types is defined for COBOL85. See “Correspondence Between ODBC-Handled Data and COBOL85-Handled Data” for more information. Data contents are assured for only the defined data correspondence as specified.

Some data types may not be handled by COBOL as host variables depending on the ODBC driver specification.

PROCEDURE DIVISION

Embedded SQL statements that can be used and the statement specification method are different between ODBC drivers.

If embedded SQL statements are specified in the COBOL source program, refer to product manuals as well as the COBOL specification for:

- ODBC driver
- Software and hardware related to the ODBC driver
- Database management system

Use the COMMIT or ROLLBACK statements to terminate transactions. Otherwise, the operation may not be assured. Before using the DISCONNECT statement to terminate a connection, execute the COMMIT or ROLLBACK statement to terminate the transaction.

The SQL descriptor area cannot be used. Therefore, do not specify the following embedded SQL statements:

- INTO and USING clauses in which a descriptor name is specified
- SQL statements for the descriptor area: ALLOCATE DESCRIPTOR, DEALLOCATE DESCRIPTOR, GET DESCRIPTOR, SET DESCRIPTOR, and DESCRIBE statements

You cannot specify the DEALLOCATE PREPARE statement.

You cannot specify any SQL statement for data definition (DDL).

When using a dynamic SQL statement such as the PREPARE, EXECUTE, or EXECUTE IMMEDIATE statement, NOT FOUND specification in the WHENEVER clause is not available.

Details on the embedded SQL statement syntax comply with the database specification and the specification of the database related products.

Notes on Executing Embedded SQL Statements

Embedded SQL statements that can be used and the methods of specifying the statements are different between ODBC drivers.

If embedded SQL statements are specified in the COBOL source program, refer to product manuals as well as the COBOL specification for:

- ODBC driver

- Software and hardware related to ODBC driver
- Database management system

To quit transactions, use the COMMIT or ROLLBACK statements; otherwise, the operation may not be guaranteed. Before using the DISCONNECT statement to terminate a connection, execute the COMMIT or ROLLBACK statements to terminate the transaction.

If X'00' is stored as a value in character data, the storage and fetch results are not assured.

Details on the embedded SQL statement syntax comply with the database specification and the specification of the database-related products.

Quantitative Limits of Embedded SQL Statement at Execution

An embedded SQL statement can be up to 4,096 bytes in length. The maximum length may be shorter depending on the ODBC driver environment.

For example, the maximum length of an embedded SQL statement may be shorter than 4,096 bytes because of the maximum length of data transferred by the software product responsible for the network.

If the ODBC driver processes data in an embedded SQL statement from COBOL, the embedded SQL statement may be longer than specified in the COBOL source program.

An area for input and output processing between a client and server can be up to 8,192 bytes in length. The sum of the input variable length and output variable length must be within the allowable range. The area may be shorter than 8,192 bytes depending on the ODBC driver environment.

Chapter 16. Distributed Development Support Functions

COBOL85 uses the SIA COBOL syntax (defined in the FUJITSU COBOL Reference Manual). A program written according to the SIA specification can be developed under this system. The developed program can operate under various types of systems.

This chapter explains distributed development with the system of GS-series (M-series) application program as a host. **Note**: This chapter is only applicable if you are running M-series or GS-series.

Outline of Distributed Development

COBOL85 enables development of a program that operates with the Global server (hereafter referred to as GS or GS program). To operate a host program that uses a function specific to the Global server system, requirements for the operating environment (movement method) must be cleared. For information about specific functions, refer to Appendix I, "GS-series Function Comparison."

For clarification between the GS-series and the previous M-series systems, the following terms are defined:

GS-series

The successor to the M-series; it is the same as the M-series.

Global server

GS-series (M-series) as it is seen from the system.

GS program

A program that works with the GS-Series (M-series).

GS-specific function

A function that can only be used with the GS-series (M-series) system.

Scope of Distributed Development Functions

The following figure is a diagram of the scope of GS-series functions and COBOL85 functions.

Operation of the common functions shown in Figure 137 can be checked under this system. To what extent GS-series functions can be developed under this system differs depending on the functions.

Refer to Appendix I, “GS-series Function Comparison” for notes on handling the functions specific to GS-host and operating host programs under this system.

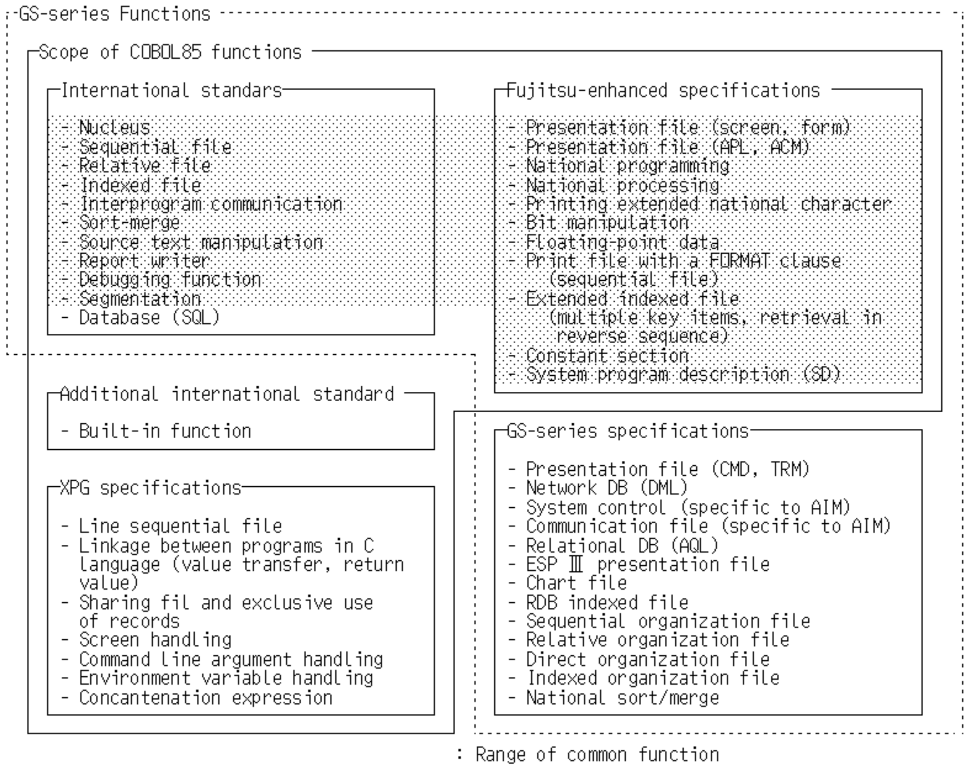

GS-series Functions
Scope of COBOL85 functions
International standars
- Nucleus
- Sequential file
- Relative file
- Indexed file
- Interprogram communication
- Sort-merge
- Source text manipulation
- Report writer
- Debugging function
- Segmentation
- Database (SQL)
Fujitsu-enhanced specifications
- Presentation file (screen, form)
- Presentation file (APL, ACM)
- National programming
- National processing
- Printing extended national character
- Bit manipulation
- Floating-point data
- Print file with a FORMAT clause (sequential file)
- Extended indexed file (multiple key items, retrieval in reverse sequence)
- Constant section
- System program description (SD)
Additional international standard
- Built-in function
XPG specifications
- Line sequential file
- Linkage between programs in C language (value transfer, return value)
- Sharing fil and exclusive use of records
- Screen handling
- Command line argument handling
- Environment variable handling
- Concantenation expression
GS-series specifications
- Presentation file (CMD, TRM)
- Network DB (DML)
- System control (specific to AIM)
- Communication file (specific to AIM)
- Relational DB (AQL)
- ESP Ⅲ presentation file
- Chart file
- RDB indexed file
- Sequential organization file
- Relative organization file
- Direct organization file
- Indexed organization file
- National sort/merge
: Range of common function

Figure 137. Scope of GS-series functions and COBOL85 functions

Distributed Development Support Functions

The distributed development support functions support the environment for operating the following two host-specific functions. The environment enables operation from compilation to module testing under this system.

- Presentation file function
- Network database function

For the presentation file module test function used with the presentation file function, see “Presentation File Module Test Function.”

Target Language Construct

This section shows the language construct of host-specific functions used as support functions.

Presentation File

Destination APL, CMD, TRM, or WST.

Network Database

Database manipulation language (DML)

- IF DB-EXCEPTION statement

System Control

- TRANSACTION management statement
- USE FOR DB-EXCEPTION statement
- USE FOR DEAD-LOCK statement

Usage

Required Resources

This section explains software and resources required for operation, from GS program compilation to the module test under this system.

Software

- COBOL85 compiler (required)
- COBOL85 run-time system (required)
- COBOL85 debugger (required)
- COBPRTST command program file (required for presentation file module test)

Resources

- Subschema definition file

  A subschema definition file is required to compile a GS program that uses network database functions. A subschema definition file contains the definition information of an area for communication (FCOM) with a GS-based advanced information management (AIM) system and an AIM expansion record area (UWA). The definition information is fetched from the AIM directory of the host machine. The extension CBL is given to the subschema name specified in the subschema name paragraph of the GS program.

To fetch the subschema definition file from the GS, use the subschema fetch tool (GETSSCH). See GETSSCH.TXT.

Compiling a GS Program

Compiler Options to be Specified

TEST: Whenever a debugger is used, specify this option.

AIMLIB (directory-name): To compile a GS program using a network database, specify this option. The directory name of the subschema definition file must be specified for directory-name.

GEN / NOGEN: To write the FCOM and UWA expansion information to the compilation list, specify GEN; otherwise, specify NOGEN.

Compilation Results

A W-level message indicating GS-specific functions is produced for the following specified in the COBOL source program:

- Network database manipulation language (DML)
- Destination APL / CMD / TRM / WST with literal specified in presentation file
- IF DB-EXCEPTION statement
- USE FOR DB-EXCEPTION statement
- USE FOR DEAD-LOCK statement
- TRANSACTION management statement

When the TRANSACTION management statement is executed, the syntax is analyzed, but no object is produced.

Linking a GS Program

Whenever a debugger is used, specify linkage option:

/DEBUG (Specify to use debugger.) and
/DEBUGTYPE{COFF|BOTH}(32), /CO(16)

Executing a GS Program

To execute the executable file of a GS program, use the COBOL85 debugger. If the file is executed without the debugger, a message is written indicating GS-specific functions cannot be executed, and the program is forcibly terminated.

To execute a GS program with the debugger:

1. Activate the debugger. Refer to the “Fujitsu COBOL Debugging Guide” for additional information. The COBOL85 Debugger window is displayed.

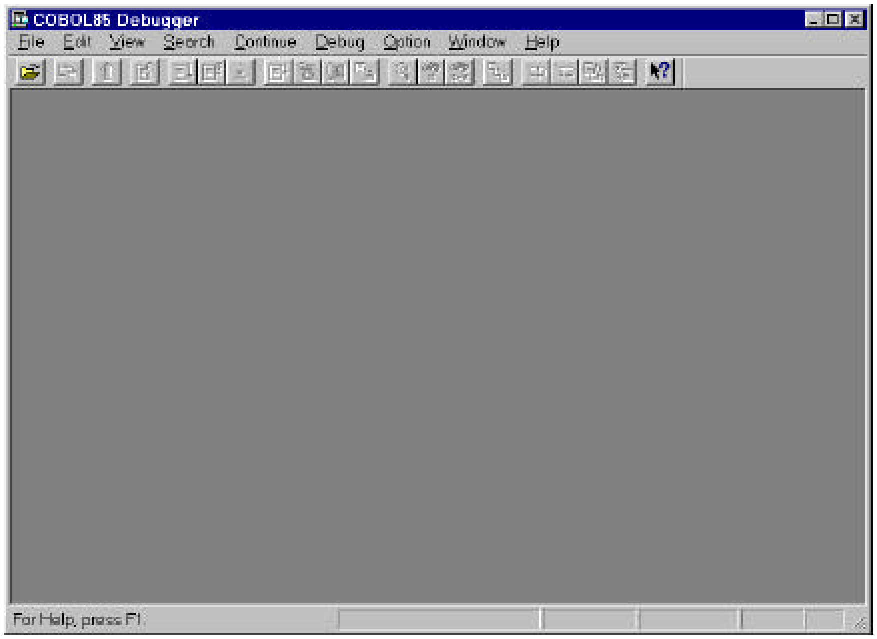
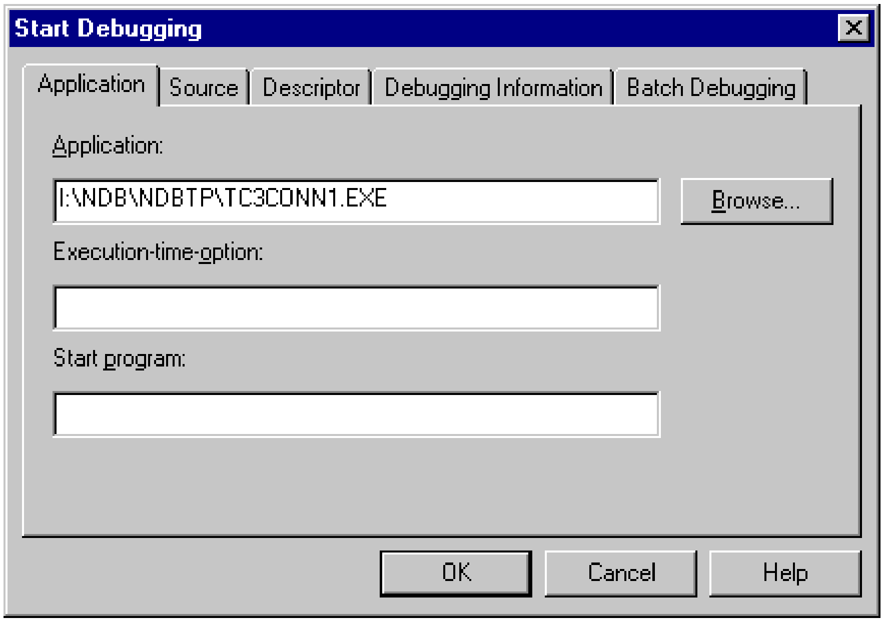


Figure 138. The COBOL85 Debugger window

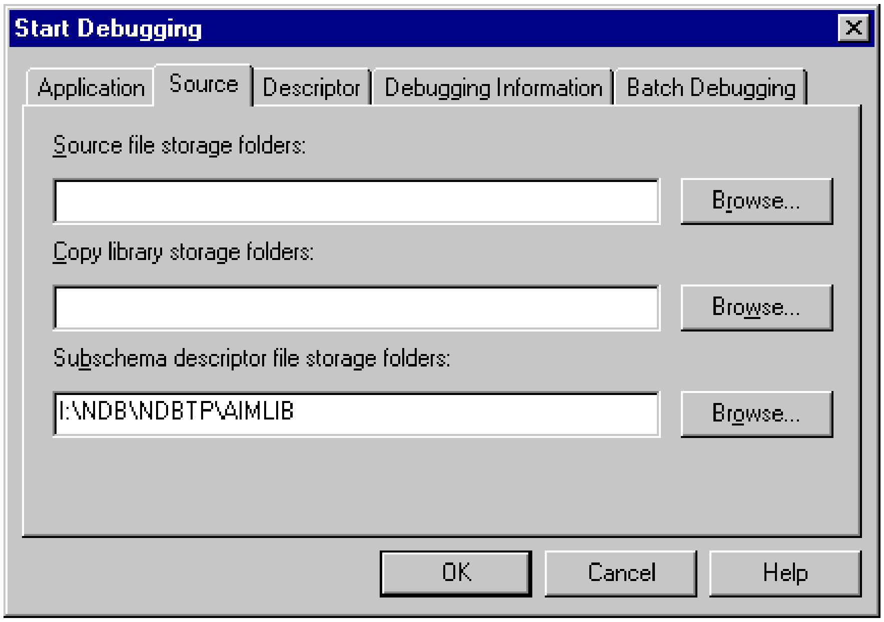
2. Select Start Debugging from the File menu to open the Start Debugging dialog box. Specify the required setup items.



Application:
I:\NDB\NDBTP\TC3CONN1.EXE
Execution-time-option:
Start program:

Figure 139. The Start Debugging dialog box, Application tab selected

Set the application name in the Application edit box.



Source file storage folders:

Copy library storage folders:

Subschema descriptor file storage folders: I:\NDB\NDBTP\AIMLIB

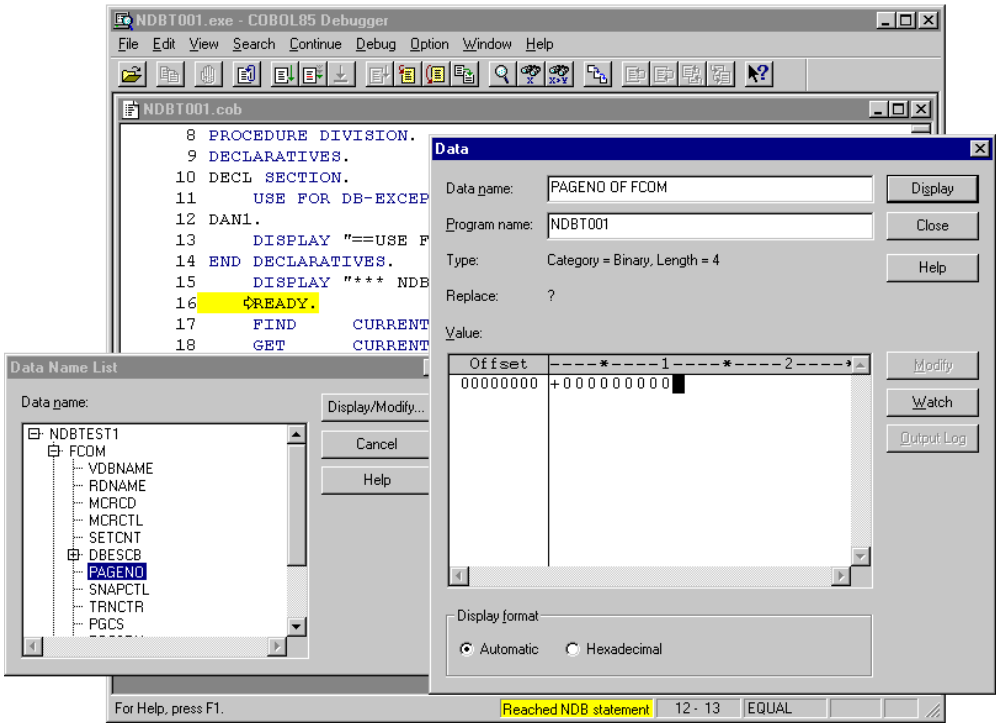
Figure 140. The Start Debugging dialog box, Source tab selected

Set the Subschema descriptor file storage folder (directory) in the Subschema descriptor file storage folders edit box.

3. Click on the OK button. The program is executed, then the debugger screen is displayed.

4. The debugger automatically suspends program execution when a statement using a GS-specific function is executed. It displays the Data Name List dialog box that shows the data names related to the network database function and presentation file function (selected when the program is executed). You can confirm the content of the data or open it by selecting the data name and clicking on the Display / Modify button. The Data dialog box opens and you can change the data if necessary.

5. Close the Data dialog box at the completion of data setup, then resume program execution.



```
PROCEDURE DIVISION.
DECLARATIVES.
DECL SECTION.
    USE FOR DB-EXCEP
DAN1.
    DISPLAY "==USE F
END DECLARATIVES.
    DISPLAY "*** NDB
    READY.
    FIND      CURRENT
    GET       CURRENT
```

Data name:

```
NDBTEST1
  FCOM
    VDBNAME
    RDNAME
    MCRCD
    MCRCTL
    SETCNT
    DBESCB
    PAGENO
    SNAPCTL
    TRNCTR
    PGCS
```

Data name: PAGENO OF FCOM
Program name: NDBT001
Type: Category = Binary, Length = 4
Replace: ?
Value:

```
Offset    ----*----1----*----2----
00000000  +000000000
```

Display format: [✓ Automatic] Hexadecimal

For Help, press F1. Reached NDB statement 12 - 13 EQUAL

Figure 141. Windows at completion of data setup

READY Statement

If the READY statement is not specified in a program and EXTERNAL is not specified in the subschema name paragraph, FCOM and UWA are defined as item names only, and no areas are allocated. If the program is executed in this condition (areas not allocated), operation is not guaranteed.

To execute the program normally, specify linkage section allocation in the debugger for the 01-level items of FCOM and UWA. The debugger allocates FCOM and UWA areas when linkage section allocation is executed. Consequently, the program can be executed in the same manner as when the READY statement is specified.

USE Statement Execution

The USE procedure is used for branching processing if an error occurs in a system for processing the presentation file or network database function called by the application. This system cannot branch processing to the USE procedure during execution because it has no system for processing the presentation file or network database function.

To execute the USE procedure, specify a breakpoint to suspend execution. Then, use the debugger move to execution start point function to move the execution start point to the first instruction of the USE procedure to resume execution.

Executing the Presentation File Function

To execute the presentation file function, assign a dummy file name (non-existent file name) to the file identifier specified in the ASSIGN clause of the file control entry.

AT END Specification

No actual input target file or database exists for statements with AT END specified. Consequently, AT END conditions are not produced and processing never branches to a statement at the AT END destination.

To execute a statement at the AT END destination, specify a breakpoint for the first-executed statement for NOT AT END, then use the debugger move to execution start point function to move the execution start point to the first-executed statement at the AT END destination to resume execution.

Data Name List Display

Fillers are not displayed.

GS-Specific Functions that Cannot be Used

The USE IN TRANSACTION statement cannot be used.

Executing the USE IN TRANSACTION statement results in a USE statement specification error and TRANSACTION statement specification error.

Presentation File Module Test Function

The presentation file module test function executes a module test during distributed development with the GS-series system. A module test is executed for an application program which performs interactive processing using the presentation file function on this system.

For more information about how to use this function, refer to the “OSIV AP/EF User’s Guide (Application Program).”

Operating Environment

This section shows the operating environment for executing the presentation file module test function.

The FORM-created screen descriptor and FORM RTS are required for using the presentation file module test function.

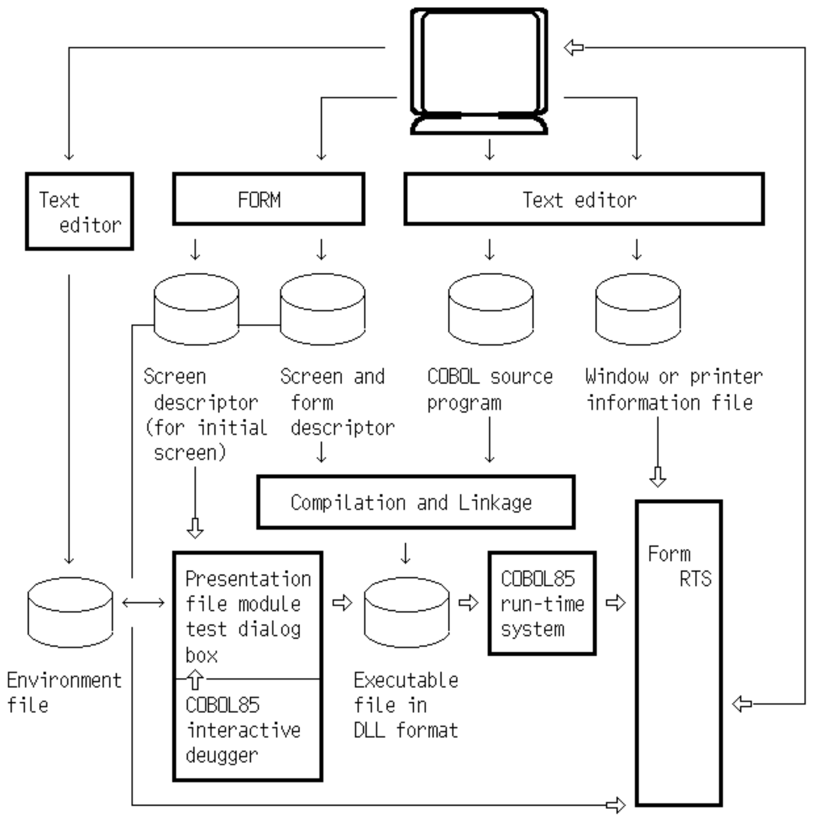

Text editor
FORM
Text editor
Screen descriptor (for initial screen)
Screen and form descriptor
COBOL source program
Window or printer information file
Compilation and Linkage
Presentation file module test dialog box
COBOL85 interactive deugger
Executable file in DLL format
COBOL85 run-time system
Form RTS
Environment file

Figure 142. The operating environment for executing the presentation file module test function

Creating Files Required for Using the Presentation File Module Test Function

The following files are required for using the presentation file module test function:

- A screen descriptor that defined the initial screen
- Other screen and form descriptors required

- Executable file in DLL format
- Window information file for the screen descriptor of the initial screen
- Other window and printer information files required
- Environment file

Before executing the presentation file module test function, take the following steps to create the required files:

1. Design an initial screen and creating screen and form descriptors
2. Create a COBOL source program
3. Compile and linking a program
4. Create window and printer information files
5. Create an environment file

Designing an Initial Screen and Creating Screen and Form Descriptors

The presentation file module test function performs all interactive processing through screens.

Before using the function, determine the correspondence between screens and programs and the hierarchical structure of the screens (screen transition).

To design the initial screen, use FORM to create a screen descriptor of the initial screen. Create more screen and form descriptors if they are required for application programs. For how to use FORM, refer to the “FORM V1.3 Manual.”

To split a screen, define a split procedure.

To define a split procedure, select split procedure definition during creation of the screen descriptor.

Split procedure definition defines operation according to input data (field input value or attention type) for any data item of the screen descriptor. For details on the split procedure, refer to the “FORM V1.3 Manual.”

Creating a COBOL Source Program

Use the editor to create a COBOL source program.

A knowledge of COBOL85 and presentation file interface is required for creating a COBOL source program. For details, refer to the “COBOL Reference Manual,” “FORM V1.3 Manual,” and the FORM RTS online help.

Compiling and Linking a Program

See “Distributed Development Support Functions” for the procedure for compiling and linking a program. The program to be used with the presentation file module test function must be created in DLL format.

Creating a Window and Printer Information Files

The presentation file module test function needs the window information file for the initial screen. If screens and forms are to be used with application programs, create the additional window information files and printer information files. For the file creation method, refer to the FORM RTS online help.

Creating an Environment File

An environment file contains the start options of the presentation file module test function. An environment file can be created by using the COBPRTST dialog box or the editor.

The following example represents a diagram of the coding format for creating an environment file with the editor.

```
1. [START]                      -> Screen-and-form-descriptors-name ‘START’
2. CBRNAME=D:\COBOL\COBOL85.CBR -> Initialization-file-name (full-path-name)
3. ICONNAME=D:\ICON\TEST.DLL    -> Icon-DLL-name (full-path-name)
4. ICONID=1                     -> Icon-resource-ID
5. DSPNAME=D:\SCREEN\DSPFILE    -> Window-information-file-name (full-path-name)
6. ENDKEY=F001                  -> Attention-value of the END-key
7. RETKEY=F002                  -> Attention-value of the RETURN-key
```

1. [section name**]**
   The section name identifies screens operating with the presentation file module test function. Specify a screen descriptor name of the initial screen.

2. CBRNAME (COBOL initialization file name)
   If a COBOL application program is to be started, specify the initialization file name (COBOL85.CBR) for COBOL execution.

3. ICONNAME (icon DLL name)
   Specify ICONNAME to change the icon of the presentation file module test function. The full path name of the icon DLL module file must be specified.

4. ICONID (icon resource ID)
   Specify a number as an ID for the icon resource stored in the icon DLL module. The default value is 1.

5. DSPNAME (window information file name)
   Specify the window information file name to change the environment information of the screen descriptor.

6. ENDKEY (END key value)
   To change the END key value, specify the item literal value specified in the attention definition of the screen descriptor. The default is F002 (F2 key).

7. RETKEY (RETURN key value)
   To change the RETURN key value, specify the item literal value specified in the attention definition of the screen descriptor. The default is F003 (F3 key).

Using the Presentation File Module Test Function

A program execution method is determined by the destination of the presentation file.

The following are presentation file destinations:

- PRT or DSP
- ACM, APL, TRM, CMD, or WST

With destination PRT or DSP, processing can be performed only by the presentation file module test function. A module test can be executed by directly starting the COBPRTST dialog box. Processing can also be done by using the COBOL85 interactive debugger.

With the other destinations, a function specific to GS-host is used. The COBOL85 interactive debugger is required to assist in processing. In this case, activate the COBOL85 interactive debugger then the COBPRTST dialog box. Execute a test while rewriting data items or record contents according to the test types.

The following sections explain procedures for processing using the presentation file module test function.

Note: When the display file unit test function is used, add the directory where FORM RTS is stored to the environmental variable PATH.

Activating the COBPRTST Dialog Box

If the presentation file is destined for PRT or DSP, activate the COBPRTST dialog box. This section explains the procedure for using the COBPRTST dialog box.

How to Activate the COBPRTST Dialog Box

Use the COBPRTST command to execute a presentation file module test. The COBPRTST dialog box can be activated as follows:

- Customize the COBPRTST command in the PROGRAMMING-STAFF Utility menu, and select it from the drop-down menu to start.

Windows 95 or Windows NT:

Execute COBPRTST.EXE.

Windows 3.1:

- Select COBPRTST.EXE in the File Manager directory window.
- Define COBPRTST.EXE as a program icon in Program Manager, then double-click on the icon.
- Select the Run command from the Program Manager menu or File Manager and enter the COBPRTST command in command form.

Specifying an Activation Name

Specify the screen descriptor name of the initial screen in the Invocation Name text box of the COBPRTST dialog box. See "Using the COBPRTST Dialog Box."

Specifying an Environment File Name

Specify an environment file name in the Environment File text box of the COBPRTST dialog box. To create a new environment file, specify the name of a non-existent file.

Changing Environment File Contents

To change environment file contents, specify the environment file name, then click on the Update button. The COBPRTST-Update dialog box is displayed. Change environment file contents. See "Using the COBPRTST-Update Dialog Box."

Starting a Presentation File Module Test

Click on the OK button at completion of input to the COBPRTST dialog box. The presentation file module test is executed.

Quitting a Presentation File Module Test

Click on the Cancel button.

Using the COBOL85 Interactive Debugger

If a presentation file is used with a function specific to GS-host and is destined for ACM, APL, TRM, CMD, or WST, the COBOL85 interactive debugger is required to assist in processing.

If the file is executed without the debugger, a message is outputted indicating host-specific functions cannot be executed under this system. The program is forcibly terminated.

This section explains how to start processing with the COBOL85 interactive debugger.

Activating the COBOL85 Interactive Debugger

Select WINSVD from the Tools menu of the P-STAFF window. Refer to the "Fujitsu COBOL Debugging Guide" for additional details.

Debugging

Open the Start Debugging dialog box. To use the presentation file module test function, specify the following:

```
Application            COBPRTST.EXE
Start program          SAMPLE
```

COBPRTST.EXE

The command to execute the presentation file module test.

SAMPLE

The program name of the DLL form which starts in the presentation file module test.

Activating the COBPRTST Dialog Box from the COBOL85 Debugger

Click on the OK button in the Start Debugging dialog box. Specify the required setup items when the COBPRTST dialog box is displayed. See “Activating the COBPRTST Dialog Box.” See also “Presentation File Module Test Dialog Boxes.”

Operation Using the COBOL85 Interactive Debugger

Specify a breakpoint with the debugger, then execute the program while manipulating data contents. See “Executing a GS Program.” If a run-time message is written during execution of the presentation file module test function, refer to the help information. (Select Help in the COBPRTST dialog box.)

COBPRTST Dialog Box

This section explains how to use the COBPRTST dialog box and COBPRTST-Update dialog box that are used with the presentation file module test function.

Using the COBPRTST Dialog Box

Use the COBPRTST dialog box to specify information for a presentation file module test. The following figure shows the COBPRTST dialog box.

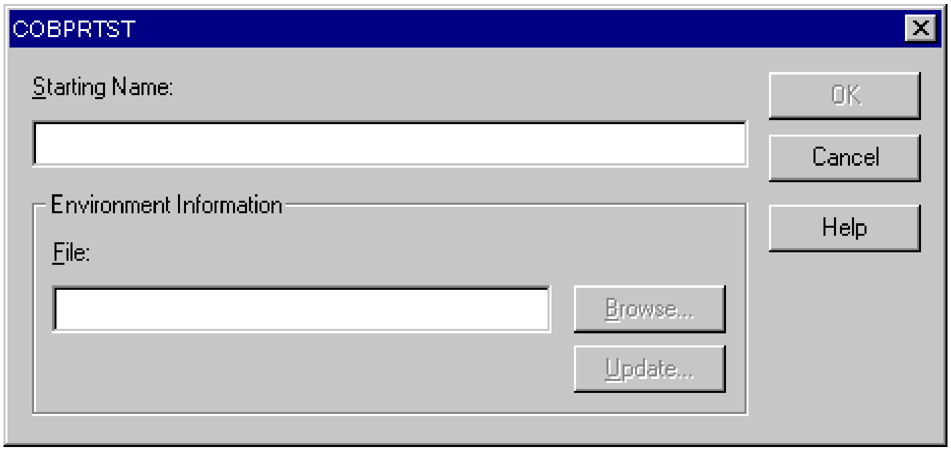


Figure 143. The COBPRTST dialog box

The COBPRTST dialog box contains the following elements:

Starting Name edit box

Specify the screen descriptor name of the initial screen.

File edit box

Specify the environment file name which enables the start option. An environment file is created by the file name specified here.

Browse button

Click to browse available files.

Update button

Click to update or change information on an environment file. When you select this button, the COBPRTST-Update dialog box is displayed.

OK button

Click to begin the presentation file module test.

Cancel button

Click to interrupt or quit the presentation file module test.

Help button

Click to access the presentation file module test function online help.

Using the COBPRTST-Update Dialog Box

To change environment file information, use the COBPRTST-Update dialog box. To display this dialog box, click on the Update button in the COBPRTST dialog box. The following figure shows the COBPRTST-Update dialog box.

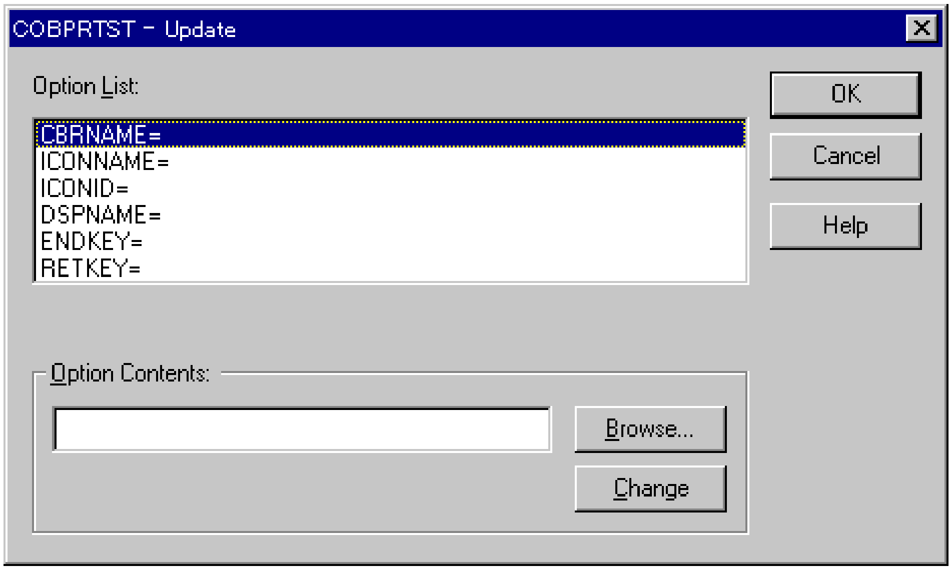


Figure 144. The COBPRTST-Update dialog box

The COBPRTST-Update dialog box contains the following elements:

Option List

Displays the options which can be specified. Change options by clicking on the name of the option.

Option Contents edit box

Displays the values set for a selected option.

Browse button

Click to browse available files.

Change button

Click to change data in the Option Contents edit box for the selected option.

OK button

Click to change the execution environment and return to the presentation file module test dialog.

Cancel button

Click to return to the presentation file module test dialog without changing the execution environment.

Help button

Click to access the presentation file module test function online help.

Appendix A. Compiler Options

This appendix explains the COBOL85 compiler options.

The first section lists compiler options. Review the list to verify the compiler option to be specified, then specify the option in the compiler option format as shown in the second section.

List of Compiler Options

This section lists compiler options. The numbers in the compiler option list correspond to the numbers of the compiler option specification formats in the second section.

Options that relate to compile time resources	Compiler option	Number
Specifies the directory of a subschema definition file. (16)	AIMLIB	1
Specifies the extension of a file containing the file descriptor.	FILEEXT	10
Specifies the directory name of a file containing the file descriptor.	FILELIB	11
Specifies the extension of the screen and form descriptor file.	FORMEXT	14
Specifies the directory name of a file containing the screen and form descriptor.	FORMLIB	15
Specifies the directory name of the libraries.	LIB	18

Options that relate to compile listings	Compiler option	Number
Determines whether to output a message indicating incompatibility between the old and new standards.	5	CONF
Displays library text.	COPY	6
Specifies the level of a diagnostic message.	FLAG	12
Displays FCOM and UWA.	GEN	16
Specifies the number of lines in a page of a compiler listings.	LINECOUNT	19
Specifies the number of characters on a line of a compiler listings.	LINESIZE	20
Determines whether to output the option information listing and compile unit statistical information listing.	MESSAGE	22
Specifies the sequence number area of a source program.	NUMBER	26
Determines whether to output compiler listings and specifies the output destination of each compiler listing.	PRINT	29
Determines whether to output a source program listing.	SOURCE	33

Options that relate to source program interpretation	Compiler option	Number
Determines how to treat the user-defined words of lowercase letters in the source program.	ALPHAL	2
Determines how to treat binary items.	BINARY	3
Determines how to treat currency symbols.	CURRENCY	7
Determines whether to display a message for the language elements in COBOL syntax.	FLAGSW	13
Specifies the ANSI COBOL standard.	LANGLVL	17
Specifies a character set of national user-defined words.	NCW	24
Determines how to treat the national spaces and ANK spaces.	NSPCOMP	25
Determines how to treat the QUOTE figurative constant.	QUOTE/APOST	30
Specifies the type of the reserved words.	RSV	31
Selection of sign adjustment for signed decimal items.	SDS	32

Specifies the type of program formats.	SRF	34
Specifies the collating sequence of alphanumeric characters.	STD1	37
Specifies a tab interval.	TAB	38
Compares the signed external decimal item with an alphanumeric item.	ZWB	42

Options that relate to object program generating	Compiler option	Number
Specifies a method of calling the specified subprogram with the CALL "literal".	DLOAD	8
Specifies a COBOL source program as a main program or subprogram.	MAIN	21
Specifies operation with the ACCEPT statement.	MODE	23
Determines whether to output the object programs.	OBJECT	27
Determines whether to create a globally optimized object programs.	OPTIMIZE	28

Options that relate to run-time control	Compiler option	Number
Specifies a method of processing data having the same key in the SORT statement.	EQUALS	9
Selection of column truncation.	TRUNC	41

Options that relate to run-time resources	Compiler option	Number
Specifies the input destination of data specified in the ACCEPT statement.	35	SSIN
Specifies the output destination of data specified in the DISPLAY statement.	SSOUT	36

Options that relate to debugging function	Compiler option	Number
Determines whether to use the CHECK function.	CHECK	4
Determines whether to use the interactive debugger.	TEST	39
Determines whether to use the TRACE function.	TRACE	40

Compiler Option Specification Formats

This section explains the compiler option specification formats.

The compiler options are listed in alphabetical order.

There are three methods of specifying a compiler option:

1. Using the Compiler Options dialog box
2. Using the -WC command option
3. Using the compiler directing statement (@OPTIONS) in the source program. (Priority: 3-2-1)

If a compiler option is specified in the compiler directing statement of the source program, some separately compiled programs may not be specified depending on the specification of the compiler option.

Symbols shown with compiler option specification formats are:

- WINCOB

  The compiler option can be specified on the WINCOB compiler option specification screen (option file).

- -WC

  The compiler option can be specified in the -WC command option.

- @

  The compiler option can be specified in the compiler directing statement.

- (F)

  The compiler option can be specified if the directory name is omitted.

Specify the absolute path name or relative path name for a directory name and file name. If a relative path name is specified, the directory name of the source file to be compiled is prefixed. When compiling from WINCOB, the source file to be compiled will be stored in the current directory.

The compiler options that specify a directory name cannot be specified in the -WC command option.

1 (16) WINCOB

AIMLIB(directory-name[;directory-name])

Specify the directory of the subschema definition file that is specified in the SUBSCHEMA-NAME paragraph.

If a subschema definition file exists in more than one directory, specify the directory names separating them by a semicolon. The directories are searched for in the order they were specified.

If the option is specified in duplicate, the directory search sequence is as follows:

1. Directories specified in the -A option
2. Directories specified in the compiler option AIMLIB

2 WINCOB,-WC,@

$$\left\{\begin{array}{l}\underline{\text{ALPHAL}}\\ \text{NOALPHAL}\end{array}\right\}$$

Specify ALPHAL to treat user-defined words with lowercase letters in the source program as uppercase letters. Otherwise, specify NOALPHAL.

ALPHAL treats character constants as follows:

Program name constant: Uppercase and lowercase letters are treated as uppercase letters.

Constants other than program name: Constants are treated as written.

The constant specified for a program called by a statement is included as written in the program name constant.

3 WINCOB,-WC,@

$$\text{BINARY(}\left\{\begin{array}{l}\underline{\text{WORD}}\text{[,}\left\{\begin{array}{l}\underline{\text{MLBON}}\\ \text{MLBOFF}\end{array}\right\}\text{]}\\ \text{BYTE}\end{array}\right\}\text{)}$$

BINARY(WORD) assigns the elementary item of binary data to an area length of 2, 4, or 8 words; BINARY(BYTE) assigns the elementary item of binary data to an area length of 1 to 8 bytes.

The area length is determined by the number of PIC digits. How to treat the high order end bit of an unsigned binary item can also be specified.

- BINARY(WORD,MLBON): The high order end bit is treated as a sign.
- BINARY(WORD,MLBOFF): The high order end bit is treated as a numeric value.

If BINARY(BYTE) is specified, the high order end bit is treated as a numeric value.

The following table shows the relationship between the number of declared digits and the area length.

Table 64. Length of Assigned Area from Number of PIC Digits

Number of PIC Digits		**Assigned Area Length**	
Signed	Unsigned	BINARY(BYTE)	BINARY(WORD)
1 - 2	1 - 2	1	2
3 - 4	3 - 4	2	2
5 - 6	5 - 7	3	4
7 - 9	8 - 9	4	4
10 - 11	10 - 12	5	8
12 - 14	13 - 14	6	8
15 - 16	15 - 16	7	8
17 - 18	17 - 18	8	8

4 WINCOB,-WC,@

$$\left\{ \begin{array}{l} \text{CHECK[(n)]} \\ \\ \underline{\text{NOCHECK}} \end{array} \right\}$$

To use the CHECK function, specify CHECK; otherwise, specify NOCHECK.

n indicates the number of times a message is displayed. Specify n with an integer 0 to 999,999. The default value is 1.

While the CHECK function is in use, program processing continues until a message is displayed up to n times. However, the program may fail to operate as expected if an error occurs (for example, area destruction). If 0 is specified for n, program processing continues regardless of the number of times a message is displayed.

If CHECK is specified, the above check processing is incorporated into the object program. Consequently, execution performance is decreased. When the debugging function terminates, specify NOCHECK, then recompile the program.

Refer to “Using the CHECK Function” in the “Fujitsu COBOL Debugging Guide” for more information.

5 WINCOB,-WC,@

$$\left\{\begin{array}{l} \text{CONF([}\left\{\begin{array}{c}\text{68}\\ \text{74}\\ \text{OBS}\end{array}\right\}\text{])} \\ \\ \underline{\text{NOCONF}} \end{array}\right\}$$

Specify CONF to indicate incompatibility between the old and new COBOL standards; otherwise, specify NOCONF. If CONF is specified, an incompatible item is indicated by I-level diagnostic messages.

- CONF(68): Indicate items that is interpreted differently between '68 ANSI COBOL and '85 ANSI COBOL.
- CONF(74): Indicate items that is interpreted differently between '74 ANSI COBOL and '85 ANSI COBOL.
- CONF(OBS): Indicates obsolete language elements and functions.

The compiler options CONF(68) and CONF(74) are effective only if the compiler option LANGLVL(85) is specified.

CONF is effective when a program created according to the existing standard is changed to '85 ANSI COBOL.

6 WINCOB,-WC,@

$$\left\{\begin{array}{l}\text{COPY}\\ \\ \underline{\text{NOCOPY}}\end{array}\right\}$$

To display library text incorporated by the COPY statement in the source program listing, specify COPY; otherwise, specify NOCOPY.

COPY is only effective when the compiler option SOURCE is specified.

7 WINCOB,-WC,@

$$\text{CURRENCY(}\left\{\begin{array}{l}\underline{\$}\\ \\ \text{currency-symbol}\end{array}\right\}\text{)}$$

Specify CURRENCY($) to use $ for a character used as a currency symbol; specify CURRENCY(currency-symbol) to use another symbol. If CURRENCY(currency-symbol) is specified, refer to the CURRENCY SIGN clause explained in the “COBOL85 Reference Manual” for the currency symbols that can be used.

8 WINCOB,-WC,@

$$\left\{\begin{array}{l}\text{DLOAD}\\ \\ \underline{\text{NODLOAD}}\end{array}\right\}$$

To dynamically call the subprogram specified with the CALL "literal", specify DLOAD; otherwise, specify NODLOAD.

Refer to “Linkage Types and Program Structure” in Chapter 4.

9 WINCOB,-WC,@

$$\left\{\begin{array}{l}\text{EQUALS}\\ \underline{\text{NOEQUALS}}\end{array}\right\}$$

If records having the same key are input by a SORT statement, specify EQUALS to ensure that the record sequential at output of the SORT statement is identical to the record sequential at inputted. Otherwise, specify NOEQUALS.

If NOEQUALS is specified, the sequence of records having the same key is not defined when the SORT statement outputs records.

If EQUALS is specified, special processing to ensure the input sequence is done sorting operation. Consequently, execution performance decreases.

10 WINCOB

FILEEXT(extension)

Specify the extension of a file containing the file descriptor. Any character string can be specified as the extension. If a file name has no extension, specify character string "None" as the extension. Do not specify more than one extension.

The following is the priority if the option is specified in duplicate:

1. Compiler option FILEEXT
2. FFD_SUFFIX environment variable (32)
3. Default value (FFD)

11 WINCOB

FILELIB(directory-name[;directory-name]...)

If the COPY statement with IN / OF XFDLIB specified is used to fetch a record descriptor from a file descriptor, specify the directory of the file containing the file descriptor.

If a file containing the file descriptor exists in more than one directory, specify the directory names separated by a semicolon. The directories are searched for in the order specified.

Specify directory-name + file-descriptor-name within 79 bytes. (16)

Specify directory-name + file-descriptor-name within 126 bytes. (32)

If the option is specified in duplicate, the directory search sequence is as follows:

1. Directories specified in the -f option
2. Directories specified in the compiler option FILELIB
3. Directories specified in the FILELIB environment variable (32)

12 WINCOB,-WC,@

$$\text{FLAG(}\left\{\begin{array}{c}\text{I}\\ \\ \text{E}\end{array}\right\}\qquad\text{W}\qquad\text{)}$$

Specify the diagnostic messages to be displayed.

- FLAG(I): Displays all diagnostic messages.
- FLAG(W): Displays diagnostic messages of only W-level or higher.
- FLAG(E): Displays diagnostic messages of only E-level or higher.

The diagnostic message specified in the compiler option CONF is displayed regardless of the FLAG specification.

13 WINCOB,-WC,@

$$\left\{\begin{array}{l}\text{FLAGSW}\;\;(\left\{\begin{array}{l}[\left\{\begin{array}{c}\text{STDM}\\ \text{STDI}\\ \text{STDH}\end{array}\right\}]\;[\text{,RPW}]\\ \text{SIA}\end{array}\right\})\\ \\ \underline{\text{NOFLAGSW}}\end{array}\right\}$$

To display a message indicating a language construct in COBOL syntax, specify FLAGSW; otherwise, specify NOFLAGSW.

The following are language constructs that can be indicated:

- FLAGSW(STDM): Language elements that are not in the minimum subset of the standard COBOL

- FLAGSW(STDI): Language elements that are not in the intermediate subset of the standard COBOL
- FLAGSW(STDH): Language elements that are not in the high subset of the standard COBOL
- FLAGSW(RPW): Language elements of the report writer function of the standard COBOL
- FLAGSW(SIA): Language elements that are not in the range of Fujitsu System Integrated Architecture (SIA)

Use FLAGSW(SIA) to create a program that will operate under another system.

14 WINCOB

FORMEXT(extent)

Specify the extent of the screen and form descriptor file. Any character string can be specified as the extent. If a file name has no extent, specify character string "None" as the extent. Do not specify more than one extent.

The following is the priority if the option is specified in duplicate:

1. Compiler option FORMEXT
2. SMED_SUFFIX environment variable (32)
3. Default value (PMD)

15 WINCOB

FORMLIB(directory-name[;directory-name]...)

If the COPY statement with IN / OF XMDLIB specified is used to fetch a record definition from the screen and form descriptor, specify the directory of the screen and form descriptor file.
If the screen and form descriptor files exist in more than one directory, specify the directory names separated by a semicolon. The directories are searched for in the order specified.

Specify directory-name + screen-and-form-descriptor-name within 79 bytes. (16)

Specify directory-name + screen-and-form-descriptor-name within 126 bytes. (32)

If the option is specified in duplicate, the directory search sequence is as follows:

1. Directories specified in the -m option
2. Directories specified in the compiler option FORMLIB
3. Directories specified in the FORMLIB environment variable (32)

16 WINCOB,-WC,@

$$\left\{\begin{array}{l}\text{GEN}\\ \underline{\text{NOGEN}}\end{array}\right\}$$

Specify GEN to display the communication area with AIM DBMS (FCOM) and the AIM expansion record area (UWA) in the source program listing. Otherwise, specify NOGEN.

17 WINCOB,-WC,@

$$\text{LANGLVL} \; (\left\{ \begin{array}{l} \underline{85} \\ 74 \\ 68 \end{array} \right\})$$

Specify a standard for an item where source program interpretation is different between the old and new COBOL standards.

- LANGLVL(85): '85 ANSI COBOL
- LANGLVL(74): '74 ANSI COBOL
- LANGLVL(68): '68 ANSI COBOL

18 WINCOB

LIB(directory-name[;directory-name]...)

If the source text manipulation function (COPY statement) is used, specify the directory of libraries. If libraries exist in more than one directory, specify the directory names separated by a semicolon. The directories are searched for in the order specified.

The following is the priority if the option is specified in duplicate:

1. -I option
2. Compiler option LIB

19 WINCOB,-WC,@

LINECOUNT(n)

Specify the number of lines in a page of a compiler listing. n can be specified with an integer of up to 3 digits. The default value is 60.

If a value 0 to 12 is specified, display without editing.

20 WINCOB,-WC,@

LINESIZE(n)

Specify the maximum number of characters (value resulting from conversion of alphanumeric characters displayed in the list) for a line in a compiler listing. Value can be specified with 80, or a 3-digit integer more than 120. The default value is 136.

The source program list, option information list, and diagnosis message list are output with a fixed number (120) of characters regardless of the maximum number of characters specified in LINESIZE.

The permitted maximum number of characters is 136. If a value greater than 136 is specified in LINESIZE, 136 is used.

21 WINCOB,-WC,@

$$\left\{\begin{array}{l}\text{MAIN}\\ \underline{\text{NOMAIN}}\end{array}\right\}$$

Specify MAIN to use a COBOL source program as a main program; specify NOMAIN to use a COBOL source program as a subprogram.

Specify MAIN for the COBOL source program to be used as a main program. If compiling the COBOL programs by Continuous Compilation mode or in a project, click on the Main Program button.

22 WINCOB,-WC,@

$$\left\{\begin{array}{l}\text{MESSAGE}\\ \\ \underline{\text{NOMESSAGE}}\end{array}\right\}$$

To output an option information listing and to compile unit statistical information listing, specify MESSAGE; otherwise, specify NOMESSAGE.

23 WINCOB,-WC,@

$$\text{MODE (}\left\{\begin{array}{c}\underline{\text{STD}}\\ \\ \text{CCVS}\end{array}\right\}\text{)}$$

If an ACCEPT statement where a numeric item is specified as the receiving item in the format of "ACCEPT unique-name[FROM mnemonic-name]" is executed, specify MODE(STD) to move a right-justified numeric data to a receiving item.

Specify MODE(CCVS) to move a left-justified character string to a receiving item:

If MODE(CCVS) is specified, only an external decimal item can be specified as a receiving item in the ACCEPT statement.

24 WINCOB,-WC,@

$$\text{NCW}\ (\left\{\begin{array}{c}\underline{\text{STD}}\\ \\ \text{SYS}\end{array}\right\})$$

Specify this option for a national character set that can be defined as user-defined words. Specify NCW(STD) to use the national character set as a national character set common to systems; specify NCW(SYS) to use the national character set as a national character set of the computer. The following are national character sets that can be used:

NCW(STD):

０，１，・・・，９
Ａ，Ｂ，・・・，Ｚ
ａ，ｂ，・・・，ｚ
ぁ，あ，ぃ，い，・・・ん
ァ，ア，ィ，イ，・・・ン，ヴ，ヵ，ヶ
ー（prolonged sound），―（hyphen），－（minus sign），etc.

NCW(SYS):

- Character set with STD specified
- Extended character
- JIS Level non-kanji (1)
- Extended non-kanji
- User-defined characters

(1) The following characters cannot be used.

、 。 ， ． ・ ： ； ？ ！ ゛
゜ ｀ ＾ ￣ ＿ ／ ＼ ｜ （ ）
［ ］ ｛ ｝ 「 」 ＋ ＝ ＜ ＞
￥ ＄ ￠ ￡ ％ ＃ ＆ ＊ ＠

25 WINCOB,-WC,@

NSPCOMP({ <u>NSP</u> | ASP })

Specify this option to determine how to treat a national space at comparison. Specify NSPCOMP(NSP) to treat the national space as a national space; specify NSPCOMP(ASP) to treat the national space as an ANK space.

A national space treated as an ANK space is assumed to be equivalent to a 2-byte ANK space.

NSPCOMP(ASP) is effective for the following comparisons:

- National character comparison with a national item as an operand
- Character comparison with a group item as an operand

NSPCOMP(ASP) is not effective for the following comparisons:

- Comparison between group items that do not include any national item
- Comparison between group items including an item whose attribute does not specify explicit or implicit display

A national space is not treated as an ANK space for the following comparison even if the NSPCOMP(ASP) option is specified:

> Character comparison and national character comparison done by using the INSPECT, STRING, or UNSTRING statement or by operating the indexed file key.

If NSPCOMP(ASP) is specified, an ANK space is treated as a national space under the class condition JAPANESE.

A national space is equivalent to a 2-byte ANK space in the host code system whereas it is not in the Windows code system. Thus, specify NSPCOMP(ASP) for a COBOL program which has been used with a host to operate under this system.

26 WINCOB,-WC,@

$$\left\{\begin{array}{l}\text{NUMBER}\\ \underline{\text{NONUMBER}}\end{array}\right\}$$

Specify this option to determine the value used as the line number in line information. The line information identifies each line of a source program within lists generated at compile time or run-time. Specify NUMBER to use the value of the sequence number area of the source program; specify NONUMBER to use the value generated by the compiler.

NUMBER: If the sequence number area includes a nonnumeric character or if the sequence number is not in ascending order, the line number is changed to the previously correct sequence number + 1.

NONUMBER: Line numbers are assigned in ascending order starting from 1. The increment between successive line numbers is 1.

If NUMBER is specified, a sequence number including identical values consecutively is not regarded as an error.

If NUMBER is specified, the error search function cannot be used.

27 WINCOB,-WC(D)

$$\left\{\begin{array}{l}\underline{\text{OBJECT}}[(\text{directory-name})] \\ \\ \text{NOOBJECT}\end{array}\right\}$$

Specify OBJECT to generate an object program; otherwise, specify NOOBJECT. If an object program is outputted, the file is usually created in the directory of the source program. To create the file in another directory, specify the directory name.

Refer to “Resources Necessary for Compilation” in Chapter 3.

28 WINCOB,-WC,@

$$\left\{\begin{array}{l}\text{OPTIMIZE} \\ \\ \underline{\text{NOOPTIMIZE}}\end{array}\right\}$$

Specify OPTIMIZE to generate a global-optimized object program; otherwise, specify NOOPTIMIZE.

If this option is specified with TEST, OPTIMIZE is displayed as the options established. The program, however, is compiled on the assumption that NOOPTIMIZE is specified. (No global optimization)

See Appendix C, “Global Optimization.”

29 WINCOB

PRINT[(directory-name)]

Specify this option to generate a compiler listing. If a compiler listing is output, the file is usually created in the directory of the source program. To create the file in another directory, specify the directory name.

30 WINCOB,-WC,@

$$\left\{\begin{array}{l}\underline{\text{QUOTE}}\\ \text{APOST}\end{array}\right\}$$

Specify QUOTE to use quotation marks as the values of the QUOTE and QUOTES figurative constants; specify APOST to use apostrophes as the values.

Either quotation marks or apostrophes can be used as delimiter of nonnumeric literal in source programs regardless of the specification of this option. The right and left side delimiters must be identical.

31 WINCOB,-WC,@

$$\text{RSV}\ (\left\{\begin{array}{l}\underline{\text{ALL}}\\ \text{V111}\\ \text{V112}\\ \text{V122}\\ \text{V125}\\ \text{VSR2}\\ \text{VSR3}\end{array}\right\}\ )$$

Specify the type of a reserved word. The following are the names of reserved-word sets:

- RSV(ALL): For COBOL85 V20 or later
- RSV(V111): For GS series COBOL85 V11L11
- RSV(V112): For GS series COBOL85 V11L20
- RSV(V122): For GS series COBOL85 V12L20
- RSV(V125): For COBOL85 V12L50
- RSV(VSR2): For VS COBOL II REL2.0
- RSV(VSR3): For VS COBOL II REL3.0

32 WINCOB,-WC,@

$$\left\{\begin{array}{l}\underline{\text{SDS}}\\ \text{NOSDS}\end{array}\right\}$$

Specify this option to determine how to move a signed internal decimal item to another signed internal decimal item. Specify SDS to move the sign of the sending item as is; specify NOSDS to move a converted sign. X'B' and X'D' are treated as minus signs.

The other values are treated as plus signs. A plus sign of the sending item is converted to X'C'; a minus sign of the sending item is converted to X'D'.

33 WINCOB,-WC,@

$$\left\{\begin{array}{l}\text{SOURCE}\\ \underline{\text{NOSOURCE}}\end{array}\right|$$

Specify SOURCE to output a source program listing; otherwise, specify NOSOURCE. If SOURCE is specified, the source program listing is output to the directory specified in the compiler option PRINT or the -dp option.

A source program listing is generated only if this option is specified with the compiler option PRINT or the -dp option.

34 WINCOB,-WC

$$\text{SRF(}\left\{\begin{array}{lll}\text{FIX} & & \text{FIX}\\ \text{FREE} & [, & \text{FREE} \quad]\\ \underline{\text{VAR}} & & \text{VAR}\end{array}\right\}\text{)}$$

Specify the reference formats of a COBOL source program and library. Specify FIX for the fixed-length format; specify FREE for the free format, specify VAR for the variable-length format.

Specify the reference format of COBOL source program, then specify that of the library. If the two reference formats are the same, there is no need to specify the library program format.

35 WINCOB,-WC,@

$$\text{SSIN (}\left\{\begin{array}{l}\text{run-time environment information name}\\ \\ \underline{\text{SYSIN}}\end{array}\right\}\text{)}$$

Specifies the data input destination of the ACCEPT statement for ACCEPT/DISPLAY function.

- SSIN(run-time environment information name): A file is used as the data input destination. At run-time, specify the path name of the run-time environment information name.
- SSIN(SYSIN): The console window is used as the data input destination.

The run-time environment information name can be specified with up to eight uppercase letters and numeric characters beginning with an uppercase letter (A to Z).

The run-time environment information name must be unique. The name must be different from a run-time environment information name (file-identifier) used with another file.

Refer to “ACCEPT / DISPLAY Function” in Chapter 11.

36 WINCOB,-WC,@

$$\text{SSOUT (} \left\{ \begin{array}{l} \text{run-time environment information name} \\ \\ \underline{\text{SYSOUT}} \end{array} \right\} \text{)}$$

Specifies the data output destination of the DISPLAY statement for ACCEPT / DISPLAY function.

- SSOUT(run-time environment information name): A file is used as the data output destination. At run-time, specify the path name of the run-time environment information name.
- SSOUT(SYSOUT): The console window is used as the data output destination.

The run-time environment information name can be specified with up to 8 uppercase letters and numeric characters beginning with an uppercase letter (A to Z).

The run-time environment information name must be unique. The name must be different from a run-time environment information name (file-identifier) used with another file.

Refer to “ACCEPT / DISPLAY Function” in Chapter 11.

37 WINCOB,-WC,@

$$\text{STD1 (}\begin{Bmatrix}\text{ASCII}\\ \text{JIS1}\\ \underline{\text{JIS2}}\end{Bmatrix}\text{)}$$

To determine how to treat an alphanumeric code (standard code of a 1 byte characters) specified for EBCDIC in the ALPHABET clause, specify this option.

Specify ASCII to treat the alphanumeric code as an ASCII code; specify JIS1 to treat it as a JIS 8-bit code; specify JIS2 to treat it as a JIS 7-bit Roman character code.

EBCDIC is specified in the ALPHABET clause, the character code system used depends on the specification of this option.

- STD1(ASCII): EBCDIC (ASCII)
- STD1(JIS1) : EBCDIC (kana)
- STD1(JIS2) : EBCDIC (lowercase letters)

38 WINCOB,-WC

$$\text{TAB (}\begin{Bmatrix}\underline{8}\\ \\ 4\end{Bmatrix}\text{)}$$

Specifies whether tabs are to be set in units of 4 columns (TAB(4)) or 8 columns (TAB(8)).

39 WINCOB,-WC(D),@(D)

$$\left\{\begin{array}{l}\text{TEST[(directory-name)]}\\ \\ \underline{\text{NOTEST}}\end{array}\right\}$$

To use a debugger, specify TEST; otherwise, specify NOTEST. If TEST is specified, the debugging information file used with the debugger is created in the directory of the source program. To create the file in another directory, specify the directory name.

If this option is specified with OPTIMIZE, OPTIMIZE is displayed as the options established. The program, however, is compiled on the assumption that NOOPTIMIZE is specified (no global optimization).

Refer to “Resources Necessary for Compilation” in Chapter 3, and the “Fujitsu COBOL Debugging Guide”.

40 WINCOB,-WC,@

$$\left\{\begin{array}{l}\text{TRACE[(n)]}\\ \\ \underline{\text{NOTRACE}}\end{array}\right\}$$

To use the TRACE function, specify TRACE; otherwise, specify NOTRACE.

n indicates the number of the trace information items to be output. Specify n with an integer 1 to 999,999. The default value is 200.

If TRACE is specified, processing for displaying trace information is incorporated into the object program. Consequently, execution performance decreases. When the debugging function terminates, specify NOTRACE, then recompile the program.

Refer to “Using the TRACE Function” in the “Fujitsu COBOL Debugging Guide”.

41 WINCOB,-WC,@

$$\left\{\begin{array}{l}\text{TRUNC}\\ \\ \underline{\text{NOTRUNC}}\end{array}\right\}$$

Specify a method of truncating high-order digits when a number is moved with a binary item as a receiving item.

- TRUNC: The high-order digits of the result value are truncated according to the specification of the PICTURE clause of the receiving item. The result value after truncation is stored in the receiving item.

  If this option is specified with the compiler option OPTIMIZE, the high-order digits of a variable moved from an external decimal item or internal decimal item through optimization are also truncated. Digits are truncated as explained above only if the number of digits in the integer part of the sending item is greater than that of the receiving item.

- NOTRUNC: The execution speed of the object program is of top priority. If the execution is faster without truncation than with truncation, digits are not truncated.

For example, specification in the PICTURE clause:

- Moving S999VP (3 digits in the integer part) to S99V99 (2 digits in the integer part): Truncation
- Moving SPV999 (0 digit in the integer part) to S99V99 (2 digits in the integer part): No truncation

If TRUNC is specified, the value to be set into the integer area of the sending item depends on the hardware.

If NOTRUNC is specified, the program must be designed so that a digit count exceeding the digit count specified in the PICTURE clause is not stored in the receiving item even when no digit is truncated.

If NOTRUNC is specified, whether to truncate digits depends on the compiler. Thus, a program with no truncation is used by specifying NOTRUNC may be incompatible with another system.

42 WINCOB,-WC,@

$$\left\{\begin{array}{l}\underline{\text{ZWB}}\\ \\ \text{NOZWB}\end{array}\right\}$$

Specify this option to determine how to treat a sign part when a signed external decimal item is compared with an alphanumeric field.

Specify ZWB for comparison ignoring the sign part of the external decimal item; specify NOZWB for comparison including the sign part.

Alphanumeric characters include alphanumeric data items, alphabetic data items, alphanumeric edited data items, numeric edited items, nonnumeric literals, and figurative constants other than ZERO.

For example:

- 77 ED PIC S9(3) VALUE +123.
- 77 AN PIC X(3) VALUE "123".

The conditional expression ED = AN is defined as shown below in this example:

- With ZWB specified: True
- With NOZWB specified: False

Appendix B. I-O Status List

This appendix lists the meanings of the values (status values) set when input-output statements are executed. The values are set in a data name specified in the FILE STATUS clause of the file control entry when input-output statements are executed.

Table 65. I-O Status List

Classification	**I-O Status Value**	**Detailed Information**	**Meaning**
Successful	00	--	Input-output statements were executed successfully.
	02	--	The status is one of the following: - The value of the reference key of the record read by executing the READ statement is the same as that of the next record. - The record having the same record key value as the record written by executing the WRITE or REWRITE statement already exists in a file. This is not an error, however, because a record key value may be duplicated.

Table 65. I-O Status List (cont.)

Classification	I-O Status Value	Detailed Information	Meaning
Successful	04	--	The length of an input record is greater than the maximum record.
		FORM RTS	One of the following status occurred in the presentation file. - An input error occurred in required full-field input items or required input items. - The data error occurred. (A national language data error, an ANK data error, a numeric item configuration data error, a numeric item sign error, a numeric item decimal point error, a redundancy check error) - Data entered in the data item is invalid.
		ACM	The input message is longer than the specified maximum record.
	05	--	One of the following status occurred in the file where the OPTIONAL clause is specified. - The OPEN statement in INPUT, I-O, or EXTEND mode was executed for a file, but the file has yet to be created. - The OPEN statement in INPUT mode was executed for a nonexistent file. In this case, the file is not created and at end condition occurs (input-output status value = 10) during execution of the first READ statement. - The OPEN statement in I-O or EXTEND mode was executed for a nonexistent file. In this case, the file is created.
	07	--	Input-output statements were successfully executed, however, the file referenced by one of the following methods exists in a non-reel or unit medium. - OPEN statement or CLOSE statement with NO REWIND specified - CLOSE statement with REEL/UNIT specified
	0A	ACM	There was no message to be input to the specified logical destination.

Table 65. I-O Status List (cont.)

Classification	I-O Status Value	Detailed Information	Meaning
Successful	0B	ACM	The number of output messages exceeds the maximum number of messages stored at the specified logical destination.
AT END condition	10	--	AT END condition occurred in a sequentially accessed READ statement. - File end reached. - The first READ statement was executed for a non-existent optional input file. - That is, a file with the OPTIONAL clause specified was opened in INPUT mode but the file was not created.
	14		AT END condition occurred in a sequentially accessed READ statement. - The valid digits of a read relative record number are greater than the size of the relative key item of the file.
Invalid key condition	21	--	Record key order is invalid. One of the following status occurred. - During sequential access, the prime record key value was changed between the READ statement and subsequent REWRITE statement. - During random access or dynamic access of a file with DUPLICATES specified as the prime key, the prime record key value was changed between the READ statement and subsequent REWRITE or DELETE statement. - During sequential access, prime record key values are not in ascending order at WRITE statement execution.
	22	--	During WRITE or REWRITE statement execution, the value of prime record key, or alternate record key to be written already exist in a file. However, DUPLICATES is specified in the prime record key or alternate record key.

Table 65. I-O Status List (cont.)

Classification	I-O Status Value	Detailed Information	Meaning
Invalid Key Condition	23	--	The record was not found. - During START statement or random access READ, REWRITE, or DELETE statement execution, the record having the specified key value does not exist in the file. - Relative record number 0 was specified for a relative file.
	24	--	One of the following status occurred. - Area shortage occurred during WRITE statement execution. - During WRITE statement execution, the specified key is out of the key range. - After overflow writing, an attempt was made to execute the WRITE statement again.
Permanent error condition	30	--	A physical error occurred.
		ACM	ACM detected error.
	34	--	Area shortage occurred during WRITE statement execution.
	35		The OPEN statement in INPUT, I-O, or EXTEND mode was executed for a file without the OPTIONAL clause specification. The file has yet to be created, however.
	37	--	The specified function is not supported.
	38		The OPEN statement was executed for a file for which CLOSE LOCK was executed before.
Logical error condition	39	--	During OPEN statement execution, a file whose attribute conflicts with the one specified in the program was assigned.
	41	--	The OPEN statement was executed for an opened file.
	42	--	The CLOSE statement was executed for an unopened file.

Table 65. I-O Status List (cont.)

Classification	I-O Status Value	Detailed Information	Meaning
Logical Error Condition	43	--	During sequential access or DELETE or REWRITE statement execution for a file with DUPLICATES specified as the prime key, the preceding input-output statement was not a successful READ statement.
	44	--	One of the following status occurred. - The record length during WRITE or REWRITE statement execution is greater than the maximum record length specified in the program. Or, an invalid numeric was specified as the record length. - During REWRITE statement execution, the record length is not equal to that of the record to be rewritten.
	46	--	During sequential call READ statement execution, the file position indicator is undefined due to one of the following reasons. - The preceding START statement is unsuccessful. - The preceding READ statement is unsuccessful (including at end condition).
	47	--	The READ or START statement was executed for a file not opened in INPUT or I-O mode.
	48 49	-- --	The WRITE statement was executed for a file not opened in OUTPUT, EXTEND (sequential, relative, or indexed), or I-O (relative or indexed) mode. The REWRITE or DELETE statement was executed for a file not opened in I-O mode.

Table 65. I-O Status List (cont.)

Classification	I-O Status Value	Detailed Information	Meaning
Other errors	90	--	These errors are other than those previously classified. The following conditions are assumed. 1) File information is incomplete or invalid. 2) During OPEN or CLOSE statement execution, an error occurred in a OPEN or CLOSE function. 3) An attempt was made to execute an input-output statement for a file where the CLOSE statement was unsuccessful due to input-output status value 90. 4) Resources such as main storage are unavailable. 5) An attempt was made to execute the OPEN statement for a file not closed correctly. 6) An attempt was made to write records after an error occurred due to overflow writing. 7) An attempt was made to write records after a no-space condition occurred. 8) An invalid character is contained in a text file record. 9) Some characters cannot be interpreted from a native code to the code set. 10) The lock table is full. 11) An error other than the above occurred. This is the only information about input-output operations where the error occurred. 12) A system error occurred.
		FORM RTS	FORM RTS detected an error.
		ACM	ACM detected an error.
	91	--	- No file is assigned. - At OPEN statement execution, a file identification does not correspond to a physical file name.
	92	--	An exclusive error occurred. (RDM file)
	93	--	An exclusive error occurred. (File lock)

Table 65. I-O Status List (cont.)

Classification	I-O Status Value	Detailed Information	Meaning
Other Errors	99	--	An exclusive error occurred. (Record lock)
		ACM	A system error occurred.
	9B	ACM	A destination error occurred.
	9E	FORM RTS	The execution waiting by the READ statement was compulsorily released.
	9F	ACM	- A monitoring time error occurred. - A destination error stopped processing.
	9G	ACM	The number of messages at the logical destination exceeds the specified maximum.
	9H	ACM	A file area or storage became insufficient at the logical destination.

'FORM RTS' in the Detail code column means a FORM RTS error code. For details, refer to the FORM RTS online help.

Appendix C. Global Optimization

This appendix explains the global optimization performed by the COBOL85 compiler.

Optimization

Global optimization divides the procedure division into sets of statements (called basic blocks), each consisting of one entry, one exit, and statements executable in physical sequence between the entry and the exit. Control flow and data use status are then analyzed, mainly in connection with loops (program parts to execute repeatedly).

Specifically, the following operations occur during global optimization:

- Removing common expressions
- Shifting an invariant
- Optimizing induction variables
- Optimizing PERFORM statements
- Integrating adjacent moves
- Eliminating unnecessary substitutions

Removing a Common Expression

For arithmetic or conversion processing, if possible, the previous results are reserved and used without additional execution of arithmetic or conversion processing.

Example 1:

```
77            I   PIC S99 BINARY.
77            J   PIC S99 BINARY.
77            K   PIC S99 BINARY.
01            REC-1
              02  A0  OCCURS 25 TIMES.
                  03 A       PIC XX OCCURS 10 TIMES.
              02  B0  OCCURS 35 TIMES.
                  03 B       PIC XX OCCURS 10 TIMES.
                  :
              MOVE SPACE TO A   (I,J).   ...     (1)
                  :
              MOVE SPACE TO B   (I,K).   ...     (2)
```

If I does not change in value between (1) and (2) in Example 1, the address calculation formula for A(I,J) "A - 22 + I * 20 + J * 2" and the address calculation formula for B(I,K) "B - 22 + I * 20 + K * 2" will have "I * 20" in common.

Therefore, COBOL optimizes the latter to use the result of the former. A is assumed to indicate the address A(1,1) here.

Example 2:

```
77           Z1  PIC S9(9)  DISPLAY.
77           Z2  PIC S9(9)  DISPLAY.
77           B1  PIC S9(4)  BINARY.
77           B2  PIC S9(4)  BINARY.
        :
     COMPUTE Z1 = B1 * B2.       ...      (1)
        :
     COMPUTE Z2 = B1 * B2.       ...      (2)
```

In Example 2, B1 and B2 remain unchanged between (1) and (2), B1 * B2 becomes common and (2) is optimized so it uses results of (1).

Shifting an Invariant

For arithmetic or conversion processing done within a loop, the processing can be performed outside the loop if the same results would be obtained regardless of whether the processing is done within or outside the loop.

Example 1:

```
77           I      PIC S9(4)  BINARY.
77           ZD     PIC S9(7)  DISPLAY.
01           REC-1.
             02 B1  PIC S9(7)  BINARY OCCURS 20 TIMES.
             :
             MOVE 1  TO  I.
LOOP-1                                    }
             IF B1(I) = ZD GO TO EXIT-1.       }
             :                                 }   Loop
             ADD 1 TO I.                       }
             IF I IS <= 20 GO TO LOOP-1.       }
```

If ZD never changes throughout the loop in Example 1, the compiler shifts ZD-to-Binary conversion processing by the IF statement to the outside of the loop.

Optimizing an Induction Variable

If the compiler finds a loop containing a partial expression that uses an item (induction item) defined recursively only by a constant (as in ADD 1 TO I., for example) or by an item with an unchanging value, it introduces a new induction variable and changes the multiplication of the subscript calculation formula to an addition.

Example 1:

```
77           I   PIC S9(4)  BINARY.
01           REC-1.
             02     A PIC X(10)    OCCURS 20 TIMES.
               :
LOOP-1.                                           }
      IF  A(I) =   ...                 ...(1)    }
        :                                         }   Loop
      ADD 1 TO I.                      ...(2)    }
      IF I IS <= 20   GO TO LOOP-1.    ...(3)    }
```

Because I is defined recursively by a constant within the loop in Example 1, the compiler introduces a new induction variable to replace the multiplication "I * 10" in the address calculation formula for A(I) "A - 10 + I * 10" with “t”, and generates "ADD 10 TO t" after (2).

Furthermore, if I is not used anywhere else in the loop and, concurrently, the value of I calculated inside the loop is not used after control exits from the loop, the compiler replaces (3) with "IF t IS <= 20 GO TO LOOP-1" and deletes (2).

Optimizing a PERFORM Statement

The compiler expands the PERFORM statement into some instructions for saving, setting and restoring the return address for the return mechanism. The exit of the PERFORM statement transfers control to the other statement, generally. Otherwise, some of the machine instructions for the return mechanism turn out to be redundant.

The compiler then removes these redundant machine instructions.

Integrating Adjacent Moves

If different alphanumeric move statements transfer data from contiguous items to identically contiguous items, the compiler integrates these move statements into one.

Example 1:

```
          02  A1  PIC X(32).
          02  A2  PIC X(16).
    :
          02  B1  PIC X(32).
          02  B2  PIC X(16).
            :
          MOVE A1 TO B1.                      ...(1)
            :
          MOVE A2 TO B2.                      ...(2)
```

If A2 and B2 do not change between (1) and (2) and, concurrently, B2 is not referenced in Example 1, the compiler deletes MOVE statement (2). Then MOVE statement (1) sets A1 and A2, and B1 and B2, as one area, respectively, to perform a move.

Eliminating Unnecessary Substitutions

Substitutions are eliminated for data items that are not going to be either implicitly or explicitly referenced.

Notes

If the compiler option OPTIMIZE is specified, the compiler generates a globally optimized object program. For details, refer to Appendix A, “Compiler Options.”

Notes on Global Optimization

The following explains notes on global optimization:

- Use of the linkage function

  Some parameters for a called program share all or part of a storage area (for example, CALL "SUBPROG" USING A, A. or CALL "SUBPROG" USING A, B. where A and B share part of the area).

  If the content of the area is overwritten by the called program, the optimization of the called program may not lead to the expected result. So, do not specify the compiler option OPTIMIZE for such a program.

- No execution of the global optimization

  The compiler does not perform global optimization on the following programs:

  – Programs that do not define the items and index names having the attributes to target by global optimization

  – Programs using the segmentation function (segmentation module)

  – Programs specifying a compiler option TEST

- Less effective global optimization

Global optimization is less effective on the following programs:

- Programs mainly performing I/O operations, and programs that usually do not use the CPU
- Program using no numeric items but only alphanumeric items
- Programs referencing nondeclaratives from declaratives
- Programs referencing declaratives from nondeclaratives
- Programs specifying the compiler option TRUNC. For details, refer to Appendix A, “Compiler Options.”

Notes on Debugging

Since global optimization causes the deletion, shifting and modification of one or more statements, a program interruption (for example, data exception) can occur a different number of times or at a different location.

If the program is interrupted while data items are begin written in, those data items may not actually be set.

With compiler option NOTRUNC, the program may not operate properly if the internal or external decimal item is to be recursively defined. For details, refer to Appendix A, “Compiler Options.”

Appendix D. Intrinsic Function List

The following table lists the Intrinsic functions that can be used with COBOL85.

Table 66. COBOL85 Intrinsic Functions

Classification	Function	Explanation
Length	LENGTH	Obtains the length of a data item or literal.
Size	MAX	Obtains the maximum value.
	MIN	Obtains the minimum value.
	ORD-MAX	Obtains the ordinal position of the maximum value.
	ORD-MIN	Obtains the ordinal position of the minimum value.
Conversion	REVERSE	Reverses the order of character strings.
	LOWER-CASE	Converts uppercase characters to lowercase characters.
	UPPER-CASE	Converts lowercase characters to uppercase characters.
	NUMVAL	Converts numeric characters to numeric values.
	NUMVAL-C	Converts numeric characters including a comma and currency sign to numeric values.

Table 66. COBOL85 Intrinsic Functions (cont.)

Classification	**Function**	**Explanation**
Character operation	CHAR	Obtains a character at a specified position in the collating sequence of a program.
	ORD	Obtains the ordinal position of a specified character in the collating sequence of a program.
Numeric value operation	INTEGER	Obtains the maximum integer within a specified range.
	INTEGER-PART	Obtains integer parts.
	RANDOM	Obtains random numbers.
Calculation of interest rate	ANNUITY	Obtains the approximate value of the equal payment rate to 1 (the principal) according to the rate of interest and the period.
	PRESENT-VALUE	Obtains the current price according to the reduction rate.
Date operation	CURRENT-DATE	Obtains the current date and time and the difference between Greenwich Mean Time.
	DATE-OF-INTEGER	Obtains the date corresponding to the day of the year.
	DAY-OF-INTEGER	Obtains the year and day corresponding to the day of the year.
	INTEGER-OF-DATE	Obtains the day of the year corresponding to the date.
	INTEGER-OF-DAY	Obtains the day of the year corresponding to the year and day.
	WHEN-COMPILED	Obtains the date and time the program was compiled.

Table 66. COBOL85 Intrinsic Functions (cont.)

Classification	**Function**	**Explanation**
Arithmetic calculation	SQRT	Obtains the approximate value of a square root.
	FACTORIAL	Obtains factorials.
	LOG	Obtains natural logarithms.
	LOG10	Obtains common logarithms.
	MEAN	Obtains average values.
	MEDIAN	Obtains medians.
	MIDRANGE	Obtains the average values of the maximum and minimum.
	RANGE	Obtains the difference between the maximum and minimum.
	STANDARD-DEVIATION	Obtains standard deviations.
	MOD	Obtains a specified value in specified modulus.
	REM	Obtains remainders.
	SUM	Obtains sums.
	VARIANCE	Obtains variances.
Trigonometric function	SIN	Obtains the approximate value of a sine.
	COS	Obtains the approximate value of a cosine.
	TAN	Obtains the approximate value of a tangent.
	ASIN	Obtains the approximate value of an inverse sine.
	ACOS	Obtains the approximate value of an inverse cosine.
	ATAN	Obtains the approximate value of an inverse tangent.

Appendix E. Special Registers Used with Screen and Form Functions

This appendix explains the values of special registers used with screen and form functions. Some functions cannot be used depending on the FORM RTS version and level. For details, refer to the FORM RTS online help.

EDIT-MODE

Before execution of the WRITE statement, the output mode of a data item can be specified for the EDIT-MODE special register. The following table explains the values that can be specified.

Table 67. Values that can be specified for EDIT-MODE

Value	Explanation	Value	Explanation
" " (Blank)	Applies output processing.	"*"	Expands and edits characters.
"X"	Does not apply output processing.	"K"	Does not output the initial value.
"N"	Outputs the data as national data. (National data items only)	"B"	Outputs bit map data.
"A"	Outputs the data as alphanumeric data. (National data items only)	"O"	Outputs OLE objects.

EDIT-STATUS

Before execution of the READ statement, the input mode of a data item can be specified for the EDIT-STATUS special register. The input results are posted after execution of the READ statement. The following table explains the values that can be specified.

Table 68. Values that can be specified for EDIT-STATUS

Value	Explanation	Value	Explanation
" " (Blank)	Applies input processing.	"S"	Enables selection status display.
"X"	Does not apply input processing.	"D"	Does not apply input processing and enables selection status display.
"B"	Disables the specification for input highlighting.		

The following table explains the return values.

Table 69. Values returned to EDIT-STATUS

Value	Explanation	Value	Explanation
"E"	The input data contains an error.	"S"	The item has been selected. (Only when an item is selected for input)
"Z"	Input was omitted for an item.	"I"	The item was entered through the ID card.
" " (Blank)	The data was inputted as national data. (National data items only)	"U"	The data was not changed.
"A"	The data was inputted as alphanumeric data. (National data items only)		

EDIT-COLOR

Before execution of the WRITE statement, the display color of a data item can be specified for the EDIT-COLOR special register. The following table explains the values that can be specified.

Table 70. Values that can be specified for EDIT-COLOR

Value	Explanation	Value	Explanation
" " (Blank)	Does not change the previous information.	"O"	Outputs data in dark gray.
"M"	Outputs data according to the color attribute of the screen descriptor.	"K"	Outputs data in dark blue.
"N"	Outputs data in non-display mode.	"F"	Outputs data in dark red.
"B"	Outputs data in blue.	"J"	Outputs data in dark pink.
"R"	Outputs data in red.	"I"	Outputs data in dark turquoise.
"P"	Outputs data in pink.	"C"	Outputs data in dark yellow.
"G"	Outputs data in green.	"E"	Outputs data in dark green.
"T"	Outputs data in turquoise.	"D"	Outputs data in gray.
"Y"	Outputs data in yellow.	"W"	Outputs data in white.
"A"	Outputs data in black.	"H"	Outputs data with high brightness. (*1)
		"L"	Outputs data with low brightness. (*1)

*1 High brightness cannot be distinguished from low brightness under this system.

EDIT-OPTION

Before execution of the WRITE statement, attributes of a data item can be specified for the EDIT-OPTION special register. The following table explains the values that can be specified.

Table 71. Values that can be specified for EDIT-OPTION

Value	Explanation	Value	Explanation
" " (Blank)	Does not change the previous information.	"D"	Outputs data without using any underline, blink, or reverse.
"M"	Outputs data according to the item attribute of the screen descriptor.	"V"	Outputs data using blink and reverse. (*1)
"U"	Outputs data using underline.	"A"	Outputs data using underline, blink, and reverse. (*1)
"B"	Outputs data using blink. (*1)	"-" (Hyphen)	Outputs deletion lines. This value is for a printed form function. Do not specify this value for a screen display function.
"R"	Outputs data using reverse.		
"S"	Outputs data using underline and blink. (*1)		
"T"	Outputs data using underline and reverse.		

*1 Blink cannot be used under this system.

EDIT-CURSOR

Before execution of the READ statement, positioning of the cursor can be specified for the EDIT-CURSOR special register. The following table explains the values that can be specified.

Table 72. Values that can be specified for EDIT-CURSOR

Value	Explanation	Value	Explanation
" " (Blank)	Does not position the cursor.	"X"	Suppresses the specification of the cursor position item. (Cursor position item only)
"C"	Positions the cursor.		

Appendix F. Message Lists

This appendix explains messages written by the WINCOB and compile commands, the COBOL85 compiler, and the COBOL85 run-time system.

Note: Messages listed in this chapter are 32-bit COBOL messages. 32-bit COBOL messages are generally more descriptive and apply to both 32 and 16-bit COBOL. While 16-bit users may see a slightly different message, the message number will provide an easy mapping between the older 16-bit messages and the more descriptive 32-bit messages.

Messages Output by WINCOB and Compile Commands

This section explains messages written by the WINCOB and compile commands, system action, programmer response, and parameters. A variable character string is expressed as $s in the messages below.

```
'$s' is an invalid source file name.
```

Compilation terminated.

Response
Specify the correct file name, then re-execute.

```
An invalid suboption is specified for '$s'.
```

Compilation terminated.

Response
Specify the correct sub-option, then re-execute.

```
'-P' option is missing.  '$s' option is ignored.
```

Ignores the option ($s) and continues compilation.

Response
To enable the option ($s), specify it with the -P option, then re-execute.

```
Source file name is missing.
```

Compilation terminated.

Response
Specify the name of the file to be compiled, then re-execute.

```
'$s' is not a COBOL85 Compiler option.
```

Compilation terminated.

Response
Specify the correct option, then re-execute.

```
Argument of '$s' is not specified.
```

Compilation terminated.

Response
Specify the argument for the option ($s) correctly, then re-execute.

```
'$s' was not found.
```

Compilation terminated.

Response
Specify the correct file name, then re-execute. The file name is written for $s.

```
System error '$s1' occurred on '$s2'.
```

Terminates compilation abnormally.

Response
Check and eliminate the cause of the error. System error code errno=0xXXX XXX (expressed in hexadecimal) is written for $s1. The name of the processing in which the system error occurred is written for $s2.

```
Number of users exceeded the specified maximum.
```

Compilation terminated.

Response
Start processing when the number of users becomes smaller than the specified maximum.

```
Error occurred in LANPACK environment.  '$s'.
```

Compilation terminated.

Response
Check and eliminate the cause of the error. The character string indicating the error information is written for $s. (See Table 72.)

Table 72. Error information

Character String	Error Information	Programmer Response
SETERR	The environment variable for LANPACK is not specified correctly.	Check if LANPACK is installed correctly.
NOUSFILE	The control file for LANPACK is not found.	____________
OTHERERR	Other error.	The error is probably caused by LANPACK. Collect the error information and call your Fujitsu SE.

Messages Produced by the COBOL85 Compiler

This section explains messages written by the COBOL85 compiler.

The following shows the message format:

```
message-number  line-number  message-text
```

- message-number
- message number is displayed as shown below.

```
JMNnnnnI-S
```

JMN: Prefix indicates a COBOL compiler message.

nnnn: Message serial number.

I: Indicates that operator response is not required.

S: Indicates the severity code. (See Table 73.)

Table 74. Severity codes of messages output by the COBOL85 compiler

Severity Code	Level	Meaning	Return Code
I (INFORMATION)	Transmitting message	Information detected by the compiler.	0
W (WARNING)	Warning error	The object program has been generated. However, check whether the compilation results are as expected.	
E (ERROR)	Recoverable error	The object program has been generated. An error (e.g., compiler option error) has occurred.	1
S (SERIOUS)	Serious error	The object program has not been generated.	2
U (UNRECOVERABLE)	Unrecoverable error	Compilation has been halted.	3

- line-number

  The line number of the line containing an error is displayed. Messages indicating the same error information are displayed collectively. In this case, more than one line number is displayed.

- message-text

  Explains the error conditions.

Messages Produced by the COBOL85 Run-time System

This section explains messages written by the COBOL85 run-time system.

The following shows the message format:

```
message-number  message-text
```

- message-number

  The message number is displayed as shown below:

```
JMPnnnni-w
```

JMP: Prefix indicates a run-time system message.

nnnn: Message serial number.

i: Characters to indicate severity and response required.

I: Operator response is unnecessary.

A: Operator response is necessary.

w: Severity code. (See Table 74.)

Table 75. Meaning of severity codes in execution-time message and their effect on return codes

Severity Code	**Level**	**Meaning**	**Effect on Return Code**
I (INFORMATION)	Transmitting message	Information detected by the run-time system. The program can be executed normally.	Not affected
W (WARNING)	Warning error	The program can be executed normally. However, check whether the execution results are as expected.	Not affected

E (ERROR)	Recoverable error	There is an error, but processing is continued according to the appropriate assumption.	Affected(*1)
U (UNRECOVERABLE)	Unrecoverable error	The system stops program execution and performs termination processing.	Not affected (*2)

*1 If an E-level message output, a return code is set when the COBOL execution unit terminates. The E-level message return code is assumed to be 12. It is compared to the value in special register PROGRAM-STATUS, and the larger value is set as the return code.
*2 The program terminates abnormally.

Message-text explains the error conditions.

The following explains an error address written for a parameter indicating variable information in the message text:

- error address

  The place where the error was detected is displayed in the following form.

  - When either compile option TRACE or CHECK is specified:

  ```
  PGM= External program name. LINE= Statement number.
  ```

  Statement number: [COPY modification value -] line number. Verb number

  - For cases except those mentioned above:

  ```
  PGM= External program name. ADR= nnnnnnnn.
  ```

  nnnnnnnn : Application address where the error is detected.

The following explains messages written by the COBOL85 run-time system, system action, and programmer response in the following format:

```
message-number
   message-text (*1)
```

System action or run-time system action

Explanation
Explanation of the message text

Response
Action to be taken by the programmer

*1 $n in message text is a parameter indicating variable information. The parameter is replaced by the program name or file name in the actual message.

The following explains an access name or file name written for a parameter indicating variable information in message text:

- Access name:

  An access name is used for connecting a program and file entity. The access name corresponds to the run-time environment information name specified with the file entity name at run-time. The access name also corresponds to the file identifier specified in the ASSIGN clause.

- File name:

  A file name is a file entity name. The file name corresponds to the file identifier literal or data name specified in the ASSIGN clause. "?" is displayed if the data name is not recognized.

  If DISK or PRINTER is specified in the ASSIGN clause, the file name corresponds to the file name specified in the SELECT clause.

  The file name specified in the SELECT clause is enclosed in parentheses in the message text.

```
JMP0001I-U
OPEN ERROR.  FILE=$1.
```

Abnormally terminates the program.

Response

- Check whether the access name ($1) corresponding to the function name SYSIN or SYSOUT is assigned correctly.

- Before activating Windows, start the SHARE command in MS-DOS command mode. (16)

```
JMP0002I-U
PROGRAM '$1' IS RECURSIVELY CALLED.
```

Abnormally terminates the program.

Explanation
The program ($1) was called, and is re-called before it terminates (before execution of the STOP RUN or EXIT PROGRAM statement).

Response
Check the CALL statement in the called program.

```
JMP0007I-W
INVALID OPTION.  $1.
```

Ignores the run-time option ($1), and continues processing.

If the specified run-time option is SWITCH, the SWITCH run-time option is enabled as follows:

- If the digit count of the number in the specified numeric part is smaller than eight, the number is written left-justified.
- If the digit count is greater than eight, the high-order eight digits are valid.

Response
Specify the correct run-time option, then re-execute.

```
JMP0009I-U
INSUFFICIENT STORAGE AVAILABLE.
```

Abnormally terminates the program.

Response
Do the following, then re-execute:

- Stop all applications that are executing at the same time
- Confirm the size of real storage and increase it if necessary

- Confirm the size of virtual storage and increase it if necessary

```
JMP0010I-U
LIBRARY WORK AREA IS BROKEN.
```

Abnormally terminates the program.

Explanation
The work area of the COBOL85 run-time system is broken.

Response
Check the program as follows:

- If an item including a subscript, index, reference modification, or OCCURS DEPENDING ON clause is referred to, confirm that the item is within the reference range. This check can be made by specifying the CHECK option.
- If a parameter was specified for transferring data via the CALL statement, confirm that the sender and receiver have the same parameter attributes. Especially, check the length attribute.
- If a file is used, confirm that the record is referenced in the correct length. For example, if the maximum record length is used instead of the actual record length and the remaining portion is filled with spaces, the record is referenced incorrectly.

```
JMP0012I-U
INTERNAL PROGRAM IS RECURSIVELY CALLED IN PROGRAM $1.
```

Abnormally terminates the program.

Explanation
The internal program included in the outermost program ($1) was called, and the internal program is re-called before it terminates (before execution of the STOP RUN or EXIT PROGRAM statement).

Response
Check the CALL statement in the called program.

```
JMP0015I-U
CANNOT CALL PROGRAM '$1'.  $2.
```

Abnormally terminates the program.

Response

- Eliminate the cause of the error according to the error code written for $2, then re-execute.

(16): Refer to “Error Codes of the Load Library function.”

(32): Refer to Visual C++ on-line help information.

- If no error code is written for $2, check whether the entry point ($1) specified in the CALL statement exists in the DLL. (16)

```
JMP0016I-U
READ/WRITE ERROR.  FILE='$1'.
```

Abnormally terminates the program.

Response
Check the file assigned to the access name ($1).

```
JMP0018I-U
CLOSE ERROR.  FILE='$1'.
```

Abnormally terminates the program.

Response
Check the file assigned to the access name ($1).

```
JMP0019I-U
EXTERNAL DATA ATTRIBUTE ERROR.  DATA=$1. PGM=$2,$3.
```

Abnormally terminates the program.

Response
Confirm that the program ($2) in which the error was detected and the program ($3) which defined the EXTERNAL data ($1)

have the same $1 definition. A data name is usually written for $1. If no data name is defined (e.g., FILLER), the file definition number that appeared in the source program is written.

```
JMP0020I-U
INVALID ENTRY INFORMATION.  PGM=$1.
```

Abnormally terminates the program.

Response

(16): Check whether the correspondence between the DLL name, program name, and secondary entry point name of the called program ($1) is correctly defined in the initial file for execution.

(32): Check whether the entry information (DLL name, program name and the second entry point name of the called program ($)) is correctly defined in the entry information file specified for @CBR_ENTRYFILE or in the initialization file for execution.

```
JMP0022I-U
EXTERNAL FILE ATTRIBUTE ERROR.  FILE=$1. PGM=$2,$3. '$4'.
```

Abnormally terminates the program.

Response
Confirm that the program ($2) in which the error was detected and the program ($3) that is first defined the EXTERNAL file ($1) have the same $1 definition. A file name is usually written for $1. If the file name cannot be written, the file definition number that appeared in the source program is written. One of the character strings shown in Table 75 is written for $4.

Table 76. Character strings written for JMP0022I-U $4

Common file organizations

$4	Error Contents
ACCESS-MODE	Access mode
ACCESS-NAME	Access name
FILE-ORG	File organization
LOCK-MODE	LOCK MODE clause specification
MAXRL	Maximum record length
MINRL	Minimum record length
OPTIONAL	Presence or absence of OPTIONAL clause specification
REC-MODE	Record format

Relative/Indexed file

$4	Error Contents
ALT-KEY	Number of alternate record keys
KEY-ATR	Key item attribute
KEY-DISP	Displacement of the key item in a record
KEY-DUPL	Presence or absence of DUPLICATES clause specification
KEY-LEN	Key length
REL-ATR	Attribute of the relative key item
REL-COL	Column count of the relative key item
REL-KEY	Presence or absence of RELATIVE KEY clause specification
REL-LEN	Length of the relative key item
REL-NAME	Name of the relative key item

Print file

$4	Error Contents
ADVANCING	Presence or absence of the WRITE statement in which ADVANCING is specified
CHAR-TYPE	Presence or absence of specification of how to write the CHARACTER TYPE clause for format 3
CODE	Presence or absence of specification of the CODE clause
CTLCHR	Attribute of the control statement area
LINAGE	Presence or absence of specification of the LINAGE clause
LNG-ATR	Attribute and column count of the LINAGE data item
LNG1-INTEG	Integer of LINAGE data item 1
LNG1-NAME	Name of LINAGE data item 1
LNG2-INTEG	Integer of LINAGE data item 2
LNG2-NAME	Name of LINAGE data item 2
LNG3-INTEG	Integer of LINAGE data item 3
LNG3-NAME	Name of LINAGE data item 3
LNG4-INTEG	Integer of LINAGE data item 4
LNG4-NAME	Name of LINAGE data item 4
REC-LEN	Record length

```
JMP0026I-W
NO 'ON EXCEPTION' STATEMENT EXISTS, WHEN EXCEPTION OCCURRED ON
'$2 $1' STATEMENT.
```

Continues processing.

Response
Specify the ON EXCEPTION clause in the $2 statement (ACCEPT or DISPLAY statement) corresponding to the $1 function name (ARGUMENT-VALUE or ENVIRONMENT-VALUE).

```
JMP0092I-U
CANNOT REGISTER WINDOWCLASS '$1'.
```

Abnormally terminates the program.

Response

Do the following, then re-execute:

- Confirm that the program name does not contend with another process.
- Check whether the EXPORT statement is specified correctly in the module definition file.
- Stop all applications that are executing at the same time
- Confirm the size of real storage and increase it if necessary
- Confirm the size of virtual storage and increase it if necessary
- Check whether the initial file format is correct. (16)

```
JMP0093I-U
CANNOT CREATE WINDOW '$1'.
```

Abnormally terminates the program.

Response

Do the following, then re-execute:

- Confirm that the program name does not contend with another process.
- Check whether the stack size is large enough.
- Stop all applications that are executing at the same time
- Confirm the size of real storage and increase it if necessary
- Confirm the size of virtual storage and increase it if necessary
- Check whether the initial file format is correct. (16)

```
JMP0096I-U
RUN-TIME INITIAL FILE'S PATH-NAME IS INVALID.
```

Abnormally terminates the program.

Response

Check whether the path name of the initial file for execution is specified correctly.

```
JMP0097I-U
RUN-TIME SYSTEM IS NOT INSTALLED PROPERLY. FILE=$1.
```

Abnormally terminates the program.

Response

- Confirm the file($1) is correctly installed.
- Confirm the run-time system is correctly installed. Or, confirm the installation directory of the run-time system is correctly set in the environmental variable PATH.

```
JMP0099I-U
FORCED TERMINATION.  CODE=$1.
```

Terminates the program.

Response
Eliminate the cause of the error according to the previous output message. If no message was written, an internal error probably occurred. Call your Fujitsu SE. The ABEND (abnormal end) code is written for $1.

```
JMP0200I-E
INSUFFICIENT DATA ACCEPTED FROM SYSIN.
```

Continues processing assuming that the remaining part of the inputted data area is blank.

Response

- Prepare the number of data items that can be input by executing the ACCEPT FROM SYSIN statement.
- Confirm that the ACCEPT statement is not executed a number of times equal to or greater than the number of data items.

```
JMP0201A-I
AWAITING A REPLY.
```

After receiving response, continues processing.

Explanation
The system asks the operator to input data by using the ACCEPT statement.

Response
Input any necessary data.

```
JMP0202A-I
'$1'
```

After receiving response, continues processing.

Explanation
The STOP literal statement is being executed.

Response
Input any character. The literal specified in the STOP literal statement is written for $1.

```
JMP0204I-U
STATEMENT SEQUENCE ERROR.  STM=$1. PGM=$2. LINE=$3. RPT=$4.
```

Abnormally terminates the program.

Response
Correct the execution sequence of statements for the report or detail phrase indicated in the message, then re-execute. The type of statement (INITIATE, GENERATE, or TERMINATE) is written for $1. The outermost program name is written for $2. The statement number ([copy-qualification-value-]line-number.verb-number.) is written for $3. The report name or data name (detail phrase name) is written for $4.

```
JMP0206I-W
SCREEN ITEM SIZE IS LARGER THAN LOGICAL SCREEN.
```

The screen item is displayed partially.

Response
Specify a logical screen size large enough to display the entire item.

```
JMP0207I-U
LOGICAL SCREEN SIZE IS TOO LARGE.
```

Abnormally terminates the program.

Response
Specify the logical screen size so that (column count + 1) * line count is equal to no more than 16,250, then re-execute.

```
JMP0208I-E
INVALID NUMERICAL VALUE ACCEPTED.
```

Writes zero to the receive data item, then continues processing.

Response
Specify a value for the numeric item of the ACCEPT statement correctly, then re-execute.

```
JMP0301I-E
'$1'($2) OPENED BY '$3' IS NOT CLOSED.
```

Terminates the program without performing close processing for the file.

Response
Correct the program ($3) so that the CLOSE statement is executed for the file ($1) before termination of the program ($3). The open mode of the file ($1) is written for $2.

```
JMP0302I-U
CLOSE ERROR DURING PROGRAM TERMINATION.  PGM=$1. FILE=$2.
```

Abnormally terminates the program.

Response
Check and eliminate the cause of the error. If another message is written, refer to it to determine the cause. The program name is written for $1. The access name or file name is written for $2.

```
JMP0310I-I/U
$1 ERROR.  FILE=$2. '$3'. $4
```

Continues program processing if the severity code is I; aborts the program if the severity code is U. The severity code is determined as follows:

I: The FILE STATUS clause is specified for file definition.

U: The FILE STATUS clause and error procedures are not specified.

Response
Eliminate the cause of the error according to the character string written for $3, then re-execute. OPEN or CLOSE is written for $1. The access name or file name is written for $2. One of the character strings shown in Tables 77 and 78 are written for $3. The error address is written for $4.

Table 77. Character strings written for JMP0310I-I/U $3 (1)

$3	Error Contents	Programmer Response
ACC-METHOD	The specified file access method is invalid. (*1)	Specify the file access method correctly.
BLKED-FILE	The file cannot be used because of a CLOSE statement error.	Check and eliminate the cause of the CLOSE statement error.
ERFLD=xxxx	Error code from the system xxxx: Expressed in hexadecimal	Check for and eliminate the cause of the error according to the system error code. (*2)
ERRCD=xxxx	A presentation file access error occurred.	Check for and eliminate the cause of the error according to the manuals of the connected products. (*3)
EXCL-ERROR	An exclusive error occurred.	Re-execute. If the exclusive error occurs repeatedly, confirm that the error does not affect operation.
FCB	The FCB control statement is invalid	Check the FCB control statement.
FILE-LOCK	The file is exclusively used by another user, or the file cannot be used exclusively because it is being used by another user.	
FTNCD=xxxx	Error code from the system xxxx: Expressed in hexadecimal	Check for and eliminate the cause of the error according to the system error code. (*2)
LOAD	The subprogram could not be loaded.	Check the program run-time environment.
LOCK-FULL	The lock table is full.	Re-execute. If the error occurs repeatedly, confirm that the error does not affect operation.
NON-FILE	The file did not exist when the OPEN statement without OUTPUT specified was executed.	Before executing the program, create the file.
NON-REEL	The CLOSE statement with REEL or UNIT specified was executed.	Correct the program.

Table 77. Character strings written for JMP0310I-I/U $3 (1) (cont.)

$3	Error Contents	Programmer Response
NON-UNIQUE	Allocated file has a duplicate key.	Make the keys duplicate in the program specification.
OPEN-MODE	The specified file open mode is invalid.	Specify the correct open mode, then open the file.
READ-ONLY	A read-only file is used, or the specified file name is invalid.	Change the file attribute, or specify the correct file name.
REC-MODE	The specified record format is invalid.	Specify the correct record format.
RMERR=xxxx	Error code from the system xxxx: Expressed in hexadecimal	A system error probably occurred. Collect the error information, then call your Fujitsu SE.
UNSUPPORT	The file does not support the specified function.	Check the file attribute, or confirm that there is no physical error.

*1 An attempt was made to access a different file organization or to access an indexed file requiring recovery.
*2 - If a message is generated showing error information ERFLD=101 when a print file (without the FORMAT clause specified) is executed, ensure the following:

Only the required number of printer drivers are installed.
The FCB control statement is less than 2,048 bytes.
All the specified forms overlay patterns are correct.
All the registered forms overlay pattern names are correct.
The spool area is large enough.
(Confirm the disk capacity of the environment variable (TMP.) directory.
Stop all other applications that are executing at the same time.
Confirm the size of real storage and increase it if necessary.
Confirm the size of virtual storage and increase it if necessary.
If no problems are found in the above checks, a system error probably occurred.
Collect the error information, then call your Fujitsu SE.

- For Btrieve file error codes, refer to the “Btrieve Manual” issued by NOVELL Inc.
- See “System Error Codes” for details on character strings indicating errors in the file message text other than that above.

*3 If ERRCD=90xx is output while FORM RTS is used, refer to the FORM RTS online help for xx.

If any of the character strings shown in Table 77 are displayed, eliminate any conflicts between the file or record definition in the program and the attribute of the file to be processed.

Table 78. Character strings written for JMP0310I-I/U $3 (2)

$3	**Error Contents**	**Check Item in Program**
INV-BLKSZ	The block length is invalid.	Length specified in the BLOCK CONTAINS clause
INV-CODEST	The code set is invalid.	Character code specified in the CODE SET clause
INV-COLSEQ	The collating sequence is invalid.	Alphabet name specified in the PROGRAM COLLATING SEQUENCE clause
INV-DELMTR	The record delimiter is invalid.	Data specified in the ORGANIZATION, RECORD, and RECORD DELIMITER clauses
INV-KEYDUP	Determination of whether to specify a key in duplicate (DUPLICATES) is incorrect.	Presence or absence of specification of the RECORD KEY, ALTERNATE RECORD KEY, and DUPLICATES clauses
INV-KEYLEN	The key length of the allocated file conflicts with the definition in the program.	Data name length specified in the RECORD KEY and ALTERNATE RECORD KEY clauses
INV-KEYSTR	The key structure is invalid.	Data name length and data name count specified in the RECORD KEY and ALTERNATE RECORD KEY clauses, or the relative position in the record of the data name specified in the clauses.
INV-LRECL	The record length of the allocated file conflicts with the definition in the program.	Data specified in the ORGANIZATION clause. Record length specified in the RECORD CONTAINS clause.
INV-RKP	The relative key position of the assigned file conflicts with the definition in the program.	Relative position in the record of the data name specified in the RECORD KEY and ALTERNATE RECORD KEY clauses
KEY-ATTR	The key attribute of the allocated file conflicts with the definition in the program.	Data name attribute specified in the RECORD KEY and ALTERNATE RECORD KEY clauses

Table 78. Character strings written for JMP0310I-I/U $3 (2) (cont.)

$3	Error Contents	Check Item in Program
KEY-ESDS	A sequential file was not allocated when specified.	Data specified in the ORGANIZATION clause
KEY-KSDS	An indexed file was not allocated when specified.	
KEY-RRDS	A relative file was not allocated when specified.	
KEY-TEXT	A text file was not allocated as a print file.	

```
JMP0311I-I/U
MISSING ALLOCATION.  FILE=$1. $2.
```

Continues program processing if the severity code is I; aborts the program if the severity code is U. The severity code is determined as follows:

I: The FILE STATUS clause is specified for file definition.

U: The FILE STATUS clause and error procedures are not specified.

Response

Allocate the file ($1) if necessary, or specify the file correctly, then re-execute. The error address is written for $2.

```
JMP0313I-I/U
INSUFFICIENT STORAGE AVAILABLE.  STM=$1. FILE=$2. $3
```

Discontinues error file processing, then continues program processing if the severity code is I; aborts the program if the severity code is U. The severity code is determined as follows:

I: The FILE STATUS clause is specified for file definition.

U: The FILE STATUS clause and error procedures are not specified.

Response
Do the following, then re-execute:

- Stop any other applications that are executing at the same time
- Confirm the size of real storage and increase it if necessary
- Confirm the size of virtual storage and increase it if necessary

The COBOL statement (OPEN, CLOSE, START, READ, WRITE, REWRITE, or DELETE) that caused the virtual storage to become insufficient is written for $1. The file name or access name of the processed file is written for $2. The error address is written for $3.

```
JMP0320I-I/U
INPUT/OUTPUT ERROR.  STM=$1. FILE=$2. '$3'. $4
```

Discontinues error file processing, then continues program processing if the severity code is I; aborts the program if the severity code is U. The severity code is determined as follows:

I: The FILE STATUS clause is specified for file definition.
U: The FILE STATUS clause and error procedures are not specified.

Response
Eliminate the cause of the error according to the character string written for $3. The COBOL statement (START, READ, WRITE, REWRITE, or DELETE) that caused the input-output error is written for $1. The file name or access name of the processed file is written for $2. One of the character strings shown in Table 79 is written for $3. The error address is written for $4.

Table 79. Character strings written for JMP0320I-I/U $3

$3	Error Contents	Programmer Response
ALTD-LEN	The record length specified in the REWRITE statement is different from the length of an existing record.	Do not change the length of an existing record when updating the record.
BLKED-FILE	The file cannot be used because of a CLOSE statement error.	If message JMP0310I-I is displayed before JMP0320I-I, eliminate the cause of the error according to message JMP0310I-I.
ERRCD=xxxx	A presentation file access error occurred.	Check for and eliminate the cause of the error according to the manuals of the linked products. (*1)
EXCEED-LEN	The length of the record to be written exceeds the maximum record length specified in the program.	Set the record length to within the specified maximum record length.
EXCL-ERROR	An exclusive error occurred.	Re-execute. If the exclusive error occurs repeatedly, confirm that the error does not affect operation.
FCB	The FCB contents are invalid.	Check the FCB contents.
FDBK=xxxx	Error code from the system xxxx: Expressed in hexadecimal	Check for and eliminate the cause of the error according to the system error code. (*2)
FONT	A font that did not exist in the system was specified for output.	Install necessary fonts in the system.
FOVL	The specified directory containing the forms overlay definition is invalid, or the forms overlay definition is invalid.	Check the initial file to determine if the directory containing the correct forms overlay definition is specified.
FTNCD=xxxx	Error code from the system xxxx: Expressed in hexadecimal	Check for and eliminate the cause of the error according to the system error code. (*2)

Table 79. Character strings written for JMP0320I-I/U $3 (cont.)

$3	Error Contents	Programmer Response
INV-CHAR	A line sequential file record or print file record contains an invalid character.	Check the data contents.
INV-LEN	The record length specified in the WRITE or REWRITE statement is invalid.	Specify the correct length of the record to be written.
KEY-CHANGE	The key value at request for updating the record was different from the key value at reading of the record.	Correct the program so that the key value is not changed at updating of the record.
LOAD	The subprogram could not be loaded.	Check the program run-time environment.
NO-TRANS	Some characters in native mode cannot be converted into a code set.	Check the data contents.
PHYSIC-ERR	A physical error occurred.	Check the file contents (physical structure).
READ-ONLY	A read-only file is used.	Change the file attribute.
REC-LOCK	The record is exclusively used by another user, or the record cannot be used exclusively because it is being used by another user.	Re-execute. If the exclusive error occurs repeatedly, confirm that the error does not affect operation.
R.KEY-ERR	The relative key contents conflict with the definition specified in the PICTURE clause during random or dynamic access.	Specify a data item size in the PICTURE clause large enough to store the maximum value of the relative key.
	The relative key item value is 0.	Specify a non-zero value for the relative key item.

Table 79. Character strings written for JMP0320I-I/U $3 (cont.)

$3	Error Contents	Programmer Response
RMERR=xxxx	Error code from the system xxxx: Expressed in hexadecimal	A system error probably occurred. Collect the error information, then call your Fujitsu SE.
RRN>R.KEY	When the READ or WRITE statement is executed for sequential processing, the relative record number exceeds the maximum value of the relative key item.	Specify a size of the relative key item large enough to store the maximum relative record number of the file

*1 If ERRCD=90xx is output while Form RTS is used, refer to the Form RTS HELP for xx.

*2 • If a message is output showing error information FDBK=101 when a print file (without the FORMAT clause specified) is executed, ensure the following:

Only the required number of printer drivers are installed.
The FCB control statement is less than 2,048 bytes.
All the specified forms overlay patterns are correct.
The spool area is large enough.
(Confirm the disk capacity of the environment variable (TMP.) directory.
Stop all other applications that are executing at the same time.
Confirm the size of real storage and increase it if necessary.
Confirm the size of virtual storage and increase it if necessary.
All the registered forms overlay pattern names are correct.
If no problems are found in the above checks, a system error probably occurred.
Collect the error information, then call your Fujitsu SE.

- For Btrieve file error codes, refer to the “Btrieve Manual” issued by NOVELL Inc.
- See “System Error Codes” for details on character strings indicating errors in the file message text other than the above.

```
JMP0321I-U
AT END CONDITION.  STM=READ. FILE=$1. $2
```

Abnormally terminates the program.

Response

- Specify AT END phrase in the READ statement to perform file end processing.

- Check whether the number of input data items is correct.

The file name or access name of the file with which a file end condition was produced is written for $1. The error address is written for $2.

```
JMP0322I-U
KEY SEQUENCE ERROR.  STM=$1. FILE=$2. $3
```

Abnormally terminates the program.

Response

- If $1 is the WRITE statement, correct the program so that the key values are written in ascending order.
- If $1 is the REWRITE or DELETE statement, correct the program so that the key value of the record read by the previous READ statement is not changed.

The file name or access name of the file with which an invalid key condition was produced is written for $2. The error address is written for $3.

```
JMP0323I-U
DUPLICATE KEY ERROR.  STM=$1. FILE=$2. $3
```

Abnormally terminates the program.

Response
Correct the program so that a key is not duplicated. The error COBOL statement (WRITE, REWRITE or CLOSE) is written for $1. The file name or access name of the error file is written for $2. The error address is written for $3.

```
JMP0324I-U
RECORD NOT FOUND.  STM=$1. FILE=$2. $3
```

Abnormally terminates the program.

Response

- Correct the program so that attempts to access a non-existent record are inhibited.
- Correct the program by using the FILE STATUS or INVALID KEY clause so that the program checks an attempt to access a non-existent record.

The error COBOL statement (READ, REWRITE, START, or DELETE) is written for $1. The file name or access name of the error file is written for $2. The error address is written for $3.

```
JMP0325I-I/U
BOUNDARY VIOLATION ERROR.  STM=$1. FILE=$2. $3
```

Continues program processing if the severity code is I; aborts the program if the severity code is U. The severity code is determined as follows:

I: The FILE STATUS clause is specified for file definition.

U: The FILE STATUS clause and error procedures are not specified.

Response
Increase the size of the file space, then re-execute. The error COBOL statement (WRITE, REWRITE, READ, START, or CLOSE) is written for $1. The file name or access name of the error file is written for $2. The error address is written for $3.

```
JMP0326I-U
NO-SPACE CONDITION.  STM=$1. FILE=$2. $3
```

Abnormally terminates the program.

Response
Increase the size of the file space, then re-execute. The error COBOL statement (WRITE, REWRITE, READ, START, or CLOSE) is written for $1. The file name or access name of the error file is written for $2. The error address is written for $3.

```
JMP0327I-U
INVALID KEY CONDITION.  STM=$1. FILE=$2. '$3'. $4
```

Abnormally terminates the program.

Response
If $3 is RRN>R.KEY, correct the program so that the relative record number does not exceed the maximum value permitted for the relative key item when records are written in sequential processing. Alternatively, correct the program by using the INVALID KEY clause so that processing for an invalid key condition is performed. The error COBOL statement (WRITE) is written for $1. The file name or access name of the error file is written for $2. The error address is written for $4.

```
JMP0328I-I/U
DEPENDING ON OBJECT VALUE IS OUT OF RANGE.  STM=WRITE. FILE=$1.
$2
```

Continues program processing if the severity code is I; aborts the program if the severity code is U. The severity code is determined as follows:

I: The FILE STATUS clause is specified for file definition.

U: The FILE STATUS clause and error procedures are not specified.

Response
Correct the program so that the DEPENDING ON value is defined in the specified range. The file name or access name of the error file is written for $1. The error address is written for $2.

```
JMP0330I-I/U
STATEMENT SEQUENCE ERROR.  STM=$1. FILE=$2. '$3'. $4
```

Continues program processing if the severity code is I; aborts the program if the severity code is U. The severity code is determined as follows:

I: The FILE STATUS clause is specified for file definition.

U: The FILE STATUS clause and error procedures are not specified.

Response

Eliminate the cause of the error according to the character string written for $3, then re-execute. The COBOL statement (OPEN, CLOSE, READ, WRITE, REWRITE, START, or DELETE) with which an input-output error occurred is written for $1. The file name or access name of the processed file is written for $2. One of the character strings shown in Table 80 is written for $3. The error address is written for $4.

Table 80. Character strings written for JMP0330I-I/U $3

$3	Error Information
AT-END	After a file at end condition was produced, the READ statement was re-executed.
DUPL-OPEN	The OPEN statement was executed for an open file.
LOCKED	After the CLOSE statement with LOCK phrase specified was executed an attempt was made to open the file.
NO-READ	The previous statement is not a successful READ statement.
NOSPACE	After a no-space condition was produced, the WRITE statement was re-executed.
NOT-OPENED	An input-output statement was executed for a closed file.
OPEN-MODE	The OPEN mode is invalid.
POS-ERROR	The file position indicator is undefined.

```
JMP0340I-U
CONTROL RECORD FORMAT ERROR.  STM=WRITE. FILE=$1. '$2'. $3
```

Abnormally terminates the program.

Response

Eliminate the cause of the error according to the character string written for $2, then re-execute. The file name or access name of the processed file is written for $1. One of the character strings shown in Table 81 is written for $2 and shows the relationship between the character strings and control record contents. The error address is written for $3.

Table 81. Character strings written for JMP0340I-U $2

$2	Field Contents	$2	Field Contents
BIND	Binding direction	OSTK	Offset stack
CMOD	Copy modification module name	PFRM	Printing format
C-NO	Number of copies	POST	Printing surface positioning
FCB	FCB name	PRTA	Unprintable area
FID	Format definition deck name	RSV	Reserved area (not blank)
FORM	Forms code	R-NO	Forms overlay printing count
FOVL	Forms overlay module name	SIDE	Printing surface specification
F-NO	Number of forms overlay fields	SIZE	Forms size
HOP	Hopper	STK	Stacker
ID	Control record ID	S-NO	Copy correction start number
LEN	Control record length	T-NO	Number of character array table for copy correction
LOAD	Dynamic loading	WDTH	Binding width
MODE	Control mode	XTB	Name of character array table for copy correction
OFST	Print origin position		

```
JMP0350I-U
 USE PROCEDURE IS RECURSIVELY CALLED.
```

Abnormally terminates the program.

Response

If execution of an input-output statement during an error procedure causes re-execution of the error procedure, do not let control reach the last statement of the first-executed error procedure.

```
JMP0360I-U
PS FILE ACCESS ERROR.  CODE=$1.
```

Abnormally terminates the program.

Response
Check the contents of the destination specified in the SYMBOLIC DESTINATION clause. If the contents are correct, a system error probably occurred. Collect the error information, then call your Fujitsu SE. The internal code indicating detailed error information is written for $1.

```
JMP0363I-U
SYMBOLIC DESTINATION ERROR.  FILE=$1.
```

Abnormally terminates the program.

Response
Specify the destination name in the SYMBOLIC DESTINATION clause correctly, then re-execute. The file name or access name is written for $1.

```
JMP0364I-U
CLAUSE '$1' ERROR.  FILE=$2.
```

Abnormally terminates the program.

Response
Specify the correct data item in the $1 clause, then re-execute. The file name or access name is written for $2.

```
JMP0365I-U
DUPLICATE FONT NAMES WERE FOUND.  '$1'.
```

Abnormally terminates the program.

Response
Multiple fonts indicated by $1 were installed. Use the control panel font application to delete unnecessary fonts, then re-execute: Ensure that:

Windows 95

“Print True Type as Graphic” is checked in the printer Properties dialog box.

Windows 3.1

- The same font type is not specified for the WIN.INI [fonts] section and WIFEMAN.INI [FontPackages] section.
- Mincho, Gothic, or Courier is generally used as the device font of the printer. None of the above three fonts is specified for the WIN.INI [fonts] section and WIFEMAN.INI [FontPackages] section.

```
JMP0370I-U
INSUFFICIENT STORAGE AVAILABLE TO PERFORM SQL.
```

Abnormally terminates the program.

Response
Do the following, then re-execute.

- Stop all applications that are executing at the same time
- Confirm the size of real storage and increase it if necessary
- Confirm the size of virtual storage and increase it if necessary

```
JMP0371I-U
ENVIRONMENT INFORMATION FILE ERROR TO PERFORM SQL.  '$1'.
```

Abnormally terminates the program.

Response
Specify data in the environment information file correctly, then re-execute. The character string indicating the error information is written for $1. (See Table 81.)

Table 82. Character strings written for JMP0371I-U $1

(With SequeLink)

$1	Error Information
@SQL_CLI	No client interface type was specified, or the specified character is not supported.
@SequeLink_Inf	No SequeLink information file was specified, or the specified SequeLink information file does not exist.
@SQL_HOSTNM	No host name was specified.
@SQL_SERVER	No server name was specified.
@SQL_USERID	No user ID was specified.
@SQL_PASSWORD	No password was specified.
@SQL_DBKIND	No database type was specified, or the specified database type is not supported.
@SQL_SERVICE	No service name was specified.
@SQL_LOGPRM1	Logon parameter 1 was specified.
@SQL_LOGPRM2	Logon parameter 2 was specified.

(With ODBC)

$1	Error Information
@ODBC_Inf	No ODBC information file was specified, or the specified ODBC information file does not exist.
@SQL_DATASRC	No data source name was specified.
@SQL_SERVER	No server name was specified.
@SQL_USERID	No user ID was specified.
@SQL_PASSWORD	No password was specified.

```
JMP0372I-U
START ENVIRONMENT ERROR TO PERFORM SQL.  '$1'. $2
```

Abnormally terminates the program.

Response

Eliminate the cause of the error according to the character string written for $1, then re-execute. (See Table 82.)

Table 83. Character string written for JMP0372I-U $1

$1	Error Contents	Programmer Response
LOAD	The run-time library could not be loaded.	Eliminate the cause of the error according to the detail code (Load Library function error code) written for $2.
FREE	The run-time library could not be loaded.	Eliminate the cause of the error according to the detail code (Load Library function error code) written for $2.

```
JMP0373I-I
$1 ERROR.  STM=$2. SERVER=$3. '$4'.
```

Continues program processing.

Response
Eliminate the cause of the error according to the error code written for $4, then re-execute. The type of the client interface is written for $1. The error SQL statement is written for $2. The name of the error server is written for $3.

For the error code, refer to the error code manual of the client interface.

```
JMP0374I-U
UNRECOGNIZABLE SQL STATEMENT WAS FOUND.
```

Abnormally terminates the program.

Response
Delete the SQL statement that cannot be used, then re-execute. For SQL statements that cannot be used, see the requirements for each database environment and notes on the environment.

```
JMP0392I-U
THE NETWORK DATABASE FUNCTION CANNOT BE USED.
```

Abnormally terminates the program.

Response

- Change the program so that it does not use functions specific to GS-series hosts.
- To use distributed development support functions, re-execute under the debugger. (16)

```
JMP0400I-U
ACCESS ENVIRONMENT ERROR OCCURRED. SYS=$1,FILE=$2,BLK=$3,$4=$5
```

Abnormally terminates the program.

Response
A system error probably occurred. Collect the error information, then call your Fujitsu SE. The following information is written for $1 to $5:

$1: Error information from the system, or the character string NONE if the system sends no error information

$2: System information

$3: Control block address at which the error was detected

$4: ERR or REQ

$5: The following is set as the internal error information:

If $4 is ERR: Error-detected location

If $4 is REQ: Processing code

```
JMP0410I-U
STOP RUN STATEMENT MUST NOT BE EXECUTED USING JMPCINT2.
```

Abnormally terminates the program.

Response
Replace the STOP RUN statement in the application program with the EXIT PROGRAM statement.

```
JMP0420I-I
CANNOT OBTAIN WINDOW HANDLE.  '$1'. $2
```

Returns -1.

Response
Eliminate the cause of the error, then re-execute. The character string indicating the error information is written for $1. (See Table 84.)

Table 84. Character strings written for JMP0420I-I $1

$1	Error Contents	Programmer Response
ACC-ERROR	A window handle allocation error occurred.	Check for and eliminate the cause of the error according to the FORM RTS code written for $2. (*1)
INV-ENV	An operating environment error occurred.	Check the COBOL operating environment.
INV-KIND	The file type corresponding to the specified file-identifier is invalid.	Specify the presentation file-identifier.
NON-FILE	The file corresponding to the specified file-identifier does not exist.	Specify the file-identifier correctly, or check if the file was opened normally.

*1 Refer to the FORM RTS online help.

```
JMP0600I-I/U
INSUFFICIENT STORAGE AVAILABLE TO PERFORM SORT OR MERGE
STATEMENT.
```

Continues program processing if the severity code is I; aborts the program if the severity code is U. The severity code is determined as follows:

I: SORT-STATUS is referred to for file definition.

U: SORT-STATUS is not referred to.

Response
Do the following, then re-execute.

- Stop all applications that are executing at the same time
- Confirm the size of real storage and increase it if necessary
- Confirm the size of virtual storage and increase it if necessary

```
JMP0601I-I/U
SORT OR MERGE STATEMENT ERROR.  $1. $2
```

Continues program processing if the severity code is I; aborts the program if the severity code is U. The severity code is determined as follows:

I: SORT-STATUS is referred to for file definition.

U: SORT-STATUS is not referred to.

Response
Eliminate the cause of the error according to the value written for $1, then re-execute. The file information (file name or access name) or detailed information is written for $1. The error address is written for $2. Table 85 explains the values written for $1.

Table 85. Values written for JMP0601I-I/U $1

$1	Error Contents	Programmer Response
51	An error occurred during input processing for the record to be sorted.	If the same error occurs when the program is re-executed, a system error probably occurred. Collect the error information, then call your Fujitsu SE.
52	Memory is insufficient.	Modify the environment to allocate enough memory.
53 54 56 59 62 101	An error occurred during input processing for the record to be sorted.	If the same error occurs when the program is re-executed, a system error probably occurred. Collect the error information, then call your Fujitsu SE.
102	An error occurred during output processing for the record to be sorted.	
103	An error occurred during input processing for the record to be merged.	
110	The address of the record to be sorted could not be obtained.	
111	The record length is invalid.	Check whether the record length complies with the COBOL85 specification.
112 113	An error occurred during input processing for the record to be sorted.	If the same error occurs when the program is re-executed, a system error probably occurred. Collect the error information, then call your Fujitsu SE.
114	An error occurred during input processing for the record to be merged.	
208	An error occurred when a temporary file (for sorting and merging) was opened.	Modify the environment (e.g., hard disk) to allocate a large enough temporary file. If the same error occurs when the program is re-executed, a system error probably occurred. Collect the error information, then call your Fujitsu SE.
212	The size of the temporary file (for sorting and merging) is insufficient.	Modify the environment (e.g., hard disk) to allocate a large enough temporary file.

Table 85. Values written for JMP0601I-I/U $1 (cont.)

$1	Error Contents	Programmer Response
214	A temporary file (for sorting and merging) cannot be created.	Modify the environment (e.g., hard disk) to allocate a large enough temporary file.
224	An error occurred during write to a temporary file (for sorting and merging).	If the same error occurs when the program is re-executed, a system error probably occurred. Collect the error information, then call your Fujitsu SE.
250	A sort-merge program error occurred.	
Other	A PowerSORT error occurred while using PowerSORT.	Refer to the PowerSORT online help.

Do the following, then re-execute.

- Stop all applications that are executing at the same time
- Confirm the size of real storage and increase it if necessary
- Confirm the size of virtual storage and increase it if necessary

```
JMP0608I-I/U
RELEASE STATEMENT CAN NOT BE EXECUTED IN OUTPUT PROCEDURE. $1
```

Continues program processing if the severity code is I; aborts the program if the severity code is U. The severity code is determined as follows:

I: SORT-STATUS is referred to for file definition.
U: SORT-STATUS is not referred to.

Response
Correct the program. The error address is written for $1.

```
JMP0609I-I/U
RETURN STATEMENT CAN NOT BE EXECUTED IN INPUT PROCEDURE. $1
```

Continues program processing if the severity code is I; aborts the program if the severity code is U. The severity code is determined as follows:

I: SORT-STATUS is referred to for file definition.
U: SORT-STATUS is not referred to.

Response
Correct the program. The error address is written for $1.

```
JMP0612I-I/U
RETURN STATEMENT CAN NOT BE EXECUTED AFTER AT END. $1
```

Continues program processing if the severity code is I; aborts the program if the severity code is U. The severity code is determined as follows:

I: SORT-STATUS is referred to for file definition.
U: SORT-STATUS is not referred to.

Response
Correct the program so that the RETURN statement is not executed after a file at end condition is produced. The error address is written for $1.

```
JMP0613I-I/U
SORT/MERGE LIBRARY CANNOT BE LOADED. $1  $2
```

Continues program processing if the severity code is I; aborts the program if the severity code is U. The severity code is determined as follows:

I: SORT-STATUS is referred to for file definition.
U: SORT-STATUS is not referred to.

Response
Remove the cause of the error, then re-execute. The library file name is set in $1 and the error code is set in $2.

Refer to the Visual C++ online help.

Note: If $1 and $2 are not output, confirm whether or not PowerSORT is correctly installed.

```
JMP0701I-U
GCB IS BROKEN.
```

Abnormally terminates the program.

Response
The program is broken. Correct the program as follows, then re-execute:

- If an item including a subscript, index, reference modification, or OCCURS DEPENDING ON clause is referred to, confirm that the item is within the allowable reference range.
- If there is a parameter for transferring data through the CALL statement, confirm that the sender and receiver have the same parameter attributes. (Be sure to check the length attribute.)
- If a file is used, confirm that the correct record length is referenced. (For example, if SPACE clear processing is performed with the maximum record length instead of the actual record length when a variable-length input file is used, the record length referenced is incorrect.)

```
JMP0702I-U
LIA IS BROKEN.
```

Abnormally terminates the program.

Response
See JMP0701I-U.

```
JMP0703I-U
RCB IS BROKEN.
```

Abnormally terminates the program.

Response
See JMP0701I-U.

```
JMP0704I-U
DEBUG TABLE IS BROKEN.
```

Abnormally terminates the program.

Response
See JMP0701I-U.

```
JMP0705I-W
INSUFFICIENT STORAGE AVAILABLE.  FUNC=$1.
```

Cancels the '$1' function, then continues processing.

Response
Do the following, then re-execute.

- Stop all applications that are executing at the same time
- Confirm the size of real storage and increase it if necessary
- Confirm the size of virtual storage and increase it if necessary

The character string indicating the function is written for $1.
(TRACE: Trace function / COUNT: Count function (16))

```
JMP0724I-W
FILE CAN NOT BE OPENED.  FILE=$1. '$2' OPTION IGNORED.
```

Continues processing without outputting '$2' information.

Response
Check whether the file is allocated correctly. The access name is written for $1. The debug function name is written for $2.

```
JMP0725I-W
FILE CAN NOT BE OPENED.  '$1' OPTION IGNORED.
```

Continues processing without outputting '$1' information.

Response
Check whether the area is large enough to create a file. The debug function name is written for $1.

```
JMP0770I-W
DEBUGGER CANNOT BE ACTIVE.  TEST OPTION IGNORED.  $1. '$2' $3.
```

Ignores the compiler option TEST and continues processing.

Response
Eliminate the cause of the error according to the character string written for $2, then re-execute. The name of the error program is written for $1. The following explains the character strings written for $2:

'MEMORY SHORTAGE': The memory is insufficient.

'FILE NOT FOUND': The $1 file is not found.

'PATH NOT FOUND': The search path is not found.

'PATH NOT SET': No search path was specified.

'LOADING ERROR': Loading failed.

The error code (Load Library function error code) was written for $3.

```
JMP0820I-E/U
SUBSCRIPT/INDEX IS OUT OF RANGE.  PGM=$1. LINE=$2. OPD=$3 ($4)
```

E level: Uses the subscript or index value as is, then continues processing.

U level: Aborts the program.

Response
Specify a correct value for the subscript or index indicated in the message, then re-execute. The outermost program name is written for $1. The statement number (line-information[copy-qualification-value-]line-number.verb-number) is written for $2. The operand name is written for $3. The number of dimensions is written for $4.

```
JMP0821I-E/U
REFERENCE MODIFIER IS OUT OF RANGE.  PGM=$1. LINE=$2. OPD=$3.
```

E level: Performs reference modification with an incorrect value, then continues processing.

U level: Aborts the program.

Response
Correct the program so that reference modification of the data item indicated in the message is performed within the specified range. The outermost program name is written for $1. The statement number (line-information[copy-qualification-value-]line-number.verb-number) is written for $2. The operand name is written for $3.

```
JMP0822I-E/U
ODO OBJECT VALUE IS OUT OF RANGE.  PGM=$1. LINE=$2. OPD=$3.
ODO=$4.
```

E level: Uses the object value specified in the OCCURS DEPENDING ON clause as is, then continues processing.

U level: Aborts the program.

Explanation
The object value specified in the OCCURS DEPENDING ON clause is not within the permitted range.

Response
Specify a correct value for the object (in the OCCURS DEPENDING ON clause) indicated in the message, then re-execute. The outermost program name is written for $1. The statement number (line-information[copy-qualification-value-]line-number.verb-number) is written for $2. The name of the operand that refers to the data including the OCCURS DEPENDING outermost-program-name ON clause is written for $3. The object name specified in the OCCURS DEPENDING ON clause is written for $4.

```
JMP0899I-U
LOGICAL ERROR.
```

Abnormally terminates the programs.

Response
The program or dynamic area may be damaged because of a program error. Check for the cause of the error in the same manner as when JMP0701I-U is written.

```
JMP0901I-E
FUNCTION $1 ERROR, ARGUMENT ABSOLUTE VALUE IS MORE THAN 3.53E+15.
```

Returns 0, then continues processing.

Response
Specify the absolute value of the argument within the calculation enabled range, then re-execute. The name of the error function (SIN or COS) is written for $1.

```
JMP0902I-E
FUNCTION $1 ERROR, ARGUMENT ABSOLUTE VALUE IS MORE THAN 1.0.
```

Returns 0, then continues processing.

Response
Specify the absolute value of the argument that can be stored in the definition area, then re-execute. The name of the error function (ASIN or ACOS) is written for $1.

```
JMP0903I-E
FUNCTION $1 ERROR, ARGUMENT VALUE IS OUT OF RANGE.
```

Returns 0, then continues processing.

Response
Specify the value of the argument that can be stored in the definition area, then re-execute. The name of the error function (LOG, LOG10, ANNUITY, FACTORIAL, RANDOM, SQRT, or TAN) is written for $1.

```
JMP0904I-E
FUNCTION $1 ERROR, ARGUMENT VALUE IS INVALID.
```

Returns 0, then continues processing.

Response
Specify the value of the argument that satisfies the condition of each function, then re-execute. The name of the error function is written for $1.

```
JMP0905I-E
FUNCTION ANNUITY ERROR, ARITHMETIC EXCEPTION.
```

Returns 0, then continues processing.

Response
Specify the value of the argument that does not cause zero division, then re-execute.

System Error Codes

This section explains system error codes.

The meanings of the error codes are different between Windows 3.1 and Windows 95/Windows NT. See the error code table for each system.

- With Windows 3.1: See Table 86.
- With Windows 95/Windows NT: See Table 87.

Error Codes (16)

Table 86. System error codes (16)

CODE	Explanation	Programmer Response
1 (0x01)	Invalid function code	Before activating Windows, confirm that the SHARE command is active in MS-DOS command mode.
4 (0x04)	The number of files to be opened is too large.	Specify the correct value for the SHARE command option.
5 (0x05)	Access cannot be made.	Check if the file is allocated correctly, or specify the correct value for the SHARE command option.
15 (0x0F)	Invalid drive name.	Allocate the file to a drive that can be used.
20 (0x14)	A disk unit is defective.	Confirm that the disk unit is not damaged.
36 (0x24)	The shared buffer is full.	Specify a larger value for the /L operand of the SHARE command.

Refer to Chapter 5 for more information about the SHARE command.

Error Codes (32)

Table 87. System error codes (32)

CODE	Explanation	Programmer Response
2 (0x02)	The specified file cannot be found.	Check to see if the product is correctly installed.
4 (0x04)	The file cannot be accessed.	Check the run-time environment.
5 (0x05)	Access cannot be made.	Check whether the file is allocated correctly.
8 (0x08)	Command cannot be processed because the applicable storage area is insufficient.	Do the following, then re-execute: ■ Stop all applications that are executing at the same time ■ Confirm the size of real storage and increase it if necessary ■ Confirm the size of virtual storage and increase it if necessary
15 (0x0F)	Invalid drive name.	Check whether the drive name is correct.
19 (0x13)	Write-protected disk.	Check the disk.
20 (0x14)	A disk unit is defective.	Confirm that the disk unit is not damaged.
21 (0x15)	The drive cannot get ready.	Check the drive.
34 (0x22)	The incorrect disk is in the drive.	Check the disk.
53 (0x35)	The network path is not found.	Check the specified network path name.
59 (0x3B)	An unexpected network error occurred.	Check whether the network environment is set up correctly.
127 (0x7F)	The specified procedure cannot be found.	Check to see if the product is correctly installed.
206 (0xCE)	The file name is too long.	Check the file name.
1785 (0x6F9)	The disk may be not formatted.	Check whether the disk is formatted.

Appendix G. Writing Special Literals

This appendix explains how to write literals for specifying names (for example, program and file names) defined in the system.

Program Name Literal

Any characters can be used for a literal specifying a program name in the PROGRAM-ID paragraph, CALL statement, or CANCEL statement. There are no configuration rules. You must determine if the literal complies with the linker rules.

Text Name Literal

For the text name literal to be written in the COPY statement, specify the name of the file containing library text in the following format:

```
"[drive-name: ] [path-name]  file-name [extension]"
```

[drive-name]

Specify the drive-name by a single character from A to Z. When the drive-name is omitted, the current drive will be used.

[path-name]

Specify the directory storing the file in the following format:

```
[\][directory-name [\ directory-name] ...]\
```

When the path-name is omitted, the file will be placed in the directory specified by compile option LIB. When compiling from the WINCOB window, the source file to be compiled is stored in the current directory. When the relativity path-name is specified, the current directory is added to the front.

[File-name]

Specify the file-name.

[extension]

Specify the file extension if possible. **Note**: "CBL" and "COB" may not be used. For example:

- "C:\COPY\A.CBL"
- "A.CPY"
- "C:\COPY\A"

File-Identifier Literal

For the file-identifier literal to be written in the ASSIGN clause of the file control entry, specify the file to be processed in the following manner:

```
    { drive-name }
"[  {            } :] [path-name] file-name [.extention]"
    { port-name  }
```

[drive-name]

Specify a drive name with an alphabetic character from A to Z. A name not specified with an alphabetic character from A to Z is regarded as a port name. When the drive name is omitted, the current drive will be used.

[port-name]

A port name can be specified for sequential files only. If a port name is specified, specification of a path name and file-reference-name is invalid.

To specify a printer with a port name, use port name LPT1, LPT2, or LPT3.

[path-name]

Specify the directory storing the file in the following format:

```
[\][directory-name[\directory-name] ...]\
```

If a path name is omitted the current directory is used.

[file-name]

Specify a file name.

[extension]

Specify any character string for identifying the type of the file.

For example:

- "A:\COBOL\A.DAT"
- "LPT1:"
- "B.TXT"

Appendix H. High-Speed File Processing

Record sequential files and line sequential files can be accessed faster by specifying an available range. This appendix explains methods of specification for high-speed file processing and notes on high-speed file processing.

Specification Methods

This section explains methods of specification for high-speed file processing. The specification methods apply to both record sequential files and line sequential files.

When defining a file-identifier as a file-reference-identifier in the program, specify ",BSAM" following the file-name to be allocated upon setting of environment variable information. Refer to “Environment Variables” in Chapter 5 for details on setting environment variables information.

```
file-identifier=[path-name]file-name,BSAM
```

When defining a data name as a file-reference-identifier in the program, specify ",BSAM" following the file-name to be allocated at definition of the data-name in the program.

```
MOVE "[path-name]file-name,BSAM" TO data-name.
```

When defining a file-identifier literal as a file-reference-identifier in the program, specify ",BSAM" following the file-name to be allocated at definition of the file-identifier literal in the program.

```
ASSIGN TO "[path-name]file-name,BSAM"
```

Notes

Records cannot be updated (REWRITE statement cannot be executed). If a record is updated, an error occurs during execution (Record sequential files only).

If files are shared and you want to permit file sharing among different processes, all files must be in shared mode and opened with INPUT specified. Operation is not guaranteed if a file is opened without specifying INPUT. File sharing is not permitted in the same process. Operation is not guaranteed if a file is shared with in the same process.

High-speed file processing cannot be used if DISK is specified as the file-reference-identifier.

If a record read from a line sequential file includes a tab, the tab code is not replaced by a blank.

Appendix I. GS-series Function Comparison

Table 88 compares functions between the GS-series (M-series) and COBOL85 for Windows. Refer to Chapter 16, “Distributed Development Support Functions” for a definition of GS-series terms.

The following symbols are used in the "Comparison" column of Table 88:

A: Can be used in the same manner as the GS-series.

B: Can be used under given conditions in the same manner as the GS-series.

C: Cannot be used with the GS-series. (The function is specific to this system or incompatible with the GS-series.)

D: Compilation can be done, but the function is disabled during execution.

E: Can be used, but the function works differently than when it works with the GS-series.

F: Cannot be used with this system.

Table 88. Function comparison between the GS-series and this system

Function Classification			Comparison	Remarks
Classification		**Function outline**		
Character set		All types of characters that can be used in the program	A	
COBOL words	User-defined word	All types of user-defined words	A	Use of the underscore (_) character has a specific function in this system.
	Figurative constant	All figurative constants that can be used in the program	A	
	Special register	SHIFT-IN,SHIFT-OUT, SORT-CORE-SIZE, SORT-MESSAGE	D	
		PROGRAM-STATUS, RETURN-CODE	B	The attributes are different. GS-series: S9(4)BINARY This system: S9(9)COMP-5
		Other than the above	A	
	Function name	SYSPUNCH,STACKER-01 to 12, CSP,S01,S02 SYSPCH,BUSHU,SOKAKU, ON-YOMI, KUN-YOMI	D	
		SWITCH-8	C	
		CHANNEL02 to12 C02 to C12	B	The FCB control statement must be specified.

Table 88. Function comparison between the GS-series and this system (cont.)

Function Classification			**Comparison**	**Remarks**
Classification		**Function outline**		
COBOL words	Function name	Other than the above	A	
	Literal	National item literal National alphanumeric literal National association literal National language literal	D	
		Hexadecimal literal National hexadecimal literal National code literal	E	Note differences among codes.
		Other than the above	A	
	Others	Specification of quotation mark as a constant	C	GS-Series: Compile option APOST/QUOTE is followed This system: Automatically determined.
Method of writing program	Reference format	Sequence number	A	
		Fixed format, variable format	A	
Data definition	Data description	All clauses that can be described in the data description entry	A	
	Data type	COMP-5	C	
		Other than the above	A	

Table 88. Function comparison between the GS-series and this system (cont.)

Function Classification			**Comparison**	**Remarks**
Classification		**Function outline**		
Expression	Arithmetic expression	Binary arithmetic operator, unary arithmetic operator	A	
	Conditional expression	All relational operators that can be used	A	
	Linkage expression	Use of linkage expression	C	
	Class condition	All class conditions that can be used	A	
	Other conditions	Condition-name condition, sign condition, switch-status condition	A	
Nucleus	Environment definition	SUBSCHEMA-NAME paragraph	B (16)	Operation can be checked with the debugger.
			D (32)	
		Other than the above	A	
	Basic instruction	All nucleus module statements	A	

Table 88. Function comparison between the GS-series and this system (cont.)

Function Classification			Comparison	Remarks
Classification		**Function outline**		
Sequential file	Environment definition	APPLY WRITE-ONLY clause MULTIPLE FILE TAPE clause RERUN clause PASSWORD clause RESERVE AREA clause	D	
		Data name specified in the ASSIGN clause DISK specified in the ASSIGN clause PRINTER specified in the ASSIGN clause LOCK MODE clause	C	
		Other than the above	A	
	File definition	CODE-SET clause	D	
		BLOCK CONTAINS clause	B	No functional meaning for this system. The program that was operating with the GS-series operates as before.
		Other than the above	A	

Table 88. Function comparison between the GS-series and this system (cont.)

Function Classification			Comparison	Remarks
Classification		Function outline		
Sequential file	Input-output statement	WITH LOCK specified in an input-output statement UNLOCK statement	C	
		Other than the above	A	
Line sequential file		All	C	
Relative file	Environment definition	PASSWORD clause RERUN clause	D	
		Data name specified in the ASSIGN clause DISK specified in the ASSIGN clause LOCK MODE clause	C	
		Other than the above	A	
	File definition	CODE-SET clause	D	
		BLOCK CONTAINS clause	B	No functional meaning for this system. The program that was operating with the GS-series operates as before.

Table 88. Function comparison between the GS-series and this system (cont.)

Function Classification			**Comparison**	**Remarks**
Classification		**Function outline**		
Relative file	File definition	Other than the above	A	
	Input-output statement	WITH LOCK specified in an input-output statement UNLOCK statement	C	
		Other than the above	A	
Indexed file	Environment definition	PASSWORD clause RERUN clause	D	
		Data name specified in the ASSIGN clause DISK specified in the ASSIGN clause LOCK MODE clause	C	
		Permission to form a single key with multiple data items	B	Can be used with this system under given conditions.
		Other than the above	A	

Table 88. Function comparison between the GS-series and this system (cont.)

Function Classification			**Comparison**	**Remarks**
Classification		**Function outline**		
Indexed file	File definition	CODE-SET clause	D	
		BLOCK CONTAINS clause	B	No functional meaning for this system. The program that was operating with the GS-series operates as before.
		Other than the above	A	
	Input-output statement	WITH LOCK specified in an input-output statement UNLOCK statement	C	
		POSITIONING POINTER specified in the START statement	F	
		Other than the above	A	
Sort-merge	Mnemonic name	BUSHU,SOKAKU, ON-YOMI,KUN-YOMI	D	
	Special register	SORT-CORE-SIZE, SORT-MESSAGE	D	
		Other than the above	A	
	Others	All	A	

Table 88. Function comparison between the GS-series and this system (cont.)

Function Classification			**Comparison**	**Remarks**
Classification		**Function outline**		
Inter-program communication	PROCEDURE DIVISION	WITH specification	C	
	CALL statement	BY VALUE specification	C	
		WITH specification	C	
		Other than the above	A	
	Others	All	A	
Source text manipulation	COPY statement	OF/IN SYSDBDCT specification	F	
		OF/IN XMDLIB and XFDLIB specification	C	
		Library text name literal	C	
		JOINING specification only	C	
Report writer feature	File definition	BLOCK CONTAINS clause CODE clause	D	
		Other than the above	A	
	Others	All	A	

Table 88. Function comparison between the GS-series and this system (cont.)

Function Classification			**Comparison**	**Remarks**
Classification		**Function outline**		
Presentation file	Environment definition	APL, CMD, TRM, or WST specified in the SYMBOLIC DESTINATION clause	B (16)	Operation can be checked with the debugger.
			D (32)	
		APPLY MULTICONVERSATION-MODE clause	D	
		PROCESSING TIME clause	C	
		DESTINATION CONTROL clause	D	
		MESSAGE SEQUENCE clause	D	
		MESSAGE CODE clause (not 4 bytes)	D	
		Other than the above	A	
	File definition	All	A	
	Input-output statement	All	A	
	Special register	All	A	
Debugging functions		All	D	Compilation only if used with the GS-series
Segmentation		All	D	Compilation only if used with the GS-series

Table 88. Function comparison between the GS-series and this system (cont.)

Function Classification			**Comparison**	**Remarks**
Classification		**Function outline**		
Communication		All	D	
Extension	System control	All	D	
	Network database	All	B (16)	Operation can be checked with the debugger.
			D (32)	
	AIM / RDB	All	F	
	SD function	All	A	
Built-in function		All	C	
Screen handling		All	C	
Command line argument handling and environment variable handling		All	C	

Checking Program Operation

Programs created in common function scope can be checked under this system. Some functions may cause program execution methods and execution results to differ between systems.

When operating a program using the inter-program communication function, if the program is activated through a system, the method of specifying the parameters to be passed to the program is different.

To pass parameters in the global server system format under this system, use the initialization file (COBOL85.CBR) or Run-time Environment Setup window to specify the parameters. Refer to “Setting Run-time Environment Information” in Chapter 5 for the method of specifying the global server system format parameters in the initialization file (COBOL85.CBR) and the Run-time Environment Setup window.

For example, a COBOL program description is as follows:

Example: COBOL program description

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. A.
     :
     :
LINKAGE SECTION.
01 parameter.
     03 parameter-length PIC 9(4) BINARY.
     03 parameter-character-string.
         05 character-1 PIC X
            OCCURS 1 TO 100 TIMES
            DEPENDING ON parameter-length.
PROCEDURE DIVISION USING parameter.
     :
     :
```

[Execution method and result with M-series host]
[Input command]

```
CALL 'X9999.A LOAD(A)' 'ABDCE'
```

[Parameter contents]

9	A	B	C	D	E

[Initialization file for getting same results with M-series host]
[The contents of the initialization file]

```
[A]
 :
 :
@MGPRM="ABCDE"
 :
```

If a program uses a function specific to the GS-series, special processing is required for operating the program under this system.

To substitute another resource and operate a program using the communication function, execute input-output statements for the sequential file. Check the contents of the sequential file to determine whether the program operates as expected.

If skipping a non-executable statement by using an interactive debugger, and the file to be processed or the subprogram does not exist, use the interactive debugger to skip input-output and CALL statements. For details on how to use the interactive debugger, see the “Fujitsu COBOL Debugging Guide.”

Notes

If a hexadecimal non-numeric literal and national hexadecimal nonnumeric literal are used in the program, take the code into consideration.

If the presentation file function is used, the expansion format of a presentation file record fetched from a screen and form descriptors (this system) or format descriptor (GS-series) is different when the COBOL source program is compiled.

If a function name (CHANNEL02 to CHANNEL12, or C02 to C12) is used, the FCB control statement is required. The FCB control statement format used with the global server ADJUST is also used under this system.

Appendix J. Command Formats

This appendix explains compiler commands and linker commands.

The command formats differ between Windows 95/Windows NT and Windows 3.1.

Compiler Commands

Use the compiler commands to compile a COBOL source program and to generate an object program.

For details on generated files and files required for compiler command execution, refer to “Resources Necessary for Compilation” in Chapter 3. For compiler command specified location and basic usage, refer to “Using Commands to Compile” also in Chapter 3.

The following are compiler commands:

- (16): COBOL command
- (32): COBOL command, or COBOL32 command

Input Format

Command	**Operand**
COBOL	[option-list] file-name …
COBOL32 (32)	[option-list] file-name …

Operands

At least one space is needed between the command name and each operand. Items enclosed in "[" and "]" can be omitted. An absolute or relative path name can be specified for a directory name and file name in the explanation below.

[option-list]

Specify the information to be posted to the COBOL85 compiler. See Table 88 for the specification.

The following shows the priority determined by the option specification methods:

1. Using the compiler directing statement in the source file (Refer to “Compiler Directing Statement” in Chapter 2)
2. Using the -WC option (See Table 88)
3. Using an option other than the -WC option (See Table 88)
4. Using the option file (See Table 88)
5. Using the environment variable (32) (Refer to “Setting Up Environment Variables” in Chapter 1)

[file-name]

Specify the name of the source file to be compiled. One or more file names can be specified.

Table 89. Compiler command options

Specification	**Specification Format**
Directory of library files	[-I directory-name]
Directory of screen and form descriptor files	[-m directory-name]
Directory of file descriptor files	[-f directory-name]
Whether to output a compiler listing file, and the output destination file name	[-P file-name]
Directory of a compiler listing file	[-dp directory-name]
Use of the TRACE function	[-Dr]
Use of the CHECK function	[-Dk]

Table 89. Compiler command options (cont.)

Specification	**Specification Format**
Use of an interactive debugger	[-Dt]
Directory of a debugging information file	[-dd directory-name]
Directory of an object file	[-do directory-name]
Compilation of main program	[-M]
Global optimization	[-O]
Option file	[-i file-name]
Directory of a SUBSCHEMA definition file (16)	[-A directory-name]
Compiler option	[-WC,"compiler-option"]

No space is needed before file-name or directory-name in the specification format.

[-I directory-name]

If the library function (COPY statement) is used, specify the directory of libraries. If libraries exist in more than one directory, specify the -I option more than once. The directories are searched for in the order they were specified.

A COPY statement with IN / OF specified is ignored.

Note: The -I option functions the same as compiler option LIB. For further information, refer to Appendix A, “Compiler Options.”

[-m directory-name]

If the COPY statement with IN / OF XMDLIB specified is used for fetching a record definition from screen and form descriptors, specify the directory of the screen and form descriptor file. If screen and form descriptor file exists in more than one directory, specify the -m option more than once. The directories are searched for in the order specified.

Note: The -m option functions the same as compiler option FORMLIB. For further information, refer to Appendix A, “Compiler Options.”

[-f directory-name]

If the COPY statement with IN / OF XFDLIB specified is used for fetching a record definition from file descriptor, specify the directory of the file descriptor file. If file descriptor file exists in more than one directory, specify the -f option more than once. The directories are searched for in the order specified.

Note: The -f option functions the same as the compiler option FILELIB. For further information, refer to Appendix A, “Compiler Options.”

[-P file-name]

Determine whether to output a compiler listing file, and specify the file name. If the file name is not specified, the file is output to the directory of the source file.

If the -P option is specified with the -dp option, the file linked with the directory specified in the -dp option is the final compilation list file. Do not specify an absolute path name for file-name.

[-dp directory-name]

A compiler listing file is usually under the directory of the source file. Specify the -dp option to define a new directory for the compilation list file.

Note: The -dp option functions the same as compiler option PRINT. For further information, refer to Appendix A, “Compiler Options.”

[-Dr]

Specify the -Dr option to use the TRACE function.

For details on using the TRACE function, refer to the “Fujitsu COBOL Debugging Guide.”

If the -Dr option is specified, processing for displaying trace information is incorporated into the object program. Consequently, execution performance lowers. When the debug function terminates, recompile the program without the -Dr option specified.

Note: The -Dr option functions the same as compiler option TRACE. For further information, refer to Appendix A, “Compiler Options.”

[-Dk]

To use the CHECK function, specify the -Dk option.

For details on using the CHECK function, refer to the “Fujitsu COBOL Debugging Guide.”

If the -Dk option is specified, processing for checking subscripts, indexes, and reference modification is incorporated into the object program. Consequently, execution performance is degraded. When the debug function terminates, recompile the program without the -Dk option specified.

Note: The -Dk option functions the same as compiler option CHECK. For further information, refer to Appendix A, “Compiler Options.”

[-Dt]

Specify the -Dt option to use an interactive debugger.

Note: The -Dt option functions the same as the compiler option TEST. For further information, refer to Appendix A, “Compiler Options” and the “Fujitsu COBOL Debugging Guide.”

[-dd directory-name]

Specify a directory of a debug information file. If the -dd option is not specified, the debug information file is created in the directory of the source file.

The -dd option is effective only when it is specified with the -Dt option or the TEST compiler option.

[-do directory-name]

Specify a directory for storing an object file. If the -do option is not specified, the object file is created in the directory of the source file.

[-M]

Specify the -M option to compile a COBOL program serving as the main program for execution.

Note: The -M option functions the same as compiler option MAIN. For further information, refer to Appendix A, “Compiler Options.”

[-O]

Specify the -O option to create a globally optimized object program.

Note: The -O option functions the same as compiler option OPTIMIZE. For further information, refer to Appendix A, “Compiler Options” and Appendix C, “Global Optimization.”

[-i file-name]

If a compiler option is specified in an option file (storing character strings as compiler options), specify the name of the option file. An option file can be created by selecting items from the list box using the WINCOB command.

The following example shows the WINCOB command specification format for creating an option file:

```
WINCOB -i path-name-of-option-file
```

[-A directory-name] (16)

Specify the directory of the SUBSCHEMA definition file which is specified in the SUBSCHEMA-NAME paragraph. If a SUBSCHEMA definition file exists in more than one directory, specify the -A option more than once. The directories are searched for in the order they were specified.

Note: The -A option functions the same as compiler option AIMLIB. For further information, refer to Appendix A, “Compiler Options.”

[-WC,"compiler-option"]

Specify a compiler option for the COBOL85 compiler. For compiler options and their specification formats, refer to Appendix A, “Compiler Options.”

More than one compiler option can be specified. The specified compiler options must be separated by a comma. If the same compiler option is specified more than once, the compiler option specified last is effective.

The priority level of compiler instructions is as follows:

- Compiler option specified by compiler instruction statement in source program.
- Compiler option specified by the -WC option of the COBOL32 command.
- Option specified for the COBOL32 command.
- Compiler option in the option file specified by -I option of COBOL32 command.

Do not specify the following options in the -WC option:

```
AIMLIB, FILEEXT, FILELIB, FORMEXT, FORMLIB, LIB, OBJECT, PRINT, TEST
```

Linker Commands

Use the linker commands to link a program and generate an executable program.

For details on generated files and required files for link command execution, refer to “Resources Required for Linking” in Chapter 4. For linker command specified location and basic usage, refer to “Using Commands to Link” in Chapter 4.

The following are linker commands:

- (16): LINK command, LIB command, IMPLIB command
- (32): LINK command, LIB command

LINK Command (16)

This section explains the LINK command for Windows 3.1.

Input Format

Command	Operand
LINK (16)	[option-list] file-name-list

Operands

At least one space is needed between the command name and each operand. Items enclosed in "[" and "]" can be omitted. An absolute or relative path name can be specified for a directory name and file name in the explanation below.

[option-list]

Specify LINK command options. See Table 89 for the specification.

[file-name-list]

File names can be specified following the option list. The following shows the specification format:

```
object-file-name,[output-file-name],[map-file-name],[library-name],
module-definition-file
```

[object-file-name]

Specify one or more object files to be linked. The specified object file names must be separated by a space or plus sign (+). The default file name extension is OBJ.

[output-file-name]

Specify an executable file or DLL. If an executable file is specified, the default file name extent is EXE. If DLL is specified, the default file name extension is DLL.

[map-file-name]

Specify a map file. The default file name extension is MAP.

[library-name]

Specify a standard (object code) library or import library. If a standard object library is specified, only the object module required for external reference is linked.

[module-definition-file-name]

Specify a module definition file. The default file name extension is DEF.

Table 90. LINK command options (16)

Specification	**Specification Format**
Prepares for debugger option	[/CO]
Restricts the linker to use only standard libraries	[/NOD]
Disables the library extended dictionary.	[/NOE]
Outputs a list of public symbols to the map file.	[/M]

For details on the above options, refer to “Setting Linker Options” in Chapter 4.

LIB Command (16)

This section explains the LIB command for Windows 3.1.

The LIB command does the following:

- Creates library files
- Deletes, adds, and replaces library modules
- Copies and moves one module from the library to another file
- Outputs library modules to the public symbol list

For details on generated files and files for LIB command execution, refer to “Resources Required for Linking” in Chapter 4.

Input Format

Command	Operand
LIB (16)	library-name [option-list][command][,[list-output-destination] [, [new-library-name]]][;]

Operands

At least one space is needed between the command name and each operand. Items enclosed in "[" and "]" can be omitted. An absolute or relative path name can be specified for a directory name and file name in the explanation below.

[library-name]

Specify the library to be processed. The default file name extension is LIB.

[option-list]

Specify the information to be posted to the LIB command. For the specification, see Table 90.

[command]

Specify the file to be processed following the command symbol. By specifying the file name, the contents of the library specified for library-name can be changed, a new library can be created, and one module can be copied or moved from the library to another file. For the command symbols and the files to be specified, see Table 91.

[list-output-destination]

The following information is output to the file specified for list-output-destination:

- Public symbol lists in the library (in alphabetical order)
- Module lists in the library

[new-library-name]

Specify new-library-name to save a library as another library when the library contents are changed. If a new library is not specified, the contents of the current library are updated. The original contents are saved in a file with which the current library name extension is changed to BAK.

Table 91. LIB command options (16)

Specification	**Specification Format**
Changing or setting a library page size	[/PAGESIZE:n]
Whether to distinguish uppercase letters from lowercase letters	[/N | /I]

[/PAGESIZE:n]

For n, specify a value (16 to 32, or 768) raised to the power of 2. The default page size of a new library is 16 bytes.

[/N | /I]

Specify /N to distinguish uppercase letters from lowercase letters. Otherwise, specify /I. The default is /I.

If a library is created with /N specified, a mark is given to the library indicating creation with /N specified. If any of the libraries to be linked was created with /N specified, the output library is treated as a library created with /N specified.

Table 92. Command symbols

Symbol	Explanation	File Specified
+	Addition of module	Specify name of the object file to be added. The object file name extension OBJ can be omitted.
	Linkage of library	Specify the library file to be linked. The library file name extension LIB must be specified.
-	Deletion of module	Specify name of the object module to be deleted. A path or extension cannot be assigned to the object module name.
-+	Replacement of module	Specify name of the object module to be replaced. A path or extension cannot be assigned to the object module name. To replace a module, the object file name extension must be OBJ and the file must be in the current directory.
*	Module copy to another file	Specify name of the object module to be copied. The copy destination file is created in the current directory. The name of the copy destination file is the object module name with extension OBJ added.
-*	Module move to another file	Specify name of the object module to be moved. The move destination file is created in the current directory. The name of the move destination file is the object module name with extension OBJ added.

Press the CTRL and C keys together to cancel LIB command processing.

IMPLIB Command (16)

This section explains the IMPLIB command of Windows 3.1.

Use the IMPLIB command to create import libraries.

For import libraries, refer to “Resources Required for Linking” in Chapter 4.

Input Format

Command	Operand
IMPLIB (16)	import-library-name module-definition-file-name | dll-name

Operands

At least one space is required between the command name and the operand.

import-library-name

Specify name of the import library to be created.

module-definition-file-name or dll-name

Specify a module definition file for a DLL or the DLL name. An import library is created according to the contents of the module definition file for a DLL specified here.

LINK Command (32)

This section explains the LINK command for the Windows 95 and Windows NT version.

Input Format

Command	Operand
LINK (32)	file-name-list [option-list]

Operands

At least one space is needed between the command name and each operand. Items enclosed in "[" and "]" can be omitted. An absolute or relative path name can be specified for a directory name and file name in the explanation below.

[file-name-list]

The following shows the specification format:

```
object-file-name library-name
```

[object-file-name]

Specify one or more object files to be linked. The specified object file names must be separated by a blank or tab.

[library-name]

Specify a standard (object code) library, import library, or export file. If a standard object library is specified, only the object module required for external reference is linked.

The following files must be specified:

- COBOL import library
- LIBC.LIB,KERNEL32.LIB,USER32.LIB

[option-list]

Specify LINK command options. For the specification, see Table 93.

Table 93. LINK command options (32)

Specification	**Specification Format**
Default base address of the executable file or DLL	/BASE:address
Preparing for use of the COBOL85 interactive debugger	/DEBUG /DEBUGTYPE:COFF
(For linkage with C language using Visual C++)	/DEBUGTYPE:BOTH
Creating a dynamic link library (DLL)	/DLL
Program execution entry point	/ENTRY:symbol
Name of the output main file	/OUT:filename

LIB Command (32)

This section explains the LIB command for Windows 95 and Windows NT.

Use the LIB command to create a standard library or import library.

For details on generated files and files for LIB command execution, refer to “Resources Required for Linking” in Chapter 4.

Input Format When Creating a Library File

Command	**Operand**
LIB (32)	file-name-list [option-list]

Operands

At least one space is needed between the command name and each operand. Items enclosed in "[" and "]" can be omitted. An absolute or relative path name can be specified for a directory name and file name in the explanation below.

[file-name-list]

Specify one or more object files. Existing libraries can also be specified.

[option-list]

Specify the information to be posted to the LIB command. For the specification, see Table 93.

Input Format When Creating an Import Library

Command	Operand
LIB (32)	/DEF:module-definition-file-name /MACHINE:IX86 /OUT:import-library-name [option-list] object-file-name-list

Operands

At least one space is needed between the command name and each operand. Items enclosed in "[" and "]" can be omitted. An absolute or relative path name can be specified for a directory name and file name in the explanation below.

[/DEF:]

The /DEF option specifies a module definition file for creating an import library. Specify the module definition file following the colon (:).

[/MACHINE:]

The /MACHINE option specifies the type of the CPU that executes the program.

[/OUT:]

The /OUT option specifies the import library to be created.

[option-list]

Specify the information to be posted to the LIB command. For the specification, see Table 93.

[object-file-name-list]

Specify the object file for creating an import library.

Table 94. LIB command options (32)

Specification	Specification Format
Copying a specified object member from a specified library	/EXTRACT:object-member-name
Deleting a specified object member from a specified library	/REMOVE:object-member-name

Press the CTRL and C keys together to cancel LIB command processing.

When an import library is created, an export file is created under the import library name with the extension LIB changed to EXP.

Appendix K. FCB Control Statement

The following shows the specification format of the FCB control statement:

```
[LPI ((line-spacing,line-count) [/line-spacing,line-count)]...)]
  [,CHm(line-number[,line-number]...) [,CHm(line-number[,line-number]...)]...]


                 ⎧ paper-length ⎫
    ⎧ SIZE (     ⎨              ⎬ )                    ⎫
    ⎪            ⎩ 110          ⎭                      ⎪
    ⎪                                                  ⎪
[   ⎨              ⎧ A3 ⎫                              ⎬  ]
    ⎪              ⎪ A4 ⎪        ⎧ PORT ⎫              ⎪
    ⎪              ⎪ B4 ⎪        ⎪      ⎪              ⎪
    ⎩ FORM (       ⎨ A5 ⎬ ) (,   ⎨      ⎬ ] )          ⎭
                   ⎪ B5 ⎪        ⎩ LAND ⎭
                   ⎩ LT ⎭
```

- LPI information

  LPI information can be sequentially specified in format of (line-spacing, line-count) from the top of the page. Specify 6, 8, or 12 (lpi) for line spacing.

- CH information

  Specify a line number for CHANNEL-01 to CHANNEL-12. Channel number 1 identifies the first line that can be printed.

- SIZE information

  Specify the length of paper in units of 1/10 inch. The default is 110 (11 inches).

- FORM information

  Specify paper in fixed size. If a fixed size is specified, the length of the paper is uniquely determined for PORT (portrait) or LAND (landscape).

Appendix L. Indexed File Recovery

This appendix explains the indexed file recovery function and indexed file simple recovery function.

Indexed File Recovery Function

Function

The indexed file recovery function refreshes normal sections from the beginning of the file, and outputs abnormal sections to another file. This allows you to recover an unusable indexed file (unusable because it was not closed normally).

The indexed file recovery function works the same as the Recovery command of the COBOL85 FILE UTILITY.

If an indexed file in the unusable state is opened, 90 is returned as the I-O status.

Coding Format

```
#include "f1bcfutc.h"  /* function-declaration (16) */
#include "f3bifutc.h"  /* function-declaration (32) */
 signed long int CFURCOV( char far*ixdfilename,
     char far *blkdatname,
     char far *message);
```

- [ixdfilename]

  Name of the file to be recovered

- [blkdatname]

  Name of the file containing data which could not be recovered

- [message]

  Area storing the execution result (message) of the indexed file recovery function

CFURCOV Function

The following are explanations of the parameters of this function.

[ixdfilename]

Specify the address of the area storing the name (character string) of the indexed file to be recovered. The character string must end with a NULL (0x00) or blank (0x20).

[blkdatname]

Specify the address of the area storing the name (character string) of the file where unrecoverable records are written. The character string must end with a NULL (0x00) or blank (0x20). Specify 0 if a file for writing unrecoverable records is unnecessary.

[message]

Specify the address of the area storing a message that indicates the execution result of the indexed file recovery function. Allocate the storage area to the calling source (512 bytes required). Specify 0 if a message is unnecessary.

Return Value

The code corresponding to the message is returned as the execution result of the indexed file recovery function. See Table 94 for codes and messages.

The following is an example how to use the indexed file recovery function (with Windows 95 and Windows NT) (32):

```
#include "f3bifutc.h"
void callcobfrcov(void)
 {
    char ixdfilename[512] = "c:\\ixdfile\0";
    char blkdatname[512] = "c:\\blkdat\0";
    char message[512];
    CFURCOV(ixdfilename,blkdatname,message);
       return;
 }
```

Indexed File Simple Recovery Function

Function

Resets flags in an indexed file in unusable state so that the file is enabled. Unlike the indexed file recovery function, the indexed file simple recovery function does not recover abnormal sections in the indexed file. Consequently, access to the indexed file after flag reset can result in an error because of conflicting data.

If an indexed file in the unusable state is opened, 90 is returned as the I-O status.

Coding Format

```
#include "f1bcfutc.h"  /* function-declaration (16) */
#include "f3bifutc.h"  /* function-declaration (32) */
 signed long int CFURCOVS( char far*ixdfilename,
     char far *message);
```

- ixdfilename

  Name of the file to be recovered

- message

  Area storing the execution result (message) of the indexed file simple recovery function

CFURCOVS Function

The following are explanations of the parameters of this function.

ixdfilename

Specify the address of the area storing the name (character string) of the indexed file to be recovered. The character string must end with a NULL (0x00) or blank (0x20).

message

Specify the address of the area storing a message that indicates the execution result of the indexed file simple recovery function. Allocate the storage area to the calling source (512 bytes required). Specify 0 if a message is unnecessary.

Return Value

The code corresponding to the message is returned as the execution result of the indexed file simple recovery function. For codes and messages, see Table 94.

The following is an example of how to use the indexed file simple recovery function (with Windows NT) (32):

```
#include "f3bifutc.h"
void callcobfrcovs(void)
 {
    char ixdfilename[512] = "c:\\ixdfile\0";
    char message[512];
    CFURCOVS(ixdfilename,message);
    return;
 }
```

Notes

At creation, include the following files for calling from a C program:

- (16): f1bcfutc.h
- (32): f3bifutc.h

At compilation, for calling from a C program, refer to “Compiling Programs” in Chapter 10.

At linkage, use the LINK command to link the following files:

- (16): f1bcfutc.lib
- (32): f3bifutc.lib

For calling from a C program, refer to “Linking Programs” in Chapter 10.

At execution, set up the following environment:

- Before starting Windows, execute the SHARE command. (16)
- Specify the directory storing the following files for the PATH environment variable:
  - (16): f1bcfutc.dll, f1bcfute.dll, f1bcfrm.dll
  - (32): f3bifutc.dll, f3bifute.dll, f3bifrm.dll

Examples of Calling from COBOL

This section shows examples of programs that perform recovery processing according to the open status.

The following example calls the indexed file recovery function when a file for storing unrecoverable data and a message are unnecessary:

```
00010 IDENTIFICATION DIVISION.
00020   PROGRAM-ID   WORK1.
00030 ENVIRONMENT DIVISION.
00040 CONFIGURATION SECTION.
00050 SPECIAL NAMES.
00060    ENVIRONMENT-NAME ENV-NAME.
00070    ENVIRONMENT-VALUE ENV-VALUE.
00080 INPUT-OUTPUT SECTION.
00090 FILE CONTROL.
00100   SELECT IXDFILE ASSIGN TO FILE1
00110    ORGANIZATION IS INDEXED
00120    RECORD KEY IS IXDFILE-RECKEY
00130    FILE STATUS IS FSDATA
00140 DATA DIVISION.
00150 FILE SECTION
00160 FD IXDFILE.
00170 01 IXDFILE-REC.
00180    03 IXDFILE-RECKEY PIC X(8).
00190    03 IXDFILE-DATA PIC X(20).
00200 WORKING STORAGE SECTION
00210 77 IXDFILE-NAME PIC X(512).
00220 77 BLKFILE-NON PIC S9(9) COMP-5 VALUE 0.
00230 77 MESSAGE-NON PIC S9(9) COMP-5 VALUE 0.
00240 77 FSDATA PIC XX.
00250 77 OPEN-FSDATA PIC XX.
00260 PROCEDURE DIVISION.
00270    OPEN I-O IXDFILE.
00280    MOVE FSDATA TO OPEN-FSDATA.
00290    CLOSE IXDFILE.
00300      EVALUATE OPEN-FSDATA
00310        WHEN '00'
00320         GO TO WORK-START
00330        WHEN '90'
00340*        Fetches the file name
00350        MOVE ALL SPACE TO IXDFILE-NAME
00360        DISPLAY 'FILE1' UPON ENV-NAME
00370        ACCEPT IXDFILE-NAME FROM ENV-VALUE
00380*        Calls the indexed file recovery function
00390        CALL 'CFURCOV' USING BY REFERENCE IXDFILE-NAME
00400                             BY VALUE BLKFILE-NON
00410                             BY VALUE MESSAGE-NON
00420      EVALUATE TRUE
00430        WHEN PROGRAM-STATUS<=1
00440         GO TO WORK-START
00450        WHEN OTHER
00460         GO TO WORK-STOP
00470      END-EVALUATE
00480     WHEN OTHER
00490      GO TO WORK-STOP
00500    END-EVALUATE.
00510 WORK-START.
00520   OPEN I-O IXDFILE
            :
00580   CLOSE IXDFILE.
00590 WORK-STOP.
00600 EXIT PROGRAM.
```

The following example calls the indexed file recovery function when a file for storing unrecoverable data and a message are necessary:

```
00010 IDENTIFICATION DIVISION.
00020   PROGRAM-ID   WORK2.
00030 ENVIRONMENT DIVISION.
00040 CONFIGURATION SECTION.
00050 SPECIAL NAMES.
00060    ENVIRONMENT-NAME ENV-NAME
00070    ENVIRONMENT-VALUE ENV-VALUE.
00080 INPUT-OUTPUT SECTION.
00090 FILE CONTROL.
00100   SELECT IXDFILE ASSIGN TO FILE1 ORGANIZATION IS INDEXED
00110    RECORD KEY IS IXDFILE-RECKEY FILE STATUS IS FSDATA.
00120 DATA DIVISION.
00130 FILE SECTION.
00140 FD IXDFILE.
00150 01 IXDFILE-REC.
00160    03 IXDFILE-RECKEY PIC X(8).
00170    03 IXDFILE-DATA PIC X(20).
00180 WORKING-STORAGE SECTION
00190 77 IXDFILE-NAME PIC X(512).
00200 77 BLKFILE-NAME PIC X(11) VALUE 'C:\BLKDAT'.
00210 77 MESSAGE-AREA PIC X(512.
00220 77 FSDATA PIC XX.
00230 77 OPEN-FSDATA PIC XX.
00240 PROCEDURE DIVISION.
00250    OPEN I-O IXDFILE.
00260    MOVE FSDATA TO OPEN-FSDATA.
00270    CLOSE IXDFILE.
00280      EVALUATE OPEN-FSDATA.
00290        WHEN '00'
00300         GO TO WORK-START
00310        WHEN '90'
00320*        Fetches the file name
00330        MOVE ALL SPACE TO IXDFILE-NAME
00340        DISPLAY 'FILE1' UPON ENV-NAME
00350        ACCEPT IXDFILE-NAME FROM ENV-VALUE
00360*         Calls the indexed file recovery function
00370        CALL 'CFURCOV' USING BY REFERENCE IXDFILE-NAME
00380                             BY REFERENCE BLKFILE-NAME
00390                             BY REFERENCE MESSAGE-AREA
00400      EVALUATE TRUE
00410        WHEN PROGRAM-STATUS<=1
00420         GO TO WORK-START
00430        WHEN OTHER
00440         GO TO WORK-STOP
00450      END-EVALUATE
00460     WHEN-OTHER
00470      GO TO WORK-STOP
00480    END EVALUATE.
00490 WORK-START.
00500   OPEN I-O IXDFILE.
              :
00560   CLOSE IXDFILE.
00570 WORK-STOP.
00580  EXIT PROGRAM.
```

Codes and Messages

Table 95. Codes and messages returned by the indexed file recovery function (and simple recovery function)

Code(Decimal)	Message
0	n Records were restored. (*1)
1	There was n records which were not able to be restored. (*2)
10	Recovery file not found.
11	Recovery file not the indexed file.
12	Could not access recovery file.
13	Sequential file exists. (*2)
14	Other process accessing the file.
15	Other process recovering the file.
16	Insufficient disk space. (*2)
128	Insufficient memory space.
129	Not DOS 3.1 or later.
130	Not share mode.(DOS SHARE command unload.)
131	Record in recovery file not found. (*2)
132	Contradiction in the file information.

*1 The simple recovery function does not return the message.
*2 The simple recovery function does not return the code and message.

Appendix M. Using Other File Systems

This appendix explains how to use COBOL85 with the following file systems:

- Btrieve
- RDM

Btrieve File

A Btrieve file can also be used just like a file system as a sequential file or indexed file supported by COBOL85.

The Btrieve file access function is supported within the scope of functions supported by the COBOL file system. Btrieve-specific functions, however, are not supported.

This section explains how to use the Btrieve file with COBOL85.

For details on how to use the Btrieve file, refer to Btrieve manuals.

Specifying File Environment

Specify a file-reference-identifier in the ASSIGN clause of the file control entry to determine whether to use the COBOL file system or Btrieve Record Manager.

If a file-identifier literal is specified as the file-reference-identifier, specify the file-identifier literal in the following format:

$$[\text{path-name}]\ \text{file-name}\left\{\begin{array}{l}\text{no-specification}\\ \\ \text{,BTRV}\end{array}\right\}$$

FILE-NAME

Specify the file name or path name of the input-output target file.

NO-SPECIFICATION

The COBOL file system is used.

BTRV

The Btrieve Record Manager is used.

If a data name is specified as the file-reference-identifier, use the file-identifier literal format to specify the file where the data name is specified.

If the character string DISK is specified as the file-reference-identifier, the Btrieve Record Manager cannot be used. The COBOL file system is used.

If a file-identifier is specified as the file-reference-identifier, specify the same data as specified in the file-identifier literal format for the environment variable that defines the file-identifier as the environment variable name.

The environment variable specification format is:

$$\text{environment-variable} = [\text{path-name}]\ \text{file-name}\left\{\begin{array}{l}\text{no-specification}\\ \\ \text{,BTRV}\end{array}\right\}$$

Function Outline

For sequential files, indexed files, COBOL syntax, and basic usage, refer to the “COBOL85 Reference Manual.”

If the function is more enhanced than the COBOL file system, the key item attributes in index files can be Alphanumeric characters, National characters, External decimal numbers, Internal decimal numbers and Binary numbers.

Unlike the COBOL file system, the following restrictions are placed on use of the Btrieve Record Manager:

- A relative file cannot be specified
- A data item which is part of a record key cannot be specified in the START statement
- DISK cannot be specified as the file-reference-identifier
- FILE STATUS = 02 cannot be returned

The following table shows the differences between limitations for the COBOL file system (COBOL) and the Btrieve record manager (Btrieve).

Table 96. COBOL File System and Btrieve Limits

Description	COBOL	Btrieve
Maximum record length	32,760	4,090 (fixed length) 32,760 (variable length)
Maximum number of data items specified in the RECORD KEY clause	254	24 (16bits) 119 (32bits)
Maximum length of all data items specified in the RECORD KEY clause	254	255
Maximum number of data items specified in the ALTERNATE RECORD KEY clause	254	23 (16bits) 118(32bits) *1
Maximum length of all data items specified in the ALTERNATE RECORD KEY clause	254	255

*1 The total of the number of data items specified in the RECORD KEY clause and that specified in the ALTERNATE RECORD KEY clause can be up to N (24 for 16bits and 119 for 32bits). This value decreases as the number of data items specified in the RECORD KEY clause increases.

NOTES

A record key must be within an area of 1,024 bytes from the beginning of the record (16).

A record key must be within an area of 4,088 bytes from the beginning of the record (32).

The length of a fixed length record and the minimum length of a variable length record must be in the range 4 to 4,088 bytes. If a variable length record is used, the Btrieve Record Manager requires 4 bytes for the overhead.

If a duplicated key (per item) is used, the Btrieve Record Manager requires 8 bytes for the overhead. The length of a fixed length record and the minimum length of a variable length record are decreased by 4 or 8 bytes.

If a record with a file position indicator defined is deleted by the DELETE statement with random access, the file position indicator is undefined.

Do not use the extension PRE for a file name to be accessed in an application. In order to use the pre-image function, do not access multiple Btrieve files having identical names with extensions that are unique within an application. The pre-image function fails to define the pre-image file for files when multiple Btrieve files are accessed.(16)

Up to 18 files for 16bit or 20 files for 32bit can be opened in the application at the same time.

To use NetWare Btrieve and client-format Btrieve (supported by this product) together, rename client-format Btrieve WBTRCALL.DLL to WBTRLOCL.DLL. For details, refer to the Btrieve manuals.(16)

A Btrieve file created under NetWare 4.1J cannot be accessed.

To enable access to Btrieve files, specify "yes" to "Create Btrieve Files in Pre v6.x Format" in the Btrieve setup utility. For details, refer to the Btrieve manuals.(16)

This product contains Btrieve Workstation Engine for Windows NT/Windows 95 v6.15 (32) and Btrieve Workstation Engine for Windows NT/Windows 95 v6.10 (16).

For ERFLD=xx and FDBK=xx status codes (xx: hexadecimal), refer to the “BTRIEVE Programmer’s Manual” and the “Btrieve for Windows NT/Windows 95 Installation and Operation” by Btrieve Technologies, Inc..

External Decimal Data Form Conversion (32)

The internal form of this NUMERIC differs from COBOL85 and is called 88consortium. It is equivalent to the data type of an external decimal with the sign which is not SEPARATE in COBOL from the NUMERIC type of Btrieve file (Refer to the “BTRIEVE Programmer’s Manual” for an internal form of the NUMERIC data type for Btrieve).

Therefore, a Btrieve file should be used and the data converted into each form as either an input or output data item before writing and after reading when an external decimal item with the not SEPARATE sign is used.

This conversion can be done easily in COBOL85 by using a CALL statement.

Description Form (32)

Conversion from 88consortium to COBOL85 is as follows:

```
Call "#DEC88TOFJ" USING [BY REFERENCE] name
```

Conversion from COBOL85 to 88consortium is as follows:

```
Call "#DECFJTO88" USING [BY REFERENCE] name
```

Function Outline (32)

At "#DEC88TOFJ", the program name which is called by the CALL statement, will convert an internal form of an external decimal item with the without all SEPARATE specification included in the name or the name from 88consortium into an internal form of COBOL85.

At "#DECFJTO88", the program name which is called by the CALL statement, will convert an internal form of an external decimal item with the without all SEPARATE specification included in the name or the name from COBOL85 into an internal form of 88consortium.

If the item is not an external decimal item, using the without SEPARATE specification will not perform a conversion.

When the name is a group item and the following items are included in the subordinate items, the item will not be converted:

- Items other than external decimal items with the sign using the without SEPARATE specification.

- Items subordinate to items which specify the REDEFINES clause as well as the item itself.
- Items which specify the RENAMES clause.

In the following cases, the execution results are not guaranteed.

- The program name called by a CALL statement is specified by the name and "#DEC88TOFJ" or "DECFJTO88" is specified by the name.
- "#DEC88TOFJ" or "DECFJTO88" is called by a form different from the above-mentioned description form (Two or more names are specified for USING phrase, BY CONTENT phrase is specified.).
- The name is a group item and the item uses the OCCURS clause with DEPENDING specification in the subordinate item.
- When data items are defined in the constant paragraph.
- Data items that refer the part, applies by the name, and is done.

Notes (32)

Immediately before passing a record to a Btrieve file or immediately after having received one, do this function call. The external decimal number of 88consortium form of conversion by immediately after having input from Btrieve file and this function cannot be treated by COBOL.

“It is not possible to be treated by COBOL” means that the results of ‘operates’, ‘compares’ or ‘post as external decimal item’ are not guaranteed. The value of the external decimal of the 88consortium form is guaranteed only in records containing the external decimal item of the 88consortium form when the group item is posted to other records.

The sorting and merging function cannot be used for Btrieve files.

Incorrect function calls will be called if an external reference error of "#DECFJTO88" or "#DEC88TOFJ" occurs when linking.

Example (32)

```
        :
FILE-CONTROL.
     SELECT file     ASSIGN TO "filename.BTRV"
                     ORGANIZATION IS INDEXED
                     ACCESS MODE SEQUENTIAL
                     RECORD KEY IS mainkey OF record.
DATA DIVISION.
FD  file.
01  record.
  02  mainkey      PIC 9(4).
  02  data1        PIC S9(8).
  02  data2        PIC X(50).
WORKING-STORAGE SECTION.
01  workarea.
  02  mainkey      PIC9(4).
  02  data1        PICS9(8).
  02  data2        PIC X(50).
PROCEDURE DIVISION.
        :
*Writing in Btrieve file.
    OPEN OUTPUT file.
    CALL "#DECFJTO88" USING workarea.                              ----- *1
    WRITE record FROM workarea.
    CLOSE file.
        :
*Reading from Btrieve file.
    OPEN INPUT file.
    MOVE 6 TO mainkey OF record.
    START file.
    READ file INTO workarea.
    CALL "#DEC88TOFJ" USING workarea.                              ----- *2
    DISPLAY "data1 :" data1 OF workarea.
    CLOSE file.
        :
```

*1: External decimal data which becomes the written data before writing in Btrieve file with all signs in “workarea” which are not SEPARATE is converted from COBOL85 form into 88consortium form.

*2 : External decimal data which before reading after reading of the record from Btrieve file with all signs in “workarea” which are not SEPARATE is converted from 88consortium form into COBOL85 form.

RDM File

An RDM file can be used just like a file system as a sequential file, relative file, or indexed file supported by COBOL85. This section explains how to use the RDM file with COBOL85.

Specifying File Environment

A file-reference-identifier specified in the ASSIGN clause of the file control entry determines whether to use the COBOL file system or RDM file system.

If a file-identifier literal is specified as the file-reference-identifier, specify the file-identifier literal in the following format:

```
                            ┌ no-specification ┐
[path-name] file-name      {                    }
                            └ ,RDM             ┘
```

FILE-NAME

Specify the file name or path name of the input-output target file.

NO-SPECIFICATION

The COBOL file system is used.

RDM

The RDM file system is used.

If a data name is specified as the file-reference-identifier, use the file-identifier literal format to specify the file with the data name specified.

If the character string DISK is specified as the file-reference-identifier, the RDM file system cannot be used. The COBOL file system is used.

If a file-identifier is specified as the file-reference-identifier, specify the same data as specified in the file-identifier literal format for the environment variable that defines the file-identifier as the environment variable name.

The environment variable specification format is:

```
                                                   { no-specification }
environment-variable = [path-name] file-name       {                  }
                                                   { ,RDM             }
```

To use the extended function of the RDM file, specify the following environment variable in addition to the above specification:

```
CRDB_XLIB_OVR = file-name,RDM,character-stringindicating-
extended-function [:...]
```

Function Outline

If the function is more enhanced than COBOL file system functions, check the key item attributes of indexed files (alphanumeric data item, national item, external decimal item, internal decimal item, binary item).

Unlike the COBOL file system, the following restrictions are placed on use of then RDM file system:

- DISK cannot be specified as the file-reference-identifier
- File sharing and exclusive use of records cannot be specified in the program
- Variable length records cannot be used
- An optional file can be used in only input open mode. (No file creation)

The following table shows the differences between quantitative restrictions for the COBOL file system (COBOL) and RDM file system (RDM).

Table 97. COBOL File System and RDM File System Limits

Description	**COBOL**	**RDM**
Maximum number of data items specified in the RECORD KEY clause	254	128
Maximum length of all data items specified in the RECORD KEY clause	254	255
Maximum number of data items specified in the ALTERNATE RECORD KEY clause	254 (*1)	128
Maximum length of all data items specified in the ALTERNATE RECORD KEY clause	254	255

*1 The total of the number of data items specified in the RECORD KEY clause and that specified in the ALTERNATE RECORD KEY clause can be up to 255. This value decreases as the number of data items specified in the RECORD KEY clause increases.

If MANUAL is specified in the LOCK MODE clause, records cannot be locked.

Execute "Relative File Creation" to use a relative file with a COBOL program. Execute the OPEN statement with OUTPUT specified to execute "Relative File Creation."

An error message is written or I-O status occurs indicating that the record does not exist (run-time error JMP0324I-I/U or FS=23) if the OPEN statement with I-O or EXTEND specified is used to open a relative file without "Relative File Creation." The WRITE statement is then executed.

If the disk size is insufficient, the following error message is output when the CLOSE statement is executed:

JMP0310I-I/U CLOSE ERROR. FILE='file-name or access-name'. 'ERFLD=1C'

Appendix N. A COBOL-Supported Subroutine

This appendix explains a COBOL-supported subroutine.

Subroutine for Receiving the Window Handle

This subroutine receives the FORM RTS window handle when linked with the presentation file edit function operating under Windows.

Specification Method

Data definition with Windows 3.1 (16):

```
01  data-name-1    PIC X(8).
01  data-name-2    PIC S9(4) COMP-5.
```

With Windows 95 and Windows NT (32):

```
 01  data-name-1    PIC X(8).
 01  data-name-2    PIC S9(9) COMP-5.
```

Specification of the CALL statement:

```
CALL"JMPBGWDH" WITH C LINKAGE USING data-name-1
                                    data-name-2
```

Interface

For data-name-1, specify the file-identifier of the presentation file for receiving the window handle.

For data-name-2, specify the area for storing the window handle received by the subroutine.

The presentation file specified for data-name-1 must be specified for DSP in the SYMBOLIC DESTINATION clause and be opened.

Message

A message is written if:

- The file is not the presentation file
- The file does not exist

Return Codes

Special register PROGRAM-STATUS is used to receive return codes from the subroutine.

- Return code 0: The window handle was received normally.
- Return code -1: The window handle could not be received.

Notes

This subroutine cannot be used under AP/EFW, or with the presentation file module test function. (16)

The file-identifier specified for data-name-1 must correspond to a single file in the run unit. If the file-identifier is associated with multiple files, the execution result is not guaranteed.

When the DLOAD option is specified at compilation of the COBOL program that calls this subroutine, the following entry information must be specified as shown below.

(16)

```
       :
[program-name.ENTRY]
 JMPBGWDH=F1BCILNG.DLL
       :
```

(32)

```
       :
[program-name.ENTRY]
 JMPBGWDH=F3BIILNG.DLL
       :
```

Refer to “Entry Information” in Chapter 5 for details on specifying entry information.

Using the Subroutine from an Application

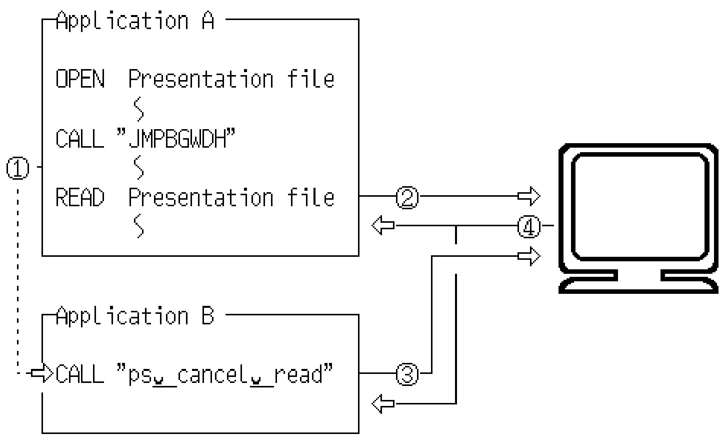


Figure 145. A sample application using the window handle subroutine

*1: The window handle acquired according to the window handle acquisition subroutine is notified of application B (Delivery of the data between processes).

*2: READ statement of the display file is executed and enters the state of the waiting for input from the screen.

*3: The window handle notified from application A is specified for the function of FORM RTS and the input interruption of the screen is ordered.

*4: The input completion is notified to interrupt the input waiting for the screen and to have been done to application A compulsorily. The input interruption is notified of application B.

9E is notified of the I/O state value of READ sentence of the display file and 9E5A is notified of detailed information.

Subroutine for Receiving the Instance Handle

The instance handle of the windows COBOL application can be received.

Specification Method

Data definition:

```
01  data-name
02 data-name-1    PIC S9(9)    COMP-5.
02 FILLER         PIC X(12).
```

Specification of the CALL statement:

```
CALL "JMPBWINS" WITH C LINKAGE USING data-name.
```

Interface

For data-name, specify the area for storing the instance handle received by the subroutine. The instance handle is stored in data-name.

Notes

When the DLOAD option is specified for a compilation of the COBOL program by which this subroutine is called, the following entry information will be needed. Refer to Chapter 5, “Entry Information” for the method of specifying entry information.

```
[program-name.ENTRY]
JMPBWINS=F3BIPRCT.DLL
```

Index