Transitioning to ACUCOBOL-GT

Version 8.1.3

Micro Focus
9920 Pacific Heights Blvd.
San Diego, CA 92121
858.795.1900

E-01-UG-100501-Transitioning-8.1.3

Contents

1 Introduction

Key Topics

1.1 Transitioning Your COBOL

Businesses convert their mission-critical applications to ACUCOBOL-GT for many reasons:

- Some are forced to migrate off their current platform, but they don't want to lose their investment in COBOL.
- Others are fed up with paying to maintain an obsolete platform, and they want to move to open systems.
- Others feel that their COBOL no longer meets their needs. They may have issues with their vendor. They may desire modern features.

Whatever your reason, converting your programs to ACUCOBOL-GT will let you breathe new life into your COBOL code without losing functionality or assuming the cost of rewriting.

To smooth your transition, ACUCOBOL-GT provides compatibility with many other COBOL dialects, RM COBOL, HP COBOL, VS COBOL, ICOBOL, IBM Mainframe COBOL, and more. We offer:

- Compatibility switches
- Conversion tools
- Professional Services Organization consultation
- Migration partners that offer everything from toolkits to turnkey solutions

Once your applications are in ACUCOBOL-GT, you can take advantage of our many COBOL modernization and interoperability features to help you reach your business objectives. And you can re-host from one hardware platform to another—all without rewriting your proven business applications.

1.2 Organization

This guide, *Transitioning to ACUCOBOL-GT,* offers information on how to convert from several of the most popular COBOLs to ACUCOBOL-GT. It is organized as follows:

- Chapter 1: Introduction
- Chapter 2: RM COBOL Conversions
- Chapter 3: ICOBOL Conversions
- Chapter 4: HP COBOL Conversions
- Chapter 5: IBM DOS/VS COBOL Conversions

1.3 Technical Services

For the latest information on contacting customer care support services go to:

http://www.microfocus.com/about/contact

For worldwide technical support information, please visit:

http://supportline.microfocus.com/xmlloader.asp?type=home

2 RM/COBOL Conversions

Key Topics

2.1 Compile-Time Options

This chapter provides information about how to convert programs written for the RM/COBOL compiler to ACUCOBOL-GT. This is particularly directed at users who are new to ACUCOBOL-GT and are not familiar with its many options. The information contained here is specifically directed to programs written for version 2.X of the RM/COBOL 1974 compiler. Most of this information is also applicable to versions 1.5 and 1.6 of that compiler, as well as to RM/COBOL-85. See **section 2.3.1** for information about converting RM/COBOL data files.

Several compile-time options should be used when you are compiling a program written for RM/COBOL with ACUCOBOL-GT. You should always use the "-Cr" flag to indicate RM/COBOL compatibility mode. You will also probably want to use the "-R8vais" flags to cause certain reserved words to be treated as user-defined words. This will cause all words not reserved by RM/COBOL to be treated as user-defined words. If you are converting programs from RM/COBOL-85, then you should use "-Rvais" instead.

Many applications will also need to use the "-Ds" flag. This flag specifies that the PICTURE element "S" for a USAGE DISPLAY item is treated as a separate character. This is the same as implying the SIGN IS TRAILING SEPARATE clause for that item. RM/COBOL version 1.6 behaved this way. RM/COBOL version 2.X can optionally behave this way. For applications that date from earlier versions of the RM/COBOL compiler, you will probably need to use this flag. If the application was written originally with version 2.0 or later, then you may or may not need to use this flag. You will need to examine the source to see if the "S" picture element counts in the size of an item or not. It is important to determine whether or not this flag should be used, particularly if you plan to convert existing data files. Otherwise, records loaded from your RM/COBOL files may not match the record layout of your new ACUCOBOL-GT files.

You may also want to use either the "-Vh" or "-Vl" flags. These flags determine the default video intensity to use when it is not explicitly stated in an ACCEPT or DISPLAY statement. The default action for ACUCOBOL-GT is to use the terminal's default intensity. The "-Vh" flag causes ACUCOBOL-GT to default to high intensity and the "-Vl" flag to low intensity. With RM/COBOL, the default intensity for UNIX machines is high intensity and for MS-DOS machines is low intensity. You may want to match the default intensity used by RM/COBOL for the machine the programs were written for.

By default, in the Identification Division the compiler allows an Area A line to start in column 8 or 9 (ANSI format), or 1 or 2 (terminal format). The "--noRmMargin" option can be used to apply a more stringent rule. When specified, lines in Area A of the Identification Division must begin in column 8 (ANSI format) or column 1 (terminal format).

The table below shows the RM/COBOL Compatibility Switches ("*" indicates an optional flag).

	Version 1.6	**Version 2.*x***	**RM-85**
General	-Cr	-Cr	-Cr
Reserved words	-R8vais	-R8vais	-Rvais
Data storage	-Ds	-Ds*	-Ds*
Video	-Vh (for UNIX) -Vl (for DOS)	-Vh (for UNIX) -Vl (for DOS)	-Vh (for UNIX) -Vl (for DOS)
Area A	--noRmMargin*	--noRmMargin*	--noRmMargin*

You will probably want to set the COPYPATH environment variable in order to locate your COPY libraries on your system. For details on how to do this, see the *ACUCOBOL-GT User's Guide* Chapter 2, "Compiler and Runtime."

2.2 Runtime Options

There are several runtime options that you will probably want to employ. The "-w" flag suppresses warning messages from the runtime system. In particular, it inhibits the warning message "NONNUMERIC DATA IN NUMERIC ITEM". This message is printed whenever a numeric field is used in a *numeric* fashion and that field does not contain a valid number. The default action for ACUCOBOL-GT is to print the warning message and then treat the value as zero. RM/COBOL does not check for this case and proceeds to do some undefined action. Because it does not test for this condition, it shows up in programs surprisingly often. When the "-w" runtime flag is specified, ACUCOBOL-GT does not print a message and acts on the field with its current value. The results are not well defined but usually are the same as the same program run under RM/COBOL.

By default, ACUCOBOL-GT uses 1985 ANSI standard file status codes. These codes differ from those used by RM/COBOL in several respects. If your programs are written to the 1974 RM/COBOL standard, you should add the following line to the configuration file to cause ACUCOBOL-GT to use those file status codes:

```
FILE-STATUS-CODES  74
```

Note: If you are converting programs from RM/COBOL-85, and use the default 1985 standard status codes, then you will *not* want to add this line to the configuration file.

RM/COBOL (but not RM/COBOL-85) also automatically closes all non-print files in a subprogram when that program exits. This is in conflict with the ANSI standard which states that a subprogram remains in the same state until explicitly canceled. ACUCOBOL-GT follows the ANSI standard. If your programs depend on this behavior, however, you can add the following line to the configuration file to cause ACUCOBOL-GT to act the same way that RM/COBOL does:

```
CLOSE-ON-EXIT  1
```

You will also need to edit the configuration file to describe which printers are attached to your system. You may also want to set the PATH entry to describe where your object-code files reside. These procedures are described in Chapter 2 of the *ACUCOBOL-GT User's Guide*, "Compiler and Runtime."

2.3 Memory Management

RM/COBOL supports two forms of memory management. One form is the standard ANSI COBOL form. This is also the memory management scheme used by ACUCOBOL-GT. If your RM/COBOL application uses the ANSI-style of memory management, then you do not need to make any changes to run under ACUCOBOL-GT.

The RM/COBOL implementation of the ANSI memory management is very restrictive, however, in that the main program plus the set of all called programs must fit in 64K bytes (except for those that are canceled). For this reason, many applications use RM/COBOL's alternate mode of memory management. In this mode, all of the currently active programs must fit in 64K bytes, but programs that are not currently active need not. This has the effect of dynamically canceling inactive subprograms as needed to free up enough space. This has the advantage that the programmer largely does not need to worry about canceling subprograms. When this style of memory management is used, however, the programmer cannot depend on a subprogram's remaining loaded in memory. This can cause variables to lose values. RM/COBOL also causes open files in a subprogram to be closed when that subprogram exits (except for files assigned to a printer).

ACUCOBOL-GT has a much larger address space, and thus has no need for this scheme. Furthermore, this form of memory management results in unpredictable behavior, since the state of a called and exited subprogram is unknown. For these reasons, ACUCOBOL-GT does not directly support the alternate form of RM/COBOL memory management.

ACUCOBOL-GT does, however, have several techniques that can be used when you are converting programs that use this style of memory management. Virtually any RM/COBOL program will run under the ACUCOBOL-GT memory management scheme. The only significant problem that applications face has to do with handling menus. These applications have one or more master menus that the user selects specific tasks from. Usually, each of these tasks is a separate program that is called from the menu program when selected. Since ACUCOBOL-GT abides by the ANSI/ISO standard, each of these programs remain in memory until explicitly canceled. As the user chooses more and more menu selections, the memory used can become large. This could eventually overrun the amount of memory available in the machine.

The ANSI/ISO solution to this problem is to cancel the task-handling subprograms when they are finished. This would normally happen when control is returned to the menu program. ACUCOBOL-GT has an extension to the CANCEL verb to make this task easier. This is the statement "CANCEL ALL". This statement has the effect of canceling every program except for those that are currently active. This has two advantages over the standard CANCEL statement:

1. You do not need to know the name of the program being canceled. Thus you can place this one statement in the main loop in your menu program to handle all of the menu selections.

2. The CANCEL ALL statement also cancels subprograms called by intervening programs. Suppose the menu program calls program A, which in turn calls program B. If you explicitly cancel program A, the ANSI standard states that program B will remain unconcealed. The CANCEL ALL statement, on the other hand, will cancel both program A and B.

This is probably the easiest way to handle the memory management problem. Another technique, however, that gives you greater control is to use the INITIAL PROGRAM attribute. Any program that has the initial attribute is automatically canceled whenever it exits. You can specify the initial attribute by either of these two methods:

1. By specifying “IS INITIAL PROGRAM” in the PROGRAM-ID clause.

2. By using the “-Zi” compile-time flag.

You can use this technique if you want greater control over memory management than the CANCEL ALL statement gives you.

Finally, do not overlook the standard CANCEL verb. This has the advantage that it is part of both the 1974 and 1985 COBOL standards.

For more information about runtime memory management, see section 6.3, “Memory Management,” in the *ACUCOBOL-GT User’s Guide*.

2.3.1 Converting RM/COBOL Data Files

The exact procedure depends on the version of RM/COBOL used to create the data files.

Sequential and relative files created by RM/COBOL version 2 programs are directly usable by ACUCOBOL-GT programs. These files have exactly the same internal format. (Note, however that this is not true for variable-length binary sequential files or variable-length relative files.)

RM/COBOL version 2 produces two types of indexed files: single-file format and dual-file format. The dual-file format consists of two physical files for each logical file. One of these files is a binary sequential file that contains the raw data records. The other is a file that contains the key information. You can convert one of these files by using the “load” option of **vutil** (described in the Debugger and Utilities chapter). To do this, first create an empty ACUCOBOL-GT indexed file either with a COBOL program (by doing an OPEN OUTPUT) or with the “generate” option of **vutil**. Then use the “load” option of **vutil**, specifying the RM/COBOL data file as the input file. Note that the RM/COBOL key file is not used.

A single-file format indexed file must first have its records unloaded into a binary sequential file with an RM/COBOL program. Then this file can be loaded into an ACUCOBOL-GT indexed file with the procedure described for the dual-file format.

Line sequential and fixed-length binary sequential files created by RM/COBOL-85 are directly usable by ACUCOBOL-GT. Use the following procedures to convert other RM/COBOL-85 formats to fixed-length binary sequential.

2.3.1.1 Converting relative files with variable-length records

You can write a program in RM/COBOL-85 to convert variable-length relative files to fixed-length binary sequential files that ACUCOBOL-GT can use. To do this:

1. Write a program that reads the desired file using a DEPENDING ON phrase in the RECORD clause of the input file’s FD. The variable depend will be filled in with the record size.
2. Define the output file as follows:
   - Records should be fixed-length binary sequential.
   - Each record must begin with two bytes that are COMP-4, and those two bytes should be filled in with the value of depend.
   - The remainder of the record should be the actual record contents from the input file.

3. Write the output records to a new file. Use this new fixed-length binary sequential as your ACUCOBOL-GT data file.

Note: ACUCOBOL-GT binary sequential and relative files have essentially the same format.

2.3.1.2 Converting binary sequential files with variable-length records

To convert binary sequential files with variable-length records, follow the same procedure (steps 1-3 above) used for variable-length relative files.

2.3.1.3 Converting relative files with fixed-length records

To convert relative files with fixed-length records, write an RM/COBOL program that reads the file and writes the records to a fixed-length binary sequential file. The resulting relative file is usable by ACUCOBOL-GT.

2.3.1.4 Converting indexed files

vutil can convert an indexed file created by RM/COBOL-85 into a Vision file. This is useful when you are moving data from an RM/COBOL-85 application to an ACUCOBOL-GT application. The command is:

```
vutil  -convert  [ -ac ]  [ +c ]  [ -f # ]  [ -2345 ]
                 [ -q ]  [ -d dir ]  [ files ]
```

The “convert” option starts with the same letter as the “check” option described earlier. You must use at least two letters of the word “convert” in order to specify this option. If you just use “-c”, **vutil** will assume that you are specifying the “check” option.

The “convert” function will take each named file and convert it from an RM/COBOL-85 indexed file to a corresponding Vision file. The Vision file replaces the original RM/COBOL-85 file, so you should have a current backup of the original files. If no files are specified, then the standard input is read for a list of files to convert.

Normally the resulting Vision file will be compressed if the original file has compressed records (this is the RM/COBOL-85 default). Specifying the “-c” option will cause the resulting file to have uncompressed records regardless of the original file; using “+c” will cause the resulting records to be compressed.

The “-f #” option sets the compression factor to be used when the file is converted. This option does not force the use of compression, it merely sets the compression factor if compression is used. The compression factor, a numeric literal, specifies how much of the space saved by compression is actually to be removed from the record.

Normally **vutil** will warn the user about the impending conversion and ask if he or she wants to continue. The “-a” (for “automatic”) option suppresses this warning. This can be useful when you are calling **vutil** from another program.

The “-5” option specifies that you want the resulting file to be in Vision Version 5. The “-4” option specifies a Vision Version 4 file. A “-3” means you want a Version 3 file, and “-2” means you want a Version 2 file.

The “-d” option specifies that you want the converted files to be placed in a new directory. Dir should be the name of a directory on the machine other than the directory containing the files to be converted. When “-d” is specified, the original files are not destroyed. Instead, the converted files are placed in dir. The “-d” option implies the “-a” option.

The “-q” option causes **vutil** to exit (with status 99) if user interaction is required.

There are a few types of files that cannot be converted due to restrictions in Vision. Any of the following properties will cause **vutil** to print a message and leave the file alone:

1. A record size or block size greater than 32 KB.

2. More than 120 keys.

3. An individual key with more than 250 bytes in it.

4. A non-ASCII code-set or collating sequence.

Some files cannot be converted due to unresolved format differences. A file with split keys cannot be converted for this reason.

If the original file has variable size records, **vutil** will convert these to fixed size records with space padding. These files should be compressed to keep disk usage down.

vutil makes a copy of the file while it is converting it. You must have adequate disk space for **vutil** to complete its conversion. Also, RM/COBOL-85 indexed files and Vision files differ in the amount of disk space they use. This difference is fairly unpredictable and can vary quite widely. Sometimes the Vision files are smaller, and sometimes the RM/COBOL-85 files are smaller. Be sure to have plenty of spare disk space when you start converting files to accommodate the potential difference.

3 ICOBOL Conversions

Key Topics

3.1 Compile-Time Options

This chapter discusses how to convert a program written for Data General ICOBOL to ACUCOBOL-GT. ACUCOBOL-GT provides a great deal of compatibility with ICOBOL, but there are a few differences that you will want to address.

It is important that you use the correct compile-time options to provide the greatest compatibility with your existing ICOBOL programs. The most important option is the “-Ci” option that puts the compiler into its ICOBOL compatibility mode. When you use this option, most ICOBOL syntax will be accepted by ACUCOBOL-GT. In addition, the meaning of certain phrases will be altered to match the ICOBOL usage. When ICOBOL compatibility mode is specified, the “-Zr” compile-time option is automatically implied. This option allows for recursive PERFORM statements.

There are a few other options that you will probably want to use. The “-Vx” option enables the entry of function keys on all ACCEPT statements. If you do not specify this, then only ACCEPT statements with an ON ESCAPE clause will allow the entry of function keys. If you want to use ICOBOL’s default handling of high and low intensity, then you should specify “-Vah”. This causes all input and update fields to be displayed in high intensity, while all literal and output fields are shown in low intensity.

You may also want to use the “-Dz” compile-time option to modify the way that COMPUTATIONAL data items are handled. Specifying this option causes the runtime system to compute the size error condition based on the physical size of the data item in memory, not on its PICTURE clause. For example, this would allow the value 255 to be placed in a PIC 99 COMP data item. Without this option, the largest value that can be placed in such an item is 99.

Finally, you may need to suppress certain reserved words used by ACUCOBOL-GT. If you want to suppress all of the reserved words not used by ICOBOL, you should specify “-Rarsv”. If you plan to use some of the features of ACUCOBOL-GT not found in ICOBOL, then you will probably have to allow for some more reserved words. For details on your choices about reserved words, see Appendix B of the ACUCOBOL-GT Manual Set and section 2.1.8 of the *ACUCOBOL-GT User’s Guide*.

Note: When you select ICOBOL compatibility mode, the storage of COMPUTATIONAL items is changed to match that of ICOBOL. You should not specify any of the compile-time options that affect COMPUTATIONAL storage, because they will override the ICOBOL settings. The following table summarizes the storage of COMPUTATIONAL items.

# of 9's	**Unsigned**	**Signed**
1-2	1	1
3-4	2	2
5-6	3	3
7	3	4
8-9	4	4
10-11	5	5
12	5	6
13-14	6	6
15-16	7	7
17-18	8	8

3.2 Runtime Options

When you use ICOBOL compatibility, the compiler generates code that has the same meaning used by ICOBOL. The default screen interface, however, is controlled by the runtime system. If you want to simulate ICOBOL keyboard handling, you should add the following lines to your configuration file. (Section 2.7 of the *ACUCOBOL-GT User's Guide* discusses the configuration file in detail.)

```
SCREEN Input-Display=Prompt
SCREEN Numeric-Updates=Unchanged
SCREEN Edited-Updates=Left-Adjust
KEYBOARD   Cursor-Past-End=Yes
KEYSTROKE  Edit=Next  Terminate=0 ^M
KEYSTROKE  Edit=Next  Terminate=0 ^J
KEYSTROKE  Edit=Next  Terminate=0 kd
KEYSTROKE  Edit=Previous  ku
KEYSTROKE  Edit=Next  Terminate=0 ^I
KEYSTROKE  Exception=1  27
KEYSTROKE  Exception=2  k1
KEYSTROKE  Exception=3  k2
KEYSTROKE  Exception=4  k3
KEYSTROKE  Exception=5  k4
KEYSTROKE  Exception=6  k5
KEYSTROKE  Exception=7  k6
KEYSTROKE  Exception=8  k7
KEYSTROKE  Exception=9  k8
```

The exact meaning of these entries is described in Chapter 4 of the *User's Guide*, "Terminal Manager." At some point, you should read this chapter to acquaint yourself with the various options available. The listing above covers only function keys 1 - 8. If you use more of the function keys, you will need to make additional KEYSTROKE entries to define those keys.

Use the ICOBOL_FILE_SEMANTICS runtime configuration variable to get ICOBOL compatible behavior when reading past the beginning or ending of a file. For more details, see section 3.2.1 below.

You will also probably want to use file status codes that are compatible with those used by ICOBOL. You can do that by entering the following line to your configuration file:

```
     FILE-STATUS-CODES  DG
```

3.2.1 ICOBOL Runtime Configuration Variable

The ICOBOL_FILE_SEMANTICS* configuration variable affects the behavior of indexed and relative files when reversing direction after reading past the beginning or end of a file. Normally, if you perform a series of READ NEXTs that reach to the end of the file (returning file status "10"), a subsequent READ PREVIOUS will return the last record in the file. The file pointer's position after each READ NEXT is just past the end of the last record. Similarly, reading past the beginning of the file and then doing a READ NEXT will return the first record in the file.

Under ICOBOL, these conditions produce different results. The record returned by the READ PREVIOUS is the second-to-last record in the file, and the record returned by the READ NEXT is the second record in the file. Essentially, when a series of READs passes either end of the file, the record pointer remains on the first or last record.

Setting ICOBOL_FILE_SEMANTICS to "1" (on, true, yes) will cause the runtime to emulate ICOBOL's handling. This is useful when porting ICOBOL programs to ACUCOBOL-GT. This option is effective only in programs that have been compiled for ACUCOBOL-85 2.0, or later. The default value is "0" (off, false, no).

3.3 Differences

ACUCOBOL-GT contains certain differences from ICOBOL, even when you are using ICOBOL compatibility mode. These differences are detailed here.

ASSIGN TO DISPLAY and KEYBOARD

ICOBOL allows an ASSIGN clause to use the word DISPLAY to refer to the user's screen and KEYBOARD to refer to the user's input device. ACUCOBOL-GT allows these terms, but when they are used a file name must be specified. This allows (as ICOBOL does) the specification of a text file. However, ACUCOBOL-GT does not support the direct assignment to the user's screen or keyboard. To do this, you will have to add code to identify the device name of the user's terminal and ASSIGN to that name. You can use the ACCEPT FROM SYSTEM-INFO verb to return the user's station id.

ACCEPT FROM LINE NUMBER

This form of the ACCEPT statement returns a 3-digit number corresponding to the console device that is controlling the executing program. Because most of the machines that run ACUCOBOL-GT do not use device numbers, but use alphanumeric names instead, ACUCOBOL-GT computes the device number by the following procedure. First, the device name is converted to uppercase and hyphens are converted to underscores (on UNIX systems, the initial "/dev/" is removed). This name is then searched for in the environment (including the configuration file). If it is found, then the value of the name is returned. If it is not found, a number is constructed from any digits found in the device name. If no digits are present, the value “0” is used.

ACCEPT FROM EXCEPTION

This statement, which returns the reason the last CALL statement failed, is not supported. A similar function can be coded using the C$CALLERR library routine described in Appendix I of the ACUCOBOL-GT Manual Set.

Library Routines

ICOBOL contains several library routines that can be called. These routines start with a “#” character. ACUCOBOL-GT does not support these routines and will ignore them if you call them with the CALL PROGRAM verb. You will have to rename the routines and code them in C or COBOL if you want to use them.

4 HP COBOL Conversions

Key Topics

4.1 Introduction to HP COBOL Compatibility

ACUCOBOL-GT provides extensive compatibility with HP COBOL II/XL. On the HP e3000, in HP COBOL compatibility mode, ACUCOBOL-GT can compile and run HP COBOL programs that use HP COBOL extensions, HP e3000 intrinsics, VPLUS forms, and KSAM and IMAGE data sources. On other platforms, ACUCOBOL-GT supports HP COBOL syntax extensions and the HP COBOL preprocessor functions. HP COBOL syntax extensions have the same meaning and generate the same results on all platforms. The ACUCOBOL-GT HP COBOL preprocessor provides the same functional capabilities on all platforms.

ACUCOBOL-GT and HP COBOL share a common core of ANSI/ISO standard COBOL. In addition, most HP COBOL extensions to the ANSI/ISO standard are supported by ACUCOBOL-GT via the "-Cp" compiler option. The "-Cp" option, set either on the command line or in the CBLFLAGS environment string, places the ACUCOBOL-GT compiler in HP COBOL compatibility mode. In this mode, ACUCOBOL-GT performs actions and accepts features of HP COBOL that it does not otherwise perform or allow.

Note: We use the term *compatibility* to describe how ACUCOBOL-GT is compatible with HP COBOL. It does *not* refer to the *compatibility mode* on HP 3000 series 900 computers that allows MPE/iX V applications to run on MPE/iX XL machines without recompilation. This is commonly referred to as *compatibility mode* (CM) by HP e3000 users.

Note: The specific version of HP COBOL with which ACUCOBOL-GT is compatible is defined in "HP COBOL II/XL Reference Manual." Further references to "HP COBOL" in this chapter refer to HP COBOL II/XL.

Users of ACUCOBOL-GT's HP COBOL compatibility mode, in addition to getting HP COBOL support, gain the benefit of many useful ACUCOBOL-GT extensions (see Appendix A of the ACUCOBOL-GT Manual Set, section A.3 for a list of ACUCOBOL-GT extensions to the ANSI/ISO standard). Users can also benefit from a powerful set of complementary technologies (see the *ACUCOBOL-GT User's Guide*, Chapter 1, section 1.1, "Product Overview" for a brief introduction to these technologies).

We also offer an integrated development environment (IDE) for COBOL called AcuBench. AcuBench extends and enhances the ACUCOBOL-GT compiler and runtime system with an advanced suite of GUI-based development tools for COBOL. With AcuBench you can develop and maintain your COBOL applications in an integrated, developer-friendly Microsoft Windows environment and deploy them on your HP e3000 system, or any system supported by ACUCOBOL-GT.

4.2 ACUCOBOL-GT in MPE/iX Environments

ACUCOBOL-GT supports HP COBOL compatibility mode in MPE/iX environments on the HP e3000 platform. In addition, ACUCOBOL-GT can be used with MPE/iX emulators on non-HP e3000 hardware. This section discusses how to use ACUCOBOL-GT (and its debugger) in these environments. It also discusses how to access MPE KSAM, relative, and sequential files from an ACUCOBOL-GT program.

Qedit source files

When the ACUCOBOL-GT compiler is started, in addition to accepting source files in text file format, the compiler also recognizes and accepts source files in Qedit format. To support the compilation of a Qedit file, the compiler makes a temporary text file from the original file. The temporary file is removed after compilation. If the compiler is also invoked with the "-v" (verbose) option, a message similar to the following is displayed for every Qedit file:

```
filename is a Qedit file
Processing Qedit source file:/tmp/SRCxxxxxx from:filename
```

KSAM COPY files and libraries

ACUCOBOL-GT can read KSAM COPY files and COPY libraries on HP e3000 systems. On other platforms, ACUCOBOL-GT requires COPY files and COPY libraries to be plain text files. ACUCOBOL-GT includes a utility called **libutil** that can be used to easily convert KSAM COPY libraries into text format COPY libraries. For more information about **libutil**, see **section 4.2.7, “The libutil Utility.”**

Runtime configuration file

Many aspects of the ACUCOBOL-GT runtime system can be controlled through configuration variables. Configuration variables are maintained in a configuration file. This is a standard text file that can be modified by the host system’s text editor. For more information about runtime configuration, see section 2.7, “Runtime Configuration,” in Book 1 of the ACUCOBOL-GT documentation set. For more information about runtime configuration variables, see Appendix H of the ACUCOBOL-GT Manual Set.

Note: Configuration file entries are not recognized when the configuration file contains line numbers along the right-hand column. Such numbers are included by default by the MPE/iX **Editor** program. To remove line numbering, specify “unn” when you save the file. For example:

```
:editor
/a
     1 DEFAULT_FILESYSTEM MPE
     2 //
/k CBLCONFI, unn
/exit
```

4.2.1 Using ACUCOBOL-GT in Traditional MPE/iX Environments

If you want to use ACUCOBOL-GT in the traditional MPE/iX environment (outside of POSIX), you should set the following MPE/iX environment variables.

```
    :SETVAR HPPATH HPPATH+”,.,/ACUCOBOL/bin”
    :SETVAR A_TERM "hp”
    :SETVAR A_TERMCAP "/ACUCOBOL/etc/a_termcap”
```

Note: If when you installed ACUCOBOL-GT you performed every step of the installation instructions, these environment variables should already be set. See the *Getting Started* booklet, section 1.6, "Installing on HP e3000 Machines" for details.

4.2.1.1 Compiling and running in the MPE/iX environment

To run the compiler and runtime from the MPE/iX command line, you should use the following general syntax.

```
    :RUN <command>;INFO="args"
```

or:

```
    :<command> "<args>"
```

where:

command could be "ccbl" (for the compiler) or "runcbl" (for the runtime), and

args are the options to the compiler or runtime. The options must be enclosed in quotes.

For example:

```
    :CHDIR /ACUCOBOL/sample
    :ccbl "tourhp.cbl"
    :runcbl "tourhp.acu"
```

Note: ACUCOBOL-GT filenames are case sensitive and must be entered in the case shown. See the installation instructions to help resolve any errors you encounter when compiling or running the "tourhp" sample.

Examples of compiling programs with HP COBOL

With HP COBOL, the syntax for compiling and linking a COBOL program might look like this:

```
    :BUILD HELLOLST;REC=-80,1,F,ASCII;DISC=2000
    :COB85XLK HELLOCBL,HELLOEXE,HELLOLST
```

or like this:

```
:FILE COBTEXT=HELLOCBL
:FILE COBOBJ=HELLOOBJ
:FILE COBLIST=HELLOLST
:BUILD HELLOLST;REC=-80,1,F,ASCII;DISC=2000
:RUN COBOL.PUB.SYS;PARM=7
:LINK FROM=HELLOOBJ;TO=HELLOEXE
```

Examples of compiling programs with ACUCOBOL-GT

With ACUCOBOL-GT, the syntax for compiling the same COBOL program might look like this:

```
:ccbl "-Cp –Lof HELLOLST –o HELLOOBJ HELLOCBL”
```

where:

-*Cp* tells the compiler to use HP COBOL compatibility mode;

-*Lof HELLOLST* creates a full listing file called HELLOLST;

-*o HELLOOBJ* creates an object file called HELLOOBJ; and

HELLOCBL is the name of the source COBOL program.

For a quick reference to all of the compiler options type:

```
:ccbl “-help”
```

Note: You should use the command line options in lieu of the HP COBOL $CONTROL directives. See **section 4.4.2** of this guide for a table that maps HP COBOL $CONTROL directives to the ACUCOBOL-GT command line options for the compiler and the runtime.

Here are some other options that you might use while compiling HP COBOL programs with ACUCOBOL-GT:

“-Rw” - suppresses individual ACUCOBOL-GT reserved words. If your HP COBOL program uses words that are not HP COBOL reserved words, but *are* ACUCOBOL-GT reserved words, a compile time error occurs. For example, if your HP COBOL program has a working storage item called BELL or BEEP, the program will not compile unless you use the "-Rw" option because these words are ACUCOBOL-GT reserved words. You can suppress reserved words with the "-Rw" option on the compiler command line, like this:

```
:ccbl "-Rw BELL –Rw BEEP -Cp –Lof HELLOLST
  –o HELLOOBJ HELLOCBL"
```

"-Di" - causes the compiler to initialize Working-Storage. Many of the problems that surface due to differences in the that way HP COBOL and ACUCOBOL-GT initialize (or not initialize) data items by default, can be avoided by compiling with "-Di". When "-Di" is specified, data items are initialized according to their type. For more information about "-Di", see Book 1 of the ACUCOBOL-GT Manual Set, section 2.1.9, "Data Storage Options."

"-Gd" - compiles for source debugging (this option includes the source code in the compiled object file). You then invoke the debugger with the "-d" option on the ACUCOBOL-GT runtime command line ("runcbl").

"-v" - tells the compiler to output information about what it is doing during compilation.

Examples of running programs with HP COBOL

With HP COBOL, the syntax for running a COBOL program with an XL (Executable Library) in your group library is:

```
:RUN program;
   INFO='options'; PARM=parameter;
   XL='xl.group.account'
```

or

```
:RUN program;
   INFO='options'; PARM=parameter;
   LIB=G
```

where:

program is the name of the program you want to run;

options are any options to your program;

parameter specifies any software switches you need to set;

xl.group.account is the name of your XL; and

LIB=G means the program's group library is searched first, next its public account library is searched, and finally the system library is searched to resolve the program's external references. You can also specify LIB=P or LIB=S.

Examples of running programs with ACUCOBOL-GT

With ACUCOBOL-GT, the syntax for running a COBOL program with an XL (Executable Library) is:

MPE:

```
:RUN /ACUCOBOL/bin/runcbl;
   INFO='options program'; PARM=parameter;
   XL='xl.group.account'
```

POSIX:

```
shell/iX> callci "RUN /ACUCOBOL/bin/runcbl;
   INFO='options program'; PARM=parameter;
   XL='xl.group.account'"
```

where:

options are any options to "runcbl";

program is the name of the program you want to run;

parameter specifies any software switches you need to set; and

xl.group.account is the name of your XL

Note: You must specify the full or relative pathname of "runcbl" if it is not in your current directory or if your HPPATH environment variable is not set correctly. See the *Getting Started* booklet, section 1.6, "Installing on HP e3000 Machines" for instructions on how to set the HPPATH environment variable.

Note: If you change the name or case of “ccbl” or “runcbl”, you must also change the name of its associated license file. The license file must have the same name and case as the program, plus the file extension “.alc” in lowercase. For example, if you rename “runcbl” to “RUNCBL” you must rename the license file from “runcbl.alc” to RUNCBL.alc”.

If the ACUCOBOL-GT programs are renamed to uppercase names and are in an MPE/iX MPE/iX group like:

```
RUNCBL.group.account
```

then you can use the “LIB=” option to specify searching the group library, public account library, or the system library. For example:

MPE:

```
:RUN RUNCBL; INFO=‘options program’; LIB=G
```

POSIX:

```
shell/iX> callci "RUN RUNCBL; INFO=‘options program’;
   LIB=G”
```

“LIB=G” means the program’s group library is searched first, then its public account library is searched, and finally the system library is searched to resolve the program’s external references. You can also specify LIB=P or LIB=S.

“runcbl” needs to be either in your current directory or in one of the directories specified in your HPPATH.

If you do not have an XL that you need to specify, you can use the implied version of RUN. For example:

MPE:

```
:runcbl; INFO=‘options program’
```

or

```
:runcbl ‘options program’
```

POSIX:

```
shell/iX> callci "runcbl; INFO=‘options program’”
```

or

```
shell/iX> callci "runcbl ‘options program’”
```

If you do not specify the full pathname of “runcbl”, then it needs to be located in one of the directories specified in your HPPATH.

4.2.1.2 Linking

To HP developers in general, the term *linking* means running a command or series of commands that arrange all of a program’s objects so that they can run as a single application.

Within ACUCOBOL-GT, the term *linking* is used to describe the linking of C subroutines or HP RLs into the ACUCOBOL-GT runtime so that the runtime can CALL the functions that these routines provide. Programs compiled with ACUCOBOL-GT do not require linking.

To help illustrate this, suppose you have three COBOL objects. If these three objects are HP COBOL objects, when you link them together you get an HP executable program:

OBJECT1 + OBJECT2 + OBJECT3 = HPEXE

To run the HP executable enter:

```
:RUN HPEXE
```

If these three objects are ACUCOBOL-GT objects, you do not link them together; they remain separate object files:

OBJECT1
OBJECT2
OBJECT3

The ACUCOBOL-GT runtime is used to execute these objects. Assuming that OBJECT1 is the main program, the command to run might look like this:

```
:RUN /ACUCOBOL/bin/runcbl;INFO="OBJECT1”
```

or:

```
:runcbl "OBJECT1”
```

To execute OBJECT2 and OBJECT3, the main program (OBJECT1) makes a CALL to one of the other objects, and the ACUCOBOL-GT runtime dynamically loads and runs the called object. The main program may CALL all of the other objects one by one in the following pattern:

OBJECT1 CALLs OBJECT2; OBJECT1 CALLs OBJECT3; etc.,

or the CALLed object can CALL the next object in the execution sequence, in the following pattern:

OBJECT1 CALLs OBJECT2; OBJECT2 CALLs OBJECT3; etc.

4.2.1.3 Object libraries

There is an ACUCOBOL-GT utility called **cblutil** that you can use to combine multiple ACUCOBOL-GT objects into one ACUCOBOL-GT object library. This is not the same as HP *linking*. It is useful if you want to combine all of your objects into one file so that you don’t have a lot of separate objects.

For example:

```
:cblutil "-lib –v –o OBJLIB OBJECT1 OBJECT2 OBJECT3”
```

creates an ACUCOBOL-GT object library called OBJLIB. (See Book 1, *User’s Guide*, section 3.2, “Object File Utilities-cblutil”, for more information.) Then, when you start the program with:

```
:RUN /ACUCOBOL/bin/runcbl;INFO="OBJLIB”
```

or

```
:runcbl "OBJLIB”
```

the runtime loads all of the ACUCOBOL-GT objects in OBJLIB and starts executing the main program.

4.2.1.4 Using XLs and RLs with ACUCOBOL-GT

The use of XLs and RLs is supported by ACUCOBOL-GT on the HP e3000. XLs should be specified on the "runcbl" command line when the program is started. RLs must be linked into the ACUCOBOL-GT runtime. Relinking the runtime requires a C compiler. In the past, HP has made a C compiler available to DSPP members. By arrangement, Micro Focus's Consulting services can provide assistance or perform relinking for you. The need to relink the runtime should arise only if you are calling functions stored in an RL.

Note: XL and RL programs sometimes change the current working directory, which can have the affect of causing the runtime to fail to locate a subsequently called program. To prevent such a failure, you should set the CODE_PREFIX runtime configuration variable. The CODE_PREFIX variable defines a series of directories that the runtime searches to locate object files. CODE_PREFIX can include absolute and relative paths, and can include the current working directory by specifying a period ("."). For example:

CODE_PREFIX . ; /ACUCOBOL/SAMPLE ; /MYAPP/CBLOBJ

For more information about how the runtime locates program object files, see section 2.7.2, "Code and Data File Search Paths," in Book 1 of the ACUCOBOL-GT Manual Set. For more information about the CODE_PREFIX configuration variable, see Appendix H of the same set.

XLs

XLs should be specified on the command line when you start the program. On the command line, after the "XL=" argument, specify the name of the XL. For example:

MPE:

```
:RUN /ACUCOBOL/bin/runcbl;
   INFO='options program'; PARM=parameter;
   XL='xl.group.account'
```

POSIX:

```
shell/iX> callci "RUN /ACUCOBOL/bin/runcbl;
   INFO='options program'; PARM=parameter;
```

```
        XL='xl.group.account'"
```

where:

options are any options to “runcbl”

program is the name of the program you want to run

parameter specifies any software switches you need to set

xl.group.account is the name of your XL

If you do not want to specify the XL on the command line, you can choose to link the name of the XL into the ACUCOBOL-GT runtime. To link an XL name into the runtime:

Edit /ACUCOBOL/lib/Makefile.

Change: LDFLAGS = -s

to: LDFLAGS = -s -WL,XL=/Account/group/name

where /Account/group/name is the name of your XL file.

RLs

To access functions in an RL, you must link the RL into the ACUCOBOL-GT runtime. To link an RL into the runtime, perform the following:

Edit /ACUCOBOL/lib/Makefile.

Change: LDFLAGS = -s

to: LDFLAGS = -s -WL,RL=/Account/group/name

where /Account/group/name is the name of your RL file.

Relink the runtime with the following commands:

MPE:

```
:CHDIR /ACUCOBOL/lib
:MAKE.HPBIN.SYS
```

POSIX:

```
shell/iX> cd /ACUCOBOL/lib
shell/iX> make
```

4.2.1.5 Interfacing to C subroutines

ACUCOBOL-GT applications can call C subroutines. See Chapter 4 of the *Guide to Interoperating with ACUCOBOL-GT* for details.

Note: If you use the *direct* method described in section 4.2 of the Interoperability Guide and you are preparing to relink the runtime for deployment on the HP e3000, when you apply the instructions in 4.2 you should replace all references to the file “direct.c” with “hpcobol.h”.

4.2.1.6 Privileged mode

ACUCOBOL-GT does not use the *privileged* mode.

4.2.1.7 Terminal emulators

ACUCOBOL-GT and the programs compiled with ACUCOBOL-GT are compatible with most common HP terminal emulators, such as ScreenJet, WRQ Reflection, QCTerm, and MiniSoft MS92.

4.2.1.8 ACUCOBOL-GT PA-RISC native code support

ACUCOBOL-GT PA-RISC native code support on the HP e3000 requires an executable library (XL) called ACUPAXL. This XL needs to reside in a valid MPE group and must be specified on the runtime command line. For example:

```
:RUN /ACUCOBOL/bin/runcbl;
      INFO='native.acu';
      XL='ACUPAXL.PUB.ACUCOBOL'
```

If you try to run a PA-RISC native code object and the runtime cannot find ACUPAXL, then you will get an error like the following:

```
native: Program contains object code for a different
processor
```

This is because the runtime cannot load the PA-RISC native code object without this XL.

4.2.1.9 MPE file equation restrictions

When you are using MPE file equations, you must exercise caution if you are also using the ACUCOBOL-GT FILE_PREFIX or FILE_SUFFIX runtime configuration variables. This is because the runtime prepends the FILE_PREFIX and appends the FILE_SUFFIX strings to the name of the file specified in the SELECT statement. As a result, this name no longer matches the name specified in the file equation, which can result in errors when the runtime tries to access the file.

For example, if you have a SELECT statement such as:

SELECT MYFILE ASSIGN TO “MYFILE”

And you have a file equation such as:

FILE MYFILE=FILEX

And you have a configuration variable such as:

FILE_PREFIX /ACUCOBOL/DATA

The runtime attempts to open a file called /ACUCOBOL/DATA/MYFILE when it should have been opening a file called MYFILE.

To prevent this renaming from happening, you can do either of these two things:

- Put the following “name alias” line in your runtime configuration file:

  MYFILE -F MYFILE

- Change the COBOL programs so the SELECT statement looks like this:

  SELECT MYFILE ASSIGN TO “-F MYFILE”

The “-F” in both cases tells the runtime to use the name as it appears and not to prepend the FILE_PREFIX to the name of the file. The open then treats MYFILE as a file equation and should find the correct file.

4.2.2 Using ACUCOBOL-GT in POSIX Environments

The ACUCOBOL-GT development system can be run in HP COBOL compatibility mode in the MPE/iX POSIX shell. This section discusses some basic issues related to the installation, startup, and running of ACUCOBOL-GT under the POSIX shell on HP e3000 platforms.

To install ACUCOBOL-GT on your system, you must have MPE/iX release 5.0 or later. The installation procedure, including the installation of license files, is described in detail in the *Getting Started* booklet, section 1.6, “Installing on HP e3000 Machines.” The installation procedure gives the exact command line syntax you should use, first to get from the basic MPE/iX level to the POSIX shell, and then to install and run ACUCOBOL-GT. In addition, useful information on printing from the POSIX shell is included.

To get your existing COBOL applications running with ACUCOBOL-GT, you need to recompile the source. The name of the ACUCOBOL-GT compiler executable is “ccbl” and you invoke it with the following command. (For complete information on invoking the compiler, see the *ACUCOBOL-GT User’s Guide,* section 2.1.)

```
ccbl <filename.cbl>
```

The resulting object file has the same base name as the source file plus the extension “.acu”. Unlike the native HP COBOL system, the ACUCOBOL-GT object does not require linking.

To run your program, invoke it with ACUCOBOL-GT runtime (“runcbl”). To invoke it, enter the following command. (For complete information on using the runtime, see section 2.2 in Book 1, *ACUCOBOL-GT User’s Guide*.)

```
runcbl <filename.acu>
```

Any programs that are called during execution are loaded dynamically by the runtime. Subprograms written in C may be linked into the runtime system directly, and then called by a COBOL program using the CALL verb. For details regarding linking a C program into the runtime, see Chapter 4 of the *Guide to Interoperating with ACUCOBOL-GT.*

There are many compile and runtime options that you can also include in the command arguments to compile or execute your ACUCOBOL-GT program. To view a list of compiler options, entering the following command:

```
    ccbl -help
```

Note: You should use the command line options in lieu of the HP COBOL $CONTROL directives. See **section 4.4.2** of this guide for a table that maps HP COBOL $CONTROL directives to the ACUCOBOL-GT command line options for the compiler and the runtime.

For detailed descriptions of the ACUCOBOL-GT compiler and runtime, including all of the command line options and utility programs, see Chapters 2 and 3 in Book 1 of the ACUCOBOL-GT Manual Set. Book 1 also includes sections on how to get help and how to handle compilation and runtime problems (*ACUCOBOL-GT User's Guide*, Chapter 1, section 1.7 and its subsections).

4.2.3 Using ACUCOBOL-GT with MPE/iX Emulators

Several companies offer software that to varying degrees emulates the MPE/iX operating system on non-HP e3000 hardware. In some situations, these emulators provide a viable rehosting path for moving HP COBOL applications to a new platform with minimum changes to the source code. Because they provide varying degrees of support for the full capabilities of MPE/iX, each emulator should be carefully evaluated for its ability to meet the needs of your applications. In most cases, ACUCOBOL-GT can be easily configured to run within these emulators.

The information in this section is designed to help you configure ACUCOBOL-GT for use in an emulated MPE/iX environment. Typically, the configuration steps are modest, requiring only that you enable the MPE file system and assign values to a small number of environment and runtime variables.

This section assumes that you are familiar with your emulator software and that you have successfully compiled your HP COBOL program with the ACUCOBOL-GT compiler on the new host.

4.2.3.1 Enabling the MPE file system

Off of the HP e3000, the ACUCOBOL-GT runtime does *not* come pre-configured to support the MPE file system. Therefore, if you want to access MPE files you must first enable the MPE file system in the runtime. This requires relinking the runtime. The steps for doing this are given below.

Before starting, it is helpful to verify the version and configuration of your ACUCOBOL-GT runtime. On the system, enter:

```
runcbl  -vv
```

The “-vv” option causes the runtime to display detailed version and configuration information. On an HP-UX system the output is similar to:

```
ACUCOBOL-GT runtime version 8.1.0
Serial number 999999
Licensed for 1 user(s)
AcuServer client
Vision version 5 file system
XML version expat_1.95.4 file system
Copyright (c) 1985-2008 Micro Focus (IP) Ltd.
```

The end of the first line of output indicates the version of the runtime. To get MPE file system support, you must have Version 7.0.0, or later.

To enable MPE file system support, follow the steps below. The steps include relinking the runtime. Relinking requires that you have access to the host system’s native C compiler. For complete information on relinking, see section 4.3.6 in *A Guide to Interoperating with ACUCOBOL-GT.*

Note: By default, the ACUCOBOL-GT files needed to relink the runtime are located in the “lib” subdirectory of your ACUCOBOL-GT installation.

---

1. Make a backup copy of your current **runcbl** file.
2. Optionally, include other libraries. See section 4.3 in *A Guide to Interoperating with ACUCOBOL-GT.*
3. Enable (or disable) the MPE file system. Edit “filetbl.c” (in the “lib” subdirectory of your ACUCOBOL-GT installation) and look for the list of define statements. The list contains entries such as:

   ```
   #define USE_VISION        1
   #define USE_RMS           0
   #define USE_CISAM         0
   #define USE_BTRIEVE       0
   #define USE_MPE           0
   ```

   Locate USE_MPE and set the value to “1”. The file systems that are set to “1” are those that are enabled and linked. Those set to “0” are disabled and not linked. Any or all file systems may be enabled at the same time; the more systems you link, the larger your runtime system becomes.
4. Edit “Makefile” (in the “lib” subdirectory of your ACUCOBOL-GT installation) and add “ksam.o” to the FSI_SUBS definition. The line for FSI_SUBS should look like:

   ```
   FSI_SUBS = ksam.o
   ```

   To remove support for MPE, remove “ksam.o” from FSI_SUBS.
5. Relink the runtime. Make your current working directory the “lib” subdirectory of your ACUCOBOL-GT installation (AcuGT/lib) and enter:

   ```
   make
   ```

   This compiles “sub.c” and “filetbl.c”, and links the runtime.

To verify that MPE file system support is enabled, enter "runcbl -vv". The output should now include the line:

```
KSAM version MPE/iX native and compatibility
   mode file system
```

4.2.3.2 Setting runtime configuration variables

Typically, you need to set a few runtime variables before using ACUCOBOL-GT with your MPE/iX emulator.

To ensure that the ACUCOBOL-GT runtime performs as expected, you may need to set the DEFAULT_FILESYSTEM, SHARED_LIBRARY_EXTENSION, and SYSINTR_NAME runtime configuration variables.

Note: An introduction to the function and use of the ACUCOBOL-GT runtime configuration file is located in section 2.7 of the *ACUCOBOL-GT User's Guide*. All ACUCOBOL-GT runtime configuration variables are described in Appendix H of *ACUCOBOL-GT Appendices*.

DEFAULT_FILESYSTEM

The value of this variable tells the runtime which file system to use when the application opens an existing file or creates a new file. (You can override this setting for an individual file with the *filename*_FILESYSTEM variable. See the entry for *filename*_FILESYSTEM in Appendix H.)

To direct the runtime to use the MPE file system by default, set DEFAULT_FILESYSTEM to MPE. The line in the configuration file should look like:

```
DEFAULT_FILESYSTEM MPE
```

SHARED_LIBRARY_EXTENSION

This variable defines the filename extension for UNIX shared libraries. The default value is ".so". On HP-UX systems, set the value to ".sl", as in:

```
SHARED_LIBRARY_EXTENSION = .sl
```

This variable has meaning only on systems that support UNIX shared libraries.

SYSINTR_NAME

For some MPE emulators, this variable defines the location of the SYSINTR file. This variable must be specified with HFS syntax and set to the full path of the SYSINTR file. For example:

```
SYSINTR_NAME /opt/mpe/etc/sysintr.txt
```

4.2.3.3 Setting environment variables

Depending on your host system, the runtime may require that the SHLIB_PATH or LD_LIBRARY_PATH environment variable be defined.

SHLIB_PATH (HP-UX systems only)

The environment variable SHLIB_PATH must include the path to the MPE emulator's shared libraries. This can be accomplished with the following Bourne Shell commands:

```
SHLIB_PATH=$SHLIB_PATH:/opt/mpe/lib
export SHLIB_PATH
```

Or with the following C Shell command:

```
setenv SHLIB_PATH=$SHLIB_PATH:/opt/mpe/lib
```

You can add these commands to the each user's start-up file (.profile or .cshrc), or to make the definition effective for all users, to /etc/profile. Changing /etc/profile requires root privileges.

LD_LIBRARY_PATH (UNIX/Linux systems, except HP-UX)

The environment variable LD_LIBRARY_PATH must include the path to the MPE emulator's shared libraries. This can be accomplished with the following Bourne Shell commands:

```
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/mpe/lib
export LD_LIBRARY_PATH
```

Or with the following C Shell command:

```
setenv LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/mpe/lib
```

You can add these commands to the each user's start-up file (.profile or .cshrc), or to make the definition effective for all users, to /etc/profile. Changing /etc/profile requires root privileges.

4.2.4 Accessing MPE KSAM, Relative, and Sequential Files

You can access MPE KSAM, relative, and sequential files through HP e3000 system intrinsic functions or via standard COBOL I/O statements. If your programs use system intrinsics to access MPE files, you do not need to make any changes to your code or set any special runtime configuration variables. If your programs use standard COBOL I/O statements to access MPE files, you will need to check the setting of one or more ACUCOBOL-GT runtime configuration variables. How to configure ACUCOBOL-GT to access MPE files via standard COBOL I/O statements is discussed in the remainder of this section.

Note: ACUCOBOL-GT supports duplicate primary key values. For more information, see **section 4.3.3, "File-Control Paragraph."**

On the HP e3000, ACUCOBOL-GT supports the following KSAM formats: KSAM XL, KSAM64, and KSAM/3000. KSAM XL and KSAM64 function in the native mode (NM) environment of the MPE/iX operating system. KSAM/3000 is an earlier MPE/iX V/E KSAM format that is used on the MPE/iX in HP's *compatibility mode* (CM).

The following sections describe ACUCOBOL-GT's support of MPE KSAM, relative, and sequential files on the HP e3000.

4.2.4.1 The ACUCOBOL-GT MPE file system interface

On the HP e3000, ACUCOBOL-GT uses the MPE file system by default for KSAM, relative, and sequential files. ACUCOBOL-GT also comes with its own indexed file system called Vision. See **section 4.2.4.2** for more information on selecting a file system.

The MPE file system interface routines are linked into the runtime system and are automatically invoked whenever you execute your programs. This method allows for improved portability of your application because no file system-specific commands need to be embedded in your COBOL code.

Depending on file system version, Vision and KSAM use either one or two disk files for storage. Vision Version 3 files are stored in a single disk file. This can simplify system maintenance and reduce the size of directories. Vision Version 4 and 5 files use a dual-file format, which facilitates the management of very large files. One disk file, or *segment*, contains the data records, and the other segment contains indexing information. As each segment approaches the configurable segment size limit, Vision automatically generates additional segments to hold the data or index information. Like Vision 4 and 5, KSAM/3000 files are stored in two disk files—one for indexes and one for data records. Like Vision Version 3, KSAM XL and KSAM64 use a single disk file. KSAM file size limits are fixed at the time of file creation. The default KSAM file size is limited to 10,000 records. A different file size limit can be established when the file is created by setting the file size parameter in the ASSIGN clause (this is an HP COBOL extension to the ASSIGN clause that requires compilation for HP compatibility ("-Cp")).

KSAM/3000 supports variable-length records. KSAM XL and KSAM64 support only fixed length records. Vision supports variable-length records, and also record locking, data compression, and data encryption.

The ACUCOBOL-GT MPE file system interface creates KSAM XL files by default. A KSAM/3000 file is created if variable-length records are specified in the File Descriptor (FD). The KSAM file is created with the name specified in the ASSIGN clause. The name is limited to eight characters. The KSAM/3000 index file has the same name as the data file but with a 'K' appended to the end (if the ASSIGN name is eight characters, only the first seven are used and a 'K' is appended to the index file name).

4.2.4.2 Selecting a file system

ACUCOBOL-GT allows a program to interface with more than one external file system in the same program. The file systems supported by the runtime are identified and enabled in the file "filetbl.c" (located in the "LIB" directory). On the HP e3000, support for the KSAM and Vision indexed file systems is enabled by default.

To specify the indexed file system that the program will use, you must set the DEFAULT_FILESYSTEM configuration variable. To define a file system for use with a particular file, you set the *filename*_HOST configuration variable. For an introduction to ACUCOBOL-GT runtime configuration variables and the configuration file, see section 2.7, “Runtime Configuration,” in Book 1 of the ACUCOBOL-GT Manual Set.

DEFAULT_FILESYSTEM

DEFAULT_FILESYSTEM specifies the default file system to be used for all file I/O. The default value on HP e3000 machines is “MPE”. If you want to use ACUCOBOL-GT’s Vision file system instead, add the following to the “cblconfig” file:

```
DEFAULT_FILESYSTEM Vision
```

This tells the runtime to use Vision files instead of MPE KSAM files. In addition, if you specify Vision, the relative and sequential files will be in bytestream format instead of MPE format.

*filename*_HOST

This variable specifies the file system to use for a particular file. For example, if DEFAULT_FILESYSTEM is set to MPE (the default), and the file “CUSTFILE” is a Vision file, in your “cblconfig” you could specify:

```
CUSTFILE_HOST VISION
```

This definition directs the runtime to treat “CUSTFILE” as a Vision file. Note that the file name may not include any path or directory name and should not include the file extension.

DEFAULT_FILESYSTEM and *filename*_HOST are described in detail in Appendix H, “Configuration File Entries,” in Book 4 of the ACUCOBOL-GT Manual Set.

When your program executes, each time a file is opened, the ACUCOBOL-GT runtime checks *filename*_HOST and DEFAULT_FILESYSTEM to determine which file system to use. You can change the value of these variables, just before you open the file, by including the following in your code:

```
SET ENVIRONMENT "DEFAULT_FILESYSTEM" TO value
```

or

```
SET ENVIRONMENT "filename_HOST" TO value
```

where *value* is a Working Storage item containing the name of the file system, or is a literal in quotation marks. SET ENVIRONMENT thus enables you to change file systems during the execution of your program.

4.2.4.3 File name handling

On MPE/iX systems, ACUCOBOL-GT can handle references to files, groups, and accounts in the traditional MPE syntax, e.g., FILE1.PUB.SYS. When an ACUCOBOL-GT program attempts to access a KSAM file, the runtime expects the filename to be in MPE syntax. The runtime treats the filenames of other file types differently.

If the file being accessed is *not* a KSAM file, the runtime first assumes that the filename is in Hierarchical File System (HFS) syntax. For example, if FILE1.PUB.SYS is referenced in your program, the runtime looks in the current directory for a file named “FILE1.PUB.SYS”. If the file is not found, the runtime assumes that the filename is in traditional MPE/iX syntax. It then converts the name to HFS syntax and tries again to find the file. For example, FILE1.PUB.SYS is converted to /SYS/PUB/FILE1.

In addition to ACUCOBOL-GT’s built-in method of establishing filename aliases via the runtime configuration file (see section 2.7.1, “File Name Assignments,” in Book 1 of the ACUCOBOL-GT Manual Set), ACUCOBOL-GT also supports file aliases created with the MPE/iX “FILE” command. FILE equations and runtime configuration file entries provide a simple and flexible way to map logical names to physical files.

Note: If you plan to migrate your application to another host and your application makes use of FILE equations, unless you will use an MPE emulator on the new host you will have to use other methods to accomplish file attribute specification and filename aliasing. Precisely what is required will vary depending on the host operating system. ACUCOBOL-GT always supports the following mechanisms:

1. The SET ENVIRONMENT statement can be used in the runtime configuration file and within the program to define filename aliases.

2. File information, including path and name information, can be stored in a special file that the program reads prior to manipulating files.

If you plan to deploy your application with our thin client technology, there is an additional consideration. Frequently, thin client applications are initiated with a common (shared) runtime configuration file. However, this approach prevents unique users from getting unique environment settings (such as those provided by FILE equates). A recommended solution is to prepare unique runtime configuration files for every user (or group of users), and to place those configuration files on the application server. The program is then modified to get the name of the user on startup and to then load the appropriate configuration file using the C$CONFIG library routine.

4.2.4.4 File I/O trace information

Both the ACUCOBOL-GT runtime and debugger support an option to output file I/O trace information during program execution. This facility can be very helpful when you are investigating program execution problems. When file tracing is enabled, the runtime outputs a message to the error file for every file I/O-related action that it executes.

To enable file tracing, the trace level must be set to “7” or higher.

How to enable file tracing is described in Book 1, section 1.7.2, “Handling Program Execution Problems,” of the ACUCOBOL-GT documentation set, and in the entry for the “FILE_TRACE” configuration variable in Appendix H of the ACUCOBOL-GT Manual Set. How to use the file trace facility in the debugger is described in Book 1, section 3.1.4, “File Tracing.”

4.2.4.5 KSAM system limits and ranges

KEYS		1-16 keys
KEY SIZE	byte	1-255 bytes
	integer	1-255 bytes of integer data
	real	1-255 bytes of real number data
	IEEE real	4, 8, or 16 bytes of IEEE real number data

	numeric	1-28 bytes of numeric data
	packed	1-14 bytes of packed decimal data (odd # of char)
	*packed	2-14 bytes of signed packed decimal data (even # of char)
REC SIZE		1-32,767 fixed-length and undefined-length ASCII files
		1-32,766 variable-length ASCII, fixed, variable and undefined length binary
FILE SIZE		4 gigabytes, KSAM XL file
		128 gigabytes, KSAM64 file

Other limits apply as described in Appendix A of the ACUCOBOL-GT documentation set.

4.2.4.6 Enabling and disabling the MPE interface

ACUCOBOL-GT's support for KSAM is enabled by default. However, if for some reason support has been disabled and you want to re-enable it, or if you want to disable support, you can include or remove it from the runtime by performing the following steps.

1. Make a backup copy of your current **runcbl** file.

2. Edit "filetbl.c" (in the "LIB" directory). In a text editor, look for the list of define statements. The list contains entries such as:

   ```
   #define USE_VISION   1
   #define USE_RMS      0
   #define USE_CISAM    0
   #define USE_BTRIEVE  0
   #define USE_INFORMIX 0
   #define USE_KSAM     1
   ```

   Set a value in the list to "0" (zero) to disable that system, or to "1" (one) to enable it. The file systems that are set to "1" are the ones you plan to link and use. Any or all may be enabled at the same time; the more systems you link, the larger your runtime system becomes. Only Vision and MPE KSAM, relative, and sequential files are supported on the HP e3000.

3. Locate and edit the “Makefile” (in the “LIB” directory). In a text editor, add “ksam.o” to the FSI_SUBS definition. The line with FSI_SUBS should look like this:

   ```
   FSI_SUBS  =  ksam.o
   ```

   To remove support for KSAM, remove “ksam.o” from FSI_SUBS.

4. Link the runtime system. Be sure that your current working directory is the “lib” subdirectory of your ACUCOBOL-GT directory.

   To link the runtime in the MPE/iX environment enter:

   ```
   :/usr/local/bin/make
   ```

   To link the runtime in the POSIX environment enter:

   ```
   make
   ```

   This compiles “sub.c” and “filetbl.c”, and then links the runtime.

5. To specify MPE as the default file system, edit “cblconfig” (the runtime configuration file) and add the following line:

   ```
   DEFAULT_FILESYSTEM MPE
   ```

   You can override this default in your programs via the SET ENVIRONMENT statement or by setting the *filename*_HOST variable.

6. Test the system.

   The interface is now in place. To ensure that ACUCOBOL-GT is writing and reading records via KSAM, move to the directory containing the sample program **iobench**. Create a temporary configuration file named “temp.ttt”. In this file, place one line:

   ```
   DEFAULT_FILESYSTEM MPE
   ```

   At the system prompt, type:

   ```
   :runcbl  "-v”
   :ccbl  "-x  -Cr  -Si  ACU  iobench.cbl”
   :runcbl  "-c  temp.ttt”
   ```

   This displays the version number of the interface and then compiles and executes a sample program that generates several file I/O activities.

We use the file "temp.ttt" to prevent the runtime system from using your usual "cblconfig" file. This ensures that the **iobench** program creates its files in the current directory.

4.2.5 Using the ACUCOBOL-GT Debugger in MPE/iX Environments

Built into the ACUCOBOL-GT runtime is a powerful and easy to use source-, symbolic-, and low-level debugger. Its capabilities and use are described in detail in Book 1, section 3.1, "Runtime Debugger," of the ACUCOBOL-GT Manual Set. You can use the debugger via an HP terminal or from a PC running a terminal emulator (such as ScreenJet or Reflection); however, access to the debugger's menu bar may be restricted or require changes to the keyboard configuration. These issues are described below.

In every environment, the debugger includes a menu bar and command window. On systems with mouse support, the menus can be accessed directly with the mouse. On other systems, the menu bar is activated with function key F10 and the menus are navigated with the arrow keys. The Command window supports the traditional text-based command line interface. Most actions that can be initiated from the menu bar can also be specified in the command window.

On HP terminals, because the keyboard does not have several function keys that are common on PC keyboards (F10 and the arrow keys, in particular), the debugger's menu interface is not accessible. Instead, you should simply use the debugger's command window. For a list of commands and command syntax, see section 3.1.3, "Debugger Commands" in Book 1. Functions keys F1 through F8 are available on most HP terminals and can be used to perform the actions described in Book 1, section 3.1.3.

If you are using a PC-based system with terminal emulator software, you have the option of re-mapping the keyboard and redefining the termcap definition on the HP e3000 to enable F10 and the arrow keys and, thereby, gain access to the debugger's menu bar. Terminal emulator configuration is described in the next section.

4.2.5.1 Terminal emulator keyboard configuration

If you are using the ACUCOBOL-GT debugger via a PC system with a terminal emulator, you can specially configure your system to gain access to the debugger’s menu system. Configuration requires two actions:

1. Redefinition of the terminal type on the HP e3000.
2. Re-mapping of several special keys in the terminal emulator so that key presses send the correct value to the debugger. These keys—F10 and the arrow keys—may have special purposes within the emulator which will be lost when each key is re-mapped.

Setting the terminal type

In the MPE/iX environment, or in your login UDC, set the environment variable A_TERM to one of the following values, as appropriate.

```
hpminisoft
hpreflection
hpscreenjet
```

For example:

```
:SETVAR A_TERM "hpreflection”
```

If you are using an emulator that is not listed above, but the emulator is very similar to one of the above, set the A_TERM variable to the similar emulator, remap the special keys as described in the next section, and run a program in the debugger to test for correct behavior.

Re-mapping special keys

In your terminal emulator, access the keyboard configuration facility and remap the following keys to the specified values (see your terminal emulator documentation for specific instructions on re-mapping keys):

F10	escape – y – carriage return (^[y^M)
Up Arrow	escape – A (^[A)
Down Arrow	escape – B (^[B)

Right Arrow	escape – C (^[C)
Left Arrow	escape – D (^[D)

To easily use these settings again in a subsequent session, save these mappings to a new “configuration” or “session.”

4.2.5.2 Debugging programs that use VPLUS

Debugging programs that use VPLUS is tricky because debugger I/O and VPLUS I/O cannot share the same terminal session. A common way to work around this problem is to use two terminal sessions, one to host debugger output and one to host VPLUS I/O. In the application of this method, the first terminal session hosts debugger output and establishes a FILE equation that directs VPLUS output to a second terminal session. This is the strategy we recommend you use when you want to debug an ACUCOBOL-GT program that uses VPLUS.

Note: You do not need to use two terminal sessions if the program you want to debug does not use VPLUS.

Before you can use the source-level debugger on any program, you must compile the program with the “-Gd” option.

The following steps describe one way to start your program so that debugger I/O will go to one terminal session and VPLUS I/O will go to another:

1. Get a logical device number for the session that will host the VPLUS I/O.

    a. Establish a serial connection to the HP e3000 that has the qualities of a directly connected terminal (e.g.: a DTC, modem connection, modem connection from a terminal emulator, etc.).

    b. Get the logical device number (LDEV) of the connection. The LDEV will be used later in the FILE equation that redirects VPLUS I/O to this logical device. You can use the “SHOWME” command to get the LDEV number. For example:

    ```
    :SHOWME
    $STDIN LDEV: 90    $STDLIST LDEV: 90
    ```

Make a note of the LDEV number. In the above example, the LDEV number is 90.

c. Log out of the session (VPLUS cannot use the connection if it is already in use).

2. Open a session for the ACUCOBOL-GT debugger. The connection type does not matter; the connection can be from a terminal emulator.

3. In the debugger session, set a FILE equation that points to the VPLUS terminal filename that is specified in the VOPENTERM call and set the DEV number to the LDEV number derived from step 1. For example:

   ```
   :FILE VPLUSTF;DEV=90
   ```

4. Start the ACUCOBOL-GT debugger by including the "-d" option before the name of the program. For example:

   ```
   :RUN /ACUCOBOL/bin/runcbl;INFO="-d VPLUSP"
   ```

5. Use the debugger to debug your program.

When you use the debugger, the runtime switches into block mode before calling a VPLUS intrinsic and then back into raw mode after the call completes. This is done because the runtime requires raw mode when using the debugger. When the runtime is in block mode you will not be able to use COBOL DISPLAY/ACCEPT statements, nor will runtime errors be displayed on the screen. To capture runtime error messages, you should use the "-le" runtime option to send error messages to a file.

Some VPLUS calls may cause the keyboard to become locked when using the debugger. If this happens, you must use the appropriate commands for your terminal to unlock the keyboard.

4.2.6 Terminal Configuration with VPLUS

As in all environments, what the operating system and program know about the display device affects the handling of screen I/O. This is equally true for HP e3000 COBOL applications that use VPLUS. To enable VPLUS applications to override some of its default values, VPLUS employs an environment control file, "VENVCNTL". When VPLUS starts up, VPLUS checks for the existence of VENVCNTL.PUB.SYS or a file equated to the formal file designator. If no VENVCNTL file exists, the standard VPLUS defaults are used.

VENVCNTL is a simple text file that contains only one line of data. Each override option may be activated by setting the option byte number to 1. For example, in the VENVCNTL file below, option 6 is set.

```
:PRINT VENVCNTL.PUB.SYS
000001
```

Option 6 tells VPLUS to extend terminal status reads to ensure that an input termination character is accounted for.

If you are using a VT-MGR connection you should set option 6. If you don't set option 6, the keyboard might become locked.

If you are using a TELNET or DTC connection you should *not* set option 6. If you do, a call to VCLOSETERM might result in an error 10.

If you are using ACUCOBOL-GT VPLUS programs with a variety of terminal types, you might want to create environment control files for each terminal type and then use a FILE equation to match the terminal type with the correct environment control file. For example:

```
:FILE VENVCNTL.PUB.SYS=file.group.account
```

where "VENVCNTL.PUB.SYS" is always fully specified and "*file.group.account*" is the name of the file containing your connection specific settings.

For HP documentation on this topic, see http://docs.hp.com/mpeix/onlinedocs/32209-90024/32209-90024.html and navigate to Application Notes/Using the VPLUS Environment Control File (VENVCNTL).

4.2.7 The libutil Utility

libutil is a special utility that allows you to easily convert KSAM COPY libraries into text format COPY libraries. The utility also allows you to create individual COPY files from KSAM or text format COPY libraries. **libutil** has several options.

To create individual COPY files from COPY libraries the usage is:

MPE:

```
libutil "-x[b] [copylib [group]]”
```

POSIX:

```
libutil  -x[b] [copylib [group]]
```

where:

copylib is a KSAM or text file. *copylib* defaults to COPYLIB.

group is the name of an existing group or directory. *group* defaults to the current group or directory.

If only ‘-x’ is specified, the output file type is MPE fixed ASCII. If ‘-xb’ is specified, the output file type is HFS bytestream ASCII. Use the “-xb” option if your copy libraries have copyfile names with hyphens in them.

One COPY file is created for each entry in the COPY library. The name of each COPY file is derived from the name of the entry in the COPY library. For example, if you want to extract all of your COPY files from a COPY library called MYLIB into a group called MYCPY in an account named ACUCOBOL, from MPE you can use the command:

```
libutil "-x MYLIB MYCPY.ACUCOBOL”
```

To create a text file version of a KSAM COPY library the usage is:

MPE:

```
libutil "-c[b] [KSAM-copylib [ASCII-copylib]]”
```

POSIX:

```
libutil -c[b] [KSAM-copylib [ASCII-copylib]]
```

where:

KSAM-copylib defaults to COPYLIB; and

ASCII-copylib defaults to COPYLIBA.

If only '-c' is specified, the output file type is MPE fixed ASCII. If '-cb' is specified, the output file type is HFS bytestream ASCII. Because some HP editors cannot edit files that do not have sequential line numbers, **libutil** resequences the line numbers in the output file, starting with 000001 and incrementing by one. For example, if you want to convert a KSAM COPY library called MYLIB into a text file called MYLIBA, from MPE you can use the following command:

```
libutil "-c MYLIB MYLIBA”
```

For more information on how libraries are used by the ACUCOBOL-GT compiler, see **section 4.3.1, "COPY Statement."**

4.3 The "-Cp" HP COBOL Compatibility Switch

The "-Cp" compiler switch allows ACUCOBOL-GT to accept HP COBOL extensions to the ANSI 1985 Standard. These extensions are described in the following subsections, organized in the order in which the extension occurs in a COBOL program.

The use of the "-Cp" option with the ACUCOBOL-GT compiler triggers the following HP COBOL-specific behavior.

- In addition to reading standard flat files, ACUCOBOL-GT also accepts Qedit format source files.

- The ACUCOBOL-GT precompiler for HP COBOL is invoked. The precompiler supports conditional compilation via the $IF, $SET, $COMMENT, $PREPROCESSOR, $DEFINE, and $INCLUDE directives. Macro substitutions with parameters are supported. $CONTROL directives are not supported. However, many of the actions accomplished by $CONTROL directives can be accomplished with ACUCOBOL-GT compiler switches. The precompiler is discussed in detail in section **section 4.4**.

- STRING/UNSTRING statements and relation conditions are accepted.

- The following HP COBOL extensions are accepted:

  - The percent character, “%”, is allowed as an indicator of an octal-based numeric literal.
  - HP extensions to the COPY statement, including those that copy a file from a resident library. COPY REPLACING statements within nested Copy books are not supported.
  - Special Names Paragraph switches “SW0” through “SW15”
  - The Special Names TOP and NO SPACE CONTROL phrases with the ADVANCING clause of the WRITE statement
  - The HP meaning of RANDOM in the File-Control paragraph
  - The Procedure Division registers: CURRENT-DATE, TALLY, TIME-OF-DAY, and WHEN-COMPILED
  - HP extensions to the ACCEPT statement
  - HP extensions to the CALL statement
  - The ENTRY statement
  - The EXAMINE statement

- The following HP COBOL extensions are not supported. These extensions are not commonly used and are largely considered to be obsolete.

  The following extensions cause ACUCOBOL-GT to issue a warning but do not cause compilation to terminate:

  - COBOLLOCK
  - COBOLUNLOCK
  - EXCLUSIVE
  - UN-EXCLUSIVE

  These unsupported extensions are not recognized by the compiler and cause compilation to terminate:

  - BEGINNING

- COMMON
- ENABLE
- ENDING
- FILE-LIMITS
- MORE-LABELS
- PROCESSING
- SEEK

Successful compilation results in the creation of an ACUCOBOL-GT object file that is ready for immediate execution by the runtime. There is no link step.

Support for HP COBOL extensions and the call interface to HP e3000 system intrinsics is built directly into the HP e3000 ACUCOBOL-GT compiler and runtime. ACUCOBOL-GT supports KSAM, IMAGE, and VPLUS through the HP e3000 intrinsic procedures and functions specific to those components. KSAM can also be used with standard COBOL I/O (see **section 4.2.4, "Accessing MPE KSAM, Relative, and Sequential Files"**). For more information about ACUCOBOL-GT support for system intrinsics, see **section 4.5, "System Intrinsics."**

Programs compiled with "-Cp" and run on the HP e3000 have the following MPE-specific behaviors and restrictions:

- Files referenced by OPEN OUTPUT statements must have MPE filenames.
- OPEN OUTPUT statements create MPE format files.
- MPE file locking is used.

4.3.1 COPY Statement

The HP COBOL syntax of the COPY statement is:

```
    COPY source-file [ {OF} library-name ] [ NOLIST ]
```

```
                           {IN}

     [ REPLACING  { { old-text } BY { new-text } } ... ].
```

If the “-Cp” option is specified, ACUCOBOL-GT supports the HP COBOL variant of the COPY statement that copies a file from a resident library. In this usage, *library-name* is a one to eight character alphanumeric name that specifies the resident library in which the source file is located. This library is a KSAM or text file that contains one or more source files.

If *library-name* is not specified, the compiler assumes that the library name is COPYLIB. If *source-file* cannot be found in the library, then ACUCOBOL-GT checks to see if *source-file* is in the current directory.

Note: HP COBOL assumes the name of the source file is in uppercase. If an uppercase name is not found in the current directory, the rules that ACUCOBOL-GT uses for *source-file* and *library-name* will be used (described below).

If the “-Cp” option is not specified, the standard rules that ACUCOBOL-GT applies to *source-file* and *library-name* are used. This means that *library-name* is interpreted as a directory name. If you have a resident library, you will need to unload its contents. In the working directory in which you perform compilations, create a subdirectory of the same name and case as *library-name*. Unload the library’s source files into this subdirectory using the same name and case as specified by *source-file*. The next time you compile, ACUCOBOL-GT will find *source-file* in the directory you created. For example, if your COBOL COPY statement reads:

```
COPY support IN common
```

During compilation, ACUCOBOL-GT will translate that statement into:

```
COPY "./common/support"
```

If *library-name* is not specified, ACUCOBOL-GT checks to see if *source-file* is in the current directory. If *source-file* is not found in the current directory, the rules that ACUCOBOL-GT uses for finding *source-file* are applied. (For more information about ACUCOBOL-GT’s treatment of the COPY statement and the use of the COPYPATH environment variable for locating copy files, see the *ACUCOBOL-GT User’s Guide*, section 2.5, “COPY Libraries”.)

NOLIST is supported by ACUCOBOL-GT in HP COBOL compatibility mode. NOLIST has the same meaning as SUPPRESS in ACUCOBOL-GT's default mode. See the *ACUCOBOL-GT Reference Manual*, section 2.4.1, "The COPY Statement," for details.

old-text and *new-text* have the same meaning in HP COBOL as in ACUCOBOL-GT. See the reference manual, section 2.4.1, "The COPY Statement," for details.

Note: COPY REPLACING within nested Copy books is not supported.

4.3.2 Special-Names Paragraph

This section discusses ACUCOBOL-GT's compatibility with the non-ANSI features of HP COBOL that affect the Special-Names Paragraph of the Environment Division.

4.3.2.1 Program switches

ACUCOBOL-GT in its HP COBOL compatibility mode allows you to specify the HP COBOL program switches in the Special-Names Paragraph. HP COBOL specifies those switches as "SW0" through "SW15". ACUCOBOL-GT in the default mode uses "SWITCH-1" through "SWITCH-26".

4.3.2.2 TOP and NO SPACE CONTROL

With ACUCOBOL-GT in its HP COBOL compatibility mode, you can include TOP and NO SPACE CONTROL as Special Names in the ADVANCING clause of the WRITE statement. These Special Names can be used in Format 1 of the WRITE statement when the program is accessing sequential files.

Format 1

```
WRITE record-name [ FROM source ] [ {BEFORE} ADVANCING {mnemonic-name} ...]
                                    {AFTER }
```

Syntax Rules

1. *record-name* is the name of a record associated with a file described in the File Section of the Data Division. The associated file may not be a sort file. (See the *ACUCOBOL-GT Reference Manual*, section 6.6, "Procedure Division Statements," WRITE Statement.

2. *source* is a data item or literal. It may not share any storage area with *record-name*. (See the *Reference Manual*, section 6.6, "Procedure Division Statements," WRITE Statement.)

3. *mnemonic-name* is a user-defined word that may be used to change the state of the associated program switch or to refer to a device. (See the *Reference Manual*, section 4.2.3, "Special Names Paragraph," syntax rule 2.)

General Rules

1. *mnemonic-name* assigned to TOP advances the line to the next page boundary (same as specifying PAGE).

2. *mnemonic-name* assigned to NO SPACE CONTROL suppresses spacing and keeps the line printer from advancing.

TOP and NO SPACE CONTROL are *system-names* accepted by ACUCOBOL-GT in HP COBOL compatibility mode. (See the *Reference Manual*, section 4.2.3, "Special Names Paragraph," syntax rule 3.)

4.3.2.3 CONDITION-CODE

As with HP COBOL, CONDITION-CODE is set to the condition code returned by an MPE/iX intrinsic function when the function is called with the CALL statement. It can also be set by a called C program if the following code is included in the C program.

```
Extern   int32_t   Ahp_ccode;

   Ahp_ccode = 0;
```

CONDITION-CODE takes the value of "Ahp_ccode" in the COBOL program.

4.3.3 File-Control Paragraph

This section discusses ACUCOBOL-GT's compatibility with the non-ANSI features of HP COBOL that affect the File-Control Paragraph of the Environment Division.

4.3.3.1 RANDOM and DYNAMIC keywords

In HP COBOL compatibility mode, the keyword RANDOM means the same as the keyword DYNAMIC (RANDOM loses its default ACUCOBOL-GT meaning). For example, either

```
    ACCESS IS RANDOM
```

or

```
    ACCESS IS DYNAMIC
```

can be used to specify DYNAMIC access to a file. See section 4.3.1, "File-Control Paragraph" in Book 3 of the ACUCOBOL-GT documentation set, *Reference Manual,* for usage syntax and general rules for the keyword DYNAMIC.

When compiling for object versions 8.1 and greater, the compiler accepts the ACTUAL KEY feature for relative files opened in RANDOM access mode. For such files, the relative key numbers will be zero-based, rather than one-based, as with RELATIVE KEY. For example, if you have a relative file with a fixed record length of 3 bytes and a relative file with the following contents:

```
AAABBBCCC
```

The record keys for the different modes are:

```
     RELATIVE  ACTUAL
AAA:    1        0
BBB:    2        1
CCC:    3        2
```

4.3.3.2 WITH DUPLICATES on primary keys

In HP COBOL compatibility mode, duplicate primary key values are allowed if the indexed file system supports them and the WITH DUPLICATES phrase is specified in the File-Control paragraph. Vision and KSAM support duplicate primary key values. The syntax is:

```
RECORD KEY IS key-name [= seg-name] [WITH [NO] DUPLICATES]
```

The phrase “WITH NO DUPLICATES” is commentary because, by default, duplicate values are not allowed for the primary key.

The phrase “WITH DUPLICATES” indicates that duplicate primary key values are allowed. If “WITH DUPLICATES” is used with a file system that doesn’t support duplicate primary keys, when the file is created via the OPEN statement a status of “0M” is returned, indicating that the file was successfully created but that duplicate primary keys are not supported.

When “WITH DUPLICATES” is used with Vision, KSAM, and other file systems that support it, the rules that govern how REWRITE and DELETE operations are handled are determined by the file system. This is because support for duplicate primary keys is not part of the ANSI 1985 Standard, few files systems support it, and therefore no common rules exist. The rules for Vision are given in General Rule 14 of section 4.3.1, “File-Control Paragraph,” of the *ACUCOBOL-GT Reference Manual*.

There are some small but important differences between how KSAM and Vision handle REWRITE and DELETE when duplicate primary keys are allowed.

Under KSAM, the first record is rewritten or deleted unless the file is open with ACCESS SEQUENTIAL mode. These latter programs should convert easily to Vision because Vision’s default lock handling will cause the correct record to be locked and result in the same set of records being updated.

For RANDOM and DYNAMIC access files, there could be a difference between how KSAM and Vision update records depending on how the records are read before they are rewritten or deleted. (Note that the HP COBOL II Reference Manual advises against allowing duplicates primary key values with RANDOM or DYNAMIC access files.) KSAM’s behavior can be mimicked with Vision by always reading the record immediately before rewriting or deleting it. This ensures the correct record is accessed.

Otherwise, KSAM and Vision rules are generally compatible unless the program does a series of READ NEXT's and then rewrites or deletes a record other than the one last read.

4.3.4 Procedure Division Register Extensions

HP COBOL supports special register words. These words reference memory storage that is generated at compile time and initialized at compile time or run time. They are treated as reserved words by ACUCOBOL-GT when it is in HP COBOL compatibility mode. In that mode, you cannot use any of the HP COBOL-specific special register words as user-defined words.

4.3.4.1 CURRENT-DATE

This register stores the current date. CURRENT-DATE can be used with a MOVE or DISPLAY statement.

Format

```
    MOVE CURRENT-DATE TO {dest-item} ...
    DISPLAY CURRENT-DATE [ UPON new-window ] ...
    DISPLAY CURRENT-DATE [ UPON mnemonic-name ] ...
```

The value of CURRENT-DATE is stored in an 8-character alphanumeric field in the format "MM/DD/YY", where "MM" is the month (01 for January, 02 for February, etc.), "DD" is the day of the month, and "YY" is the last two digits of the year. For example, the date February 15, 2001 is formatted as "02/15/01".

4.3.4.2 TALLY

See **section 4.3.5.3, "EXAMINE statement,"** for information about the special register TALLY.

4.3.4.3 TIME-OF-DAY

This register stores the current time of day. TIME-OF-DAY can be used with a MOVE or DISPLAY statement.

Format

```
MOVE TIME-OF-DAY TO {dest-item} ...
DISPLAY TIME-OF-DAY [ UPON new-window ] ...
DISPLAY TIME-OF-DAY [ UPON mnemonic-name ] ...
```

The value of TIME-OF-DAY is stored in the format “hhmmss”, where “hh” is the hour, “mm” is the minute, and “ss” is the second, based on a 24-hour clock. For example, 5:30 p.m. is stored as “173000”.

The DISPLAY output of TIME-OF-DAY is edited to include colons (“:”) as separators, so the displayed format is “hh:mm:ss”. For example, 5:30 p.m. is displayed as “17:30:00”.

4.3.4.4 WHEN-COMPILED

This register stores the date and time that the program was compiled. It can be used with a MOVE or DISPLAY statement.

Format

```
MOVE WHEN-COMPILED TO {dest-item} ...
DISPLAY WHEN-COMPILED [ UPON new-window ] ...
DISPLAY WHEN-COMPILED [ UPON mnemonic-name ] ...
```

The value of WHEN-COMPILED is stored in the format “MM/DD/YY hh:mm:ss”.

4.3.5 Procedure Division Statements

This section discusses ACUCOBOL-GT’s compatibility with the non-ANSI features of HP COBOL that affect the Procedure Division.

The HP COBOL Procedure Division supports special formats of the ANSI-standard statements, as well as non-ANSI verbs. They are presented here in alphabetical order.

4.3.5.1 ACCEPT statement

CONVERT phrase

HP COBOL programs allow numeric or numeric edited receiving fields on the ACCEPT statement by default.

With ACUCOBOL-GT in HP COBOL compatibility mode, the CONVERT phrase is implied on numeric ACCEPT statements. When the "-Cp" option is specified, the compiler sets the variable AUTO_CONVERT to "TRUE".

ACUCOBOL-GT, in the default mode, requires that you use the CONVERT phrase on the ACCEPT statement, or use the "-Vc" compiler option that sets the variable AUTO_CONVERT to "TRUE".

See ACCEPT statement, Format 1, syntax rule 1, in section 6.6 of Book 3, *Reference Guide*, for details.

FREE phrase

The ACUCOBOL-GT compiler in its HP e3000 compatibility mode supports the following HP COBOL formats of the ACCEPT statement.

Note: Format 10, **Format 11**, and **Format 12** are the numbers given to the ACCEPT statement formats. ACUCOBOL-GT formats of the ACCEPT statement are documented in the *ACUCOBOL-GT Reference Manual,* section 6.6, "Procedure Division Statements."

Format 10 *(HP COBOL)*

```
ACCEPT identifier [FREE]  [FROM { SYSIN        }]
                                { CONSOLE      }
                                { mnemonic-name }
```

Format 11 *(HP COBOL)*

```
ACCEPT identifier FREE    [FROM { SYSIN        }]
                                { CONSOLE      }
                                { mnemonic-name }

[ON INPUT ERROR imperative-statement-1]
```

```
[NOT ON INPUT ERROR imperative-statement-2]

[END-ACCEPT]
```

Format 12 *(HP COBOL)*

```
ACCEPT identifier FROM { DATE        }
                       { DAY         }
                       { DAY-OF-WEEK }
                       { TIME        }
```

Syntax Rules

1. *identifier* is a data item that receives the accepted data.
2. *mnemonic-name* is a user-defined word declared in Special-Names that refers to a display device, or is the name of a display device. See the entry for ACCEPT in the *ACUCOBOL-GT Reference Manual*, section 6.6, “Procedure Division Statements.”
3. *imperative-statement-1* and *imperative-statement-2* are COBOL statements. The INPUT ERROR phrase in which the statement appears can be used only if the FREE phrase is used. (See General Rule 3.)

General Rules

1. CONSOLE and SYSIN are system devices. See *Reference Manual*, section 4.2.3, “Special-Names Paragraph.”
2. The FREE phrase allows the use of the free-field format. The free-field format allows you to continue the input into the next line when the input field is smaller than the defined length of identifier. In this format, the ampersand character (“&”) acts as a continuation character when it is the last non-blank character in the input line, thus allowing the input to be continued in the next line. The pound sign (“#”) acts as a line terminator (but is not required when the input comes from the terminal). To specify the “#” character and circumvent its use as a line terminator, use two “#” characters in succession.
3. The ON INPUT ERROR and NOT ON INPUT ERROR phrases can be used only with the FREE statement. They allow the handling of the following input error conditions:

- An illegal character in a numeric item.
- Too many digits in a numeric item.
- An input string that is too long for the receiving field.

4.3.5.2 CALL statement

The ACUCOBOL-GT compiler in its HP e3000 compatibility mode supports the following ***HP COBOL*** extensions to the CALL statement.

Note: **Format 4** is the number given to the HP CALL statement extensions. Format 4 is described below. Other CALL statement formats are described in section 6.6, "Procedure Division Statements," in the *ACUCOBOL-GT Reference Manual*.

Format 4 *(HP COBOL)*

```
CALL { identifier-1           } [ USING { \\             } ... ]
     { [ INTRINSIC ] literal-1 }         { @identifier-2  }
                                         { identifier-2   }
                                         { literal-2      }
                                         { \identifier-2\ }
                                         { \literal-2\    }
     [ GIVING identifier-n ]

     [ ON {EXCEPTION} statement-1 ]
          {OVERFLOW }

     [ NOT ON {EXCEPTION} statement-2 ]
              {OVERFLOW }

     [ END-CALL ] ]
```

Syntax Rules

1. *Identifier-1* is an alphanumeric data item whose value is a program name. See General Rules (below) for a description of conditions that may occur when you use this operation.

2. *Literal-1* is a nonnumeric literal whose value is either an operating system intrinsic name or a program name. When you are naming an intrinsic, *literal-1* must be preceded by the keyword INTRINSIC.

3. "\\" represents a null value passed as a parameter to an intrinsic or to a System Programming Language (SPL) program that includes the OPTION VARIABLE clause. When the intrinsic option is not specified, the compiler assumes a one-word parameter.

4. *@identifier-2* indicates that the byte address of the data item represented by *identifier-2* is to be passed as a parameter. These types of parameters may be passed only to non-COBOL programs.

5. *Identifier-2* is the name of any data item in the calling program, or is a file named in an FD-level entry in the File Section of your program.

6. *Identifier-2* cannot be a function-identifier (i.e., it cannot reference an intrinsic function).

General Rules

1. When you use the *identifier-1* form of the CALL statement, the value of *identifier-1* determines which subprogram is called. When the program is executed, an attempt is made to load the subprogram. If the load fails, an exception condition occurs.

2. *Identifier-2* must be described in the File, Working-Storage, or Linkage section of your program. When this value is passed to another program, it is passed by reference.

3. If *identifier-2* names a file, the called program must not be a COBOL program.

4. *\identifier-2\* and *\literal-2\* indicate that the literal or data item enclosed in backslashes is to be passed by value to the called program or intrinsic. This may be used only when the called program is not a COBOL program. If an identifier is used in this way, it must represent a numeric data item of not more than 18 digits.

5. *Identifier-n* is the name of a binary data item in the calling program. It is used in calls to non-COBOL programs and in calls to COBOL programs that return a value in the RETURN-CODE special register. *Identifier-n* cannot be a function-identifier (i.e., it cannot reference an intrinsic function).

CALLing intrinsics

The INTRINSIC phrase is used to indicate that the CALL statement is calling an operating system intrinsic function, rather than a subprogram. When the INTRINSIC phrase is used, *literal-1* must be used and must name an operating system intrinsic. The USING phrase specifies the various parameters to be passed to the intrinsic. When the intrinsic is a "typed procedure," the GIVING phrase specifies the parameter to be returned by the intrinsic.

As with subprograms, the parameters passed to intrinsics are specified by position. If a parameter of an intrinsic is optional, and you do not want to pass a value for that parameter, you must specify two consecutive backslashes ("\\") in the position within the USING phrase of the CALL statement that corresponds to that parameter's position.

Unlike subprograms, if an intrinsic expects a parameter to be passed by value rather than by reference, you don't need to enclose the literal or identifier in backslashes. The intrinsic automatically assumes that the parameter is being passed by value.

The special register CONDITION-CODE holds the condition code returned by the intrinsic function. The following special relation operators can be used after the call to check the condition code:

```
mnemonic-name { [ NOT ] { <  } } 0
              {         { =  } }
              {         { >  } }
              {         { >= } }
              {         { <= } }
              {         { <> } }
```

When CALL INTRINSIC is used for intrinsics in a system file:

- the special symbols "@" and "\" are optional.
- data conversions for parameters passed by value are performed automatically.

Note: On the HP e3000, ACUCOBOL-GT supports calls to most system intrinsic functions. Nothing in ACUCOBOL-GT's CALL interface limits support for intrinsics. However, we have not tested support for all intrinsic functions. In particular we have not tested the more obscure functions or variants of functions that take unusual or undocumented parameter types. Should you encounter unexpected behavior, please contact **our Technical Services**.

USING phrase (non-COBOL subprograms)

Data is passed from the calling program to the called program on a positional basis, rather than by name. Therefore, the third name in a USING phrase of a calling program corresponds to the third name in the arguments of the called program.

This positional correspondence extends to non-COBOL called programs. Thus, for example, if the called program is a Pascal program, then the names in the parameter list of the procedure declaration in that program are identified with those data items whose names appear in the corresponding position of the USING phrase in the calling program.

As stated above in the description of *identifier-2*, *identifier-3*, and so forth, these identifiers may name files to be passed to a called program. Furthermore, although you can enclose such a file identifier between backslashes (which are ignored), preceding it with an “@” symbol results in an error.

If the file name from the FD is passed, the file number of that file is passed by value. If the subprogram is an SPL procedure, the procedure parameter corresponding to the file name must be declared as type INTEGER or LOGICAL, and it must be specified as a value parameter. The file must be opened in the calling program.

To pass a data item by value, you must enclose the associated identifier in backslashes. If the value passed is a literal, the backslashes are optional. Passing a data item by value leaves the data item in the calling program unchanged following execution of the called program.

If an identifier is not passed by value (that is, is not enclosed in backslashes), it is passed as a byte pointer (that is, by reference). Thus, the data in the calling program can be altered by the called program if it is passed in this manner. In calls to COBOL programs, this is the standard method of referencing common data.

Two consecutive backslashes ("\\") may be specified in a USING phrase of a CALL statement if the called program is a Pascal procedure with OPTION DEFAULT_PARMS or EXTENSIBLE. Using two consecutive backslashes indicates that a parameter is not being sent and should not be expected.

Whenever an OPTION VARIABLE SPL procedure is called, an additional parameter must be added to the end of the USING parameter list. This parameter is called a *bit mask* and is used to tell the SPL procedure which parameters are being passed. The bit mask consists of one or two 16-bit binary words, where a "0" represents a missing parameter and a "1" represents an existing parameter (this allows up to 32 parameters to be passed). A parameter in the bit mask must be a numeric data item, and it represents the value derived from the bit mask.

Parameters are matched, starting from the right, in both the bit mask and the USING list, excluding the value in the USING list used for the bit mask parameter. For example,

```
CALL "SPLPROC" USING \TESTER\ \\ @RESULT \ERROR\ \%13\
```

The bit mask in this case is 0000000000001011, which is represented by the octal value "\%13\", showing that the fourth, third, and first parameters are being passed, while the second parameter is not being passed.

The bit mask is generated automatically by the compiler if you specify the INTRINSIC option.

Passing file handles to subroutines

In HP COBOL compatibility mode, you can pass the operating system's file handle for an open COBOL file to a subroutine by referring to that file's SELECT name in the CALL statement. For example:

```
ENVIRONMENT DIVISION.
FILE-CONTROL.
SELECT MY-FILE
ASSIGN TO DISK
```

```
SEQUENTIAL.

DATA DIVISION.
FILE SECTION.
FD MY-FILE.
01 RECORD-1     PIC X(80).

PROCEDURE DIVISION.
MAIN-LOGIC.
   OPEN INPUT MY-FILE.
   CALL "SUB" USING MY-FILE.
   CLOSE MY-FILE.
```

One reason for doing this is to call an operating system function that allows the file to retrieve some information that is not available through COBOL.

Several special rules apply:

1. For compatibility with HP COBOL, a file handle is automatically passed BY VALUE unless it is immediately preceded by a BY REFERENCE or BY CONTENT specification. You will need to do this if you pass an open file handle to a COBOL subroutine because COBOL routines cannot take BY VALUE parameters.
2. If the called subroutine is a COBOL routine, the handle passed is PIC S9(4) COMP-5. You can override this with the compiler option "--fileIdSize=#" where "#" is either "2","4" or "8" to specify the number of bytes you want in the passed integer.
3. If the called subroutine is not COBOL, the handle is passed as a signed native integer using the host's default integer size.
4. The file handle passed is the host file system's identifying value for the open file. For all current implementations, this is the value returned by the C "open" function. This may change in a future implementation.
5. If the host file system does not have this information available, then "-1" is used instead. This can happen if the host system is not a file (e.g. Acu4GL for Oracle) or the host system does not provide a way of obtaining the handle (e.g. C-ISAM interface). Files served by AcuServer also use "-1" since there is no useful way to use a remote process' open file handle.

6. For Vision files in the multi-file format, the handle used is the handle of the first data segment (this is the same file specified by the file name used when opening the file).

7. It is best to avoid performing actual I/O on the file using this file handle because the COBOL file system will be unaware of any state changes to the file and may perform incorrectly. It is possible to corrupt data this way.

Recursive CALLs

In ACUCOBOL-GT, as in HP COBOL, a program may directly or indirectly call itself. Such a CALL statement is termed a *recursive call*. By default, ACUCOBOL-GT creates a new copy of data for each instance of the program. To share the initial data with all recursive calls, as is the default with HP COBOL, set the RECURSION_DATA_GLOBAL configuration variable. For more information, see the descriptions of the RECURSION and RECURSION_DATA_GLOBAL configuration variables in Appendix H of the ACUCOBOL-GT documentation set. See also section 2.9 in the *ACUCOBOL-GT User's Guide.*

4.3.5.3 EXAMINE statement

The EXAMINE statement is used to count the number of occurrences of a given character in a data item. ACUCOBOL-GT accepts the EXAMINE statement in the ***HP COBOL*** compatibility mode. HP COBOL's EXAMINE statement duplicates some of the functionality of the ACUCOBOL-GT Format 1 INSPECT statement, but with slightly different syntax. (See the *ACUCOBOL-GT Reference Manual*, section 6.6, "Procedure Division Statements," for a detailed discussion of the INSPECT statement.)

Format 1

```
EXAMINE identifier-1 TALLYING {ALL        } literal-1 [REPLACING BY literal-2]
                              {LEADING    }
                              {UNTIL FIRST}
```

Format 2

```
EXAMINE identifier-1 REPLACING {ALL        } literal-1 BY literal-2
                               {FIRST      }
                               {LEADING    }
                               {UNTIL FIRST}
```

Syntax Rules

1. *identifier-1* is the name of a data item containing characters to be counted or replaced. Its USAGE must be DISPLAY (implicitly or explicitly).

2. Each *literal* is a single-character literal or one of the following figurative literals:

   ZERO
   ZEROS
   ZEROES
   SPACE
   SPACES
   QUOTE
   QUOTES
   LOW-VALUE
   LOW-VALUES
   HIGH-VALUE
   HIGH-VALUES

3. ACUCOBOL-GT does not enforce the one-character limitation. If a string literal has two or more characters, it uses only the first character.

General Rules

1. The number of characters in Format 1 that obey certain conditions are counted by a special register named TALLY. Its PICTURE is 9(5) and USAGE is COMP-N. Its default value is zero.

2. The characters to be counted and replaced are determined by the key words following TALLYING or REPLACING. Counting is done only in Format 1, and replacement is done only in Format 2 and when specified in Format 1.

Key Words	**Characters To Be Counted And Replaced**
ALL	Count and/or replace all instances of *literal-1* with *literal-2*.

FIRST	Replace the first (left-most) instance of *literal-1* with *literal-2*.
LEADING	Count and/or replace all instances of *literal-1* that appear before (to the left of) any other character in the data item.
UNTIL FIRST	Count and/or replace all characters to the left of the first (left-most) instance of *literal-1* with *literal-2*. If *literal-1* does not appear in the data item, count and replace all characters in the data item.

3. *Literal-1* is the character to be counted. *Literal-2* is the character that will replace *literal-1*. Note that the substitution of *literal-2* for characters other than *literal-1* occurs only when UNTIL FIRST is specified.

4. The contents of the TALLY register are unchanged in Format 2. In Format 1, its contents are replaced by the appropriate character count, regardless of the previous contents.

4.3.6 Conversion Issues

This section addresses source code conversion issues.

4.3.6.1 Unsupported HP COBOL extensions

The HP COBOL extensions are said to be *unsupported* by ACUCOBOL-GT when they are recognized at compile time but ignored at runtime. Such extensions, when present, cause the compiler to return a warning message indicating that this is an unsupported operation. They do not prevent the compiler from creating an object.

The following HP COBOL extensions are not supported by ACUCOBOL-GT in HP COBOL compatibility mode:

COBOLLOCK
COBOLUNLOCK
EXCLUSIVE
UN-EXCLUSIVE

Note: These extensions are currently listed as Reserved Words in the HP COBOL II/XL Reference Manual, Appendix F.

4.3.6.2 Unrecognized HP COBOL extensions

The HP COBOL extensions are said to be *unrecognized* by ACUCOBOL-GT when they prevent the compiler from successfully compiling the COBOL source. Statements that include these words cannot be compiled. The words must be removed or commented out.

The following HP COBOL extensions are not recognized by ACUCOBOL-GT in HP COBOL compatibility mode:

BEGINNING
COMMON
ENABLE
ENDING
FILE-LIMIT
FILE-LIMITS
MORE-LABELS
PROCESSING
SEEK

4.3.7 Operating System and Runtime Limitations and Differences

The following sections describe some notable differences between the way HP COBOL and ACUCOBOL-GT handle certain aspects of program execution.

4.3.7.1 *ACCEPT FROM INPUT STATUS* statement

MPE/iX does not support the "ACCEPT FROM INPUT STATUS" statement due to limitations of the MPE/iX operating system. The lack of this functionality can cause problems for multi-threaded COBOL applications. If the application executes an ACCEPT in one thread, the runtime system is unable to task-switch to any other thread until the ACCEPT terminates.

A possible workaround for this limitation is to change the ACCEPT statement to use the BEFORE TIME clause. This causes the ACCEPT statement to time out which then allows the program to switch to another thread.

4.3.7.2 Divide by zero

At run time, DIVIDE statements that do not include an ON SIZE ERROR phrase and that produce a division by zero error are handled differently by HP COBOL and ACUCOBOL-GT. Programs compiled with HP COBOL output an error message and terminate. The behavior of programs compiled with ACUCOBOL-GT is undefined. In some cases the program does not terminate. It is strongly recommended that you always use the ON SIZE ERROR phrase with any arithmetic statement that could generate a size error.

4.3.7.3 File I/O error handling

In most cases, HP COBOL and ACUCOBOL-GT handle file I/O errors in exactly the same way. Specifically, if the program includes a USE statement procedure, an INVALID KEY phrase, or an AT END phrase, and a file I/O error occurs, the appropriate code is executed. However, when a FILE STATUS variable is specified with a file, the behavior of the two COBOLs can differ. For example, when a FILE STATUS variable is specified and an OPEN INPUT statement attempts to open the file but the file does not exist, HP COBOL sets the FILE STATUS item and the program continues to execute. In contrast, ACUCOBOL-GT sets the FILE STATUS item and halts the program. ACUCOBOL-GT can be made to behave like HP COBOL by setting the ERRORS_OK configuration variable. When ERRORS_OK is set to a value of "1", file I/O errors are ignored. The result being that when ACUCOBOL-GT attempts to OPEN a file that doesn't exist, FILE STATUS is set and the program continues to execute. For complete information on the ERRORS_OK configuration variable, see Appendix H of the ACUCOBOL-GT Manual Set.

4.3.7.4 File name case

HP COBOL and ACUCOBOL-GT treat file name case differently. When HP COBOL creates a new file via the OPEN OUTPUT statement, the name of the file (as specified in the ASSIGN TO phrase) is always created in upper case. For example:

```
ASSIGN TO "myfile”
.
.
.
OPEN OUTPUT . . .
```

creates a file with the name “MYFILE”. In ACUCOBOL-GT, the name of the file retains the case specified in the ASSIGN TO phrase (in the example above, “myfile”). This difference in behavior can produce unexpected results (for example, although “MYFILE” may exist, ACUCOBOL-GT won’t find it; instead it creates a new file named “myfile”). You can make ACUCOBOL-GT behave like HP COBOL by setting the FILE_CASE configuration variable. The value of FILE_CASE causes the case of data file names to be adjusted at run time. Accepted values for FILE_CASE include:

2	Data file names are translated to upper case.
1	Data file names are translated to lower case.
0	(default) No case translation is performed.

For complete information on the FILE_CASE configuration variable, see Appendix H of the ACUCOBOL-GT documentation set.

4.3.7.5 Mismatched EXTERNAL data items

There is a small, but significant incompatibility in the way that HP COBOL and ACUCOBOL-GT treat the indexes of external tables. In ACUCOBOL-GT, indexes attached to an external table are also external. As a result, all programs that share an external table must also have matching indexes for the table. (Note that the 1985 ANSI standard leaves the handling of INDEXED BY data items up to the implementor.)

The incompatibility exhibits itself at runtime. HP COBOL programs compiled with ACUCOBOL-GT using the "-Cp" option, and that share an external table but that do not have identical indexes for that table, will get the following runtime error:

```
Mismatched EXTERNAL data item: data_item_name
```

The problem is illustrated by the following example. If the called program defines:

```
01 main-01 external.
      03 item-01   pic x occurs 20
                        indexed by idx1.
01 redef1 redefines main-01.
      03 item-01   pic x occurs 20
                        indexed by idx2.
```

and the calling program defines:

```
01 main-01 external.
      03 item-01  pic x occurs 20
                       indexed by idx1.
```

the "Mismatched EXTERNAL data item" error is raised at run time. Although "redef1" is a REDEFINES of "main-01", "idx2" is still a new data item and it must be present in all of the programs that share "main-01" or the index must be eliminated. Note that if a SEARCH statement references "redef1", "idx2" must be present.

The following code does not raise the error. The called program defines:

```
01 main-01 external.
      03 item-01  pic x occurs 20
                       indexed by idx1.
01 redef1 redefines main-01.
      03 item-01  pic x occurs 20
                       indexed by idx2.
```

and the calling program defines:

```
01 main-01 external.
      03 item-01  pic x occurs 20
```

```
                                    indexed by idx1, idx2.
```

4.4 Preprocessor for HP COBOL

The ACUCOBOL-GT compiler includes a preprocessor that is written specifically for providing compatibility with HP COBOL programs. This preprocessor handles conditional compilation with $IF, $SET, $COMMENT, $PREPROCESSOR, $DEFINE, and $INCLUDE directives, and macro calls with parameters. When you specify the "-Cp" option, ACUCOBOL-GT automatically invokes the preprocessor for HP COBOL.

Note: The $CONTROL directives are ignored by the ACUCOBOL-GT preprocessor for HP COBOL. The $CONTROL directives are handled by the compiler command arguments. See **section 4.4.2, "$CONTROL Directive,"** for details.

The compiler determines the source format automatically by examining the first character of the first non-blank line. If that character is blank or a digit, the file is assumed to be an ANSI file, otherwise it is assumed to be in terminal format. You can also specify the format of the source with the "-Sa" or "-St" compile options. See the *ACUCOBOL-GT User's Guide*, section 2.1.7, "Source Options."

The ACUCOBOL-GT preprocessor for HP COBOL currently supports the following directives. They are discussed in more detail below.

$COMMENT
$DEFINE
$IF
$INCLUDE
$PREPROCESSOR
$SET

The $CONTROL directive is not supported by the preprocessor; however, ACUCOBOL-GT provides compile options that handle most of the functionality supported by $CONTROL. See **section 4.4.2, "$CONTROL Directive."**

4.4.1 $COMMENT Directive

Format

```
$COMMENT [comment-text]
```

The preprocessor treats any line that has the $COMMENT directive as whitespace. If the line ends with the line continuation character, (ampersand (“&”) is the default line continuation character), the following line is also treated as a commented line.

4.4.2 $CONTROL Directive

The $CONTROL directive in HP COBOL provides a method for specifying compilation and list options. The $CONTROL directive is not supported by ACUCOBOL-GT and is not included in the compiled object or listing file.

ACUCOBOL-GT uses command line arguments to perform the same functions that $CONTROL directives perform for HP COBOL. The following table correlates HP COBOL $CONTROL options to equivalent ACUCOBOL-GT options. The string “<default>” indicates that the option is enabled by default in ACUCOBOL-GT. To make it easier to specify multiple compiler options, you can use the CBLFLAGS environment variable. For information on the use of CBLFLAGS, see section 2.1.15 in the *ACUCOBOL-GT User’s Guide*.

HP COBOL $CONTROL option	**ACUCOBOL-GT compiler option**
ANSISORT	<none>
ANSISUB	<default>
BOUNDS	-Za
CHECKSYNTAX	-Lf
CODE	<default>
NOCODE	<none>
CROSSREF	-LC
NOCROSSREF	<default>

DEBUG	-Gd
DIFF74	<none>
DIFF74=OBS	<none>
DIFF74=INC	<none>
DYNAMIC	-Zi
ERRORS=number	<none>
LINES=number	-Ll
LIST	-Lf
NOLIST	<default>
LOCKING	<none>
LOCOFF	<default>
LOCON	-Lf
MAP	-Ls
NOMAP	<default>
MIXED	<none>
NOMIXED	<default>
QUOTE = “ ’	<none>
SOURCE	-Lo
NOSOURCE	<default>
STAT74	use the runtime configuration variable: FILE_STATUS_CODES = 74
STDWARN	<none>
NOSTDWARN	<none>
SUBPROGRAM	<none>
SYMDEBUG	-Gy
SYNC16	-D12
SYNC32	-D14
USLINIT	-Di
VERBS	-Lc

NOVERBS	<default>
WARN	-a
NOWARN	-w

4.4.3 $DEFINE Directive

The $DEFINE preprocessor directive associates a string of text with a macro name. When the preprocessor encounters a defined macro name in the source program, it invokes the macro and passes the associated string of text.

The string of text may specify up to nine formal parameters. If the string of text uses parameters, then the macro call can specify actuals for these parameters; otherwise, they are ignored.

Format

```
    $DEFINE macro-name=[string-text]#
```

where:

macro-name is the name of the macro. Macro names begin with a non-alphanumeric character. By default, it is the percent symbol (“%”);

string-text is a string of text to replace occurrences of the macro call within the body of the program code. The string of text may contain formal parameters (which are sometimes referred to as *variables*). These are designated by an exclamation point followed by an integer in the range of 1 to 9. *string-text* may also contain other macro calls, which recursively expand; and

string-text is delimited by the equal sign (“=”) and pound sign (“#”). The pound sign marks the end of the definition. The pound sign is the default delimiter and it can be changed with the $PREPROCESSOR command.

Note: Use of an “&” character is interpreted as part of the macro definition and not as a continuation (as it would in the $COMMENT command).

If the string-text starts on the same line as the macro name, the replacing text is expanded from exactly the same column as the macro call. Otherwise, the replacing text starts at the beginning of the next line.

The two different forms of the macro call are:

```
    macro-name
```

and

```
    macro-name(p1#,p2#,...,pn#)
```

where

“p1”, “p2”, ..., “pn” are actual parameters that replace the formal parameters in the string of text. Each actual parameter can be a null character or any combination of characters, including spaces;

n is an integer in the range of 1 to 9; and

“#” is a delimiter.

Formal parameters that are not specified with an actual parameter in the macro call are ignored. You can specify which formal parameters should be ignored by entering only a pound sign (“#”) in the appropriate position within the macro call. For example, to ignore the first parameter, you write the call like this:

```
    macro-name(#,p2#,...,pn#)
```

4.4.4 $PREPROCESSOR Directive

The default delimiter characters used in macro text strings and macro names are the pound (“#”), percent (“%”), and the exclamation point (“!”). These characters can be changed with the $PREPROCESSOR directive.

Format

```
    $PREPROCESSOR parameter=new-character
       [, parameter=new-character, ...]
```

where:

parameter specifies which character to modify; and

new-character specifies the new character.

The following *parameters* can be modified.

"KEYCHAR" - defines the initial character of a macro name. The default character is "%".

"DELIMITER" - defines the character that delimits the end of macro text strings and the actual parameters used with macro calls. The default character is "#".

"PARMCHAR" - defines the initial character of each formal parameter within macro text string. The default character is "!".

4.4.5 $IF and $SET Directives

The $IF and $SET directives allow for partial compilation of a source file based on the states of software switches. Ten software switches (0 through 9) can be independently changed between binary states ("on" or "off") using the $SET directive. Sections of the source program can then be processed or skipped, depending on the state of any one or a combination of switches used in the $IF directive.

The $IF and $SET directives can be specified on the command line used to invoke the compiler. When specified in that way, the directive must be preceded with a hyphen ("-"). For example:

```
:ccbl "-Cp -$SET X0=ON HELLOCBL"
```

$SET Format

```
$SET [Xn={ ON } [, Xr={ ON }]]
         { OFF}       { OFF}
```

where:

"Xn" and "Xr" are software switches; and

n and *r* are integers in the range of 0 to 9.

All switches are “OFF” by default.

When the $SET directive is used by itself, it sets all switches to “OFF”.

$IF Format

```
$IF [ Xn={ ON } ]
         { OFF}
```

where “Xn” is a software switch.

When the $IF directive is used by itself, it evaluates to *true* and any following program source is processed.

A $IF directive always ends the influence of any preceding $IF directive.

4.4.6 $INCLUDE Directive

The $INCLUDE directive allows you to include a file as part of your source file.

Format

```
$INCLUDE filename
```

where *filename* is the name of the file. The contents of *filename* are inserted beginning where the $INCLUDE statement appears.

4.5 System Intrinsics

On the HP e3000, ACUCOBOL-GT supports calls to most system intrinsic functions. Most notably, this includes support for the TurboIMAGE/XL database and VPLUS interface. Nothing in ACUCOBOL-GT’s CALL interface specifically limits support for system intrinsics. However, we have not tested support for all intrinsic functions. In particular, the more obscure functions and variants of calls that take unusual or undocumented parameter types have not been tested.

On platforms *other than HP e3000* that support TurboIMAGE/XL, VPLUS, or other intrinsics, if the emulator or environment does not provide transparent support, you can access the intrinsics from ACUCOBOL-GT using the *direct method* of interfacing to C subroutines (see Chapter 4 of the *Guide to Interoperating with ACUCOBOL-GT*, section 4.2). All you need to do is edit the “LIBDIRECT” structure in the file “direct.c”. After editing “direct.c”, you must relink the runtime. Create a new runtime using the method described in section 4.5 of the interoperability guide.

4.5.1 CREATEPROCESS Intrinsic Function

In HP COBOL compatibility mode, ACUCOBOL-GT supports the use of the CREATEPROCESS intrinsic function to start separate, directly executable programs. CREATEPROCESS statements that launch such programs do not have to be modified in any way. CREATEPROCESS statements cannot, however, be used to start programs that have been compiled with ACUCOBOL-GT. Such programs can be started directly with the CALL or CALL RUN statement. Existing CREATEPROCESS statements that call programs that are now compiled with ACUCOBOL-GT, must be converted to use the CALL or CALL RUN statement.

4.5.2 CREATE/ACTIVATE Intrinsic Functions

In HP COBOL compatibility mode, ACUCOBOL-GT supports the use of the CREATE and ACTIVATE intrinsic functions to load and start separate, directly executable programs. Processes started in this way run asynchronously to the calling program (in a separate *thread*).

CREATE and ACTIVATE statements that load and launch separately executable programs do not have to be modified in any way. CREATE and ACTIVATE cannot, however, be used to load and launch programs that have been compiled with ACUCOBOL-GT. Existing CREATE and ACTIVATE statements that call programs that are now compiled with ACUCOBOL-GT, must be converted to use the CALL THREAD statement.

4.5.3 GETINFO Intrinsic Function

The GETINFO intrinsic function returns the string specified in the INFO argument of the command line. A typical HP COBOL command line includes program parameters in the INFO argument. The structure of the ACUCOBOL-GT command line includes the name of the COBOL program as well as the program parameters in the INFO argument.

To cause GETINFO to return only the arguments that follow the COBOL program name (identical behavior to HP COBOL), you must include a “ ;” delimiter between the COBOL program name and the other parameters in the INFO argument.

For example, if the HP COBOL command line looks like:

```
:RUN MYPROG;INFO='info1 info2'
```

a call to GETINFO will return “info1 info2”. To get the same result with ACUCOBOL-GT, the command line must look like:

```
:RUN /ACUCOBOL/bin/runcbl;INFO='MYPROG ;info1 info2'
```

If the “ ;” delimiter is left out of the INFO argument, a call to GETINFO will return “MYPROG info1 info2”. Note that there must be a space before the semi-colon and no space after the semi-colon.

4.6 AcuBench

You can use our Windows-based integrated development environment (IDE), AcuBench, to develop new COBOL applications for your HP e3000 systems or to enhance your existing COBOL applications. AcuBench is particularly helpful in developing new graphical user interfaces (GUI) for existing applications.

AcuBench is a Windows-based tool. To enhance existing HP e3000 applications, you may need to use some additional tools in conjunction with AcuBench to ensure that all libraries and data files are in formats that AcuBench can handle and to allow your Windows systems and your HP e3000 systems to pass files to each other as required.

This section outlines the use of AcuBench to develop HP e3000 applications. Please contact your Micro Focus ***extend*** Systems Engineering representative for additional information and assistance.

4.6.1 Using AcuBench With Existing Applications

AcuBench is an excellent tool for developing a Windows-native graphical user interface (GUI) to replace an application's existing user interface. Depending on various characteristics of your applications, such as the format of source and library files, the use of VPLUS, the use of Qedit files, and other HP e3000 specific attributes, some special needs may have to be addressed. These special needs are discussed in this section.

Source code residency

In order for AcuBench to access program source files, the source files must reside in a local Windows directory or in a Windows shared-drive. If you want to keep your COBOL objects and files on the HP e3000, you can install special software (such as SambaIX) on your HP e3000 to allow your Windows machine to map directories on the HP e3000.

Compiling with AcuBench

AcuBench uses the Windows version of the ACUCOBOL-GT compiler to compile source code. The ACUCOBOL-GT compiler includes a capability called Acucorp Remote File Access Protocol, or *acurfap*. With *acurfap*, the compiler can create the resulting object file directly on the remote system, if AcuServer or AcuConnect is running on the remote machine. AcuConnect can be deployed on the HP e3000, making it is possible to compile on a Windows system and have the resulting object file placed automatically on the designated HP e3000 machine. The exact location is specified in the Project Properties area of AcuBench. For complete information on acurfap and its use with AcuBench, see the *AcuBench User's Guide* and the ACUCOBOL-GT documentation set.

Note: *acurfap* can be used only when AcuConnect is running on the HP e3000.

Handling KSAM file format and multiple-file COPY libraries

AcuBench requires that COPY files be in plain text format. AcuBench does not support the inclusion of macros in COPY files. Neither does it support COPY libraries. KSAM format COPY files and libraries must be converted to individual text files. See **section 4.2.7, "The libutil Utility,"** for information about converting KSAM files and COPY libraries.

Using terminal emulators with AcuBench

AcuBench is particularly helpful in developing a graphical user interface for your HP e3000 applications for deployment using our Thin Client technology. However, at present, this development scenario requires, in most cases, that you compile your updated application on the HP e3000 using ACUCOBOL-GT's HP COBOL compatibility option ("-Cp"). One way to facilitate this activity is to use WRQ's Reflection to help transfer files between MPE/iX MPE/iX and Windows and to initiate ACUCOBOL-GT compilation and runtime execution on the HP e3000. The following outlines the basic steps.

1. Use *Reflection* to import the source code from the HP e3000 to the Windows machine on which AcuBench is installed.
2. Use the Screen Designer to develop and generate code for your GUI screens and perform any source code editing in the AcuBench Code Editor.
3. When you are ready to compile the updated code, use *Reflection* to export the source code back to the HP e3000.
4. Use *Reflection* to invoke the ACUCOBOL-GT compiler on the HP e3000.
5. Use *Reflection* to invoke the ACUCOBOL-GT runtime and debugger on the HP e3000.

Handling source files in Qedit format

Qedit files must be converted to a flat file format before they can be used in the AcuBench Code Editor. To use Qedit files with AcuBench, you should move them as *keep files*, not as *Qedit format files*. You can use a third-party tool (such as QCOPY.QLIB.ROBELLE) to convert the file to *keep file format*, or you can use Qedit for Windows to copy and paste the file into a PC file.

Handling programs containing VPLUS screens

VPLUS screens cannot be displayed on the Windows host using the Thin Client. HP e3000 programs that contain VPLUS screens can be coded in the AcuBench Code Editor and compiled with the Windows compiler. However, runtime-related activities, such as execution and debugging, must be performed on the HP e3000. You can handle this by complementing AcuBench with a product such as *Reflection*, which contains a terminal emulator. See the subsection titled “Using terminal emulators with AcuBench” in this section.

Handling IMAGE data files

There are no limitations associated with IMAGE data. We recommend that execution be on the HP e3000. This could be considered a requirement, except that there are third-party software products (such as OMNIDEX by DISC) that allow your program to be run on Windows and access IMAGE data files on the HP e3000. The Graphical Working Storage Section of AcuBench can import existing COPY files which describe the buffers and variables used to make intrinsic calls.

4.6.2 Developing New Applications With AcuBench

If you want to develop new applications for your HP e3000 systems, AcuBench provides a complete IDE that has all of the tools necessary for developing and debugging your COBOL application on the HP e3000 platform. You work in a Windows environment on a client, and you can develop and debug your applications on the HP e3000 server. One thing to remember is that the MPE/iX layer uses the FILE.GRP.ACCT syntax for specifying files and directories, while POSIX uses Hierarchical File System (HFS) syntax, such as /ACCT/GRP/FILE, for the same purpose. For details, see “MPE/iX vs. HFS Syntax and LISTF vs. LISTFILE” in the *Getting Started* booklet, section 1.6. See the *AcuBench User’s Guide* for complete information on using that product.

4.7 Thin Client Solution on MPE/iX

If you use ACUCOBOL-GT to run your applications on the HP e3000, you may be interested in our Thin Client technology. TheThin Client technology allows you to run your application on the HP e3000 while displaying the user interface on a Windows client. This is particularly valuable if you use ACUCOBOL-GT to develop a Windows-native graphical user interface (GUI) for your HP e3000 application. Our AcuConnect component, described in the next section, is a key enabling Thin Client technology.

Note: The MPE/iX operating environment has a limitation that requires sites planning to use AcuConnect to carefully consider how they will deploy the service. In the MPE/iX environment, the operating system does not provide a way for a program to change its user ID. Therefore, the service always uses the ID of the account that starts the service. Any action the program takes is performed with that ID. This inability to change IDs imposes some limitations and requires that MPE/iX sites carefully consider how they will deploy AcuConnect. There are two approaches to managing this issue. See the *AcuConnect User's Guide* for details.

4.7.1 AcuConnect

AcuConnect is used with the Thin Client technology, an innovation that lets you display your server-based application on graphical display hosts. For a thin client to access programs on the server, AcuConnect must be running.

AcuConnect comes with a default security file known as “AcuAccess.” This file is located in the /ACUCOBOL/etc directory and is called “/ACUCOBOL/etc/AcuAccess”.

The “/ACUCOBOL/etc/AcuAccess” file contains a database of access records that determine which machines and which users are allowed to use the server. Depending on the construction of the database records, the server access file can provide many levels of system access, from very permissive to very restrictive. By default, system access is permissive.

The server software also comes with a configuration file called “/ACUCOBOL/etc/acurcl.cfg”.

For the job file (see the example under “Starting AcuConnect”) to work, you must modify the configuration file so that the following parameters are uncommented and have the following values:

Parameter	**Value**
ACCESS-FILE	/ACUCOBOL/etc/AcuAccess
DEFAULT-USER	MGR.ACUCOBOL

Note that the value for DEFAULT-USER must be specified in uppercase.

In addition AcuConnect requires a server alias file to hold all the information that will be needed to invoke the appropriate application on the server. See the *AcuConnect User’s Guide* for details on how to create this file.

4.7.1.1 Starting AcuConnect

To start the server process, you need to run the process as an MPE/iX batch job. The job file might be called JACURCLD and might look something like this:

```
!JOB JACURCLD,MGR.ACUCOBOL;PRI=CS
!RUN /ACUCOBOL/bin/acurcl;&
!  INFO=‘-start -c /ACUCOBOL/etc/acurcl.cfg –le /ACUCOBOL/bin/acurcl.err -t3’;&
!  PRI=CS
!EOJ
```

To stream the job, you could do the following, assuming you are in the MPE/iX group where the job file is:

MPE:

```
:STREAM JACURCLD
```

POSIX:

```
shell/iX> callci "STREAM ./JACURCLD”
```

To stop the job, you would do the following:

MPE:

```
:/ACUCOBOL/bin/acurcl "-kill”
```

POSIX:

```
shell/iX> /ACUCOBOL/bin/acurcl -kill
```

4.8 Backing Up ACUCOBOL-GT Software

When you do a full system backup and want to make sure that the ACUCOBOL-GT files are included in that backup, all you need to do is use a command that includes HFS files. By including HFS files you also include ACUCOBOL-GT files.

ACUCOBOL-GT files will be included in a full backup with STORE if you use a fileset like @.@.@:

```
:STORE @.@.@;;DIRECTORY
```

However, if you use a fileset like ?@.@.@:

```
:STORE ?@.@.@
```

the “?” *excludes* HFS files.

You can use the TREE option to force STORE to include HFS files if for some reason they are not being included with your fileset specification:

```
:STORE @.@.@;;TREE
```

For more details on backing up HFS files, see the *STORE and TurboSTORE/ iX Products Manual*.

5 IBM DOS/VS COBOL Conversions

Key Topics

5.1 Support for DOS/VS COBOL

The command-line option “-Cv” (either on the command line or in the CBLFLAGS environment string) allows the ACUCOBOL-GT compiler to be compatible with IBM DOS/VS COBOL. In this mode, it accepts features of IBM DOS/VS COBOL that are not otherwise accepted by ACUCOBOL-GT.

Since there are slight differences between IBM COBOL versions, “-Cv” also takes the following optional arguments:

“**-Cv=OSVS**” specifies OSVS compatibility.

“**-Cv=VSC2**” specifies VSC2 compatibility.

“-Cv” by itself defaults to OSVS mode. The two modes are very similar, except that in VSC2 compatibility mode, the following words are not reserved:

CURRENT-DATE
EXAMINE
TIME-OF-DAY
TRANSFORM

Note that CURRENT-DATE is a valid function in any compatibility mode.

This chapter provides information about the IBM DOS/VS COBOL features that the ACUCOBOL-GT compiler supports. Complete IBM DOS/VS COBOL compatibility is, unfortunately, not feasible. However, in this chapter you will find a list of IBM DOS/VS-specific features, what they do, and whether they are supported by ACUCOBOL-GT through the command line option, “-Cv.”

IBM DOS/VS COBOL compatibility is available only with Version 5.0 (or later). Therefore, the compiler aborts with its “usage” message if both “-Cv” and a backward object compatibility option, “-Z32” for example, are specified.

5.2 ACUCOBOL-GT and IBM DOS/VS COBOL-Specific Features

IBM DOS/VS COBOL-specific features that cannot be supported by the ACUCOBOL-GT compiler are reported as general errors. The following list contains the specific features of IBM DOS/VS COBOL, along with an explanation of what each feature does, whether it is accepted or not via the command line option "-Cv," and any operational information that is helpful in using the "-Cv" command line option.

5.2.1 ACTUAL KEY Clause and SEEK

In IBM DOS/VS compatibility mode, when compiling for object versions 8.1 and greater, the compiler accepts the ACTUAL KEY feature for relative files opened in RANDOM access mode. For such files, the relative key numbers will be zero-based, rather than one-based, as with RELATIVE KEY. For example, if you have a relative file with a fixed record length of 3 bytes and a relative file with the following contents:

```
AAABBBCCC
```

The record keys for the different modes are:

```
     RELATIVE  ACTUAL
AAA:    1        0
BBB:    2        1
CCC:    3        2
```

ACUCOBOL-GT does not support SEEK and reports an "Unsupported Operation" error if it is present.

5.2.2 ADVANCING in WRITE Statement

In the absence of either an ADVANCING or POSITIONING clause, a WRITE statement for a sequential PRINT file advances one line *before* printing for ACUCOBOL-GT without the "-Cv" option, and one line *after* printing for IBM DOS/VS COBOL or ACUCOBOL-GT with the"-Cv" option.

5.2.3 AFTER POSITIONING Clause

IBM DOS/VS COBOL has a Format 2 WRITE statement that is similar to the ACUCOBOL-GT Format 1 WRITE statement, except that the ADVANCING clause is replaced by a POSITIONING clause:

ACUCOBOL-GT:

```
{BEFORE} ADVANCING {number [LINE ]}
{AFTER }           {       [LINES]}
                   {PAGE          }
```

IBM DOS/VS COBOL:

```
AFTER POSITIONING {identifier} LINES
                  {integer   }
```

If the line count in the IBM form is an identifier, it must be a one-character alphanumeric item, interpreted as follows:

Character	**ACUCOBOL-GT Equivalent**
‘ ‘	AFTER ADVANCING 1
‘0’	AFTER ADVANCING 2
‘-’	AFTER ADVANCING 3
‘+’	AFTER ADVANCING 0
‘1’	AFTER PAGE
others	invalid or not supported in ACUCOBOL-GT (implemented as AFTER ADVANCING 1)

If the line count in the IBM form is an integer literal, it is interpreted as follows:

Value	**ACUCOBOL-GT Equivalent**
0	AFTER PAGE
1	AFTER ADVANCING 1
2	AFTER ADVANCING 2
3	AFTER ADVANCING 3

Value	**ACUCOBOL-GT Equivalent**
others	invalid or not supported in ACUCOBOL-GT (implemented as AFTER ADVANCING 1)

ACUCOBOL-GT accepts the IBM form (along with its own form, even in the same file) if the "-Cv" option is in effect. ACUCOBOL-GT does not enforce the one-character limit on the line count and uses only the first character or digit if there are two or more.

Note: In the absence of either an ADVANCING or POSITIONING clause, a WRITE statement for a sequential PRINT file advances one line BEFORE printing for ACUCOBOL-GT without the "-Cv" option, and one line AFTER printing for IBM DOS/VS COBOL or ACUCOBOL-GT with the "-Cv" option.

5.2.4 APPLY Clause Options

IBM DOS/VS COBOL allows the clauses of the I-O-CONTROL paragraph of the ENVIRONMENT division to appear in any order. It also accepts the following forms of the APPLY clause that ACUCOBOL-GT does not accept:

```
APPLY WRITE-ONLY ON file-name-1 [file-name-2] ...

APPLY EXTENDED-SEARCH ON file-name-1 [file-name-2] ...

APPLY WRITE-VERIFY ON file-name-1 [file-name-2] ...

APPLY CYL-OVERFLOW OF integer TRACKS ON file-name-1 [file-name-2] ...

APPLY CORE-INDEX TO data-name ON file-name-1 [file-name-2]

APPLY {MASTER-INDEX} TO device-number ON file-name-1 [file-name-2] ...
      {CYL-INDEX   }
```

These options affect the efficiency of file operations in IBM DOS/VS COBOL but not the program logic. As a result, ACUCOBOL-GT treats these forms of the APPLY clause as comments when in the IBM DOS/VS COBOL compatibility mode, and it accepts the clauses in any order regardless of the mode.

You may put APPLY clauses of the IBM DOS/VS COBOL type into the I-O-CONTROL paragraph, but only when the compiler is in the IBM DOS/VS COBOL compatibility mode. Such clauses are scanned but otherwise ignored.

The clauses of the I-O-CONTROL paragraph may be arranged in any order, and the existing APPLY clause may be used, regardless of the compiler mode.

5.2.5 Arithmetic Statements

ACUCOBOL-GT includes a compiler option, “--ArithmeticVSC2”, that causes intermediate results in arithmetic statements to be truncated following the rules of VS COBOL II and COBOL/370. Note that this option matches the Micro Focus ARITHMETIC”VSC2” compiler directive.

5.2.6 BASIS, INSERT, DELETE

The BASIS statement is a variant of the COPY statement. Because the BASIS statement is a debugging technique, it is not portable and ACUCOBOL-GT does not support it.

5.2.7 COM-REG

This special register in IBM DOS/VS COBOL corresponds to bytes 12 to 22 of the DOS Communication Region. It is not portable and is not supported by ACUCOBOL-GT.

5.2.8 COPY SUPPRESS Statement

The word SUPPRESS in the COPY statement causes the compiler to omit the copied file from the program listing. This feature of IBM DOS/VS COBOL has been incorporated into ACUCOBOL-GT. See section 2.4.1 in the *ACUCOBOL-GT Reference Manual*.

5.2.9 CURRENT-DATE

IBM DOS/VS COBOL has a CURRENT-DATE register that contains the current date in either MM/DD/YY or DD/MM/YY format, where MM is the month (01 for January, 02 for February, etc.), DD is the day of the month, and YY is the last two digits of the year. Leading zeros are used where necessary to ensure two-digit values. For example, November 2, 2000 is expressed as either 11/02/00 or 02/11/00. This register is valid only as the source of a MOVE statement without conversion.

When ACUCOBOL-GT is in the IBM DOS/VS COBOL compatibility mode, CURRENT-DATE may be used as the source of a Format 1 MOVE statement:

```
MOVE CURRENT-DATE TO {dest-item} ...
```

If there are two or more destination items in the same MOVE statement, all are guaranteed to receive the same value, even if the clock ticks and midnight passes while the statement is being processed.

As usual, the value is truncated or padded with blanks at the right end to make it fit into the destination.

The runtime configuration variable CURRENT-DATE determines which format to use. If this variable is “0”, which is the default condition, MM/DD/YY is used. If this variable is “1”, DD/MM/YY is used. Other values produce undefined results.

5.2.10 DATE-COMPILED

IBM DOS/VS COBOL replaces the comment in the DATE-COMPILED paragraph with the compilation date in the program listing. ACUCOBOL-GT does not need to do this because it puts the compilation date elsewhere in the listing.

5.2.11 Debugging Features

ACUCOBOL-GT does not support IBM DOS/VS-specific debugging features. The ACUCOBOL-GT compiler marks as errors the debugging statements that begin with the following words:

```
READY
RESET
EXHIBIT
ON
DEBUG
```

5.2.12 DISPLAY UPON SYSPUNCH

The IBM DOS/VS COBOL SYSPUNCH (or SYSPCH) option refers to punchcards, which are obsolete. The ACUCOBOL-GT compiler redirects punchcard output to a special file called SYSPUNCH.TXT.

Format 9 of the DISPLAY statement (ANSI DISPLAY) is in most cases treated in a slightly different manner by IBM DOS/VS COBOL.

IBM DOS/VS COBOL has special names for card punch output, which may appear after the reserved word UPON either as SYSPUNCH (or SYSPCH) or as a “mnemonic” associated with one of these in the SPECIAL-NAMES paragraph.

In the IBM DOS/VS COBOL compatibility mode, ACUCOBOL-GT accepts these names, or their mnemonics, and it emulates a card punch by writing the card images to a disk file (or a device that looks like a disk file to the operating system). The default file specification is SYSPUNCH.TXT (in the current directory), but you may specify another file by putting a line of the following form in the runtime configuration file:

```
SYSPUNCH = <file specification used instead of SYSPUNCH.TXT>
```

Each card image is 80 characters wide, and is terminated in the manner appropriate for the runtime environment (carriage return and line feed for Windows, line feed for UNIX).

The leftmost 72 columns contain data from the DISPLAY UPON SYSPUNCH statement. If the data is longer than this, excess characters spill over to the next card image. If the data is shorter than this, it is padded with blanks at the right end to make it 72 characters long.

The other 8 columns of each card image contain the program name from the PROGRAM-ID phrase of the IDENTIFICATION division. If the program name is more or less than eight characters long, it is truncated or padded with spaces at the right end.

If the file specified for card images already exists, the card images are added to the end of the file.

The WITH NO ADVANCING phrase may be used with DISPLAY UPON SYSPUNCH, although it is not part of IBM DOS/VS COBOL. It may cause subsequent text to go into the same card image, if it fits.

DISPLAY UPON CONSOLE, DISPLAY UPON SYSLST, DISPLAY UPON SYSLIST and DISPLAY UPON SYSOUT behave in a slightly different manner when the IBM DOS/VS COBOL compatibility mode is in effect. In this mode, each line must be 100 characters long for DISPLAY UPON CONSOLE, and 120 characters long for the other forms. A line that is too short is padded with blanks at the right end, and a line that is too long spills over to the next line.

The IBM DOS/VS COBOL compatibility mode has no effect on the DISPLAY UPON SYSERR statement.

5.2.13 EJECT, SKIP in Listing

In the IBM DOS/VS COBOL compatibility mode, the following reserved words may be used to control spacing of the program listing:

Word	**Effect**
EJECT	start a new page
SKIP1	skip one line before the next line (double spacing)
SKIP2	skip two lines before the next line (triple spacing)

Word	Effect
SKIP3	skip three lines before the next line (quadruple spacing)

IBM DOS/VS COBOL requires these words to appear in Area B. The reserved word must be the only thing on the line, other than tabs and spaces.

5.2.14 ENTER Statement

IBM DOS/VS COBOL scans but ignores the ENTER statement as follows:

```
ENTER  language-name  [routine-name].
```

ACUCOBOL-GT also ignores this statement when it is in the IBM DOS/VS COBOL compatibility mode.

5.2.15 EXAMINE

IBM DOS/VS COBOL has an EXAMINE statement, which duplicates some of the functionality of the ACUCOBOL-GT INSPECT statement, but with slightly different syntax. ACUCOBOL-GT accepts this statement in the IBM DOS/VS COBOL compatibility mode.

Format 1:

```
EXAMINE identifier-1 TALLYING {ALL        } literal-1 [REPLACING BY literal-2]
                              {LEADING    }
                              {UNTIL FIRST}
```

Format 2:

```
EXAMINE identifier-1 REPLACING {ALL        } literal-1 BY literal-2
                               {FIRST      }
                               {LEADING    }
                               {UNTIL FIRST}
```

Here *identifier-1* is the name of a data item containing characters to be counted or replaced. Its USAGE must be DISPLAY (implicitly or explicitly). Each literal is a single-character literal or one of the following figurative literals:

ZERO

```
ZEROS
ZEROES
SPACE
SPACES
QUOTE
QUOTES
LOW-VALUE
LOW-VALUES
HIGH-VALUE
HIGH-VALUES
```

ACUCOBOL-GT does not enforce the one-character limitation. If a string literal has two or more characters, it uses only the first character.

The special register TALLY, whose PICTURE is 9(5) and whose USAGE is COMP-N, counts the number of characters in Format 1 that obey certain conditions.

The characters to be counted and replaced are determined by the key words following TALLYING or REPLACING. Counting is done only in Format 1, and replacement is done only in Format 2 and when specified in Format 1.

Key Words	**Characters To Be Counted And Replaced**
ALL	Count all instances of *literal-1* in the data item and replace them with *literal-2*.
FIRST	Replace the first (leftmost) instance of *literal-1* in the data item to *literal-2*.
LEADING	Count the instances of *literal-1* in the data item that appear before (to the left of) any other character in the data item, and replace them with *literal-2*.
UNTIL FIRST	Count the characters to the left of the first (leftmost) instance of *literal-1* and replace them with *literal-2*. If *literal-1* does not appear in the data item, count and replace all characters in the data item.

It should be noted that the substitution of *literal-2* for characters other than *literal-1* occurs only in the last case.

The contents of the TALLY register are unchanged in Format 2. In Format 1, its contents are replaced by the appropriate character count, regardless of the previous contents.

5.2.16 FILE-LIMIT Clause

IBM DOS/VS COBOL permits an optional clause in a SELECT statement (also called a “file-control-entry”):

```
{FILE-LIMIT IS  }  limits-1  [limits-2] ...
{FILE-LIMITS ARE}
```

Here limits-1, limits-2, etc., are phrases of the form:

```
{data-name-1} THRU {data-name-2}
{literal-1  }      {literal-2  }
```

IBM DOS/VS COBOL treats this clause as a comment. ACUCOBOL-GT ignores it in the IBM DOS/VS COBOL compatibility mode.

Put the FILE-LIMIT clause in any convenient spot after the ASSIGN clause.

5.2.17 File Status Codes

Some of the numeric status codes returned for file operations are different in IBM DOS/VS COBOL and ACUCOBOL-GT. The following list contains codes returned for each.

Code	**IBM DOS/VS COBOL**	**ACUCOBOL-GT**
00	Success	Same
10	EOF	Same
21	Invalid key, sequence error	Same
22	Duplicate key	Same
23	Key not found	Same
24	Boundary violation (indexed VSAM	Same
30	Permanent error	Same

Code	**IBM DOS/VS COBOL**	**ACUCOBOL-GT**
34	Boundary violation (sequential VSAM)	Same
91	Password failure	Not supported closest alt.: 3707
92	Logic error	Presumably 4x
93	Resource not available	Presumably 05/35
94	No current record for seq. Request	43
95	Invalid or incomplete file information.	Not supported closest alt.: 39xx
96	No DLBL card	Not supported
Zx	User-defined errors	Not supported

5.2.17.1 FILE STATUS Extensions

The compiler supports the specification of a second FILE STATUS item, which is treated as commentary. The FILE STATUS phrase of the File-Control paragraph supports the following format:

```
[ File STATUS Is status-variable [status-variable-2] ]
```

Status-variable must be the name of an alphanumeric (or USAGE DISPLAY numeric) Working-Storage or Linkage data item with a size of 2 characters. Status-variable-2 must be the name of a group item that is 6 characters in length (this is not checked by the compiler).

When the FILE STATUS clause is specified, a value will be moved into status-variable after the execution of every statement that references the corresponding file. This value indicates the status of the statement.

5.2.18 IDENTIFICATION Division Arrangement

IBM DOS/VS accepts the AUTHOR, INSTALLATION, DATE_WRITTEN, DATE_COMPILED, SECURITY and REMARKS paragraphs in any order in the IDENTIFICATION division.

ACUCOBOL-GT does the same, even when not in the IBM DOS/VS compatibility mode. Therefore, the user may put the AUTHOR, INSTALLATION, DATE_WRITTEN, DATE_COMPILED, SECURITY and REMARKS paragraphs in the IDENTIFICATION division in any order.

5.2.19 IF OTHERWISE

IBM DOS/VS COBOL optionally accepts the reserved word OTHERWISE instead of ELSE in IF statements. In the IBM DOS/VS COBOL compatibility mode, ACUCOBOL-GT accepts OTHERWISE as a synonym for ELSE.

5.2.20 LENGTH OF Expression

The LENGTH OF construct works differently in ACUCOBOL-GT than in IBMCOBOL when the data item used is a table. In this case, ACUCOBOL-GT returns the size of the entire table, while IBM returns the size of a single element of the table. For example, consider the following:

```
01  my-data.
    03  my-table occurs 20 times.
        05  my-element-1   pic x(10).
05  my-element-2   pic 99.
MOVE LENGTH OF my-element-1 TO data-size.
MOVE LENGTH OF my-table TO data-size.
MOVE LENGTH OF my-table(1) TO data-size.
```

ACUCOBOL-GT IBM COBOL treat the first MOVE as MOVE 10 TO data-size, and the third MOVE as MOVE 12 TO data-size. The second MOVE, however, is treated differently. ACUCOBOL-GT treats the second MOVE as MOVE 240 TO data-size, while IBM COBOL treats the second MOVE as MOVE 12 TO data-size.

When using IBM compatibility mode, the ACUCOBOL-GT compiler treats LENGTH OF in the same way as IBM COBOL.

5.2.21 NOTE Statement

IBM DOS/VS COBOL accepts a NOTE statement in the PROCEDURE division:

```
        NOTE  character string(s)
```

The entire statement is treated as a comment.

If the NOTE statement is not first statement in its paragraph, then it ends

1. at the first period followed by a space, or
2. at the beginning of the next paragraph, or
3. at the end of the source code,

whichever comes first.

If the NOTE statement is the first statement in a paragraph, then it ends

1. at the beginning of the next paragraph, or
2. at the end of the source code,

whichever comes first. Hence all subsequent statements in the same paragraph are also ignored.

When it is in the IBM DOS/VS COBOL compatibility mode, ACUCOBOL-GT accepts the NOTE statement as IBM DOS/VS COBOL does, with one slight difference. When a NOTE statement is the first statement in a paragraph, IBM DOS/VS COBOL parses subsequent statements in the same paragraph, and presumably flags any syntax errors found in them. ACUCOBOL-GT, however, does not perform any syntax parsing in such cases.

5.2.22 Password Protection of Files

IBM DOS/VS COBOL implements password protection for files. You may put a PASSWORD clause into a SELECT statement, but only when the compiler is in the IBM DOS/VS COBOL compatibility mode. A SELECT statement (also called a "file-control-entry") for a password-protected file must contain a PASSWORD clause of the following format:

```
PASSWORD IS data-name-1
```

Here, *data-name-1* is the name of an alphanumeric data item in the WORKING-STORAGE section of the DATA division. This data item must contain a valid password when the file is opened; otherwise, the OPEN statement fails.

To be compatible with IBM DOS/VS, ACUCOBOL-GT ignores the PASSWORD clause when in the IBM DOS/VS compatibility mode.

Put the PASSWORD clause in any convenient spot after the ASSIGN clause.

5.2.23 PROCESSING MODE Clause

IBM DOS/VS COBOL permits an optional PROCESSING MODE clause in a SELECT statement (also called a "file-control-entry"):

```
PROCESSING MODE IS SEQUENTIAL
```

IBM DOS/VS COBOL ignores this clause. Compatibility with IBM DOS/ VS COBOL requires that ACUBOL-GT ignore it also.

You may put a PROCESSING MODE clause into a SELECT statement, but only when the compiler is in the IBM DOS/VS COBOL compatibility mode.

Put the PROCESSING MODE clause in any convenient spot after the ASSIGN clause.

5.2.24 PROGRAM-ID Program Name

In IBM DOS/VS COBOL, the program name in the PROGRAM-ID item of the IDENTIFICATION division may be placed between quotation marks. This is especially useful if the program name contains embedded spaces or is otherwise incompatible with identifier syntax.

The ACUCOBOL-GT compiler accepts a program name in quotation marks, whether it is in the IBM DOS/VS COBOL compatibility mode or not.

5.2.25 RECORDING MODE Clause

IBM DOS/VS COBOL permits an optional RECORDING MODE Clause in a SELECT statement (also called a “file-control-entry”):

```
RECORDING MODE IS mode
```

Here the *mode* must be F, V, U, or S. It describes the manner in which records are placed into blocks, which is irrelevant under DOS, Windows or Unix. Compatibility with IBM DOS/VS COBOL means that ACUCOBOL-GT ignores it.

You may put a RECORDING MODE clause into a SELECT statement, but only when the compiler is in the IBM DOS/VS compatibility mode.

Put the RECORDING MODE clause in any convenient spot after the ASSIGN clause.

5.2.26 REVERSED Sequential Input Files

IBM DOS/VS COBOL contains a REVERSED option for input files, which is specified when the file is opened:

```
OPEN INPUT file-name REVERSED.
```

The file must be a sequential file with a fixed record length. The OPEN operation fails if this is not the case.

The records in a REVERSED file are read in reverse order by ordinary READ statements. An attempt to read past the beginning of the file produces the same result as an attempt to read past the end of a regular sequential file.

ACUCOBOL-GT accepts this feature when in the IBM DOS/VS compatibility mode.

5.2.27 SORT Statement Registers

IBM DOS/VS COBOL has 11 special registers associated with the SORT statement. ACUCOBOL-GT supports five of these. When in the IBM DOS/VS compatibility mode, ACUCOBOL-GT treats these as dummy registers, as though they had been declared at the beginning of the WORKING-STORAGE section as follows:

```
01  SORT-MESSAGE PIC X(8) EXTERNAL
77  SORT-FILE-SIZE PICTURE S9(8) USAGE COMP-4 VALUE 0.
77  SORT-CORE-SIZE PICTURE S9(8) USAGE COMP-4 VALUE 0.
77  SORT-MODE-SIZE PICTURE S9(5) USAGE COMP-4 VALUE 0.
77  SORT-RETURN PICTURE    S9(4) USAGE COMP-4 VALUE 0.
```

Special registers are reserved words that name storage areas, which are generated by the compiler. They do not need to be declared in the Working-Storage Section of your program, and they can still be referenced as data items in your program. It’s important to understand the data format for which they implicitly are created and to understand that they behave as though they were declared variables. These registers cannot be used as operands of ACCEPT or DISPLAY statements (as in IBM DOS/VS COBOL), but values can be moved to and from them.

5.2.28 SPECIAL-NAMES

The following system names are supported by ACUCOBOL-GT in the SPECIAL-NAMES paragraph:

```
SYSPCH
SYSPUNCH
C01 through C12
CSP
S01 through S05
```

The SYSPCH and SYSPUNCH names do not represent an actual card punch in ACUCOBOL-GT. Instead, output directed to the card punch is written to the file SYSPUNCH.TXT. (See **DISPLAY UPON SYSPUNCH** for details.)

To enable printing to printer channels C01-C12, set the runtime configuration variable called "COBLPFORM". See Appendix H in the *ACUCOBOL-GT Appendices* manual for details on setting this variable.

5.2.29 TIME-OF-DAY

IBM DOS/VS COBOL has a TIME-OF-DAY register that contains the current time of day in HHMMSS format, where HH is the hour (on a 24-hour clock), MM is the minute, and SS is the second. Leading zeros are used where necessary to ensure two-digit values. For example, 8 AM would be 080000.

This register is valid only as the source of a MOVE statement without conversion.

When ACUCOBOL-GT is in the IBM DOS/VS COBOL compatibility mode, TIME-OF-DAY may be used as the source of a Format 1 MOVE statement:

```
MOVE TIME-OF-DAY TO {dest-item} ...
```

If there are two or more destination items in the same MOVE statement, all are guaranteed to receive the same value, even if the clock ticks while the statement is being processed.

As usual, the value is truncated or padded with blanks at the right end to make it fit into the destination.

5.2.30 TRACK-AREA Clause

IBM DOS/VS COBOL permits an optional TRACK-AREA Clause in a SELECT statement (also called "file-control-entry"):

```
TRACK-AREA IS integer-1 CHARACTERS
```

Here *integer-1* is an integer.

This clause increases the efficiency of some file operations on the IBM System/360 Disk Operating System but is meaningless on other systems. ACUCOBOL-GT ignores it to remain compatible with IBM DOS/VS.

You may put a TRACK-AREA clause into a SELECT statement, but only when the compiler is in the IBM DOS/VS compatibility mode.

Put the TRACK-AREA clause in any convenient spot after the ASSIGN clause.

5.2.31 TRANSFORM Statement

IBM DOS/VS COBOL has a TRANSFORM statement with the following syntax:

```
TRANSFORM  identifier-1  CHARACTERS FROM pattern-1 TO pattern-2
```

Each pattern is one of the following:

```
    ZERO
    ZEROS
    ZEROES
    SPACE
    SPACES
    QUOTE
    QUOTES
    LOW-VALUE
    LOW-VALUES
    HIGH-VALUE
    HIGH-VALUES
    nonnumeric-literal
    identifier-2
```

Here *identifier-2* must not be more than 255 characters long. If it is more than 255 characters long, the results in IBM DOS/VS COBOL are undefined; in ACUCOBOL-GT only the first 255 characters are used.

Each figurative constant (ZERO, ZEROS, ZEROES, SPACE, SPACES, QUOTE, QUOTES, LOW-VALUE, LOW-VALUES, HIGH-VALUE and HIGH-VALUES) is equivalent to a one-character nonnumeric literal containing the associated character. For example, ZERO is equivalent to '0' and HIGH-VALUE is equivalent to x"FF".

One of the following conditions must hold:

1. *pattern-1* and *pattern-2* are the same length;
2. *pattern-1* contains more than one character and *pattern-2* contains exactly one character.

The effect of the TRANSFORM statement on the contents of *identifier-1* is undefined if neither of these conditions holds. The effect on the contents of *identifier-1* is also undefined if *pattern-1* contains duplicate characters. However, the statement finishes in a reasonable time and control passes normally to the next statement in any case.

At run time, every character in *identifier-1* that matches a character in *pattern-1* is replaced by the character in the corresponding position in *pattern-2*. If there is no corresponding position, because *pattern-2* is shorter than *pattern-1*, then it is replaced by the sole character in *pattern-2*.

5.2.32 USE GIVING

IBM DOS/VS COBOL has a form of the USE statement in the DECLARATIVES section that normally is not recognized by ACUCOBOL-GT:

```
USE AFTER STANDARD ERROR PROCEDURE ON file-name GIVING
    data-name-1 [data-name-2]
```

This form is accepted by ACUCOBOL-GT when the "-Cv" option is in effect. See section 6.6, "USE Statement," in the *ACUCOBOL-GT Reference Manual* for the format and rules governing the usage of this statement. See **section 5.6** of this guide for the IBM DOS/VS COBOL error codes.

5.2.33 VALUE OF Clause

IBM DOS/VS COBOL ignores the following clause in a file description entry statement:

```
VALUE OF  value-pair-1  {value-pair-2} ...
```

Here each *value-pair* is of the form

```
data-name-1 IS  {data-name-2 }
                {literal-1   }
```

ACUCOBOL-GT ignores it too.

You may put a VALUE OF clause into a file description entry, but only when the compiler is in the IBM DOS/VS compatibility mode.

5.2.34 WHEN-COMPILED

IBM DOS/VS COBOL contains a reserved word, WHEN-COMPILED, that can be used as the source of a MOVE statement. The value copied into the destination or destinations is a string of the form "MM/DD/YYhh.mm.ss," representing the time and date when the compilation began, where

MM	is the month (01 = Jan., 02 = Feb., etc.)
DD	is the day (01 to 31, inclusive)
YY	is the year, modulo 100 (99 for 1999, 00 for 2000, 01 for 2001, etc.)
hh	is the hour, on a 24-hour clock (00-23)
mm	is the minute (00-59)
ss	is the second (00-59)

ACUCOBOL-GT does the same when it is in the IBM DOS/VS COBOL compatibility mode. However, ACUCOBOL-GT also allows the programmer to use WHEN-COMPILED wherever a string constant can be used, not just as the source of a MOVE statement.

Note: The ACUCOBOL-GT object file (.ACU) contains an embedded compilation time and date, which is used for other purposes but is identical to COMPILE-TIME.

5.2.35 WRITE ADVANCING Special-Name

IBM DOS/VS COBOL allows the following form of the ADVANCING clause in a Format 1 WRITE statement:

```
{BEFORE}  ADVANCING  {identifier-2 LINES}
{AFTER }             {integer LINES     }
                     {special-name      }
```

The *special-name* option is the only one that is not also in ACUCOBOL-GT. The *special-name* is one of the following special names, or a “mnemonic” for it defined in the SPECIAL-NAMES paragraph:

Special Name	**Equivalent**
CSP	0 LINES
C01	PAGE
C02	none
C03	none
C12	none
S01	none
S02	none
S05	none

Only CSP and C01 are implemented in ACUCOBOL-GT in the IBM DOS/VS COBOL compatibility mode. The others depend on particular hardware features and cannot be ported. They will generate error messages.

5.2.36 XLM GENERATE and PARSE

ACUCOBOL-GT supports the IBM Enterprise COBOL XML GENERATE and XML PARSE statements. XML PARSE gives you a way to parse XML data and process it in a COBOL program, associating processing procedures with the exception and non-exception cases that can result from the parse. XML GENERATE gives you a way to translate COBOL data into XML.

As before, ACUCOBOL-GT also includes **AcuXML**, a runtime-resident interface that transparently converts XML data to sequential files for COBOL processing, and C$XML, a library routine that gives you precise control over which elements or attributes of the data to parse. A developer utility called **xml2fd** creates File Descriptors (FDs) and SELECT statements from existing XML files to support **AcuXML**.

All three of these approaches can be used to parse records-based XML files, but note that only C$XML and XML PARSE can be used to parse non-records-based XML files.

5.3 Reserved Words

The following words are reserved in IBM DOS/VS, but not in ACUCOBOL-GT. They are, however, treated as reserved words by ACUCOBOL-GT when it is in the IBM DOS/VS COBOL compatibility mode. You need to avoid using any of the IBM DOS/VS-specific reserved words as user-defined words.

ACTUAL	DISPLAY-ST	SKIP
BEGINNING	EJECT	SORT-CORE-SIZE
C01	ENDING	SORT-FILE-SIZE
C02	ENTRY	SORT-MODE-SIZE
C03	EXAMINE*	SORT-RETURN
C04	EXTENDED-SEARCH	SUPPRESS
C05	FILE-LIMIT	SYSPCH
C06	MASTER-INDEX	SYSPUNCH

C07	NOMINAL	TALLY
C08	NOTE	TIME-OF-DAY*
C09	OTHERWISE	TRACK-AREA
C10	PASSWORD	TRANSFORM*
C11	POSITIONING	WHEN-COMPILED
C12	PROCESSING	WRITE-ONLY
COM-REG	S01	WRITE-VERIFY
CORE-INDEX	S02	
CSP	S03	
CURRENT-DATE*	S04	
CYL-INDEX	S05	
CYL-OVERFLOW	SEEK	

* In VSC2 mode these words are NOT reserved. See **Section 5.1** for details on this mode. Note that CURRENT-DATE is a valid function in any compatibility mode.

5.4 COMP-1 and COMP-2 are Floating-Point

The treatment of COMP-1 (a.k.a. COMPUTATIONAL-1) and COMP-2 (a.k.a. COMPUTATIONAL-2) is different in IBM DOS/VS COBOL. They are equivalent to FLOAT and DOUBLE, respectively, and they cannot be used with PICTURE phrases.

ACUCOBOL-GT already has a command-line option (“-Df”) to specify that COMP-1 and COMP-2 are to be treated as in IBM DOS/VS COBOL. The “-Cv” option automatically turns on this “-Df” option, and so additionally specifying “-Df” is unnecessary.

Take care to observe the proper syntax and usage of COMP-1 and COMP-2 variables for the mode in effect.

5.5 External Floating-Point (EFP)

The format of external floating-point (EFP) items is compatible with IBM DOS/VS COBOL. However, ACUCOBOL-GT will accept it even if it is not in the IBM DOS/VS compatibility mode.

Note: If a command-line option, such as "-Z4", is used to force the compiler to generate code for external floating-point items for a version of ACUCOBOL-GT prior to Version 5.0, then the compiler issues an "invalid picture" error.

5.5.1 External Floating-Point Data Type

External floating-point (EFP) is a machine-independent floating-point data type defined with a PICTURE phrase. An EFP item is a numeric item. It is not an edited numeric item.

EFP items in ACUCOBOL-GT can be written to, and read from, data files, and they can be converted to and from other numeric data types.

An EFP item may be used anywhere a floating-point item may be used. When it is used in a DISPLAY statement, the assumed decimal point is shown, but no other editing is performed.

Note: Mixed-type operations are not well-defined in COBOL. Therefore, it is not advisable to mix internal floating-point (FLOAT or DOUBLE), external floating-point, and fixed-point numeric operands, except in simple MOVE statements.

EFP operations are always rounded off, even if the ROUNDED option is not used.

No USAGE, SIGN, SYNCHRONZED, JUSTIFIED or VALUE clause may be used with an EFP item.

5.5.2 The Picture

An external floating-point (EFP) data item is defined by a picture that strongly resembles a floating-point numeric literal:

```
{+} mantissa E {+} 99
{-}            {-}
```

The mantissa is a string containing:

1. from one to sixteen “9”s, and
2. one assumed decimal point (that does not appear in the data) represented by the letter “V”, or one actual decimal point (that does appear in the data) represented by a period “**.**”.

The exponent is always exactly two digits long, so the last two characters of the picture must be “99”.

A plus sign “+” in the picture indicates that a nonnegative mantissa or exponent will be preceded by a plus sign; a minus sign “-” indicates that a nonnegative mantissa or exponent will be preceded by a space.

A negative mantissa or exponent is always preceded by a minus sign.

When a numeric value is converted to EFP format, the result is normalized so that the leftmost digit of the mantissa is nonzero, if this can be done without reducing the exponent below “-99”. Zero is represented by a mantissa and exponent consisting entirely of zeros.

When exponent overflow occurs, the mantissa of the result consists entirely of “9”s, and its exponent is “+99”.

When exponent underflow occurs, the result is not normalized, and may be zero if it is too small to be represented in this format.

Examples:

picture	**value**	**data**	**notes**
+9.999E+99	123	+1.230E+02	
+9.999E+99	123.0E99	+9.999E+99	exponent overflow

picture	**value**	**data**	**notes**
-9V999E+99	123	1230E+02	
-9V999E-99	123	1230E 02	
+V9999E-99	0.0123	+1230E-01	note normalization
+V9999E-99	0.01E-99	+0100E-99	exponent underflow
-99999.E+99	123	12300.E-02	

5.6 IBM DOS/VS Error Codes

IBM DOS/VS COBOL has a form of the USE statement in the DECLARATIVES section that is not recognized by ACUCOBOL-GT:

```
USE AFTER STANDARD ERROR PROCEDURE ON file-name GIVING
      data-name-1 [data-name-2]
```

This form is accepted by ACUCOBOL-GT when the "-Cv" option is in effect.

When an error handler introduced by this statement is invoked, the runtime puts special error codes into the eight-byte data item data-name-1. Each byte is loaded with "1"s if the corresponding error condition is true. If the corresponding error condition is false, ACUCOBOL-GT loads it with "0"s.

Here are the IBM conditions:

byte	**indexed**	**direct**	**sequential**
1 (leftmost)	DASD error	data check in count	parity error
2	wrong record length	wrong record length	wrong record length
3	prime data full	no room found	
4	cylinder index too small	data check in key or data	
5	master index too small		

byte	indexed	direct	sequential
6	overflow area full		
7	no EOF record written in prime data area		

This is how the preceding error conditions are mapped into the conditions detected by ACUCOBOL-GT:

85	74	Vax	DG	IBM	data-name-1
00	00	00	00	00	10000000
48	91	48	92	13	10000000
49	91	49	92	13	10000000
47	91	47	92	13	10000000
42	91	94	91	92	10000000
38	93	38	92	93	10000000
41	92	41	91	93	10000000
37	95	37	91	93	10000000
93	93	91	94	93	10000000
94	94	97	97	93	01000000
48	90	48	92	13	10000000
24	24	24	24	24	00100000
22	22	22	22	22	00010000
24	24	24	24	24	00010000
34	34	34	34	34	00100000
30	30	30	30	30	10000000
48	90	48	92	13	10000000
49	90	49	92	13	10000000
23	23	23	23	23	01000000
99	99	92	94	23	10000000

85	74	Vax	DG	IBM	data-name-1
44	97	44	92	21	01000000
43	90	43	92	23	01000000
47	90	47	92	13	10000000
10	10	10	10	10	01000000
42	91	42	92	92	10000000
05	00	05	00	10	10000000
39	94	39	9A	95	01000000
46	96	46	10	21	01000000
9A	9A	9A	9A	23	10000000
02	02	02	00	00	00010000
35	94	35	91	93	10000000
37	90	39	91	93	10000000
98	98	30	9B	93	01000000
94	94	39	92	93	01000000
9B	9B	9B	9B	23	10000000
02	02	00	00	00	00010000
07	00	07	00	00	10000000
14	00	14	00	00	01000000
24	00	24	00	24	10000000
21	21	21	21	21	00010000
9C	9C	9C	9C	23	10000000
0M	0M	0M	0M	00	10000000
9D	9D	9D	9D	92	10000000
9Z	9Z	9Z	9Z	92	10000000
9E	9E	9E	9E	92	10000000

Index