Liant Software Corporation

# RM/COBOL®

## User's Guide

### First Edition

LIANT

This manual is a user's guide for Liant Software Corporation's RM/COBOL language. It is assumed that the reader is familiar with programming concepts and with the COBOL language in general.

The information contained herein applies to systems running under Microsoft 32-bit Windows and UNIX-based operating systems.

The information in this document is subject to change without prior notice. Liant Software Corporation assumes no responsibility for any errors that may appear in this document. Liant reserves the right to make improvements and/or changes in the products and programs described in this guide at any time without notice. Companies, names, and data used in examples herein are fictitious unless otherwise noted.



**Liant Software Corporation**
8911 N. Capital of Texas Highway
Austin, TX 78759
U.S.A.

Phone (512) 343-1010
(800) 762-6265
Fax (512) 343-9487

Website http://www.liant.com/

---

## Documentation Release History for the RM/COBOL User's Guide:

| Edition Number | Document Part Number | Applies To Product Version | Publication Date |
|---|---|---|---|
| 1 | 401227 | RM/COBOL version 9 and later | January 2005 |

# Contents

## Appendix D: Support Modules (Non-COBOL Add-Ons) ..... 429



## Appendix E: Windows Printing ..... 447

## List of Figures

## List of Tables

# Preface

## Welcome to RM/COBOL for Windows and UNIX

RM/COBOL for Windows and UNIX is a significantly enhanced version of Liant's widely used RM/COBOL compilers, designed for new program development and execution of programs created with earlier versions of RM/COBOL. Although modeled on the American National Standard COBOL X3.23-1985, there are areas where RM/COBOL varies from the standard. A complete list of these variances is included in Appendix I: *Extension, Obsolete, and Subset Language Elements* (on page 629).

The RM/COBOL operating procedures described in this manual are for use on Microsoft 32-bit Windows and UNIX-based systems that may have remote file access using Novell NetWare (version 3.11 and later), Client for Microsoft Networks, Btrieve software, or NFS (Network File System). The new features in RM/COBOL are described in Chapter 1: *Introduction* (on page 9).

**Note 1** Beginning with version 6.5, the -85 suffix is no longer a part of the RM/COBOL product name. The -85 suffix was used to reflect current technology and to avoid confusion with an earlier product named RM/COBOL, which referred to the 1974 ANSI standard version. Support for RM/COBOL (74) ceased on December 31, 1994.

**Note 2** The term "Windows" in this document refers to Microsoft 32-bit Windows operating systems, including Windows 98, Windows Me, Windows NT 4.0, Windows 2000, Windows XP or Windows Server 2003, unless specifically stated otherwise. As you read through this guide, note that Liant may use two shorthand notations when referring to these operating systems. The term "Windows 9x class" refers to the Windows 98,or Windows Me operating systems. The term "Windows NT class" refers to the Windows NT 4.0, Windows 2000, Windows XP, or Windows Server 2003 operating systems.

# Who Should Use This Book

This book is intended for commercial application developers who are familiar with programming concepts and with the COBOL language in general, and by persons running COBOL programs developed with RM/COBOL.

# Organization of Information

This user's guide is divided into the following parts:

**Chapter 1—Introduction** describes the general concepts of the RM/COBOL compiler and runtime system and how they are used, lists the add-on development tools that are available to support RM/COBOL programs, and explains file naming conventions.

**Chapter 2—Installation and System Considerations for UNIX** explains the installation procedures for RM/COBOL and presents information about the RM/COBOL implementation on UNIX-based operating systems.

**Chapter 3—Installation and System Considerations for Microsoft Windows** explains the installation procedures for RM/COBOL and presents information about the RM/COBOL implementation on Microsoft 32-bit Windows operating systems.

**Chapter 4—System Considerations for Btrieve** presents information about the implementation of RM/COBOL for systems using Btrieve. This chapter also describes the limitations of RM/COBOL indexed files and the Btrieve MicroKernel Database Engine (MKDE).

**Chapter 5—System Verification** describes the suite of verification programs provided with RM/COBOL.

**Chapter 6—Compiling** describes RM/COBOL files, details the RM/COBOL Compile Command, **rmcobol**, and its options, defines the types of errors that can be encountered during program compilation and the messages generated as a result, illustrates and defines each section of the program listing, and presents information on RM/COBOL error recovery.

**Chapter 7—Running** details the RM/COBOL Runtime Command, **runcobol**, and its options, and defines the types of errors that can be encountered during program execution. It also lists the messages generated as a result.

**Chapter 8—RM/COBOL Features** presents information about the implementation of RM/COBOL with respect to specific COBOL statements.

**Chapter 9—Debugging** presents general debug concepts and a detailed discussion of the Debug commands.

**Chapter 10—Configuration** details the methods available for modifying the RM/COBOL default configuration.

**Chapter 11—Instrumentation** details the data-gathering Instrumentation facility. It also describes a sample data analysis program—provided with Instrumentation—called **analysis**.

**Appendix A—Runtime Messages** lists and defines the messages that may be generated during program execution.

**Appendix B—Limits and Ranges** describes RM/COBOL limits and ranges.

**Appendix C—Internal Data Formats** describes and illustrates the internal representation of the data types.

**Appendix D—Support Modules (Non-COBOL Add-Ons)** provides information on using optional support modules to add functionality to the runtime system, compiler, and Indexed File Recovery components of RM/COBOL.

**Appendix E—Windows Printing** describes the subprograms supplied with the RM/COBOL Windows runtime system that allow access to Windows printing features.

**Appendix F—Subprogram Library** describes a set of supplied subprograms that can be called by any RM/COBOL program.

**Appendix G—Utilities** describes the full range of file conversion, management, and manipulation facilities.

**Appendix H—Object Versions** lists the new object features that are incompatible with earlier releases of RM/COBOL.

**Appendix I—Extension, Obsolete, and Subset Language Elements** lists the RM/COBOL extensions to and variances from ANSI COBOL 1985. It also lists obsolete and subset language elements.

**Appendix J—Code-Set Translation Tables** lists each ASCII and EBCDIC hexadecimal value and its corresponding numeric, alphabetic or control character.

**Appendix K—Troubleshooting RM/COBOL** presents troubleshooting tips for some common problems that might occur when running RM/COBOL on different systems.

**Appendix L—Summary of Enhancements** reviews the new features and enhancements that were added to earlier releases of RM/COBOL.

The *RM/COBOL User's Guide* also includes an index (on page 689).

# Related Publications

For additional information, refer to the following publications that are available from Liant Software Corporation:

*Business Information Server (BIS) User's Guide*

*CodeBridge (Calling Non-COBOL Subprograms) User's Guide*

*CodeWatch User's Guide*

*Enterprise CodeBench Getting Started Manual*

*Relativity Client/Server Installation Guides (Windows and UNIX)*

*Relativity Data Manager Installation Guide*

*Relativity DBA Installation Guide and Help File*

*Relativity Designer Installation Guide and Help File*

*Relativity UNIX Data Client Installation Guide*

*RM/COBOL Open File Manager User's Guide*

*RM/COBOL Language Reference Manual*

*RM/COBOL Syntax Summary*

*RM/CodeBench User's Guide*

*RM/InfoExpress for TCP/IP User's Guide*

*RM/Panels Reference Manual*

*Theory of Relativity, A Primer*

*VanGui Interface Builder User's Guides (Delphi and Visual Basic )*

*WOW Extensions (for RM/COBOL) User's Guide, WOW Extensions Designer Help File, and WOW Extensions Functions and Messages Help File*

*XML Extensions for RM/COBOL*

Contact the appropriate vendor for other publications:

Btrieve products are available from Pervasive Software, Inc. (formerly Btrieve Technologies, Inc.).

NetWare products are available from Novell, Incorporated.

Microsoft products are available from Microsoft Corporation.

# Conventions and Symbols

The following conventions and symbols are used or followed throughout this guide.

1. Words in all capital letters indicate COBOL reserved words, such as statements, phrases, and clauses; acronyms; configuration keywords; environment variables; and RM/COBOL Compiler, Runtime, and Recovery Command-line options.
2. Bold lowercase letters represent names of files, directories, programs, commands, and utilities. RM/COBOL accepts uppercase and lowercase filenames. Within this document, the lowercase version is used. Remember, however, that under UNIX filenames are case-sensitive (for example, **TEST4** and **test4** represent different files).

   Bold type style is also used for emphasis on some types of lists.
3. Text that is displayed in a monospaced font indicates user input or system output (according to context as it appears on the screen). This type style is also used for sample command lines, program code and file listing examples, and sample sessions.
4. Italic type identifies the titles of other books and the names of chapters in this guide, and it occasionally is used for emphasis.

   In syntax, italic type denotes a placeholder or variable for information you supply, as described in the following item.

5. The symbols found in the syntax charts are used as follows:

| | |
|---|---|
| *italicized words* | Indicate items for which you substitute a specific value. |
| UPPERCASE WORDS | Indicate items that you enter exactly as shown (although not necessarily in uppercase). |
| . . . | Indicate indefinite repetition of the last item. |
| \| | Separate alternatives. |
| [ ] | Surround optional items. |
| { } | Surround a set of alternatives, one of which is required. |
| {\| \|} | Surround a set of unique alternatives, one or more of which is required, but each alternative may be specified only once; when multiple alternatives are specified, they may be specified in any order. |

6. All punctuation must appear exactly as shown.
7. The term “NetWare” refers to the Novell NetWare operating system.
8. The term “window” refers to a delineated area of the screen, normally smaller than the full screen. The term “Windows” refers to Microsoft 32-bit Windows operating systems, including Windows 98, Windows Me, Windows NT 4.0, Windows 2000, Windows XP, or Windows Server 2003 , unless specifically stated otherwise. As you read through this guide, note that Liant may use two shorthand notations when referring to these operating systems. The term “Windows 9x class” refers to the Windows 98 or Windows Me operating systems. The term “Windows NT class” refers to the Windows NT 4.0, Windows 2000, Windows XP, or Windows Server 2003 operating systems.
9. Examples for UNIX-based systems in this document assume the use of the Bourne Shell (**sh**) command interpreter.
10. Throughout this document, references to a printer refer to the device assigned to PRINTER, in accordance with operating system conventions.
11. RM/COBOL Compile and Runtime Command-line options may be specified either with or without a leading hyphen. Examples in this guide do not show a leading hyphen. If any option on a command line is preceded by a hyphen, then a leading hyphen is required for all options. When assigning a value to an option, the equal sign is optional if leading hyphens are used.

    Command-line options may be specified in either uppercase or lowercase characters. Examples in this guide are shown in uppercase.

    These capabilities are provided to support the command-line syntax of previous versions of RM/COBOL.
12. Any text that applies only to a specific operating system is specified in a **Note** format.
13. Key combinations with a plus sign between key names indicate to press and hold down the first key while pressing the second key. For example, “Press Alt + Esc” means to press and hold down the Alt key and press the Escape key. Then release both keys. A comma between key names means to press and release the keys one after the other.

14. If present in the electronic PDF file, this symbol represents a “note” that allows you to view last-minute comments about a specific topic on the page in which it occurs. This same information is also contained in the README text file under the section, Documentation Changes. In Adobe Reader, you can open comments and review their contents, although you cannot edit the comments. Notes do not print directly from the comment that they annotate. You may, however, copy and paste the comment text into another application, such as Microsoft Word, if you wish.

    To review notes, do one of the following:

    - To view a note, position the mouse over the note icon until the note description pops up.
    - To open a note, double-click the note icon.
    - To close a note, click the Close box in the upper-left corner of the note window.

# Registration

Please take a moment to fill out and mail (or fax) the registration card you received with RM/COBOL. You can also complete this process by registering your Liant product online at: http://www.liant.com.

Registering your product entitles you to the following benefits:

- **Customer support.** Free 30-day telephone support, including direct access to support personnel and 24-hour message service.
- **Special upgrades.** Free media updates and upgrades within 60 days of purchase.
- **Product information.** Notification of upgrades or revisions to RM/COBOL as soon as they are released.

You can also receive up-to-date information about Liant Software and all its products via our web site. Check back often for updated content.

# Technical Support

Liant Software Corporation is dedicated to helping you achieve the highest possible performance from the RM/COBOL family of products. The technical support staff is committed to providing you prompt and professional service when you have problems or questions about your Liant products.

These technical support services are subject to Liant’s prices, terms, and conditions in place at the time the service is requested.

While it is not possible to maintain and support specific releases of all software indefinitely, we offer priority support for the most current release of each product. For customers who elect not to upgrade to the most current release of the products, support is provided on a limited basis, as time and resources allow.

## Support Guidelines

When you need assistance, you can expedite your call by having the following information available for the technical support representative:

1. Company name and contact information.
2. Liant product serial number (found on the media label, registration card, or product banner message).
3. Product version number.
4. Operating system and version number.
5. Hardware, related equipment, and terminal type.
6. Exact message appearing on screen.
7. Concise explanation of the problem and process involved when the problem occurred.

## Test Cases

You may be asked for an example (test case) that demonstrates the problem. Please remember the following guidelines when submitting a test case:

- The smaller the test case is, the faster we will be able to isolate the cause of the problem.
- Do not send full applications.
- Reduce the test case to one or two programs and as few data files as possible.
- If you have very large data files, write a small program to read in your current data files and to create new data files with as few records as necessary to reproduce the problem.
- Test the test case before sending it to us to ensure that you have included all the necessary components to recompile and run the test case. You may need to include an RM/COBOL configuration file.

When submitting your test case, please include the following items:

1. **README text file that explains the problems.** This file must include information regarding the hardware, operating system, and versions of all relevant software (including the operating system and all Liant products). It must also include step-by-step instructions to reproduce the behavior.
2. **Program source files.** We require source for any program that is called during the course of the test case. Be sure to include any copy files necessary for recompilation.
3. **Data files required by the programs.** These files should be as small as possible to reproduce the problem described in the test case.

# Chapter 1: Introduction

This introductory chapter of the *RM/COBOL User's Guide* lists the new features in the version 9 release and provides an overview of the RM/COBOL product. It explains the general concepts of the RM/COBOL compiler and runtime system and how they are used, lists the additional add-on development tools that are available to support RM/COBOL programs, and explains file naming conventions.

## What's New

The following sections summarize the major new enhancements available in RM/COBOL version 9 for Windows and UNIX. This summary describes the main features of each enhancement and tells you where to look for more information about them. The *RM/COBOL Language Reference Manual* and this user's guide primarily contain the details regarding these features.

**Note** For information on the significant enhancements in previous releases of RM/COBOL, see Appendix L: *Summary of Enhancements* (on page 661).

Deficiencies that are version-specific or are discovered after printing are described in the README files contained on the delivered media. For further information, see also Appendix B: *Limits and Ranges* (on page 399).

### New Runtime System Features

The RM/COBOL version 9 for Windows and UNIX runtime system has been enhanced with the following new features:

- **Memory Fill Character.** The new F Runtime Command Option may be used to specify a fill character for read-write memory. Previously, read-write memory was filled with a space character, which is still the default fill character. The fill character can also be specified in a configuration file. See Environment Runtime Command Options (on page 181) for further details.
- **LIKE Condition Variable Pattern.** A LIKE condition pattern specified as an alphanumeric data item containing the pattern value, that is, a variable pattern, is now trailing blank stripped by default. The trailing blanks are not considered significant in the pattern value. A configuration keyword, STRIP-LIKE-PATTERN-TRAILING-SPACES, has been added to suppress this new behavior so that trailing spaces are considered significant in the pattern value; that is, they must match in the value of the subject data item for a true result. For more

information, see the keyword description in the RUN-ATTR configuration record (on page 313).

- **ANSI/OEM on Windows.** The user can now choose to use the ANSI or OEM codepage as the assumed codepage for character data on Windows. Prior to version 9, the RM/COBOL for Windows runtime assumed that the OEM codepage described character data in memory and data files. Thus, the runtime would convert displayed or printed data from OEM to ANSI if the display or printer font had a default script of ANSI. The runtime system would also convert keyboard input from ANSI to OEM. There were a variety of other necessary conversions that the runtime system performed since Windows is primarily based on the ANSI character set. This behavior was consistent with the historical use of RM/COBOL on MS-DOS, where OEM character sets were used and thus existing user data files contained OEM characters. Version 9 adds support for assuming the ANSI codepage on Windows. In the ANSI mode, the runtime will convert displayed or printed data to OEM only if a font with a default script of OEM/DOS is used and keyboard input will not be converted. For more information, see Character Set Considerations for Windows (on page 97).

  **Note** The ANSI mode is appropriate only for new projects with no existing data files or when the user is willing to convert character data in existing data files from OEM to ANSI. It would be a severe mistake to mix ANSI and OEM data in the same data file. Of course, none of this is relevant unless characters from the top half of the codepage (code points 128 – 255) are used, since code points 0–127 have a common ASCII interpretation in all codepages.

- **C$GetNativeCharset.** The C$GetNativeCharset (on page 541) library routine has been added to query the runtime about the native character set on Windows, that is, whether the program is running with the OEM or ANSI codepage assumed for characters in memory and data files. (Prior to version 9, the runtime always assumed the OEM codepage for character data.) The actual codepage number can also be obtained. On UNIX, this library routine returns "NONE" for the character set and 0 for the codepage number.

- **P$ResetPrinter and P$SetLineSpacing**. Two new P$ routines have been added. P$ResetPrinter (on page 490) resets the P$ printer. P$SetLineSpacing (on page 476) sets the number of lines per inch. For more information on the P$ routines, see Appendix E: *Windows Printing* (on page 447).

- **New Object Version Level.** Object version 12 has been introduced to support the CURSOR clause (see New Compiler Features), large debugging line numbers, new treatment of the SECURE phrase in an ACCEPT statement, and extended file parameters. For more information on object version 12, see Appendix H: *Object Versions* (on page 617).

- **Faster Initialization.** The runtime initialization process has been enhanced to be significantly faster when a second or subsequent run unit is started within a short period of time, such as in testing or batch processing.

## New Compiler Features

The RM/COBOL version 9 for Windows and UNIX compiler has been enhanced with the following new features:

- **COPY Statement SUPPRESS Phrase.** The COPY statement now allows the SUPPRESS phrase, which suppresses printing to the source listing file any source lines included because of the COPY statement. See the "COPY Statement" section in Chapter 1: *Language Structure* of the *RM/COBOL Language Reference Manual*.
- **WHEN-COMPILED Special Register.** The compiler now supports the WHEN-COMPILED special register, which provides a value indicating the date and time of the compilation. The default format for the value matches the IBM OSVS COBOL implementation of this special register (`"hh.mm.ssMMM DD, YYYY"`). There is a configuration option to use the value for the IBM VSC2 COBOL implementation (`"MM/DD/YYhh.mm.ss"`). The configuration capability is quite flexible, allowing the user to specify a specific date and time format, along with user-specified text; for example, a copyright notice of the form "Copyright YYYY/MM/DD. All rights reserved." could be configured for the value of the WHEN-COMPILED special register. The actual compilation date would replace YYYY/MM/DD in the value. For details on formatting the value of the WHEN-COMPILED special register, see the WHEN-COMPILED-FORMAT keyword (on page 300) of the COMPILER-OPTIONS configuration record and the description of special registers in the "Reserved Words" topic in Chapter 1: *Language Structure* of the *RM/COBOL Language Reference Manual*.
- **CONSOLE IS CRT Clause.** The CONSOLE IS CRT clause has been added to the Special-Names paragraph. This clause is used in some COBOL dialects to indicate that all ACCEPT and DISPLAY statements, which do not specify any phrases specific to a particular format, should be treated as implementor-defined terminal I-O statements as opposed to ISO 1989-1985 standard ACCEPT and DISPLAY statements. The clause has been added to ease conversion from such COBOL dialects and has the same meaning in RM/COBOL as in those other COBOL dialects. See Chapter 3: *Environment Division* of the *RM/COBOL Language Reference Manual*.
- **CRT STATUS Clause.** The CRT STATUS clause has been added to the Special-Names paragraph. This clause specifies a data item to receive the field termination code after ACCEPT statements are executed. See Chapter 3: *Environment Division* of the *RM/COBOL Language Reference Manual*.
- **CURSOR Clause.** The CURSOR clause has been added to the Special-Names paragraph. This clause specifies a cursor position data item for Screen Section ACCEPT statements. When specified, the cursor position is used to initially position the cursor at the beginning of the ACCEPT statement and is set to the cursor position at the end of the ACCEPT statement. This clause aids in input error processing because the cursor can be returned to the input field with the error or to the last field input by the user. See Chapter 3: *Environment Division* of the *RM/COBOL Language Reference Manual*.
- **Options Listing.** The compiler summary listing has been enhanced to produce more information about options specified for the compilation on the command line and in configuration. This ensures not only that the listing more thoroughly documents the options used, but also may be helpful when diagnosing object

behavior that is dependent on those options. See Chapter 6: *Compiling* of the *RM/COBOL User's Guide*.

- **ANSI/OEM on Windows.** The compiler can now be run in ANSI or OEM mode on Windows. This affects the way characters are interpreted for displaying or printing during compilation and parallels the runtime support for ANSI and OEM character sets. The compilation mode is recorded in the object so that utilities, such as the Map Program File (rmmappgm) utility (on page 586), can retrieve it for information, but is otherwise generally not used at execution. The runtime should be used in the same mode as used for compilation so that characters will be interpreted as they were at compilation, although this is not a requirement. The same object program can be run in either ANSI or OEM mode, but nonnumeric literals that use code points in the range 128 – 255 will have different interpretations depending on the runtime mode.

- **Long User-Defined Words.** The warning about user-defined words, such as data-names, condition-names, file-names, paragraph-names, section-names, and so forth, which exceed the standard COBOL limit of 30 characters, has been changed. The warning occurs only on words longer than 240 characters, at which length the compiler truncates the name if it is longer. The FIPS flagging option for extension flagging can be used to obtain a warning when names that exceed the standard COBOL limit of 30 characters are used. Program-names are still restricted to 30 characters in length and are truncated to that length if a longer program-name is specified.

- **Relaxed Reference Modification Rules.** The rules for reference modification have been relaxed to allow truncation without error when the leftmost-character-position or the sum of the leftmost-character-position and the length minus one exceeds the number of characters in the referenced modified data item. This behavior matches other COBOL dialects that do not enforce the standard COBOL rules for reference modification. A configuration option has been provided to enforce the strict rules for reference modification. See the STRICT-REFERENCE-MODIFICATION keyword of the COMPILER-OPTIONS configuration record (on page 287).

- **Suppressible Diagnostic Messages.** The compiler configuration has been enhanced to allow selective suppression of compilation diagnostic messages. This allows the user to choose to ignore certain warnings and errors that do not concern them. RM/COBOL is strict about enforcing standard COBOL rules, producing errors where other dialects of COBOL choose to ignore the non-standard syntax or semantics. Even though RM/COBOL notes the error, in many cases the error can be ignored when the user is not concerned about conforming to standard COBOL. See the NO-DIAGNOSTIC keyword of the COMPILER-OPTIONS configuration record (on page 287) for additional information on this enhancement.

- **Large Program Enhancements.** Additional support for compiling large programs has been added. Listing line numbers have been changed from five to six digits and the cross reference listing supports six digit line numbers, if necessary. Internal support for debugging line numbers exceeding 65535 has been added. The compiler memory allocation scheme has been enhanced to minimize the effect of failing to set a large enough initial workspace size (W Compile Command Option) so that large programs compile more quickly regardless of the initial workspace size. Support for a large number of file parameters has been expanded to about four times the previous limit.

- **ACCEPT and DISPLAY Statement Enhancements.** The ACCEPT and DISPLAY statements have been enhanced with syntax features from other common COBOL dialects. This eases the conversion effort when converting to RM/COBOL from those other COBOL dialects, since the statements do not have to be re-written to remove the other forms of syntax. These changes, in conjunction with the addition of the CONSOLE IS CRT, CRT STATUS, and CURSOR clauses to the Special-Names paragraph, significantly reduce conversion effort. Also, the SECURE phrase in an ACCEPT statement now matches the SECURE clause in the Screen Section, replacing input characters with asterisks instead of treating the SECURE phrase as a synonym for OFF.
- **Faster Initialization.** The compiler initialization process has been enhanced to be significantly faster when a second or subsequent compilation is started within a short period of time, such as in batch processing of multiple source programs.

# RM/COBOL Software

RM/COBOL, delivered on appropriate media, contains a large number of individual files and programs. The actual number of files and programs depends on the specific version of the product you purchased and whether you purchased a development or a runtime-only system. The delivered media contains one or more README files, which list the actual files and programs delivered. Please check these README files after you have installed the product to make sure that you have received all of the appropriate files and programs.

# RM/COBOL Compiler

The RM/COBOL compiler reads COBOL source code and produces object files that can be executed using the runtime system. These object files are portable, and they can be executed by an RM/COBOL runtime system on many computer configurations—even computer configurations that are different from the one used to compile the object files. For more information on compiling COBOL programs, see Chapter 6: *Compiling* (on page 135).

# RM/COBOL Runtime System

The RM/COBOL runtime system is used to execute compiled COBOL programs. Liant Software Corporation provides a different runtime system for each supported computer, and they help to insulate the COBOL programmer from the differences between computers. The runtime system also includes a debugger to assist in developing COBOL programs. For more information on running COBOL programs, see Chapter 7: *Running* (on page 177).

# CodeWatch

CodeWatch is a fully integrated development system for Windows that is included with the RM/COBOL development system. CodeWatch supports the entire development cycle, including editing, compiling, and debugging RM/COBOL applications. CodeWatch can be used to debug and change programs that are independently compiled, without requiring you to build projects. Instead, all the knowledge about the structure of your application is built up debugging sessions. For more information, see the *CodeWatch* manual.

# CodeBridge

CodeBridge is a cross-language call system included with the RM/COBOL development system. This facility simplifies communication between COBOL programs and non-COBOL subprograms (such as those written in C or C++). CodeBridge allows COBOL programmers to call external APIs or custom-developed subprograms without introducing "foreign" language and data dependencies into their programs. For more information, see the *CodeBridge* manual.

# Internal Libraries and Utility Programs

The RM/COBOL runtime system also includes several built-in library routines to perform functions not described in the COBOL standard. Among other things, these routines can be used to determine information about program arguments, control the display screen, and execute other (non-COBOL) programs. For more information, see Appendix F: *Subprogram Library* (on page 527).

In addition, a library of P$ subprograms, supplied with the RM/COBOL for Windows runtime system, allows access to Windows printing features. This library is described in Appendix E: *Windows Printing* (on page 447).

There are several utility programs delivered with RM/COBOL. These utility programs are used to manage and manipulate both data files and RM/COBOL object files. For more information on the utility programs, see Appendix G: *Utilities* (on page 581).

# Add-On Packages

Several add-on packages are available from Liant Software Corporation to support RM/COBOL programs. They include the following:

- **XML Extensions for RM/COBOL.** A facility that allows RM/COBOL applications to interoperate freely with other applications that use XML (eXtensible Markup Language, the universal standard format for structured documents and data on the Web). This capability to import and export XML documents easily to and from COBOL data structures turns RM/COBOL into an "XML Engine."

- **Relativity.** An integrated tool set that provides relational database functionality for COBOL data without any application modifications or data conversions. It also provides a full-featured, Microsoft Windows Open Database Connectivity (ODBC)-compliant relational database engine that allows SQL-based access to COBOL application data.
- **Enterprise CodeBench.** An integrated, graphical development environment for developing RM/COBOL applications on UNIX within the environment of Windows 9x-class operating systems.
- **RM/InfoExpress.** A file management system designed to optimize RM/COBOL data file access on various local area networks (LANs) and wide area networks (WANs). Implementation is available for TCP/IP (Transmission Control Protocol/Internet Protocol).
- **VanGui Interface Builder.** An interface builder that allows Visual Basic and Delphi to serve as graphical user interface (GUI) front-ends to COBOL programs running across a network.
- **RM/Panels.** A portable screen I/O package that allows WYSIWYG development of RM/COBOL displays.
- **WOW (Windows Object Workshop) Extensions.** A visual tool for developing full-featured Windows applications completely in RM/COBOL.
- **RPC (Remote Procedure Calls).** A tool for building distributed RM/COBOL applications for LANs, WANs, and the Internet.
- **Cobol-CGIX (Common Gateway Interface).** A tool for integrating RM/COBOL applications with the Internet's World Wide Web.
- **InstantSQL.** A package for embedding SQL statements in COBOL source programs so that the programs can access ODBC-enabled relational databases using SQL statements.

# File Naming Conventions

On those operating systems that support case-sensitive filenames, RM/COBOL filenames can contain any combination of uppercase and lowercase letters, and numerals.

The Windows version of RM/COBOL, like Microsoft 32-bit Windows, supports long filenames and filenames containing embedded spaces. RM/COBOL filenames can be enclosed in quotation marks (ASCII code 22 hex). RM/COBOL filenames containing embedding spaces must be enclosed in quotation marks to avoid having the embedded spaces interpreted as separators. For example:

```
"C:\My Source Directory\My COBOL Program.cbl"
```

**Note** Although 32-bit Windows stores long filenames with case preserved, filenames are always compared and searched for in a case-insensitive manner (that is, filenames that differ only in whether letters are uppercase or lowercase refer to the same physical file).

RM/COBOL uses the extensions **.cbl**, **.cob**, and **.lst** to designate the source, object and listing files of a program. This allows all three files to reside in the same directory. These extension names may be changed with the EXTENSION-NAMES configuration record (on page 308).

Source files do not need to have an extension of **.cbl**; in fact, they do not need an extension at all. If the compiler cannot locate the source file with the name given and the name does not have an extension, it will try to locate the file again, using first **.cbl** as an extension to the filename and then **.CBL**.

The RM/COBOL compiler always creates object and listing files with extensions. It will either replace the current extension of the source file, or append an extension if the source filename does not have one. The case of the extension will match the case of the first character of the source file's extension, or the first character in the source file's name if there is no extension. If there is no extension and the first character of the source filename is not a letter, the extension will be lowercase.

The RM/COBOL runtime system does not require object files to have an extension of **.cob**. However, since the compiler generates objects with the **.cob** extension, the runtime system will try to locate object files by adding first **.cob** and then **.COB**, but only if the original filename does not already have an extension. Table 1 contains sample filenames.

**Table 1: Sample Filenames**

| Source Filename | Resulting Object Filename |
|---|---|
| TESTFILE | TESTFILE.COB |
| Testfile | Testfile.COB |
| Test | Test.COB |
| Test.Cbl | Test.COB |
| Test.cbl | Test.cob |
| test.xyz | test.cob |
| test.XYZ | test.COB |
| tESTFILE | tESTFILE.cob |
| test | test.cob |
| test.CBL | test.COB |
| test.cbl | test.cob |
| 2TESTFIL | 2TESTFIL.cob |

# Chapter 2: Installation and System Considerations for UNIX

This chapter lists the hardware and software required to install the RM/COBOL product, describes how to install RM/COBOL, and provides information specific to using RM/COBOL for UNIX-based operating systems.

Your computer configuration is the assembled set of hardware and software that makes up your system. Before you can develop or run RM/COBOL programs, your configuration must meet or exceed the requirements set forth in this chapter.

## System Requirements for UNIX

The version of RM/COBOL that you have purchased is for a particular combination of hardware and operating system. Several items listed below vary depending on the actual version of the product that you have purchased.

### Required Hardware

- **Memory.** The amount of memory depends on the specific version of the product that you have purchased.
- **Hard Disk.** A hard disk drive is required for installing this product. The amount of space required is version-specific. The average minimum disk space required to install this product is 4.5 megabytes (4.5 MB) for a development system and 2.5 megabytes (2.5 MB) for a runtime system.
- **Removable Media.** One CD-ROM drive and one double-sided, high-density 3.5-inch floppy disk drive is required for installing this product.
- **Terminal.** A terminal with a minimum capacity of 80 columns.

# System Installation for UNIX

RM/COBOL is delivered on media suitable for your configuration. In order to use this product, you must install the contents of the distribution media onto your hard disk.

There are four main steps to installing RM/COBOL for UNIX:

1. Loading the license file (see the following topic).
2. Loading the distribution media (on page 21).
3. Performing the installation (on page 22).
4. Unloading the distribution media (on page 23).

## Loading the License File

The Liant license file is a normal text file distributed on an MS-DOS-formatted diskette. Not all UNIX operating systems, however, can read an MS-DOS-formatted diskette, and not all UNIX server machines have diskette drives. To make the license file available to the RM/COBOL for UNIX installation script, two techniques are provided:

1. Mounting the diskette as an MS-DOS file system (see below).
2. Transferring the Liant license file via FTP from a Windows client (on page 20).

### *Mounting the Diskette as an MS-DOS File System*

Use this option to load the license file if the UNIX operating system supports MS-DOS file systems and your hardware has a diskette drive installed. Instructions for specific platforms and versions of UNIX are provided.

- **IBM AIX 4.3 and 5, HP-UX 11.0, Intel UNIX System V Release 4, and Digital/Compaq Tru64 UNIX V4, V5**

  These platforms do not support mounting MS-DOS diskettes. Use the FTP instructions on page 20 to transfer the license file to the UNIX server.

- **Linux**

  a. Insert the diskette into the diskette drive.

  b. Log in as root and enter:

  ```
  mount –t msdos /dev/fd0H1440 /mnt/floppy
  ```

  c. Copy the license file to the /tmp directory:

  ```
  cp /mnt/floppy/liant.lic /tmp/liant.lic
  ```

  d. Dismount the diskette with the following command and then remove the diskette from the diskette drive:

  ```
  umount /mnt/floppy
  ```

- **SCO OpenServer 5**

  a. Insert the diskette into the diskette drive.

  b. Log in as root and enter:

  ```
  mount –f DOS,lower /dev/fd0 /floppy
  ```

  **Note** It may be necessary to create the mount directory, /floppy, before executing this command.

  c. Copy the license file to the /tmp directory:

  ```
  cp /floppy/liant.lic /tmp/liant.lic
  ```

  d. Dismount the diskette with the following command and then remove the diskette from the diskette drive:

  ```
  umount /floppy
  ```

- **Sun Solaris SPARC (2.6, 7, and 8) and Intel x86 (2.6)**

  a. Insert the diskette into the diskette drive.

  b. Log in as root and enter:

  ```
  volcheck
  ```

  c. Copy the license file to the /tmp directory:

  ```
  cp /floppy/floppy0/LIANT.LIC /tmp/liant.lic
  ```

  d. Dismount the diskette with the following command and then remove the diskette from the diskette drive:

  ```
  eject floppy
  ```

- **UnixWare 7.1.1** (or later)

  a. Insert the diskette into the diskette drive.

  b. Log in as root and enter:

  ```
  mount –F dosfs /dev/dsk/f0q18dt /Disk_A
  ```

  c. Copy the license file to the /tmp directory:

  ```
  cp /Disk_A/liant.lic /tmp/liant.lic
  ```

  d. Dismount the diskette with the following command and then remove the diskette from the diskette drive:

  ```
  umount /Disk_A
  ```

## Transferring the Liant License File via FTP from a Windows Client

To transfer the Liant license file from a Windows client to the UNIX server, use one of the many graphical FTP utilities available on Windows and transfer the **liant.lic** file as a text file. You can also follow this procedure:

1. On the Windows client, insert the diskette into the diskette drive.

   These instructions assume that this is drive A. If it is another drive, change the drive letter to the appropriate letter in the remaining instructions.

3. Open an MS-DOS Prompt window by clicking Start on the task bar, point to Programs, and then click MS-DOS Prompt.
4. Connect to the UNIX server by entering:

   ```
   ftp UnixServerName
   ```

   where *UnixServerName* is the network name of your UNIX server.

5. Change the directory to the /tmp directory:

   ```
   cd /tmp
   ```

6. Specify a text file transfer:

   ```
   ascii
   ```

7. Send the license file to the UNIX server:

   ```
   send A:\LIANT.LIC liant.lic
   ```

8. Disconnect from the UNIX server:

   ```
   bye
   ```

9. Close the MS-DOS Prompt window with the following command and then remove the diskette form the diskette drive:

   ```
   exit
   ```

## Loading the Distribution Media

To load the distribution media on the UNIX machine:

1. Insert the RM/COBOL for UNIX CD-ROM in the appropriate CD-ROM drive.
2. Log in as root.
3. Enter the appropriate mount command for your system. See the examples listed below.

**Note 1** In the list that follows, the standard mount directory names are used where the UNIX operating system has such a standard. If the operating system does not follow a standard, the name */cdrom* is used. In such cases, it will be necessary either to create the directory */cdrom* or to substitute the preferred mount directory name for */cdrom*.

**Note 2** The device names below are examples. The actual device name is dependent on the hardware configuration of your UNIX server. It may be necessary to substitute the proper value for your system. Consult your UNIX System Administrator for more details.

| System | Mount Command |
|---|---|
| Digital/Compaq Tru64 UNIX V4, V5 | `mount -t cdfs -o ro,noversion /dev/rz4c /cdrom` |
| HP-UX 11.0 | `mount –F cdfs –o ro,cdcase /dev/dsk/c0t4d0 /cdrom` |
| IBM AIX 4.3 and 5 | `mount –o ro –v cdrfs /dev/cd0 /cdrom` |
| Intel UNIX System V Release 4 | `mount –o ro –F cdfs /dev/cdrom/c0t4l0 /cdrom` |
| Linux | `mount –o ro –t iso9660 /dev/cdrom /mnt/cdrom` |
| SCO OpenServer 5 | `mount –o ro –f ISO9660,lower /dev/cd0 /cdrom` |
| Sun Solaris SPARC (2.6, 7, and 8) and Intel x86 (2.6) | If Solaris does not automatically load the CD-ROM, log in as root and enter: `volcheck` |
| UnixWare 7.1.1 (or later) | `mount –F cdfs –o ro /dev/cdrom/c1b0t0l0 /CD-ROM_1` |

## Performing the Installation

After the CD-ROM has been successfully mounted, you will need to do the following:

1. Change the directory to the mount point for the CD-ROM. For example, enter:

   ```
   cd /cdrom
   ```

2. From the mount point, execute the installation script using the following command:

   ```
   sh ./install.sh
   ```

3. The installation script prompts you for all the information that it needs before beginning the actual installation. Answer the prompts of the installation script accordingly.

Messages are displayed periodically indicating the status of the installation.

During the execution of this command, you are prompted about which optional features you wish to install. For additional information, see Appendix D: *Support Modules (Non-COBOL Add-Ons)* on page 429. For example:

- You are asked whether you want to use the terminfo or termcap terminal interface. For more information, see Terminal Interfaces (on page 33). Versions of RM/COBOL prior to 7.1 supported the terminfo and termcap terminal interfaces with different versions of the runtime (**runcobol**) and different versions of the indexed file recovery (**recover1**) utility. Version 7.1 of RM/COBOL uses separate support modules to support the two terminal interfaces so only a single runtime and recovery utility are present on the distribution media. If you later decide to switch from terminfo to termcap or vice versa, you will need to run the installation command again and respond appropriately to the prompts.
- You are also asked whether you want to install the FlexGen support module, the RM/InfoExpress Client support module, or the Automatic Configuration File (on page 282) support module.

  **Note** If you elected to install the Automatic Configuration File support module, you will be able to add a configuration file for the runtime system, the compiler, and/or the recovery utility, which will be used automatically without the need to specify it on the command line.
- If the installation process detects the presence of any other support modules (for example, the Enterprise CodeBench server support module) in the install directory, you will be asked whether you want to install those support modules.

RM/COBOL is distributed with a default configuration that will satisfy your system requirements. Configuration options for your system are found in Chapter 10: *Configuration* (on page 281).

## Unloading the Distribution Media

To unload (remove) the distribution media from the hardware:

1. Enter the appropriate command for your system. See the examples listed below.
2. Remove the distribution media from the CD-ROM drive.

| System | Umount Command |
|---|---|
| IBM AIX 4.3 and 5<br>HP-UX 11.0<br>SCO OpenServer 5<br>Intel UNIX System V Release 4<br>Digital/Compaq Tru64 UNIX V4, V5 | `umount /cdrom` |
| Linux | `umount /mnt/cdrom` |
| Sun Solaris SPARC (2.6, 7, and 8)<br>and Intel x86 (2.6) | `eject cdrom` |
| UnixWare 7.1.1 (or later) | `umount /CD-ROM_1` |

# System Removal for UNIX

The RM/COBOL system now comes with a command to remove the files installed in the system command directory (or other execution directory of your choice). Issue the following command to remove the RM/COBOL installed files, including any support modules:

```
./rmuninstall
```

During the execution of this command, you are asked to provide the location of the RM/COBOL installed files (that is, **/usr/bin** or the execution directory specified when the RM/COBOL files were installed). You are then asked which files you wish to remove.

You may elect to remove all of the RM/COBOL installed files, “complete (not prompted)” mode, or the specific files of your choice, “selective (prompted)” mode. If, for example, you decide that you no longer want to use the RM/InfoExpress client module, you may remove just that single file. After the RM/COBOL system is removed, it is still possible to run the installation command to reinstall RM/COBOL.

# Locating RM/COBOL Files on UNIX

## File Locations within Operating System Pathnames on UNIX

File locations are determined by the pathname of the file, according to operating system rules and conventions. A fully qualified pathname consists of an optional directory path with slash separators followed by a filename. The directory path may begin with a leading slash, tilde (~), or period (.) character. A directory path with a leading slash or tilde is fully specified and identifies a filename relative to the root file system. A directory path without a leading slash or tilde character specifies a filename relative to the current directory.

If a pathname is specified without a directory path, RM/COBOL searches the current directory.

Specifying a directory path with a leading slash or tilde indicates to RM/COBOL that an exact filename has been specified. If RM/COBOL cannot find the file in the specified location, it will not look elsewhere. If you do not specify a directory path, and RM/COBOL cannot find the file relative to the current directory, it will search for the file according to the directory search sequence. If a directory path is specified, but there is no leading slash or tilde, then the EXPANDED-PATH-SEARCH keyword (see page 317) of the RUN-FILES-ATTR configuration record determines whether the directory search sequence will be used. When the configuration keyword is set to its default value of NO, the directory search sequence will not be used. If the value is set to YES, then the entire name, including the directory path, will be appended to each entry in the directory search sequence in an attempt to locate the file.

The tilde (~) character at the beginning of a pathname is used to refer to home directories. When followed by a slash or standing alone, it expands to the user's home directory as reflected in the environment variable HOME. When followed by a name consisting of letter and digit characters, the name identifies the user whose home directory should be used.

## Directory Search Sequences on UNIX

You can direct RM/COBOL to search for a file not found in the current working directory by using a predefined directory search sequence. There are two directory search sequences: one for the compiler and one for the runtime system.

To direct the RM/COBOL compiler to use the directory search sequence, set the environment variable RMPATH as follows:

```
RMPATH=path[:path] ... ; export RMPATH
```

To direct the RM/COBOL runtime system to use the directory search sequence, set the environment variable RUNPATH as follows:

```
RUNPATH=path[:path] ... ; export RUNPATH
```

In both commands, *path* indicates the directory that is to be searched for the file and has the form:

```
[/]directory[/directory ] ...
```

where, *directory* is the name of an existing directory.

If multiple *path*s are specified, they must be separated with colons. If the file is not located in the current directory or the explicitly defined paths and if the file should be created, then the file is created in the current directory.

Figure 1and Figure 2 illustrate the compiler and runtime system search sequences on UNIX, respectively.

**Figure 1: Compiler Search Sequence**

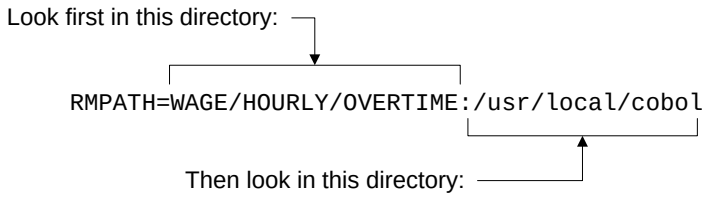
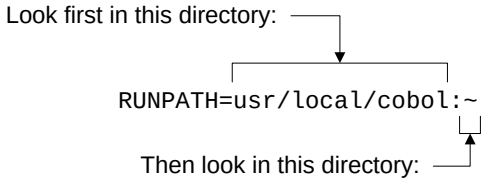



**Figure 2: Runtime System Search Sequence**




The compiler and runtime system may be executed from a directory other than the current directory if a complete pathname is specified on the command line, or if the PATH directory search feature is used. If a complete pathname is not specified, the list of directories specified by PATH is searched. Note that the current directory is not implicitly searched with the PATH environment variable.

The compiler, runtime system, and recovery utility (**recover1.exe**) require access to other files in order to operate, including the license vault. The license vault must reside in the same directory as the executable file. RM/COBOL looks for the other files first in the directory containing the executable file, then in the current directory, and finally in the directories specified in the PATH environment variable.

## File Access Names on UNIX

The file access name you specify in the COBOL source program specifies the physical file or device to which input or output is directed. For information on specifying the file access name in a COBOL source program, see the discussion of the ASSIGN clause (file control entry) in Chapter 3: *Environment Division*, and the discussion of the VALUE OF clause in Chapter 4: *Data Division*, of the *RM/COBOL Language Reference Manual*.

To establish synonymy between a file access name specified in your source program and another name specified when the program is run, use environment variables that are set before starting the runtime system.

If you specified a generic file access name for program input-output and wish to direct it to a specific device or file, enter:

```
file-access-name-1 = file-access-name-2;
    export file-access-name-1
```

A generic file access name is one that does not specify a directory path. Since the format of physical pathnames, including conventions of directory names, varies from one operating system to another, for maximum portability it is recommended that source programs specify generic file access names, preferably with eight or fewer letters. This recommendation applies only when the file access name is hard coded into the program as a literal.

For example, if the file control entry specifies:

```
SELECT REPORT-FILE ASSIGN TO PRINT, "report"
```

and no environment variable named “report” exists, RM/COBOL will create a file named **report** in the current directory.

If, prior to running the program, you enter the command:

```
REPORT=/dev/lp; export report
```

all program output written to REPORT-FILE will be written to **/dev/lp**.

If—again prior to execution—you enter the command:

```
REPORT=/output/audit.lst; export report
```

RM/COBOL will create a file **audit.lst** in the directory **/output** without any need to modify or recompile the source program.

The RESOLVE-LEADING-NAME and RESOLVE-SUBSEQUENT-NAMES keywords of the RUN-FILES-ATTR configuration record can be used to force resolution of one or more of the directory names from the environment. For more information, see the discussion of the RUN-FILES-ATTR configuration record (on page 316).

When environment variables are not used, the file access name in the COBOL program specifies the UNIX filename. The effect of a prior environment variable assignment may be canceled by the command:

```
unset file-access-name
```

Whether or not an environment variable is used to modify the file access name, if the resulting file access name does not include a directory path, RUNPATH will be used by the runtime system to obtain the fully qualified pathname. For additional information, see File Locations within Operating System Pathnames on UNIX (on page 24).

Control characters, spaces, and nonprintable characters (per the locale setting) are removed from the file access name, except that, if the path begins with a pipe character ('|'), white space characters are preserved after the first non-white space character following the pipe character.

After environment variable mapping and removal of control characters, spaces, and nonprintable characters, the file access name is checked against the DEFINE-DEVICE table, which is either the default DEFINE-DEVICE table, or, if a configuration file with one or more DEFINE-DEVICE configuration records is supplied, the specified DEFINE-DEVICE entries in the configuration. See the DEFINE-DEVICE configuration record (on page 304) for more information. If the resulting file access name matches a DEFINE-DEVICE entry, the PATH value from that DEFINE-DEVICE entry becomes the final file access name, which is not further modified. If the resulting file access name does not match an entry in the DEFINE-DEVICE table, it is not further modified.

When the resulting file access name is "*", then

- for a sequential input file, the standard input file (stdin) is read; and,
- for a sequential output file, the standard output file (stdout) is written.

When the resulting file access name has a leading pipe character ('|'), then the pipe character and any immediately following white space characters are removed. The remainder of the file access name is treated as a shell command to be started when the file is opened for input or output. The open mode of the file determines the direction of the pipe as follows:

- When the file is opened for input, the **shell** command is started with its standard output redirected to the input of the associated COBOL file. That is, the COBOL program will read the records written by the process. The shell command may be a pipeline (a series of commands separated by pipe characters), in which case the COBOL program will read the output of the rightmost command (the rightmost command must start a program that writes to standard output and output redirection using the > character must not be specified for the rightmost command). For example, an input file access name value "| sort -r -k 5 file1.txt file2.txt | uniq | grep Fail" will result in reading records from the files **file1.txt** and **file2.txt** that have been sorted and merged together in reverse order on field five of the record without any duplicate records and only records that have the word "Fail" in them.
- When the file is opened for output, the **shell** command is started with its standard input redirected to the output from the associated COBOL file. That is, the process will read the records written by the COBOL program. The shell command may be a pipeline (a series of commands separated by pipe characters), in which case the leftmost command will read the output of the COBOL program (the leftmost command must start a program that reads from standard input and input redirection using the < character must not be specified for the leftmost command). For example, an output file access name value "| sort -r | uniq | grep Pass >results.txt" will cause the records written by the COBOL program to be sorted in reverse order, duplicates removed, and only records with the word "Pass" in them written to the file **results.txt**.

If two or more COBOL files in the same run unit are open at the same time and specify the same file access name with a leading pipe character, each will start a separate process and pipe input or output from or to its associated process. In contrast, two or more files open at the same time in the same run unit will start one

process and share the pipe to that process if they have the same file access name and that file access name is resolved through a DEFINE-DEVICE record to a pipe.

When the resulting file access name is PRINTER or PRINTER1, then the default configuration writes the file to the print spooler. See Printer Support (on page 227) for additional information on printing.

When the resulting file access name is TAPE, then the default configuration writes the file to the default tape device. See Tape Support (on page 227) for additional information on tape devices.

The DEFINE-DEVICE configuration record may define other file access names that are to be treated as devices and may change the default treatment of PRINTER and TAPE. See DEFINE-DEVICE configuration record (on page 304) for additional information on configuring file access names that are to be treated as devices.

The resulting file access name should follow the operating system rules for valid filenames and pathnames.

# UNIX Resource File

A resource file capability is provided to support the C$GetSyn (on page 545) and C$SetSyn (on page 569) subprograms and to provide stored configuration information for the compiler, runtime system, and recovery utility. A resource file, similar in format to a Windows initialization (**.ini**) file, allows for permanent storage of synonym names and values on UNIX in the same way that the registry file does on Windows. You can use the resource files to customize your RM/COBOL application.

The resource files may be located in the user's home directory (local) for information that does not need to be shared or in **/etc/default** (global) for information to be shared among a group of users. For the compiler, the local resource file is named **.rmcobolrc**; the global resource file is named **rmcobolrc**. For the runtime system, the local resource file is named **.runcobolrc**; the global resource file is named **runcobolrc**. For the Indexed File Recovery (**recover1**) utility , the local resource file is named **.recover1rc**; the global resource file is named **recover1rc**.

**Note** The global resource files for the compiler, runtime system, and recovery utility (located in **/etc/default**) do not begin with a period. The local resource files for the compiler, runtime system, and recovery utility (located in the user's home directory) do include a leading period in the name so that it is not visible to the user.

The resource files in the user's home directory normally are maintained by the individual user, while the resource files in **/etc/default** usually are maintained by the system administrator. Although resource files may be maintained with the editor of your choice, no editing should ever be done when the resource file is in use. There is a simple locking mechanism to ensure that two users sharing the same resource file do not conflict with one another, but this mechanism will not prevent an editor from changing the file.

## Resource File Format

All resource files have the same general format. Each file may consist of a [Defaults] section to specify default configuration information for all programs, a [Default Synonyms] section to specify default synonyms to be used by all programs, one or more [*Program*] sections to specify configuration information when a specific

program is executed or compiled, and one or more [*Program* Synonyms] sections to specify synonyms to be used when a specific program is executed or compiled. Lines in a resource file should begin in column 1 (that is, without leading spaces) and be no more than 131 characters long. Section names, including the *Program* portion of section names, are not case-sensitive. A section name matching a prior section name, except for case, will be ignored. Comments may be included in a resource file. Comment text begins with a semicolon (";") in column 1. Lines that have ";" in column 1, as well as blank lines, are ignored in their entirety.

The configuration information specified in a [*Program*] section overrides the configuration information specified in the [Defaults] section when program *Program* is being executed or compiled. Synonyms specified in a [*Program* Synonyms] section are added to the synonyms specified in the [Default Synonyms] section with synonyms from the [*Program* Synonyms] section overriding any duplicate definitions.

**Note** For the recovery utility, *Program* is actually the indexed file name, not including any directory path, but including the extension, if any. For example, the value of *Program* for the indexed file **/usr/guest/mydata.inx** would be **mydata.inx**. In contrast, the value of *Program* for the source file **/usr/guest/myprog.cbl** or the object file **/usr/guest/myprog.cob** would be **myprog**.

## Command-Line Options

Command-line options for the compiler, runtime system, or recovery utility may be specified either in the [Defaults] or the [*Program*] sections. In each case, the command-line options are specified as:

```
Options=command line options
```

where, the *command line options* parameter specifies a series of command-line options to be passed to the compiler, runtime system, or recovery utility. The command-line options from the resource files will be processed cumulatively in the following order: global [Defaults], local [Defaults], global [*Program*], and local [*Program*]. Any options from the resource files are processed before options on the actual command line are processed so that the command-line options can override any options specified from the resource files. If duplicate options appear in the same section of any resource file, the first entry is used.

**Note** Some options for the runtime system may not be overridden by the actual command-line options because the options are cumulative; that is, multiple options of this type may be specified on the command line. The L Option (for library loads) is an example of such a parameter. For additional information, see the Runtime Command (on page 177) description and the description of the L Option on page 183.

The environment variable, RM_IGNORE_GLOBAL_RESOURCES, may be defined if you wish the compiler, runtime system, or recovery utility not to access the command-line options defined in **/etc/default**. This may be useful if you are trying to do development at the same time others are running an application in live "production mode."

## Specifying Synonyms

Synonyms for the compiler, runtime system, or recovery utility may be specified either in the [Default Synonyms] or [*Program* Synonyms] sections. These synonyms may be used to establish a connection between the open name of the file and the actual file access name. Synonyms may also be used to establish the RUNPATH and RMPATH directory search sequences. Users should not attempt to specify synonym names differing only in case. For more information, see Directory Search Sequences on UNIX (on page 24).

In each case the synonym name and value are specified as:

```
SynonymName=SynonymValue
```

When the compiler, runtime system, or recovery utility are being initialized, synonyms are added to the environment in the order specified below. Synonyms names are case-sensitive. However, a synonym whose name is the same as a prior synonym, except for case, will be initialized to the value of the prior synonym.

```
[Default Synonyms] section of the global resource file
[Default Synonyms] section of the local resource file
[Program Synonyms] section of the global resource file
[Program Synonyms] section of the local resource file
```

The environment variable, RM_IGNORE_GLOBAL_RESOURCES, may be defined if you wish the compiler, runtime system, or recovery utility not to access the global synonyms defined in **/etc/default**. This approach may be useful if you are trying to do development at the same time others are running an application in live "production mode."

The C$GetSyn and C$SetSyn subprograms may be used to retrieve and store synonym values in the resource file. Specifically, C$GetSyn retrieves synonym values from either the [*Program* Synonyms] or the [Default Synonyms] section of the local resource file (in the user's home directory) or, if the synonym was not found in the local resource file, from either the [*Program* Synonyms] or the [Default Synonyms] section of the global resource file (in **/etc/default**). C$GetSyn ignores case when searching for the synonyms. The third parameter on the C$GetSyn CALL specifies the program-name for the synonym being retrieved. Specifying SPACES indicates that the user wants the [Default Synonyms] section rather than synonyms for a particular program-name. The environment variable, RM_IGNORE_GLOBAL_RESOURCES, may be defined if you wish to always ignore the global resource file for the runtime system. In this case, C$GetSyn will only have access to the local resource file.

C$SetSyn stores synonym information in the local resource file. C$SetSyn ignores the case of the synonym name when searching for an existing synonym value to replace. It is not possible for C$SetSyn to modify the global resource file for the runtime system. C$SetSyn stores the synonym information in either the [*Program* Synonyms] or the [Default Synonyms] section depending upon the value of the third parameter on the CALL. If necessary, C$SetSyn will create the local resource file in the user's home directory.

## Example of .rmcobolrc File

The following is an example of a UNIX local resource file for the RM/COBOL compiler:

```
[Defaults]
Options=<Compile Command options>

[Default Synonyms]
PRINTER=PrinterFile.prt
RMPATH=~/default/source

[AR]
Options=-l -a -x -o=~/arobj

[AR Synonyms]
RMPATH=~/arsource
PRINTER=~/arlist/ar.prt
```

## Example of .runcobolrc File

The following is an example of a UNIX resource file (local) for the RM/COBOL runtime system:

```
[Defaults]
Options=<Runtime Command options>

[Default Synonyms]
Printer1=PrinterFile
AR-Directory=/usr/company/ar-data

[AR]
Options=<Runtime Command options>

[AR Synonyms]
RUNPATH=<pathname>
AR-FILE1=comp1/ar.dat
```

### *Example of .recover1rc File*

The following is an example of a UNIX local resource file for the RM/COBOL Indexed File Recovery (**recover1**) utility:

```
[Defaults]
Options=-l

[Default Synonyms]
PRINTER=recovery.log

[armaster.inx]
Options=-L armrec.log -K armtempl.inx -M 5

[armaster.inx Synonyms]
DROPFILE=~/ar/armdrop.fil

[artrans.inx]
Options=-L -K arttempl.inx -M 3

[artrans.inx Synonyms]
DROPFILE=~/ar/artdrop.fil
PRINTER=~/ar/artrec.log
```

# Other System Considerations for UNIX

This section describes special system considerations for using RM/COBOL under the UNIX operating system.

## Memory Available for a COBOL Run Unit on UNIX

The memory available for a run unit in the operating system environment is implementation specific. If the total memory required by a run unit exceeds the amount of available memory, runtime system errors will occur. These errors indicate the inability to obtain enough memory to perform a desired operation. The RM/COBOL runtime system does not provide a virtual memory scheme, although your system may.

Segmentation and subprograms should be used to manage the dynamic memory requirements of very large run units.

Most modern UNIX systems (for example, BSD, System V, Sun OS) are supplied with built-in virtual memory systems. These systems make it appear as though there is always sufficient memory for the runtime system, regardless of how much physical RAM is installed in the machine.

## Number of Files

The operating system determines the number of files a run unit is allowed to open. The maximum number of files that may be opened is three fewer than the maximum number of open files per process. Most UNIX systems allow this maximum to be changed by reconfiguring the kernel.

## Number of Region Locks

The runtime system uses the operating system region lock facility to provide file level locking and to control file sharing, as well as to support record locking. To implement file locking, the runtime system applies one region lock to each open file in addition to the locks applied for record locks. During an I/O statement, one or two additional region locks may be applied to a single file. If the program employs multiple record locking, these region locks remain until the program unlocks the records.

## Network File Access

It is possible to receive a 98,27 error when accessing an indexed file through the network file system (NFS) when logged in as super-user (or root). If the file permissions do not include write permission for “other”, an open operation may inadvertently succeed for modes other than input mode. This is misleading because writes to the file will appear to succeed, even though the data is not updated. This problem is undetectable and will appear as a 98,27 error on the next access of the file after writing or deleting a record.

## Terminal Input and Output

This section describes how terminal input and output is handled by the RM/COBOL runtime system.

### *Terminal Interfaces*

The runtime system uses one of two terminal interface mechanisms, termcap or terminfo, to control cursor positioning, video display attributes, and function key mapping.

The termcap version of the runtime system uses the older termcap database, which has a description of the user’s terminal in it. See Termcap Database (on page 35) for more information.

The terminfo version of the runtime system uses the terminal description in the terminfo database for both input and output control of the terminal. See Terminfo Database (on page 35) for more information.

Both the terminfo and termcap Terminal Interface support modules are present on the distribution media. During the installation process, you will be asked which Terminal Interface support module to install. To switch to the other Terminal Interface support module, you will need to run the installation command again and respond appropriately to the prompts described in Performing the Installation (on page 22), paying particular attention to the discussion of optional features.

## Cursor Types

The termcap and terminfo versions of the runtime system support two types of cursors, each of which indicates a different edit mode during ACCEPT operations.

1. The attribute cursor_normal (or cursor-on) indicates that standard overtype mode is active.
2. The attribute cursor_visible (or cursor-blink) indicates that insert mode is active.

## Terminal Attributes

Some terminals under UNIX require that special characters appear on the screen just before the start of an attribute and right after the end of it. Characters in between these special characters take on the specified attribute. To accommodate these terminals, the **oV** capability for termcap specifies the number of screen positions to be used by the **nM**, **nB**, **nR**, **nS**, **aL**, **aB**, **aR**, **aS** and **rS** capabilities. The **xmc** capability is used for the terminfo runtime system. RM/COBOL places the attribute characters at the position specified by the ACCEPT or DISPLAY operation, and moves the actual start of the field by the number of positions specified by **oV** or **xmc**. If you want, you can use the keyword MOVE-ATTR with the TERM-UNIT configuration record to specify moving the attributes back the number of positions specified by **oV** or **xmc**. However, if MOVE-ATTR causes the attribute character to move back to the next line, and such a move is prohibited by the **lA** (do not cross lines) capability described in the next paragraph, the attribute will appear on the same line that is being displayed or accepted.

The **lA** is a Boolean termcap capability and is used with terminals that require screen positions to implement attributes, as described in the preceding paragraph. The standard RM/COBOL model is to keep an attribute in effect—without regard to the number of screen lines to which it applies—until it encounters the special character that signals the end of the attribute. Some terminals, however, recognize the end of a line as the end of the attribute, without regard to the presence or absence of the ending special character. In this case, the presence of **lA** will tell RM/COBOL that a new attribute character must be placed at the start of every new line in a multiline ACCEPT or DISPLAY operation.

The **sA** is a Boolean termcap capability that is also used with terminals that require screen positions to implement attributes. The RM/COBOL model is to assume that attributes will not wrap from the bottom to the top of the screen. If your terminal behaves differently, and if you have specified the MOVE-ATTR configuration keyword, use **sA**. This allows fields placed at the home position (line 1, position 1) to have their attributes placed at the last line of the screen.

## Termcap Database

The runtime system locates the termcap database by first looking for the environment variable TERMCAP. If the TERMCAP environment variable is found and contains a valid pathname, that value is used as the pathname to the database. If the environment variable is found but it contains a valid termcap entry, that entry will be used as the terminal description. Otherwise, the filename **/etc/termcap** will be used as the name of the database.

The TERMCAP environment variable can be set as follows:

```
TERMCAP=pathname ; export TERMCAP
```

*pathname* is a pathname of the termcap file.

For example:

```
TERMCAP=/usr/sales/mytermcapfile; export TERMCAP
```

## Terminfo Database

The runtime system locates the terminfo database by first looking for the environment variable TERMINFO. If the TERMINFO environment variable is found, that value is used as the pathname to the database subdirectories. Otherwise, the path **/usr/lib/terminfo** will be used.

The TERMINFO environment variable can be set as follows:

```
TERMINFO=pathname ; export TERMINFO
```

*pathname* is a pathname of the terminfo file.

For example:

```
TERMINFO=/usr/sales/myterminfo; export TERMINFO
```

## Termcap and Terminfo

The name of the database entry that describes the behavior of your terminal is obtained from the environment variable TERM. This variable should be set to the appropriate terminal name before invoking the runtime system.

The TERM environment variable can be set as follows:

```
TERM=term-name ; export TERM
```

*term-name* is the name of your terminal as it appears in the termcap or terminfo database. The termcap or terminfo capabilities used by the runtime system (if present) are listed in the tables that follow.

Table 2 describes the information from the terminfo and termcap databases that the runtime system uses.

**Table 2: Terminfo and Termcap Names for the Runtime System**

| Terminfo Name | Termcap Name | Description |
|---|---|---|
| **Booleans** | | |
| am | am | Terminal has automatic margins. |
| bce | be | Screen erased with background color. |
| xenl | xn | Newline ignored after 80 columns. Also used to signify that the terminal's cursor will not automatically advance to the next line after column 80 is reached, but will instead wait for the next character. |
| **Numbers** | | |
| cols | co | Number of columns in a line. |
| lines | li | Number of lines on screen or page. |
| pb | pb | Lowest baud where padding is needed. |
| xmc | sg | Number of blank characters left by smso or rmso. |
| **Output Strings** | | |
| acsc | ac | Graphic charset pairs. |
| bel | bl | Audible signal (bell). |
| blink | | Turn on blinking. |
| civis | vi | Make cursor invisible. |
| clear | cl | Clear screen and home cursor. |
| cnorm | ve | Make cursor appear normal (undo vs/vi). |
| cr | cr | Carriage return. |
| cub1 | le | Move cursor left one space. |
| cud1 | do | Down one line. |
| cuf1 | nd | Nondestructive space (cursor right). |
| cup | cm | Cursor motion. |
| cuu1 | up | Upline (cursor up). |
| cvvis | vs | Make cursor very visible—insert mode. |

**Table 2: Terminfo and Termcap Names for the Runtime System (*Cont.*)**

| Terminfo Name | Termcap Name | Description |
|---|---|---|
| **Output Strings *(Cont.)*** | | |
| dim | | Turn on half-bright mode. |
| ed | cd | Clear to end of display. |
| el | ce | Clear to end of line. |
| enacs | eA | Enable alternate character set. |
| home | ho | Home cursor. |
| | ko | Termcap entries for other non-function keys. |
| ind | sf | Scroll text up. |
| pad | pc | Pad character (rather than null). |
| op | op | Set all colors to the original color pairs. |
| rev | | Turn on reverse video mode. |
| rmacs | ae | End alternate character set. |
| rmcup | te | String to end programs that use cup. |
| rmso | se | End of standout mode (if no nM or sgr0). |
| | is | Terminal initialization string. |
| rs1 | r1 | Terminal reset/initialization string 1. |
| rs2 | r2 | Terminal reset/initialization string 2. |
| rs3 | r3 | Terminal reset/initialization string 3. |
| setb | Sb | Set current background color. |
| setf | Sf | Set current foreground color. |
| sgr | | Define video attributes, 1 through 9. |
| sgr0 | me | Turn off all attributes. |
| smacs | as | Start alternate character set. |
| smcup | ti | String to begin programs that use cup. |
| | tc | Entry of similar terminal. |
| xenl | xn | Newline ignored after 80 columns. |

Table 3 describes the input sequences that may be handled by the terminfo package. For convenience, the corresponding termcap name is also given. These terminfo names are also the only names that will be recognized when using the TERM-INPUT configuration feature (on page 332) of the runtime system. Termcap names other than the ones listed in the table can be used.

**Table 3: Input Sequences for Terminfo and Termcap**

| Terminfo Name | Termcap Name | Description |
|---|---|---|
| **Input Strings** | | |
| ka1 | K1 | Upper-left of keypad. |
| Ka3 | K3 | Upper-right of keypad. |
| Kb2 | K2 | Center of keypad. |
| Kbeg | @1 | Sent by beginning key. |
| Kbs | kb | Sent by backspace key. |
| Kc1 | K4 | Lower-left of keypad. |
| Kc3 | K5 | Lower-right of keypad. |
| Kcan | @2 | Sent by cancel key. |
| Kclo | @3 | Sent by close key. |
| Kclr | kC | Sent by clear screen or erase key. |
| Kcmd | @4 | Sent by command key. |
| Kcpy | @5 | Sent by copy key. |
| Kcrt | @6 | Sent by create key. |
| Kctab | kt | Sent by clear-tab key. |
| Kcub1 | kl | Sent by terminal left arrow key. |
| Kcud1 | kd | Sent by terminal down arrow key. |
| Kcuf1 | kr | Sent by terminal right arrow key. |
| Kcuu1 | ku | Sent by terminal up arrow key. |
| Kdch1 | kD | Sent by delete character key. |
| Kdl1 | kL | Sent by delete line key. |
| Ked | kS | Sent by clear-to-end-of-screen key. |
| Kel | kE | Sent by clear-to-end-of-line key. |
| Kend | @7 | Sent by end key. |
| Kent | @8 | Sent by enter/send key. |
| Kext | @9 | Sent by exit key. |
| Kf0 | k0 | Sent by function key f0. |
| Kf1 | k1 | Sent by function key f1. |
| Kf2 | k2 | Sent by function key f2 |
| kf3 | k3 | Sent by function key f3. |
| Kf4 | k4 | Sent by function key f4. |
| kf5 | k5 | Sent by function key f5. |
| kf6 | k6 | Sent by function key f6. |
| kf7 | k7 | Sent by function key f7. |
| kf8 | k8 | Sent by function key f8. |
| kf9 | k9 | Sent by function key f9. |
| kf10 | k; | Sent by function key f10. |
| kf11 | F1 | Sent by function key f11. |
| kf12 | F2 | Sent by function key f12. |

**Table 3: Input Sequences for Terminfo and Termcap (*Cont.*)**

| Terminfo Name | Termcap Name | Description |
|---|---|---|
| **Input Strings *(Cont.)*** | | |
| kf13 | F3 | Sent by function key f13. |
| kf14 | F4 | Sent by function key f14. |
| kf15 | F5 | Sent by function key f15. |
| kf16 | F6 | Sent by function key f16. |
| kf17 | F7 | Sent by function key f17. |
| kf18 | F8 | Sent by function key f18. |
| kf19 | F9 | Sent by function key f19. |
| kf20 | FA | Sent by function key f20. |
| kf21 | FB | Sent by function key f21. |
| kf22 | FC | Sent by function key f22. |
| kf23 | FD | Sent by function key f23. |
| kf24 | FE | Sent by function key f24. |
| kf25 | FF | Sent by function key f25. |
| kf26 | FG | Sent by function key f26. |
| kf27 | FH | Sent by function key f27. |
| kf28 | FI | Sent by function key f28. |
| kf29 | FJ | Sent by function key f29. |
| kf30 | FK | Sent by function key f30. |
| kf31 | FL | Sent by function key f31. |
| kf32 | FM | Sent by function key f32. |
| kf33 | FN | Sent by function key f33. |
| kf34 | FO | Sent by function key f34. |
| kf35 | FP | Sent by function key f35. |
| kf36 | FQ | Sent by function key f36. |
| kf37 | FR | Sent by function key f37. |
| Kf38 | FS | Sent by function key f38. |
| Kf39 | FT | Sent by function key f39. |
| Kf40 | FU | Sent by function key f40. |
| Kf41 | FV | Sent by function key f41. |
| Kf42 | FW | Sent by function key f42. |
| Kf43 | FX | Sent by function key f43. |
| Kf44 | FY | Sent by function key f44. |
| Kf45 | FZ | Sent by function key f45. |
| Kf46 | Fa | Sent by function key f46. |
| Kf47 | Fb | Sent by function key f47. |
| Kf48 | Fc | Sent by function key f48. |
| Kf49 | Fd | Sent by function key f49. |
| Kf50 | Fe | Sent by function key f50. |
| Kf51 | Ff | Sent by function key f51. |
| Kf52 | Fg | Sent by function key f52. |
| Kf53 | Fh | Sent by function key f53. |
| Kf54 | Fi | Sent by function key f54. |
| Kf55 | Fj | Sent by function key f55. |
| Kf56 | Fk | Sent by function key f56. |

**Table 3: Input Sequences for Terminfo and Termcap (*Cont.*)**

| Terminfo Name | Termcap Name | Description |
|---|---|---|
| **Input Strings *(Cont.)*** | | |
| Kf57 | Fl | Sent by function key f57. |
| Kf58 | Fm | Sent by function key f58. |
| Kf59 | Fn | Sent by function key f59. |
| Kf60 | Fo | Sent by function key f60. |
| Kf61 | Fp | Sent by function key f61. |
| Kf62 | Fq | Sent by function key f62. |
| Kf63 | Fr | Sent by function key f63. |
| Kfnd | @0 | Sent by find key. |
| Khlp | %1 | Sent by help key. |
| Khome | kh | Sent by home key. |
| Khts | kT | Sent by set-tab key. |
| Kich1 | kI | Sent by insert character/enter insert mode key. |
| Kil1 | kA | Sent by insert line. |
| kind | kF | Sent by scroll-forward/down key. |
| kll | kH | Sent by home-down key. |
| kmsg | %3 | Sent by message key. |
| knp | kN | Sent by next-page key. |
| knxt | %5 | Sent by next-object key. |
| kopn | %6 | Sent by open key. |
| kopt | %7 | Sent by options key. |
| kpp | kP | Sent by previous-page key. |
| kprt | %9 | Sent by print key. |
| kprv | %8 | Sent by previous-object key. |
| krdo | %0 | Sent by redo key. |
| kref | &1 | Sent by reference key. |
| kres | &5 | Sent by resume key. |
| krfr | &2 | Sent by refresh key. |
| kri | kR | Sent by scroll-backward/up key. |
| krmir | kM | Sent by exit insert mode key. |
| krpl | &3 | Sent by replace key. |
| krst | &4 | Sent by restart key. |
| ksav | &6 | Sent by save key. |
| kslt | *6 | Sent by select key. |
| kspd | &7 | Sent by suspend key. |
| ktbc | ka | Sent by clear-all-tabs key. |
| kund | &8 | Sent by undo key. |
| kBEG | &9 | Sent by shifted beginning key. |
| kCAN | &0 | Sent by shifted cancel key. |
| kCMD | *1 | Sent by shifted command key. |
| kCPY | *2 | Sent by shifted copy key. |
| kCRT | *3 | Sent by shifted create key. |
| kDC | *4 | Sent by shifted delete-char key. |
| kDL | *5 | Sent by shifted delete-line key. |
| kEND | *7 | Sent by shifted end key. |

**Table 3: Input Sequences for Terminfo and Termcap (*Cont.*)**

| Terminfo Name | Termcap Name | Description |
|---|---|---|
| **Input Strings *(Cont.)*** | | |
| kEOL | *8 | Sent by shifted clear-line key. |
| kEXT | *9 | Sent by shifted exit key. |
| kFND | *0 | Sent by shifted find key. |
| kHLP | #1 | Sent by shifted help key. |
| kHOM | #2 | Sent by shifted home key. |
| kIC | #3 | Sent by shifted input key. |
| Kmov | %4 | Sent by move key. |
| Kmrk | %2 | Sent by mark key. |
| KLFT | #4 | Sent by shifted left arrow key. |
| KSAV | !1 | Sent by shifted save key. |
| KSPD | !2 | Sent by shifted suspend key. |
| KUND | !3 | Sent by shifted undo key. |
| KMSG | %a | Sent by shifted message key. |
| KMOV | %b | Sent by shifted move key. |
| KNXT | %c | Sent by shifted next key. |
| KOPT | %d | Sent by shifted options key. |
| KPRV | %e | Sent by shifted prev key. |
| KPRT | %f | Sent by shifted print key. |
| KRDO | %g | Sent by shifted redo key. |
| KRPL | %h | Sent by shifted replace key. |
| KRIT | %I | Sent by shifted right arrow key. |
| KRES | %j | Sent by shifted resume key. |
| Lf0 | l0 | Labels on function key f0 if not f0. |
| Lf1 | l1 | Labels on function key f1 if not f1. |
| Lf2 | l2 | Labels on function key f2 if not f2. |
| Lf3 | l3 | Labels on function key f3 if not f3. |
| Lf4 | l4 | Labels on function key f4 if not f4. |
| Lf5 | l5 | Labels on function key f5 if not f5. |
| Lf6 | l6 | Labels on function key f6 if not f6. |
| Lf7 | l7 | Labels on function key f7 if not f7. |
| Lf8 | l8 | Labels on function key f8 if not f8. |
| Lf9 | l9 | Labels on function key f9 if not f9. |
| Lf10 | la | Labels on function key f10 if not f10. |
| Nel | nw | Sent by newline key. |

Table 4 describes the additional Boolean capabilities used by RM/COBOL when accessing the termcap database.

**Table 4: Additional Boolean Capabilities**

| Termcap Name | Description |
|---|---|
| lA | Attributes will not wrap lines. |
| sA | Attributes will wrap screen. |

Table 5 describes the additional output string capabilities used by RM/COBOL when accessing the termcap database.

**Table 5: Additional Output String Capabilities**

| Termcap Name | Description |
|---|---|
| aB | Low intensity blink. |
| Ab | Low intensity underline and blink. |
| aL | Low intensity. |
| Al | Low intensity underline. |
| aR | Low intensity reverse. |
| aS | Low intensity blink and reverse. |
| nB | High intensity blink. |
| Nb | High intensity underline and blink. |
| nM | High intensity. |
| Nm | High intensity underline. |
| nR | High intensity reverse. |
| nS | High intensity blink and reverse. |
| vr | End of field. |

Table 6 describes the additional numeric capabilities used by RM/COBOL when accessing the termcap database.

**Table 6: Additional Numeric Capabilities**

| Termcap Name | Description |
|---|---|
| oV | Number of blank characters left by additional RM/COBOL attribute capabilities. |

The Boolean capabilities **sA** and **lA** cannot be added to the terminfo database since it is a closed system; these capabilities are not used by the terminfo runtime system. Under runtime systems that use terminfo for output, the **xmc** numeric capability determines the width of attribute characters and the starting position of fields. Specifying **xmc#0** indicates a physical attribute terminal for which the attributes do not occupy a screen position but still must be written at the physical start and end of each field.

Runtime systems that use the terminfo database directly for output sequences will use the set_attributes or **sgr** string for all field attributes, if it is available. The terminfo set_attributes string has nine parameters or attributes that can be set. RM/COBOL makes use of six of these parameters. The second parameter is set if the underline

attribute is requested. The third parameter is set if the reverse attribute is requested. The fourth parameter is set if the blinking attribute is requested. The fifth parameter is set if the low-intensity attribute is used. The sixth parameter is set if the high-intensity attribute is used. The ninth parameter may be used when line draw characters are requested for pop-up window borders. The only exception to requesting line draw characters in this manner is in terminals where **xmc** and **sgr** are specified (for example, physical attribute terminals). On these terminals, the alternate character set attribute can either be a field attribute or a single character attribute. Because the terminfo database does not indicate how to determine this behavior for a terminal, RM/COBOL will infer that the terminal has the alternate character set as a single character attribute, if the **smac** definition is in the terminfo database for the terminal. In this case, the **smacs** and **rmacs** sequence will be used for the writing of graphics or alternate character set data and the ninth parameter will always be specified as off.

Each of the **sgr** parameters is set to one if an ACCEPT or DISPLAY requests the corresponding attribute. Otherwise, a zero is set for the parameter. A zero is also set for all other parameters.

Attributes are reset by using the **sgr0** string if it is defined. Otherwise, they are reset using all zeroes as parameters to the set_attributes string.

If the set_attributes string is not available, the standard terminfo strings listed in Table 7 will be used.

**Table 7: Standard Terminfo Strings**

| Terminfo Name | Description |
|---|---|
| blink | High intensity blink. |
| dim | Low intensity. |
| rev | High intensity reverse video. |
| rmacs | End alternate character set. |
| rmso | Reset attributes (also used for high intensity if no sgr0). |
| sgr0 | High intensity. |
| smacs | Start alternate character set. |
| smso | High intensity (if no sgr0 or rmso). |

If color keywords are specified in the CONTROL phrase, the terminfo **setf** or **setb** sequence will be used to set the foreground or background color. These sequences accept a single numeric parameter indicating the desired color. If these sequences are not already defined for your terminal and you wish to define them, the association of colors to color numbers is normally defined in the C include file, **curses.h**.

If line draw characters are requested for either pop-up window borders, or because the GRAPHICS keyword in the CONTROL phrase was specified in an ACCEPT or DISPLAY statement, the terminfo database is examined for the **acsc** sequence. UNIX systems provide the **acsc** string to map generic (vt100) line draw characters to the correct characters for your terminal. These characters are then enabled through the ninth **sgr** parameter (see page 43). To support double-line draw characters, RM/COBOL has extended the **acsc** string to include six more mappings. These mappings extend the generic (vt100) characters by describing the double-line graphic characters with the corresponding uppercase letters, as shown in Table 8.

**Table 8: vt100 Line Graphic Characters**

| Description | Single-Line Character | Double-Line Character |
|---|---|---|
| lower-right corner | j (Ù) | J (¼) |
| upper-right corner | k (¿) | K (») |
| upper-left corner | l (Ú) | L (É) |
| lower-left corner | m (À) | M (È) |
| horizontal line | q (Ä) | Q (Í) |
| vertical line | x (³) | X (º) |

## Redirection of Input and Output

RM/COBOL supports standard piping and standard redirection of input and output.

The use of the redirection and piping operators ( `>` , `>>` , `<` , and `|` ) on the Runtime Command line affects the operations of ACCEPT and DISPLAY statements in several ways. Piping is a means of chaining the standard output (DISPLAY statements) of one run unit to the standard input (ACCEPT statements) of a second run unit; therefore, piping appears identical to redirection at the program level. Note that a Format 1 ACCEPT or DISPLAY statement that includes the FROM/UPON CONSOLE phrase or FROM/UPON *mnemonic-name* phrase where *mnemonic-name* is defined as CONSOLE IS *mnemonic-name*, is not redirected or piped unless it is configured to come from standard input or go to standard output. If this is not the case, you must use either 2> or 2>> for redirection. Note also that if an ACCEPT or DISPLAY statement contains a UNIT phrase, it will not be redirected.

### *Standard Input*

The standard input device is defined by default to be the keyboard of the terminal that started the run unit. Standard input may be redirected to a file or other device by the operating system conventions for standard input redirection and piping on the command line that starts the run unit.

For example:

```
runcobol getdata <inputfile
```

redirects standard input to the file **inputfile**, and

```
runcobol putdata | runcobol getdata
```

pipes the standard output from program **putdata** to the standard input of program **getdata**.

ACCEPT statements that do not specify the FROM CONSOLE phrase read from the standard input device.

When standard input is redirected, the ACCEPT statement (Formats 1 and 3) operation is modified. Only the SIZE, CURSOR, ECHO, CONVERT and ON EXCEPTION phrases of Format 3 are used; all other phrases are ignored. Note that Format 1 ACCEPT statements with numeric operands are treated as Format 3 ACCEPT statements unless the program containing the ACCEPT statements was compiled with the M Compile Command Option (see page 147).

At the beginning of each ACCEPT statement, the next record is read from standard input into the ACCEPT buffer. The following operations take place for each of the receiving data items in the ACCEPT statement:

1. If there are no characters in the ACCEPT buffer, the next record is read from standard input into the ACCEPT buffer.
2. If the number of characters in the ACCEPT buffer does not exceed the size of the current receiving item, those characters are transferred to the receiving item in the appropriate format (that is, left justified, space fill for all Format 1 and for alphanumeric Format 3, and with appropriate conversion for numeric Format 3).
3. If the number of characters in the ACCEPT buffer exceeds the size of the current receiving item, only the leftmost "size" characters are transferred, as described in the previous operation. The characters that remain in the ACCEPT buffer are used for the next receiving item or are discarded if the current receiving item is the last receiving item in the ACCEPT statement.

**Note** Where numeric sending and receiving data items are used with piping, the use of the CONVERT phrase with DISPLAY and ACCEPT statements is strongly recommended.

The M Runtime Command Option (see page 181) modifies the operation of Format 1 ACCEPT statements to conform to Level 2 ANSI semantics. The actions described above are modified as follows:

1. If the number of characters in the ACCEPT buffer does not equal or exceed the size of the current receiving item, one or more records are read from standard input and are concatenated until there are enough characters.
2. The leftmost "size" characters are transferred as described in steps 2 and 3 in the instructions above. The characters that remain in the ACCEPT buffer are discarded.

Note that the use of the M Runtime Command Option requires close matching of ACCEPT and DISPLAY statements when used with piping.

Also note that the M Runtime Command Option affects the operation of Format 1 ACCEPT statements which are not redirected; the console operator is required to enter enough characters to fill the receiving item. If the Enter key is pressed before enough characters have been entered, the request will be reissued until the concatenation of the characters entered is sufficient to fill the receiving item.

The M Runtime Command Option does not affect the operation of Format 3 ACCEPT statements.

An end-of-file condition is reported to Format 3 ACCEPT statements as an exception variable of 64 (Send). If an end-of-file condition occurs and there is no ON EXCEPTION phrase, a runtime system error is reported and execution ends.

## *Standard Output*

The standard output device is defined by default to be the monitor of the terminal that started the run unit. Standard output may be redirected to a file or other device by the operating system conventions for standard output redirection and piping on the command line that starts the run unit.

For example:

```
runcobol putdata >outputfile
```

redirects standard output to the file **outputfile**, and

```
runcobol putdata | runcobol getdata
```

pipes the standard output from program **putdata** to the standard input of program **getdata**.

DISPLAY statements—that do not specify the UPON or UPON CONSOLE phrase—write to the standard output device.

When standard output is redirected, all phrases, except SIZE and CONVERT, of the Format 2 DISPLAY statement are ignored. All sending operands are concatenated (within the limits of the ACCEPT/DISPLAY buffer as described in the following paragraphs) and are transferred to standard output as one or more records.

A Format 1 DISPLAY statement generates one record and may generate more than one record, depending on the presence or absence of the M Runtime Command Option (see page 181). If the M Option is not present in the Runtime Command, all sending operands are concatenated, the resulting operand is truncated to the DISPLAY buffer size, and a single record is written. If the M Option is present, all sending operands are concatenated and the resulting operand is split into zero or more records equal in length to the DISPLAY buffer size, along with a final record no longer than the DISPLAY buffer size.

If a Format 2 DISPLAY statement is redirected, one or more records are generated, depending on the size of the discrete sending items. If the size of the sending operand does not exceed the space remaining in the DISPLAY buffer, the sending operand is appended to the current buffer and the DISPLAY buffer is written if the sending operand is the last operand. If the size of the sending operand exceeds the space remaining in the DISPLAY buffer, the current DISPLAY buffer is written and the sending operand is truncated to the size of the DISPLAY buffer. The new DISPLAY buffer contents are written if the sending operand is the last operand.

## *Standard Error*

The standard error device is defined by default to be the monitor and keyboard. Interactive debug input and output, STOP *literal* statements, and runtime system messages are directed to the standard error device. These operations can be redirected by a configuration option; see the discussion of the ERROR-MESSAGE-DESTINATION keyword that begins on page 314.

These operations also can be redirected using the operating system standard-error redirection convention on the command line that starts the run unit.

For example:

```
runcobol putdata 2>error.out
```

To direct standard output and standard error to the same destination, specify:

```
runcobol putdata >all.out 2>&1
```

## Using Large Files on UNIX

RM/COBOL supports files larger than 2 gigabytes (GB). Large file support is available only on those UNIX systems that provide native support for files larger than 2 GB. The following UNIX systems provide such support: IBM AIX 4.3 and 5; Digital/Compaq Tru64 UNIX V4, V5; HP-UX 11.0; some versions of Linux, Sun Solaris 2.6; and UnixWare 7.

Many UNIX systems are configured to restrict the size of files to which normal user accounts can write. Often this limit is 2 GB or less. On systems that support large files, the system administrator may be able to configure the system or the user accounts to allow a large ULIMIT, or the user may need to run the **ulimit** command to increase the ULIMIT before creating or accessing large files.

Refer to Very Large File Support (on page 221) for more information.

## Environment Variables for UNIX

An environment variable is an operating system feature that allows a value to be equated with a name. Table 9 lists those environment variables that are used by RM/COBOL on UNIX.

**Note** In addition to the environment variables listed in the following table, RM/COBOL uses environment variables to map generic file access names as explained in File Access Names on UNIX (on page 25).

**Table 9: Environment Variables for UNIX**

| Environment Variable | Usage |
|---|---|
| HOME | Locating files. See File Locations within Operating System Pathnames on UNIX (on page 24). |
| LD_LIBRARY_PATH | Locating optional support modules (on page 431). |
| PATH | Locating files. See Directory Search Sequences on UNIX (on page 24). |
| PRINTER | Printer support (on page 227). |
| RMPATH | Locating files. See Directory Search Sequences on UNIX (on page 24). |
| RMTERM132 | ACCEPT/DISPLAY CONTROL SCREEN-COLUMNS (see page 201). |
| RMTERM80 | ACCEPT/DISPLAY CONTROL SCREEN-COLUMNS (see page 201). |
| RM_DEVELOPMENT_MODE | C$SetDevelopmentMode subprogram (on page 567). |
| RM_DYNAMIC_LIBRARY_TRACE | Tracing support module loads. See Locating optional support modules (on page 431). |
| RM_ESCAPE_TO_COMMAND | TERM-INPUT ACTION=ESCAPE-TO-COMMAND (see page 332). |
| RM_IGNORE_GLOBAL_RESOURCES | UNIX resource file. See Command-Line Options (on page 29). |
| RM_LIBRARY_SUBDIR | Locating optional support modules. See Using a Different Subdirectory (on page 433). |
| RM_LOAD_WOW_CLIENT | Loading the WOW Extensions support module, libtclnt.so. |
| RM_VERBOSE_BANNER | Compile command messages (on page 166) and runcobol banner message (on page 396). |
| RM_Y2K | COMPILER-OPTIONS ALLOW-DATE-TIME-OVERRIDE (see page 287). |
| RUNPATH | Locating files. See Directory Search Sequences on UNIX (on page 24). |
| SHELL | SYSTEM subprogram (on page 578). |
| TAPE | Tape support (on page 227). |
| TERM | Terminal I/O. See Termcap and Terminfo (on page 35). |
| TERMCAP | Terminal I/O. See Termcap and Terminfo (on page 35). |
| TERMINFO | Terminal I/O. See Termcap and Terminfo (on page 35). |
| TMPDIR | Temporary files (on page 239). |
| TZ | Standard C TimeZone variable. |

# Chapter 3: Installation and System Considerations for Microsoft Windows

This chapter lists the hardware and software required to install the RM/COBOL product, describes how to install RM/COBOL, and provides information specific to using RM/COBOL with Microsoft 32-bit Windows operating systems.

Your computer configuration is the assembled set of hardware and software that makes up your system. Before you can develop or run RM/COBOL programs, your configuration must meet or exceed the requirements set forth in this chapter.

## System Requirements for Windows

RM/COBOL runs on the IBM PC and full compatibles. Appropriately licensed versions run in conjunction with Client for Microsoft Networks or Novell NetWare software to provide support for multi-user file access.

### Required Hardware

- An IBM PC or compatible machine with a Pentium-class processor or higher is required.
- A mouse or other pointing device.
- A minimum of 16 megabytes of random access memory (RAM) is required for compiling and running. (For debugging with CodeWatch, 32 megabytes is recommended.)
- The minimum disk space varies from 4 MB to 15 MB depending on number of components that are being installed.
- An 800 x 600 x 256 color display adapter. (1024 x 768 x 256 or better is recommended.)

  **Note** Although RM/COBOL will run in 640 x 480 x 256, this is not recommended.

- One CD-ROM drive and one double-sided, high-density 3.5-inch floppy disk drive for program installation.
- Network adapter (for multi-user configurations).

## Required Software

One of the following operating systems is required:

- Microsoft Windows 98 or Windows 98 Second Edition (SE)
- Microsoft Windows Millennium Edition (Me)
- Microsoft Windows NT version 4.0 (Service Pack 6 or higher recommended)
- Microsoft Windows 2000
- Microsoft Windows XP
- Microsoft Windows Server 2003

**Note** As you read through this guide, note that Liant may use two shorthand notations when referring to these operating systems. The term "Windows 9x class" refers to the Windows 98 or Windows Me operating systems. The term "Windows NT class" refers to the Windows NT 4.0, Windows 2000, Windows XP, or Windows Server 2003 operating systems.

### *Local Area Network (LAN) Software*

To provide multi-user access, one or both of the following network programs is required:

- Novell NetWare version 3.11 or later
- Client for Microsoft Networks for 32-bit Windows

### *Btrieve Software*

To access local Btrieve files, the following software is required:

- Version 6.15 or later of Btrieve for 32-bit Windows

To access remote Btrieve files, both of the following software components are required:

- Version 6.15 or later of Btrieve MicroKernel Database Engine for NetWare or a Windows NT-class operating system
- Version 6.15 or later of Btrieve Requester for 32-bit Windows

**Note** Btrieve components are available from Pervasive Software Inc.

# System Installation for Windows

**Note** It is recommended that all other applications be closed before proceeding with the installation.

To install RM/COBOL on the Windows operating system:

1. Start Windows.
2. Insert the RM/COBOL for 32-bit Windows CD-ROM in the appropriate CD-ROM drive.
3. If the installation program does not start automatically, click Start, and then click Run.
4. In the Open text box in the Run dialog box, type the following:

   ```
   X:setup
   ```

   where, *X* is the letter assigned to the CD-ROM drive where you inserted the CD-ROM media.
5. If the installation starts automatically or if you start it using the **autorun** command, you will be presented with the following list of installation options:
   a. Click the **View the READ ME file** option to review important last-minute information about RM/COBOL for 32-bit Windows.
   b. Click the **View the On-Line Documentation** option to read or install the documentation for this product that is provided on the CD-ROM.
   c. Click the **Install the RM/COBOL 9 for 32-bit Windows** option to begin the actual installation process.
6. Click the OK command button and follow the prompts on your screen to complete the installation process.

During the installation process, you will be prompted for the Liant License floppy disk. Follow the instructions on the screen.

After installation is complete, you can display the RM/COBOL Start Menu Programs folder, which is illustrated in Figure 3. The programs are described in Table 10.

**Note** For further information on installing RM/COBOL on a Windows NT-class operating system and network client machines, see Installation Notes for Windows (on page 53).

**Figure 3: RM/COBOL Start Menu Programs Folder**

**Note** Depending upon the RM/COBOL package that you purchased, not all of the program icons in Figure 3 will appear on your system.

**Table 10: RM/COBOL Program Icons**

| Program Icon Name | Description |
|---|---|
| CodeWatch | Starts CodeWatch, a fully integrated development environment for RM/COBOL for Windows. |
| CodeWatch Help | Starts the CodeWatch help file. |
| Compiler | Starts the RM/COBOL compiler (rmcobol.exe) and prompts for a source filename. |
| doverify | Starts the RM/COBOL verification suite. |
| INI to Registry | Starts the Initialization File to Windows Registry Conversion utility (ini2reg.exe). This program takes a Windows initialization (.ini) file and inserts its entries into the Windows registry database used by RM/COBOL. |
| ixverify | Starts the RM/InfoExpress verification suite. |
| ReadMe | Starts Notepad with the README file. |
| recover1 | Starts the Indexed File Recovery utility (recover1.exe). This program is used to recover damaged indexed files. |
| Registry Configuration | Starts the RM/COBOL Configuration utility (rmconfig.exe). This program sets the runtime system (runcobol.exe), compiler (rmcobol.exe), and Indexed File Recovery utility (recover1.exe) options for RM/COBOL programs and data files. |
| rmdefinx | Starts the Define Indexed File utility program (rmdefinx.cob). |
| rmmapinx | Starts the Map Indexed File utility program (rmmapinx.cob). |

**Table 10: RM/COBOL Program Icons (*Cont.*)**

| Program Icon Name | Description |
|---|---|
| rmmappgm | Starts the Map Program File utility program (rmmappgm.cob). |
| rmpgmcom | Starts the Combine Program File utility program (rmpgmcom.cob). |
| Runtime | Starts the RM/COBOL runtime system (runcobol.exe) and prompts for a program-name. |
| Syntax Summary Help | Starts the RM/COBOL Syntax Summary help file. |
| Toolbar Editor | Starts the toolbar button editor program (rmtbedit.exe). |

# Installation Notes for Windows

The following notes apply to installing RM/COBOL on Windows systems.

## *Installation of RM/COBOL on Windows NT-Class Operating Systems*

The RM/COBOL installation procedure checks the system configuration for compatibility of other products with RM/COBOL. Certain Windows NT features can cause problems with RM/COBOL. To avoid these problems and fix incorrect Windows registry entries, see Network Redirector File Caching (on page 656) and Opportunistic Locking (on page 657).

## *Installation of RM/COBOL on Network Client Machines*

The RM/COBOL installation process has been changed to help users who wish to install RM/COBOL on a network server machine and then install RM/COBOL on multiple client machines using the RM/COBOL installation on the server.

First, install all of the RM/COBOL components that you desire onto the server machine. This may be performed either directly on the server or from a client machine via a mapped network drive. Then, on each remaining client machine, specify the mapped network drive path of the original server installation in the Select Destination Directory dialog box and click the Next button. In the Setup Type dialog box, click Network in the list box and then click Next to continue with the network installation. This causes the installation process to install the RM/COBOL program folder and icons for those components that already exist in the original server installation. Shared system DLLs (such as **CTL3D32.DLL** and **MSVCRT.DLL**) also will be installed on the client machine (if a later version does not already exist there) and appropriate Windows registry entries will be created.

## *Default Native Character Set*

After RM/COBOL installation on Windows, the default native character set for the compiler, runtime, and CodeWatch is the character set defined by the OEM codepage. Starting with version 9, RM/COBOL also has support for a native character set using the ANSI codepage. A complete explanation of native character set selection is provided in Character Set Considerations for Windows (on page 97).

# Registering the RM/COBOL Compiler and Runtime Executables

The RM/COBOL compiler and runtime system use clients and servers that conform to Microsoft Windows Component Object Model (COM) technology standard. The server must be registered with Windows, which is normally done by the Setup program during system installation. This section discusses the information that must be considered when components of the RM/COBOL compiler or runtime system are moved or renamed after installation, if the Windows registry is damaged. For information on registering the runtime, see Runtime Registration (on page 56).

## *Compiler Registration*

The RM/COBOL for Windows compiler consists of two components:

- A client, which may be either of the following:
  - The console-mode client, called **rmcobolc.exe**
  - The GUI-mode client, called **rmcobolg.exe**

  Either client may be called **rmcobol.exe**.
- A server, called **rmcbl90c.dll**

The compiler server DLL, which must be registered with Windows before RM/COBOL programs can be compiled, is automatically registered by the Setup program when the compiler is installed. If the compiler is moved to a directory other than the installation directory without a re-installation, an error message is displayed indicating that there is a registration problem. The error message is displayed either in the console window for the console-mode compiler or in a message box for the GUI-mode compiler. The text of the error message is as follows:

```
An error occurred while the RM/COBOL compiler was loading:

        Class not registered (the code is 80040154).

To resolve this problem, reinstall the RM/COBOL compiler, or register
  the RM/COBOL compiler with the /REGSERVER command.
```

As indicated, there are two ways to resolve this problem. You can do either of the following:

- Repeat the installation process.
- Use the /REGSERVER command-line option.

## Registering the Compiler

To register the RM/COBOL for Windows compiler in a directory other than the installation directory using the /REGSERVER command-line option:

1. First, make sure that **rmcbl90c.dll** is in the Windows System directory.
2. Then, either click the Windows **Start** button and select **Run** from the menu, or open an MS-DOS Prompt window.
3. Enter the following command:

   ```
   path\RMCOBOL /REGSERVER
   ```

   where, *path* is the drive and directory or the UNC location of these two files. For example:

   ```
   "c:\program files\rmcobol v9\rmcobol" /regserver
   ```

   **Note** The quotes are necessary only if the executable pathname contains spaces.
4. If the registration process is successful, a message, such as the following, is displayed:

   ```
   Server "c:\windows\system\rmcbl90c.dll"
      registration succeeded.
   ```

It does not matter which of the two clients, the console-mode or GUI-mode, is used to register the server, other than that the console-mode compiler displays the message in the console window while the GUI-mode compiler displays the message in a message box. Regardless of which client registers the server, either compiler client can use the registered server.

## Unregistering the Compiler

The RM/COBOL compiler also provides the /UNREGSERVER command-line option to unregister the compiler from Windows. Although the uninstallation program automatically unregisters the compiler, this can be done manually with the following command:

```
path\RMCOBOL /UNREGSERVER
```

When the compiler server has been properly registered, a message, such as the following, is displayed either in the console window for the console-mode compiler or in a message box for the GUI-mode compiler:

```
Server "c:\windows\system\rmcbl90c.dll"
   unregistration succeeded.
```

If the compiler server had not been properly registered or has already been unregistered, a detailed error message is displayed instead. For example:

```
The server could not be unregistered: Class not registered
```

In this error message, “Class not registered”, indicates that the server was not registered.

It is not necessary to unregister the compiler server before re-registering the compiler server from a different location.

### Showing the Compiler Registration

Finally, the following option will display the location of the currently registered compiler server:

```
path\RMCOBOL /SHOWSERVER
```

When the compiler server has been properly registered, a message, such as the following, is displayed either in the console window for the console-mode compiler or in a message box for the GUI-mode compiler:

```
Server "c:\windows\system\rmcbl90c.dll" is
   currently registered.
```

If the compiler server is not properly registered, a detailed error message is displayed instead. For example:

```
Show server failed: Class not registered
```

In this error message, “Class not registered”, indicates that the server is not registered.

## *Runtime Registration*

The RM/COBOL for Windows runtime system consists of two components:

- A client, called **runcobol.exe**
- A server, called **rmcbl90r.dll**

The runtime server DLL, which must be registered with Windows before RM/COBOL programs can be run, is automatically registered by the Setup program when the runtime system is installed. If the runtime is moved to a directory other than the installation directory without a re-installation, an error message is displayed in a message box indicating that there is a registration problem. The text of the error message is as follows:

```
An error occurred while the RM/COBOL runtime was loading:

        Class not registered (the code is 80040154).

To resolve this problem, reinstall the RM/COBOL runtime, or register
   the RM/COBOL runtime with the /REGSERVER command.
```

As indicated, there are two ways to resolve this problem. You can do either of the following:

- Repeat the installation process.
- Use the /REGSERVER command-line option.

## Registering the Runtime

To register the RM/COBOL for Windows runtime in a directory other than the installation directory using the /REGSERVER command-line option:

1. First, make sure that **rmcbl90r.dll** is in the Windows System directory.
2. Then, either click the Windows **Start** button and select **Run** from the menu, or open an MS-DOS Prompt window.
3. Enter the following command:

   ```
   path\RUNCOBOL /REGSERVER
   ```

   where, *path* is the drive and directory or the UNC location of these two files. For example:

   ```
   "C:\program files\rmcobol v9\runcobol" /regserver
   ```

   **Note** The quotes are necessary only if the pathname contains spaces.
4. If the registration process is successful, a message, such as the following, is displayed:

   ```
   Server "c:\windows\system\rmcbl90r.dll"
      registration succeeded.
   ```

## Unregistering the Runtime

The RM/COBOL runtime also provides the /UNREGSERVER command-line option to unregister the runtime from Windows. Although the uninstallation program automatically unregisters the runtime, this can be done manually with the following command:

```
path\RUNCOBOL /UNREGSERVER
```

When the runtime server has been properly registered, a message, such as the following, is displayed:

```
Server "c:\windows\system\rmcbl90r.dll"
   unregistration succeeded.
```

If the runtime server had not been properly registered or has already been unregistered, a detailed error message is displayed instead. For example:

```
The server could not be unregistered: Class not registered
```

It is not necessary to unregister the runtime server before re-registering the runtime server from a different location.

### Showing the Runtime Registration

Finally, the following option will display the location of the currently registered runtime server:

```
path\RUNCOBOL /SHOWSERVER
```

When the server has been properly registered, a message, such as the following, is displayed:

```
Server "c:\windows\system\rmcbl90r.dll" is
   currently registered.
```

If the runtime server is not properly registered, a detailed error message is displayed instead. For example:

```
Show server failed: Class not registered
```

In this error message, “Class not registered”, indicates that the server is not registered.

# System Removal for Windows

To remove RM/COBOL from your system:

1. Click the **Start** button, point to **Settings**, and then click **Control Panel**.
2. Double-click **Add/Remove Programs**.
3. From the Install/Uninstall tab in the Add/Remove Programs Properties dialog box, select **RM/COBOL** from the list of installed program packages.
4. Click the **Add/Remove** button.
5. Follow the prompts on your screen to complete the removal process.

All installed RM/COBOL system programs, files, shortcuts, and Windows registry entries are now removed. Customer files are not affected.

# System Configuration for Windows

As mentioned, RM/COBOL supports IBM PCs, full PC compatibles, and Windows systems. This section sets forth information required to configure RM/COBOL with each type of system.

## Creating a Windows Shortcut

When you create a shortcut in Windows, you must also specify the properties of the item. Properties include a description of the item (the application name) and the working directory where the application files are stored.

To create a shortcut for an application under Windows:

1. Open the folder to which you want the item added. (Note that you can also add an item directly to the desktop.)
2. Click the right mouse button to open a pop-up menu. Point to the **New** option and click **Shortcut**. The Create Shortcut dialog box opens.
3. In the Command line text box, type in a runtime system command, as described in Chapter 7: *Running* (on page 177). Click the Next button.
4. When prompted to name the shortcut, choose a name that uniquely identifies the application program. This name becomes the label that appears under the shortcut icon.
5. After Windows creates the shortcut, you must modify the properties of the shortcut in order for it to work properly. Right-click the shortcut icon and choose **Properties**. The Shortcut Properties dialog box opens.
6. Select the **Shortcut tab** in the dialog box. (Figure 4 illustrates the Shortcut Properties Tab used in this example.)
7. In the Start in text box, enter the name of the directory where the program files for this application are located and where new files will be placed. The directory you specify here becomes the current directory while the application program is running.

**Figure 4: Shortcut Properties Tab**

## Using Associations with Filename Extensions

During installation, RM/COBOL for Windows automatically sets up filename extension associations for **.cbl** and **.cob** files. These associations allow the user to compile or run source or object files by double-clicking these files when running the Windows File Manager or Windows Explorer.

Normally, you cannot pass command-line options to Windows programs executed using a filename extension association. However, using the Windows registry (on page 66), it is possible to inform the RM/COBOL compiler or runtime system of command-line options for all programs or for specific programs. For a discussion of the command-line options in the RM/COBOL configuration, see the Command Line Options property (on page 72).

Under Windows, it is also possible to drag and drop **.cbl** and **.cob** files to the RM/COBOL compiler or runtime system for execution. Dropping a **.cbl** file on a printer icon will print that source file.

## Prompting for a Filename

If the command line specified for the compiler or the runtime system has a ? character for the source or object filename, the Select an RM/COBOL Object File dialog box appears, as shown in Figure 5.

**Figure 5: Select an RM/COBOL Object File Dialog Box**


When the user selects the file from the list available in the space below the Look in drop-down list box, the filename in the File name text box replaces the ? character on the command line. To open (or start) the source or runtime system file, click the Open button. Double-clicking the name of the file also opens (or starts) the selected object file.

# Locating RM/COBOL Files on Windows

## File Locations within Operating System Pathnames on Windows

File locations are determined by the pathname of the file, according to operating system rules and conventions. A fully qualified pathname contains the drive specifier, a directory path, the filename, and the filename extension. A filename that begins with a universal naming convention (UNC) specifier (*server*) is also treated as a fully qualified pathname.

**Note** Novell NetWare syntax (*server**volume:filename*) is no longer supported. Use of UNC filename is now required (*server**volume**filename*).

If a pathname appears without a drive specifier, the current drive is assumed. If a pathname is specified without a directory path, RM/COBOL searches the current directory of the specified or assumed drive.

Specifying a directory path with a leading slash, a drive letter, or a volume name indicates to RM/COBOL that an exact filename has been specified. If RM/COBOL cannot find the file in the specified location, it will not look elsewhere. If you do not specify a directory path, and RM/COBOL cannot find the file in the assumed location, it will search for the file according to the directory search sequence. If a directory path is specified, but there is no leading slash, drive letter, or volume name, then the EXPANDED-PATH-SEARCH keyword (see page 317) of the RUN-FILES-ATTR configuration record determines whether the directory search sequence will be used. When the configuration keyword is set to its default value of NO, the directory search sequence will not be used. If the value is set to YES, then the entire name, including the directory path, will be appended to each entry in the directory search sequence in an attempt to locate the file.

## Directory Search Sequences on Windows

You can direct RM/COBOL to search for a file not found in the current working directory by using a predefined directory search sequence. There are two directory search sequences: one for the compiler and one for the runtime system.

To direct the RM/COBOL compiler to use the directory search sequence, set the RMPATH environment variable. You can do this by setting a synonym with the RM/COBOL Configuration (rmconfig) utility (on page 615). Alternatively, you can right-click the mouse button on a **.cbl** file, select the Synonyms Properties tab, and set the RMPATH synonym with the following syntax, as discussed in Setting Synonym Properties (on page 85):

```
path [;path] ...
```

To direct the RM/COBOL runtime system to use the directory search sequence, set the RUNPATH environment variable. You can do this by setting a synonym with the RM/COBOL Configuration (**rmconfig**) utility. You may also right-click the mouse button on a **.cob** file, select the Synonyms Properties tab, and set the RUNPATH synonym with the following syntax, as discussed in Setting Synonym Properties (on page 85):

```
path [;path] ...
```

For both the RMPATH and RUNPATH environment variable values, *path* indicates the directory that is to be searched for the file, and has the form:

```
[d:] [\] directory [\directory] ...
```

where, *d* is the drive specifier.

*directory* is the location of an existing file, or the location of a file that will be created.

If multiple *path*s are specified, they must be separated with semicolons.

Means other than setting synonyms can be used to set the RMPATH or RUNPATH environment variable values. Consult your operating system documentation for such methods. If synonyms are set, the synonyms will override values set by the operating system.

Figure 6 and Figure 7 illustrate the compiler and runtime system search sequences, respectively.

**Figure 6: Compiler Search Sequence**

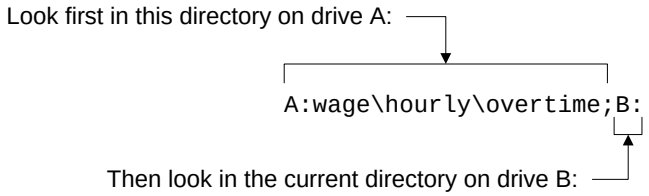
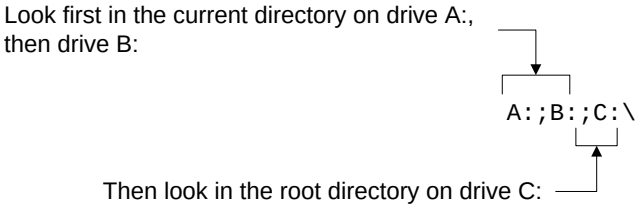



**Figure 7: Runtime System Search Sequence**




Files made to appear in the current directory by using Novell search directories when the Novell Search Mode is set to a value other than 2 will not be accessed. If a file to be accessed resides in a directory other than the current directory, that directory must be included in the RMPATH or RUNPATH directory list. This requirement also applies to files located in Novell search directories when the Novell Search Mode is set to a value other than 2.

The compiler, runtime system, and Indexed File Recovery (recover1) utility (on page 598) require access to other files in order to operate. These include the license vault and dynamic-link library files (with an extension of **.dll**). The license vault must reside in the same directory as the executable file. RM/COBOL looks for the other files first in the directory containing the executable file, then in the current directory, and finally in the directories specified in the PATH environment variable. For the dynamic-link library files, the default Windows system directory and then the default Windows directory will be searched prior to searching the directories specified in the PATH environment variable. On Windows NT-class operating systems, the search of the default Windows system directory is actually two searches. The first is a search of the default Windows 32-bit system directory while the second is a search of the Windows 16-bit system directory.

The compiler and runtime system may be executed from a directory other than the current directory if a complete pathname is specified in the command line, or if either the search directory of Novell NetWare or the DOS PATH directory search feature is used. If a complete pathname is not specified and the compiler or runtime system is not located in the current directory, the directories specified by PATH are searched.

## Novell NetWare Search Paths

Novell NetWare defines a search path for locating command files. RM/COBOL defines a search path for locating compiler files (RMPATH) and for locating runtime system files (RUNPATH). Both Novell NetWare and RM/COBOL search paths consist of a list of directories from which attempts are made to open files.

With RM/COBOL search paths, if any one of the directories in a user's path does not have search permission for the user, then the searching sequence stops for all remaining directories and a security violation is reported. This security violation indicates that the runtime system has been prevented from examining the directory for a file. If a security violation occurs, and the file is located in a directory for which the user has permission, examine the permissions for other directories in the RUNPATH sequence.

To prevent this security violation, take one of the following actions:

- Give the user search permission for all directories in RUNPATH, RMPATH, and the Novell NetWare search path.
- Alternatively, remove the directory from the search path.

**Note** This same security violation can occur when creating a new file, even if it is with OPEN OUTPUT. The RM/COBOL runtime system still searches RUNPATH to locate a file that needs to be replaced.

## File Access Names on Windows

The file access name you specify in the COBOL source program specifies the physical file or device to which input or output is directed. For information on specifying the file access name in a COBOL source program, see the discussion of the ASSIGN clause (file control entry) in Chapter 3: *Environment Division*, and the discussion of the VALUE OF clause in Chapter 4: *Data Division*, of the *RM/COBOL Language Reference Manual*.

To establish synonymy between a file access name, specified in your source program and another name specified when the program is run, use an environment variable. Environment variables may be set using the Synonyms tab of the Properties dialog

box, as illustrated in Figure 8. The Synonyms Properties tab is described in the topic Setting Synonym Properties (on page 85). Consult your operating system documentation for other methods of setting environment variables.

**Figure 8: Synonyms Tab of the Properties Dialog Box**


For example, let us say that you specified a generic file access name for program input-output and wish to direct it to a specific device or file. A generic file access name is one that does not specify a directory path or drive letter. Since the format of physical pathnames, including conventions of specifying drive letters and directory names, varies from one operating system to another, for maximum portability it is recommended that source programs specify generic file access names, preferably with eight or fewer letters. This recommendation only applies when the file access name is hard coded into the program as a literal.

For example, if the file control entry specifies:

```
SELECT REPORT-FILE ASSIGN TO PRINT, "report"
```

and no environment variable with the name “report” is found, RM/COBOL will create a file named **report** in the current directory on the current drive.

If, prior to running the program, you set the synonym “report” to a value of LPT1, all program output written to REPORT-FILE will be written to **LPT1**.

If—again prior to execution—you set the synonym “report” to a value of “A:\output\audit.lst”, RM/COBOL will create a file named **audit.lst** in the subdirectory **\output** on drive A without any need to modify or recompile the source program.

When an environment variable is not set, because there is no synonym set and no other method of setting the environment variable has been used, the file access name in the COBOL program specifies the actual filename. Synonym values can be canceled by highlighting the entry on the Synonyms tab of the Properties dialog box, clicking the Remove button, and restarting **runcobol.exe**.

Whether or not an environment variable is used to modify the file access name, if the resulting file access name does not include either a drive letter or a directory path, RUNPATH will be used by the runtime system to obtain the fully qualified

pathname. For additional information, see File Locations within Operating System Pathnames on Windows (on page 61).

The RESOLVE-LEADING-NAME and RESOLVE-SUBSEQUENT-NAMES keywords of the RUN-FILES-ATTR configuration record can be used to force resolution of one or more of the directory names from the environment. For more information, see the discussion of the RUN-FILES-ATTR configuration record (on page 316).

Control characters are removed from the file access name, but spaces are preserved since Win32 supports spaces in filenames.

The resulting file access name should follow the operating system rules for valid filenames and pathnames. If the file access name contains any of the characters

```
\   /   :   *   ?   "   <   >   |
```

except that, "\" may be used as a directory separator and ":" may be used to indicate a device, an error will occur when the file is opened.

## Windows System Print Jobs

When the resulting file access name is PRINTER or PRINTER*n*, where *n* is a decimal digit from 1 to 9, RM/COBOL refers to the Windows printer device attached to **LPT1:** or **LPT*n*:** respectively, provided that **LPT1:** or **LPT*n*:** has a Windows printer attached to it. This method causes the Windows printer device driver to be used, which does not allow the use of raw escape sequences.

When the resulting file access name is "PRINTER?", as described in the section Windows Printers (on page 306), RM/COBOL displays the standard Windows Print dialog box when the file is opened. This allows the user to select the destination Windows printer in a dynamic manner (that is, at execution). Once the "PRINTER?" device has been opened, the selected printer is remembered by the runtime, and subsequent opens do not display the standard Windows Print dialog box. The program may call the P$EnableDialog (on page 461) subprogram to force a standard Windows Print dialog box on the next open of "PRINTER?". The program may also call the P$DisableDialog (on page 460) subprogram to cause the Print dialog box not to appear when the "PRINTER?" device is opened for the first time. This feature can be useful when P$SetDialog (on page 462) has been called to preset the desired printer (obtained from P$EnumPrinterInfo or by other methods) and the application does not want the dialog to appear. The user may also set the Printer Dialog Always property (on page 79) in the Windows registry file to True to force the dialog box on every open of "PRINTER?". The program may also call the P$DisplayDialog (on page 460) subprogram at any time, to force the standard Windows Print dialog box to be displayed.

In order to send raw escape sequences to a printer, the printer must be opened directly by using the name "LPT1". The name "LPT*n*", where *n* is a decimal digit, also can be used.

The DEFINE-DEVICE configuration record may define other file access names that are to be treated as devices and change the default treatment of PRINTER and PRINTER*n*. See the DEFINE-DEVICE configuration record (on page 304) for additional information on configuring file access names that are to be treated as devices.

# Windows Registry

Beginning with version 6.5, RM/COBOL for Windows stores configuration information for the runtime system (**runcobol**), compiler (**rmcobol**), and Indexed File Recovery utility program (**recover1**) in the Windows registry. The registry is a hierarchical database used to store configuration settings and options maintained by Windows.

The Windows registry is organized much like a disk drive's directory structure. All of RM/COBOL's configuration information is stored under the "directory" path "HKEY_LOCAL_MACHINE\SOFTWARE\Liant Software Corporation\RM/COBOL\CurrentVersion". Three subdirectories underneath that path (rmcobol, runcobol, and recover1) correspond with information previously stored in the separate initialization files **rmcobol.ini**, **runcobol.ini**, and **recover1.ini**.

**Note** Previous versions of RM/COBOL for Windows stored program configuration information in separate initialization files located in the main Windows directory. These initialization files are no longer used. However, when distributing configuration information to end-users, initialization files can still be shipped with your product. To merge your program's configuration information into the Windows registry, include a call to the supplied Initialization File to Windows Registry Conversion (ini2reg) utility (on page 614) in your application's installation procedure.

You are not required to know the inner details of the Windows registry structure in order to change the properties of your programs. RM/COBOL for Windows includes Windows shell extensions that allow the manipulation of configuration information for default values as well as individual program settings without having to navigate through the Windows Registry Editor. Configuration information for a specific COBOL program may be edited by right-clicking a source or object file and choosing Properties. If a source file is chosen, the properties used when compiling that program, can be modified. If an object file is chosen, the properties used when running that program can be modified. The configuration options available in the Properties dialog box are described in the section Setting Properties (on page 68). Configuration information for programs and generic default values may also be edited by running the supplied RM/COBOL Configuration (rmconfig) utility (on page 615).

Users may migrate the complete RM/COBOL Windows registry information from one machine to another by using the Registry Editor (**regedit.exe**), which is included with Windows. This program allows entire sections of the Windows registry to be exported to a text file (with the **.reg** extension), which can then be imported into the Windows registry of another machine. Consult the Microsoft Windows help documentation for more information on **regedit.exe**.

# Windows Registry Considerations

Several Windows registry issues may be encountered when using the Initialization File to Windows Registry Conversion (ini2reg) utility (on page 614) if the RM/COBOL for Windows runtime executable has been renamed.

## *Renaming the RM/COBOL for Windows Runtime*

By default, the Windows registry key created by the **ini2reg** utility is the same as the name of the input initialization file (**.ini**). This registry key is also used by the RM/COBOL Configuration (rmconfig) utility (on page 615). The RM/COBOL for Windows runtime expects to find the configuration information under a key based on the name of the executable module. If you rename the RM/COBOL for Windows runtime executable, **runcobol.exe**, it is also necessary to rename the initialization file (**runcobol.ini**) to match the new runtime name before the **ini2reg** utility is run.

Furthermore, if the RM/COBOL for Windows runtime is renamed, the Windows Explorer SHELL/OPEN registry entry that names the runtime must be updated to reflect the new name. Otherwise, the Windows Explorer will be unable to find the runtime when a **.cob** file is opened, and the runtime will not correctly read configuration information from the registry.

The RM/COBOL installation program automatically sets the SHELL/OPEN registry entry to the drive and directory where the RM/COBOL for Windows runtime is installed. If the runtime is later renamed or moved, the Registry Editor (**regedit.exe**) supplied with Windows can be used to update the registry. The key is:

```
HKEY_LOCAL_MACHINE
  Software
    Classes
      RMCOBOL.Object
        shell
          open
            command
```

This entry must be set to the following:

```
x:\dir\filename.exe "%1"
```

where, *x:\dir* is the drive and directory containing the runtime, and *filename* is the name of the RM/COBOL for Windows runtime. If this path contains spaces, it must be surrounded by double quotes.

**WARNING** Use extreme caution when editing the Windows registry. Liant Software recommends that you do not change any other entries.

This entry can also be updated automatically using a properly prepared **.reg** file. See your Windows documentation for details.

# Setting Properties

This section describes the configuration options that can be set using the Properties dialog box.

**Note** The Properties dialog box contains a set of seven tabs. Each tab contains a set of three buttons that are active in all the tabs and which serve the same function. The OK button accepts all the settings selected on that tab and then closes the dialog box. The Cancel button closes the dialog box with saving any changes. The Apply button saves the settings specified on that tab without closing the dialog box, allowing you to select another page of options.

The following definitions explain terms used throughout this section.

| Term | Meaning |
|---|---|
| *Boolean* | Indicates a value of True or False. A value of 1 or 0 may also be used to indicate True or False. |
| *number* | Indicates a positive integer value. Valid example: "12". Invalid example: "=5". |
| *string* | Indicates alphanumeric characters. |
| *filename* | Indicates the operating system filename. |

## Selecting a File to Configure

The Select File tab, illustrated in Figure 9, allows you to select the source file, object file, or indexed file that you want to configure. The title bar on each tab of the Properties dialog box provides three important pieces of information depending on the settings selected on the Select File tab. First, it displays either the name of the specific COBOL program you selected or "Default", if you are setting system defaults for all programs. Second, it displays whether you are configuring the program for the runtime, compiler, or recovery utility. Third, it indicates the name of the custom key in the Windows registry if the default key is not being used.

**Note** If you have opened the Properties dialog box by right-clicking the mouse button on an RM/COBOL source or object file and then selecting Properties, this tab will not be available. You are only able to configure options for the default key in the currently selected individual file.

**Figure 9: Select File Tab**


The Select File tab contains the following options:

- **Configure.** The two options provided in this area allow the specification of configuration options for all programs or for a specific COBOL program.
  - **Default Properties.** When selected, the Default Properties option enables the other Properties tabs (Control, Synonyms, Colors, Toolbar, Menu Bar, and Pop-up Menu) to set system defaults for all files.
  - **Individual File.** When selected, the Individual File option enables the other Properties tabs to change the properties for the selected file. Any directory path for the selected file must not be specified. For source and object files, the extension must not be specified. For example, the source file **c:\mysrcdir\mysource.cbl** must be specified as **mysource** and the object file **c:\myobjdir\myobject.cob** must be specified as **myobject**. For indexed files, the extension, if any, must be specified. For example, the indexed file **c:\mydatdir\mydata.inx** must be specified as **mydata.inx**. The name specified is not necessarily a file at all, but corresponds to the first argument from the command line. In the case of the Runtime Command, this may be a program-name of a program within a library file.
- **Browse.** Use this button to open a dialog box that allows you to look for a file for which you want to set properties.
- **Remove.** Use this button to remove a selected file from the list of files.
- **Configure for.** The three options provided in this area determine the activity for which the Properties tabs will configure a file.
  - **Runtime.** If this option is selected, the settings for the **.cob** file will be shown and used when running the file.
  - **Compiler.** If this option is selected, the settings for the **.cbl** file are affected.
  - **Recovery.** If this option is selected, the settings used for recovering a data file with the Indexed File Recovery (recover1) utility (on page 598) are affected.

- **Key.** This option allows you to override the master key in the Windows registry that is used to store the configuration information. This is most useful if you have renamed the compiler, runtime, or recover utility program. The options in this area include:

  - **Default.** If this option is selected, the default key for each product is used. For the runtime, the default key is **runcobol**; for the compiler, it is **rmcobol**; and for the recovery utility, it is **recover1**.

  - **Custom.** Use this option to override the default key. Enter a new key name in the text box and press the **Set** button.

  **Note** The combination of the selections in the **Configure for** area and the **Key** option together affect where the values are stored.

- **Use Defaults.** The behavior of the Use Defaults button is dependent upon whether the Default Properties or the Individual File option is in effect.

  If the Default Properties option is selected, choosing the Use Defaults button causes the system defaults to be reset to the values that were in place when the product was originally installed. Note, however, that any property values set for an individual file will not be reset.

  If the Individual File option is selected, choosing the Use Defaults button causes property values that have been overridden for the selected file to be reset to use the system defaults.

## Setting Control Properties

The Control Properties tab, illustrated in Figure 10, allows you to set various properties for the file that was specified using the Select File tab. The properties that can be set or modified are discussed in the following sections.

**Figure 10: Control Properties Tab**

The Control Properties tab contains the following options:

- **Property.** This list box presents an alphabetical listing of the properties that are used to configure the physical appearance of the RM/COBOL program. An area below this list box contains a description of the selected property. (Each of these properties is discussed in the sections that follow.)
- **Default Setting.** Select this button to use the selected property's default value. That default value will be shown in the Value area (described below). See the Default Properties option on the Select File tab for information on configuring default values. For more information, see the discussion in Selecting a File to Configure (on page 68).
- **Custom Setting.** Select this button to override the default value for the selected property.
- **Value.** This area displays the value associated with the property selected in the Property list box and allows you to change it. Note that this area is disabled unless the Use Custom Settings button is selected.

## Auto Paste Property

The Auto Paste property specifies a *Boolean* value that enables or disables the Auto Paste function. Setting Auto Paste to True enables the Auto Paste feature and double-clicking the mouse button transfers the marked data to a pending ACCEPT field. If Auto Paste is set to False, double-clicking the mouse button marks a word of text. The default value for this property is False.

**Note** During installation you have the option to allow certain configuration information to be added to the Windows registry. This configuration information, included in the file **rmcobol.reg**, sets the system default value of the Auto Paste control property to the custom setting True. This default value will be used for individual files unless overridden by a custom setting.

The C$GUICFG (on page 547) subprogram can be used to change the Auto Paste property temporarily in order to manipulate the graphical user interface.

## Auto Scale Property

The Auto Scale property specifies a *Boolean* value that determines whether to implement auto scaling of fonts when the RM/COBOL runtime window is resized. Setting Auto Scale to True automatically changes the font size when the window is resized. Setting Auto Scale to False turns off this capability. The default value for this property is True. See also the Sizing Priority property (on page 82).

The setting of the Auto Scale property is ignored if the Scroll Buffer Size property is set to a non-zero value.

The C$GUICFG (on page 547) subprogram can be used to change the Auto Scale property temporarily in order to manipulate the graphical user interface.

## Command Line Options Property

The Command Line Options property defines a series of command-line options to be passed to the compiler, runtime system, or recovery utility, depending on whether "Compiler", "Runtime", or "Recovery" was selected on the Select File tab in the configuration Properties dialog box, as described in Selecting a File to Configure (on page 68). Command options are processed in the following order:

1. Options specified in the Command Line Options property for Default Properties.
2. Options specified in the Command Line Options property for the filename that was specified in the actual command line. However, for a clarification regarding the use of "?" or wildcard characters in the filename specified on the Compile Command line, see Multiple File Compilation on Windows (on page 139).
3. Options specified in the actual command line submitted by the user.

Since for most options, a later specification of the option overrides a prior specification, this means that options specified on the actual command line take precedence over command-line options specified in the registry and options specified in the Command Line Options property for a particular filename take precedence over options specified in the Command Line Options property for Default Properties. This is not true of cumulative options, such as the L Runtime Command Option, which are accumulated from left to right as the command-line options are processed in the order given above. For more information, see Compile Command Options (on page 141), Runtime Command Options (on page 179), and Recovery Command Options (on page 600).

## Cursor Overtype Property

The Cursor Overtype property determines the appearance of the cursor during ACCEPT operations when in overtype mode. For more information, see Cursor Types (on page 34).

The following values are valid:

| Value | Meaning |
|---|---|
| HorzLine | Displays the cursor as a horizontal line at the bottom of the character cell. |
| HalfBox | Displays the cursor as a half box at the bottom of the character cell. |
| FullBox | Displays the cursor as a full box. |
| VertLine | Displays the cursor as a vertical line at the right of the character cell. |

The default value for this property is HorzLine.

## Cursor Insert Property

The Cursor Insert property determines the appearance of the cursor during ACCEPT operations when in insert mode (see page 105).

The following values are valid:

| Value | Meaning |
|---|---|
| HorzLine | Displays the cursor as a horizontal line at the bottom of the character cell. |
| HalfBox | Displays the cursor as a half box at the bottom of the character cell. |
| FullBox | Displays the cursor as a full box. |
| VertLine | Displays the cursor as a vertical line at the right of the character cell. |

The default value for this property is HalfBox.

## Cursor Full Field Property

The Cursor Full Field property determines the appearance of the cursor during ACCEPT operations when in the input field is full. See Cursor Types (on page 105). The following values are valid:

| Value | Meaning |
|---|---|
| HorzLine | Displays the cursor as a horizontal line at the bottom of the character cell. |
| HalfBox | Displays the cursor as a half box at the bottom of the character cell. |
| FullBox | Displays the cursor as a full box. |
| VertLine | Displays the cursor as a vertical line at the right of the character cell. |

The default value for this property is FullBox.

## Enable Close Property

The Enable Close property specifies a *Boolean* value that enables or disables the Close menu item on the Control menu as well as the Close button in the upper-right corner of the window. Setting Enable Close to True enables the Close menu item and the Close button. Setting Enable Close to False dims and disables the Close menu item and the Close button. The default value for this property is True.

The C$GUICFG (on page 547) subprogram can be used to change the Enable Close property temporarily in order to manipulate the graphical user interface.

## Enable Properties Dialog Property

The Enable Properties Dialog property specifies a *Boolean* value that enables or disables the Properties menu item on the Control menu. Setting Enable Properties Dialog to True enables the Properties menu item. Setting Enable Properties Dialog to False dims and disables the Properties menu item. The default value for this property is True.

The C$GUICFG (on page 547) subprogram can be used to change the Enable Properties Dialog property temporarily in order to manipulate the graphical user interface.

## Font Property

The Font property specifies the typeface to use as well as point size and style. The typeface must be a fixed-width (or monospaced) font, such as Courier. Clicking the Select Fonts button opens the Fonts dialog box, which provides a list of available fonts, styles, and sizes.

## Font CharSet OEM Property

The Font CharSet OEM property determines the display character sets considered to be OEM character sets when the native character set uses the OEM codepage. In this case, RM/COBOL considers internal character data to be OEM and converts displayed characters to ANSI unless the chosen display font has an OEM character set. Fonts with the Arabic, Baltic, East Europe, Greek, Hebrew, Russian, and Turkish character sets generally require conversion from OEM to ANSI. The value NotANSI assumes all character sets other than the ANSI character set are OEM; this was the original RM/COBOL assumption. The value OEMSymbolDefault assumes that only the OEM, Symbol, and Default character sets are OEM and that all other character sets are ANSI. The default is OEMSymbolDefault. For printer character sets, as opposed to display character sets, see the Printer Font CharSet OEM property (on page 80).

**Note** The value of the Font CharSet OEM property is stored in the registry as a string value for the key FontCharsetOem. This string is a comma or space separated list of OEM character set numbers. A range of OEM character set numbers may be specified with a hyphen-separated pair of numbers. Alphabetic text or any text contained between braces is considered commentary. While the RMCONFIG user interface only allows setting NotANSI, that is, “1-255”, or OEMSymbolDefault, that is, “255,2,1”, any set of character set numbers may be listed in the registry in order to include or exclude specific character sets for the OEM to ANSI conversion done when characters are displayed. The specified string will be used until it is modified, either by RMCONFIG or other means such as regedit.

When the native character set uses the ANSI codepage, this property is ignored. In this case, code points are converted from ANSI to OEM only when the display font default script is OEM/DOS; otherwise, no conversion is necessary and none occurs.

## Full OEM To ANSI Conversions Property

The Full OEM To ANSI Conversions property specifies a *Boolean* value that determines whether to convert a character from OEM to ANSI or from ANSI to OEM when the native character set uses the OEM codepage. This property affects titles, menus, and other Windows objects. Because Windows uses the ANSI character set, the default setting (True) causes all output to be converted from the OEM character set to the ANSI character set and all input from these controls to be converted from ANSI to OEM. If, however, the user wants to avoid these conversions, this property should be set to False in order to suppress the conversion. Setting this value to False causes the runtime system to behave as it did prior to the RM/COBOL 6.5 release.

**Note** The European "Latin-1" character set is the same as the Windows native ANSI character set.

The C$GUICFG (on page 547) subprogram can be used to change the Full OEM to ANSI Conversions property temporarily in order to manipulate the graphical user interface.

When the native character set uses the ANSI codepage, this property is ignored.

## Icon File Property

The Icon File property specifies the icon *filename* from which to load icons for the toolbar. See the Name option of the Toolbar Properties tab, described in Setting Toolbar Properties. This property is used only if the Toolbar property (on page 83) is set to True. The default value is **rmtbar.vrf.** See Table 12 on page 94 for more information.

The C$GUICFG (on page 547) subprogram can be used to change the Icon File property temporarily in order to manipulate the graphical user interface.

## Load Registry On CALL Property

The Load Registry On CALL property specifies a *Boolean* value that enables or disables additional processing of the Windows registry file. If set to True, the registry is re-examined whenever a COBOL subprogram is called. The subprogram name is treated as if it were a *filename* and causes corresponding registry entries to be processed. If the value is set to False, the registry is not re-examined. The default value for this property is False.

**Note** Use caution when setting the value of the Load Registry On CALL property to True as a system default. Doing so can affect the performance of your application. This behavior can occur when using RM/Panels because an RM/Panels application uses many subprogram calls. Alternatively, you can use the C$TBar (on page 572), C$MBar (on page 552), or C$GUICFG (on page 547) subprograms to manipulate the toolbar and menu bar instead of using the Windows registry file and the Load Registry on CALL property.

## Load Registry On RETURN Property

The Load Registry On RETURN property specifies a *Boolean* value that enables or disables additional processing of the Windows registry file. If set to True, the registry is re-examined whenever a COBOL subprogram is exited. The calling program's name is treated as if it were a *filename* and causes corresponding registry entries to be processed. If the value is set to False, the registry is not re-examined. The default value for this property is False.

**Note** Use caution when setting the value of the Load Registry On RETURN property to True as a system default. Doing so can affect the performance of your application. This behavior can occur when using RM/Panels because an RM/Panels application uses many subprogram calls. Alternatively, you can use the C$TBar (on page 572), C$MBar (on page 552), or C$GUICFG (on page 547) subprograms to manipulate the toolbar and menu bar instead of using the Windows registry file and the Load Registry on RETURN property.

## Logo Bitmap Property

The Logo Bitmap property specifies a *Boolean* value that determines whether a Logo Bitmap is displayed. If the value is set to True, the file specified by the Logo Bitmap File property (described below) is displayed. If the value is set to False, it is not displayed. The default value for this property is True.

## Logo Bitmap File Property

The Logo Bitmap File property specifies the bitmap (**.bmp**) *filename* that may be displayed in the RM/COBOL runtime window when an application is started. The bitmap is centered in the RM/COBOL runtime window until an erase screen operation is encountered (DISPLAY ERASE). You can build a simple RM/COBOL program that displays a bitmap, responds to keyboard sequences (such as function keys that could be generated from the menus or toolbar), and dispatches the appropriate code. The default value is **run.bmp**, **rmc.bmp**, or **rec.bmp** for the runtime system, compiler, and Indexed File Recovery utility program, respectively. If the bitmap file is not found, or if Logo Bitmap (described above) is set to False, this property is ignored.

### Main Window Type Property

The Main Window Type property determines the style of the RM/COBOL runtime window (the window that is activated when the RM/COBOL application begins execution).

The following values are valid:

| Value | Meaning |
|---|---|
| Hidden | The window is not activated and is hidden. |
| Minimized | The window is activated and is displayed as an icon. |
| Maximized | The window is activated and is displayed in its maximized state. |
| Show | The window is activated and is displayed in its current size and position. |

The default value for this property is Show.

### Mark Alphanumeric Property

The Mark Alphanumeric property specifies a *Boolean* value that determines the terminating conditions for selecting a word from the application window. If Mark Alphanumeric is set to True, double-clicking the mouse button to mark a word selects characters until a non-alphanumeric character is encountered. If Mark Alphanumeric is set to False, selection occurs when a blank is encountered. The default value for this property is True.

The C$GUICFG (on page 547) subprogram can be used to change the Mark Alphanumeric property temporarily in order to manipulate the graphical user interface.

### Offset X Property

The Offset X property specifies a *number* that identifies the leftmost location (as a pixel offset from the left edge of the screen) of the RM/COBOL runtime window. The default value for this property is 0.

### Offset Y Property

The Offset Y property specifies a *number* that identifies the uppermost location (as a pixel offset from the top edge of the screen) of the RM/COBOL runtime window. The default value for this property is 0.

### Panels Controls 3D Property

The Panels Controls 3D property specifies a *Boolean* value that causes certain RM/Panels for Windows controls (date, time, alpha, and numeric fields) to take on a three-dimensional appearance. This capability was not available in prior versions of RM/Panels for Windows. The default value, False, causes existing applications to appear in the same manner as they did under previous versions of RM/COBOL.

### Panels Static Controls Border Property

The Panels Static Controls Border property specifies a *Boolean* value that causes the Static Text Control (a new Panels control type) to have a border. The default value, False, causes these controls to be drawn without a border.

### Paste Termination Property

The Paste Termination property specifies a *Boolean* value that affects automatic termination of fields pasted into a pending ACCEPT statement, using either the Paste function (see page 106) or the Auto Paste property (on page 71). If Paste Termination is set to True, data transfer will continue until the data is exhausted, including all tabs and carriage returns. If Paste Termination is set to False, data transfer stops when a tab or carriage return is encountered. There is a carriage return at the end of each line of text in the Windows Clipboard. The default value for this property is True.

The C$GUICFG (on page 547) subprogram can be used to change the Paste Termination property temporarily in order to manipulate the graphical user interface.

### Persistent Property

The Persistent property specifies a *Boolean* value that affects the behavior of the RM/COBOL runtime window when the RM/COBOL program, compiler, or Indexed File Recovery utility program terminates. If Persistent is set to True, the window will not close until dismissed by the user. If Persistent is set to False, the window will close immediately upon completion. The default value for this property is False.

If any RM/COBOL runtime window disappears upon completion before the user is able to read the final text displayed in that window, then set Persistent to True and close the window manually after reading the final text.

The C$GUICFG (on page 547) subprogram can be used to change the Persistent property temporarily in order to manipulate the graphical user interface.

### Pop-Up Window Positioning Property

The Pop-Up Window Positioning property determines the method used to initially position a pop-up window. The value Corrected positions the pop-up window with LINE 1 COLUMN 1 at the line and column specified in the DISPLAY statement, as specified in the documentation for pop-up windows in Line and Position Phrases (on page 206). The value Traditional positions the pop-up window as incorrectly implemented in initial releases of RM/COBOL for Windows, where the pop-up window is generally positioned lower and further to the right by a few pixels. The default value is Traditional.

### Printer Dialog Always Property

The Printer Dialog Always property specifies a *Boolean* value that affects the behavior of the RM/COBOL runtime when opening the "PRINTER?" (dynamic printer selection) device (see page 307). If Printer Dialog Always is set to True, the standard Windows Print dialog box will appear when the dynamic printer device is opened. If Printer Dialog Always is set to False, the dialog box will appear only the first time the dynamic printer is opened. The P$EnableDialog (on page 461) subprogram may be called to cause the dialog to appear on the next open of the dynamic printer). The default value for this property is False.

The C$GUICFG (on page 547) subprogram can be used to change the Printer Dialog Always property temporarily.

### Printer Dialog Never Property

The Printer Dialog Never property specifies a *Boolean* value that affects the behavior of the RM/COBOL runtime when opening the "PRINTER?" (dynamic printer selection) device, as described in Windows Printers (on page 306). If Printer Dialog Never is set to True, the standard Windows Print dialog box will never appear when the dynamic printer device is opened. In this case, the "PRINTER?" device behaves like the Printer (default) printer device. If Printer Dialog Never is set to False, the appearance of the dialog box is controlled by the setting of the Printer Dialog Always property. The default value for this property is False.

The C$GUICFG (on page 547) subprogram can be used to change the Printer Dialog Never property temporarily.

**Note** If the Printer Dialog Never property is set to True, the standard Windows Print dialog box will never appear, regardless of the state of the Printer Dialog Always property.

### Printer Enable Escape Sequences Property

The Printer Enable Escape Sequences property specifies a *Boolean* value that determines whether printing will allow embedded RM/COBOL-specific escape sequences. For a description of these sequences, see RM/COBOL-Specific Escape Sequences (on page 525). If the value is set to True, the RM/COBOL runtime system will recognize the sequences. If the value is set to False, the runtime system will ignore those escape sequences. The default value for this property is False.

**Note** Setting the Printer Enable Escape Sequences property to True affects all Windows printers that the COBOL program uses. To allow embedded RM/COBOL-specific escape sequences for only specific printers, use the P$EnableEscapeSequences (on page 483) subprogram or the ESCAPE-SEQUENCES keyword of the DEFINE-DEVICE configuration record (on page 304).

### Printer Enable Null Esc. Seq. Property

The Printer Enable Null Esc. Seq. property defines a *Boolean* value that specifies whether the ASCII NUL character will be ignored when written to a printer from within an escape sequence. When the value is set to True, NUL characters within an escape sequence are ignored and are not sent to the printer. When the value is set to False, NUL characters are changed to spaces. The default value for this property is False. For more information, see RM/COBOL-Specific Escape Sequences (on page 525).

### Printer Enable Raw Mode Property

The Printer Enable Raw Mode property specifies a *Boolean* value that determines whether Windows printers will be opened in raw mode. If the value is set to True, the runtime system will open printers in raw mode. This allows certain networked printers on Windows NT-class servers to respond to embedded escape sequences. See the P$SetRawMode (on page 491) subprogram for a more complete description of raw mode. Most P$ subprograms are not available if RAW mode is used. If the value is set to False, the runtime system will treat the printer as a normal Windows printer. The default value for this property is False.

**Note** Setting the Printer Enable Raw Mode property to True affects all Windows printers that the COBOL program uses. To allow raw mode printing for only specific printers, use the P$SetRawMode (on page 491) subprogram.

### Printer Font CharSet OEM Property

The Printer Font CharSet OEM property determines the character sets considered to be OEM character sets for printer fonts when the native character set uses the OEM codepage. In this case, RM/COBOL considers internal character data to be OEM and converts printed characters to ANSI unless the chosen printer font has an OEM character set. Fonts with the Arabic, Baltic, East Europe, Greek, Hebrew, Russian, and Turkish character sets generally require conversion from OEM to ANSI. The value NotANSI assumes all character sets other than the ANSI character set are OEM; this was the original RM/COBOL assumption. The value OEMSymbolDefault assumes that only the OEM, Symbol, and Default character sets are OEM and that all other character sets are ANSI. The default value for this property is OEMSymbolDefault. For display character sets, as opposed to printer character sets, see the Font CharSet OEM property (on page 74).

**Note** The value of the Printer Font CharSet OEM property is stored in the registry as a string value for the key PrinterFontCharsetOem. This string is a comma or space separated list of OEM character set numbers. A range of OEM character set numbers may be specified with a hyphen-separated pair of numbers. Alphabetic text or any text contained between braces is considered commentary. While the RMCONFIG user interface only allows setting NotANSI, that is “1-255”, or OEMSymbolDefault, that is, “255,2,1”, any set of character set numbers may be listed in the registry in order to include or exclude specific character sets for the OEM to ANSI conversion done when characters are printed. The specified string will be used until it is modified, either by RMCONFIG or other means such as regedit.

When the native character set uses the ANSI codepage, this property is ignored. In this case, code points are converted from ANSI to OEM only when the printer font default script is OEM/DOS; otherwise, no conversion is necessary and none occurs.

### Remove Trailing Blanks Property

The Remove Trailing Blanks property defines a *Boolean* value that specifies whether trailing blanks will be removed from the Toolbar and Menu Bar strings before they are sent to the COBOL program's ACCEPT statement for processing. The default value for this property is True. For more information, see Setting Toolbar Properties (on page 88) and Setting Menu Bar Properties (on page 91).

The C$GUICFG (on page 547) subprogram can be used to change the Remove Trailing Blanks property temporarily in order to manipulate the graphical user interface.

### Screen Read Line Draw Property

The Screen Read Line Draw property defines a *Boolean* value that enables or disables the return of DOS line draw characters in the screen read buffer for the line draw characters specified in Table 22: System-Specific Line Draw Characters in ACCEPT and DISPLAY Phrases (on page 195) when doing a screen read, as discussed in the C$SCRD (on page 559) subprogram. The default value for this property is False, which causes a screen read to return plus, hyphen, and bar characters for line draw characters.

The C$GUICFG (on page 547) subprogram can be used to change the Screen Read Line Draw property temporarily during the execution of a run unit.

### Scroll Buffer Size Property

The Scroll Buffer Size property specifies a *number* that affects the virtual size of the RM/COBOL runtime window. The number of rows initially displayed in the window is determined by the ROWS keyword (see page 330) in the TERM-ATTR configuration record. The Scroll Buffer Size property determines the number of rows that can be scrolled off the screen using the vertical scroll bar. Setting the Scroll Buffer Size to a non-zero value overrides the Auto Scale property (on page 71) and automatically turns on the vertical scroll bar. The default value for this property is 0.

The maximum value depends on the font size and is limited to approximately 1600 on Windows 9x-class operating systems and to 2400 on Windows NT-class operating systems. Values larger than the maximum may be set, but display problems can occur if more than the actual maximum number of lines is scrolled without an intervening erase. The actual maximum is a limit on the number of pixels in the virtual screen height, which is computed as the font height in pixels (typically, 15 to 20) times the quantity of the Scroll Buffer Size plus the number of rows in the actual screen area. This pixel limit is 32,767 on Windows 9x-class operating systems because of an internal Windows limit, and it is 50,000 on Windows NT-class operating systems because of an RM/COBOL implementation limit.

### Show Return Code Dialog Property

The Show Return Code Dialog property specifies a *Boolean* value that determines whether the Return Code message box (on page 106), indicating the program exit codes (on page 186), should be displayed when an error occurs. Automated systems, which handle such errors and do not require operator assistance, may wish to suppress the message box and continue processing. The default value for this property is True, which causes the message box to be displayed.

### Show Through Borders Property

The Show Through Borders property specifies a *Boolean* value that determines whether the border of an overlayed pop-up window is shown when overlayed by a pop-up window without a FILL character. When Show Through Borders is set to True, the border is visible. When Show Through Borders is set to False, the border is not visible. The default value for this property is False.

### Sizing Priority Property

The Sizing Priority property specifies whether to make the width or height a priority when auto scaling fonts. If the user resizes the window and auto scaling is on, the system will select a font to match the new size of the window. The new size will be based on the width or height of the window. The default value for this property is Width. See also the Auto Scale property (on page 71).

The C$GUICFG (on page 547) subprogram can be used to change the Sizing Priority property temporarily in order to manipulate the graphical user interface.

### Status Bar Property

The Status Bar property specifies a Boolean value that determines whether the status bar is initially visible. Setting Status Bar to True turns on the status bar. Setting Status Bar to False turns off the status bar. The default value for this property is False.

The C$GUICFG (on page 547) subprogram can be used to change the Status Bar property temporarily in order to manipulate the graphical user interface.

### Status Bar Text Property

The Status Bar Text property specifies the initial *string* of text to be placed in the status bar. The default value is an empty string. This text is displayed in the status bar whenever the mouse is in the client area of the window.

**Note** The C$SBar (on page 559) subprogram also can be used to display a status bar in the RM/COBOL runtime window.

## SYSTEM Window Type Property

The SYSTEM Window Type property determines the style of the window shown by a program run using the SYSTEM (on page 578) non-COBOL subprogram.

The following values are valid:

| Value | Meaning |
|---|---|
| Hidden | The window is not activated and is hidden. |
| Minimized | The window is activated and is displayed as an icon. |
| Maximized | The window is activated and is displayed in its maximized state. |
| Show | The window is activated and is displayed in its current size and position. |
| ShowNoActivate | The window is displayed in its most recent size and position, but is neither activated nor given focus. |
| MinimizedNoActive | The window is displayed as a minimized window, but the window is neither activated nor given focus. |

The default value for this property is Show.

The C$GUICFG (on page 547) subprogram can be used to change the System Window Type property temporarily for subsequent calls to the SYSTEM non-COBOL subprogram in the same run unit.

## Title Text Property

The Title Text property specifies the *string* of text to be placed in the runtime window of the RM/COBOL program that is currently running. The default title is "RM/COBOL" if no program-name is specified on the **runcobol** command line. Otherwise, the default value for this property is the initial program-name.

**Note** The C$Title (on page 575) subprogram also can be used to specify the text to be placed in the RM/COBOL runtime window.

## Toolbar Property

The Toolbar property specifies a *Boolean* value that determines whether the toolbar is visible initially. Setting Toolbar to True turns on the toolbar. Setting Toolbar to False turns off the toolbar. The default value for this property is False.

The C$GUICFG (on page 547) subprogram can be used to change the Toolbar property temporarily in order to manipulate the graphical user interface. In addition, the C$TBar (on page 572), C$TBarEn (on page 573), and C$TBarSeq (on page 574) subprograms can be used to affect the toolbar during execution.

### Toolbar Prompt Property

The Toolbar Prompt property specifies how to display the toolbar prompt string value when the mouse cursor hovers over a toolbar command button.

The following values are valid:

| Value | Meaning |
|---|---|
| None | The prompt is not displayed. |
| StatusBar | The prompt is displayed only in the status bar. |
| ToolTip | The prompt is displayed only as a tooltip. |
| Both | The prompt is displayed in the status bar and as a tooltip. |
| SplitNewline | The prompt is split at the first newline (`x'0a'`) character; the leading portion is displayed in the status bar and the trailing portion is displayed as a tooltip. |
| SplitColon | The prompt is split at the first colon (`:`) character; the leading portion is displayed in the status bar and the trailing portion is displayed as a tooltip. |
| SplitVertBar | The prompt is split at the first vertical bar (`\|`) character; the leading portion is displayed in the status bar and the trailing portion is displayed as a tooltip. |

The default value for this property is Both.

For information on setting toolbar prompt values, see Setting Toolbar Properties (on page 88) and C$TBar (on page 572).

The C$GUICFG (on page 547) subprogram can be used to change the Toolbar Prompt property temporarily in order to manipulate the graphical user interface. Changes to the Toolbar Prompt property do not affect the display of the prompt for an existing toolbar; the change affects only the display of the prompt for a toolbar created by calling C$TBar after the change has been made.

### Update Timeout Property

The Update Timeout property specifies a *number* that represents a delay before a screen refresh occurs. The value of *number* is specified in milliseconds. A larger number causes DISPLAY statements to occur less frequently, potentially improving screen display performance (especially when multiple DISPLAY statements of short records occur in a short period of time). This property may also be used to force DISPLAY statements to occur more frequently. The default value is 500 milliseconds (half of a second).

The C$GUICFG (on page 547) subprogram can be used to change the Update Timeout property temporarily in order to manipulate the graphical user interface.

### *Use Windows Colors Property*

The Use Windows Colors property defines a *Boolean* value that specifies whether the standard Windows colors, as set in the Windows Control Panel (Display Properties dialog box, Appearance tab), are used as the RM/COBOL default foreground and background colors. If Use Windows Colors is set to True, the standard Windows colors will be used. If Use Windows Colors is set to False, BLACK will be used for the background and WHITE for the foreground. A value of False allows the same behavior as that found in versions of RM/COBOL prior to 6.0. The default value for this property is True.

## Setting Synonym Properties

The Synonyms Properties tab, illustrated in Figure 11, allows you to establish synonym *name(s)* and their *values* for the file that was specified using the Select File tab, as described in Selecting a File to Configure (on page 68). The *name* is a string that is the name of a variable placed in the program's environment. The *value* is a string that is the value of *name* in the environment. A synonym can be used to specify the actual file access name for a COBOL program, or to specify other environment variables such as the RMPATH and RUNPATH directory search sequences described in Directory Search Sequences on Windows (on page 61).

**Figure 11: Synonyms Properties Tab**


The Synonyms Properties tab contains the following options:

- **Name.** The value entered in this list box is the name of the synonym to which you are assigning a value.
- **Value.** The value in this text box is the value assigned to the synonym selected in the Name list box.
- **Remove.** Use this button to clear the value for the currently selected synonym name and remove it from the list.
- **Remove All.** Use this button to clear all synonym values for the currently selected program.

These synonyms are used to set environment variables for the runtime, compiler, or recovery utility (per the Select File tab setting, as described in Selecting a File to Configure (on page 68).) Synonyms override environment variable settings that may already exist because of operating system methods of setting environment variables, such as the DOS SET command or the Environment Variables system property on Windows NT-class operating systems. However, environment variables set with CodeWatch cause any matching synonym names to be ignored, so that the environment variables will have the values specified in CodeWatch (see the "Creating a Workspace" topic in the CodeWatch manual for information on setting environment variables with CodeWatch). As a result, these synonyms may be used to establish a connection between the open name of the file, *literal-1* or *data-name-1* (see the "Input-Output Section" in Chapter 3: *Environment Division* of the *RM/COBOL Language Reference Manual* for more information), and the actual file access name.

If either the Load Registry On CALL property (on page 75) or Load Registry On RETURN property (on page 76) is set to True, synonyms will be reprocessed whenever a subprogram is called or exited. Synonym assignments are cumulative. For example, if a synonym is assigned for a called subprogram, its value is unchanged when the subprogram exits unless Load Registry On RETURN is set to True and the synonym is defined for the calling program.

## Setting Color Properties

The Colors Properties tab, illustrated in Figure 12, allows you to control color mapping for the file that was specified using the Select File tab, as described in Selecting a File to Configure (on page 68).

**Figure 12: Colors Properties Tab**


The Colors Properties tab contains the following options:

- **Color.** Use this list box to select the color you want to change. The first eight colors in this list box correspond to the *color-names* for the keywords (FCOLOR and BCOLOR) allowed in a CONTROL phrase of an ACCEPT or DISPLAY statement. These colors are displayed if low intensity is selected. The remaining eight colors correspond to the same *color-names* if high intensity is selected. Note that GRAY is “HIGH BLACK” and YELLOW is “HIGH BROWN.” The current color setting is displayed to the right of each name. For more information, see ACCEPT and DISPLAY Phrases (on page 195).

  **Note** An asterisk (*) after the name indicates that the default color has not been overridden and the default will be used. If the Change button (see the following item) is used to override the default, the overriding color is displayed on the right.

- **Change.** Use this button to display a Color Selection dialog box that allows you to select a color to override the selected color name.

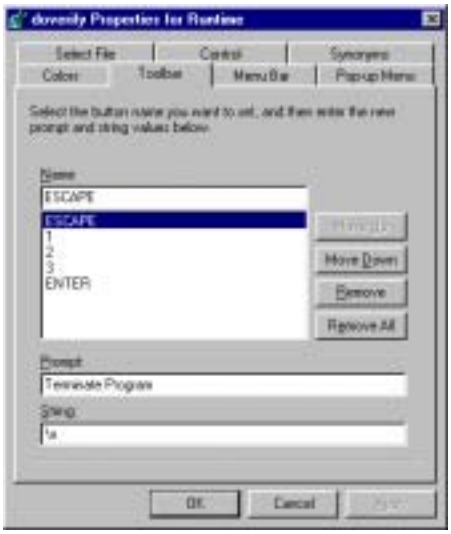
- **Use Default.** Use this button to clear the overriding color for the currently selected color name, thereby using the default color.

## Setting Toolbar Properties

The Toolbar Properties tab, illustrated in Figure 13, allows you to define the string that is to be sent to the program through the COBOL ACCEPT statement when the corresponding toolbar button is pressed.

The C$TBar (on page 572) subprogram also can be used to display a toolbar in the RM/COBOL runtime window.

**Figure 13: Toolbar Properties Tab**




The Toolbar Properties tab contains the following options:

- **Name.** The value entered in this text box is the name of the icon stored in the filename specified by the Icon File property (on page 75).
- **Prompt.** The value entered in this text box is an optional text string that is displayed whenever the mouse cursor hovers over the toolbar icon that is specified by the icon name. The text string may be displayed in the status bar, as a tooltip, or both as specified by the Toolbar Prompt property (on page 84). The text string may contain one of the separator characters newline (x'0a'), colon (":"), or vertical bar ("|") to divide it into separate status bar and tooltip text. The appropriate separator character is determined by the Toolbar Prompt property.
- **String.** The value entered in this text box is the ASCII text string returned when the toolbar icon is clicked. This text string may also contain special characters for the Return, Tab, Escape, or Function keys. If the first character is a greater than character (>), the characters that follow are executed as a command. The special characters are described in Table 11. (These characters are interpreted by the COBOL ACCEPT statement, as configured by the TERM-INPUT configuration record, described on page 332, or by the default configuration supplied by the runtime. The default TERM-INPUT configuration is specified by the Windows Example, which is described beginning on page 348.)

  **Note 1** The modifiers "\a" (Alt), "\c" (Ctrl), "\g" (AltGr), and "\s" (Shift), are not required before ASCII character values, but are necessary to modify non-character items such as function keys when the modifier is desired. The modifier "\a" (Alt) is actually shorthand for "\c\s" (Ctrl+Shift), the Windows

substitute for the Alt key. The Alt key is trapped by the Windows operating system and is therefore not available to applications. The modifier "\g" (AltGr) is actually shorthand for Alt+Ctrl (but not "\a\c"), the Windows substitute for the AltGr key.

**Note 2** When the characters "a" through "z" (lowercase only) are preceded by "\c" (Ctrl) or either of the modifiers "\a" or "\g", which imply Ctrl, they are converted to 1 through 26 (SOH through SUB). Since the values 1 through 26 are not normally configured as data characters, this means that the configured TERM-INPUT virtual-key code will be used to determine the action. The toolbar button generated virtual-key code is the uppercase equivalent of the letter, that is, "A" through "Z", plus any modifier flags for Ctrl, Alt, or Shift. Thus, "\c" followed by "a" through "z" matches the default Windows configuration for Ctrl+"a" through Ctrl+"z", respectively.

**Table 11: Special Characters for the Button Character-String**

| Special Character | Description |
|---|---|
| >*commandline* | execute *commandline*: |
| \a | Alt |
| \b | Backspace |
| \\ | Backslash character |
| \c | Control |
| \d | Delete |
| \e | Escape |
| \f0 | Function key 10 |
| \f1 | Function key 1 |
| \f2 | Function key 2 |
| \f3 | Function key 3 |
| \f4 | Function key 4 |
| \f5 | Function key 5 |
| \f6 | Function key 6 |
| \f7 | Function key 7 |
| \f8 | Function key 8 |
| \f9 | Function key 9 |
| \fa | Function key 10 |
| \fb | Function key 11 |
| \fc | Function key 12 |
| \fd to \fn | Function key 13 to Function key 23 |
| \g | AltGr (TERM-INPUT: NUL WAGR) |
| \i | Insert |
| \n | Newline |
| \p | Pause (TERM-INPUT: NUL PAUSE) |
| \qa | ATTN (TERM-INPUT: NUL ATTN) |
| \qc | Caps Lock (TERM-INPUT: NUL CAPITAL) |

**Table 11: Special Characters for the Button Character-String (*Cont.*)**

| Special Character | Description |
|---|---|
| \qp | PA1 (TERM-INPUT: NUL PA1) |
| \s | Shift |
| \t | Tab |
| \wa | Applications (TERM-INPUT: NUL APPS) |
| \wc | CRSEL (TERM-INPUT: NUL CRSEL) |
| \we | EREOF (TERM-INPUT: NUL EREOF) |
| \wl | Left Windows Logo (TERM-INPUT: NUL LWIN) |
| \wp | PLAY (TERM-INPUT: NUL PLAY) |
| \wr | Right Windows Logo (TERM-INPUT: NUL RWIN) |
| \wx | EXSEL (TERM-INPUT: NUL EXSEL) |
| \x | Exit program |
| \zb | Begin |
| \zc | Clear |
| \zd | Down Arrow |
| \ze | End |
| \zh | Home |
| \zl | Left Arrow |
| \zm | ZOOM (TERM-INPUT: NUL ZOOM) |
| \zn | Next (Page Down) |
| \zp | Prior (Page Up) |
| \zr | Right Arrow |
| \zs | Scroll Lock (TERM-INPUT: NUL SCROLL) |
| \zu | Up Arrow |
| \z9 | Num Lock (TERM-INPUT: NUL NUMLOCK) |

The string "\g" is used as a modifier corresponding to the AltGr (alternate graphics) key found on many international keyboards. Windows supports the AltGr key with the key combination Alt+Ctrl, which can be entered even on a keyboard that does not have an AltGr key. In a button string, the escape "\g" is normally followed by another escape, such as "\f1", to represent AltGr+F1.

To be effective in a button string, these keys must be configured in the TERM-INPUT records of the configuration. The commonly used keys, such as F1 through F12, are configured by the default Windows configuration, but several of the less common keys such as F13 through F23, CRSEL, EXSEL, PA1, and ZOOM are not configured in the default Windows configuration. (For the keys configured by the default configuration, see the Windows Example (on page 348); additionally, the **windows.cfg** file, which is provided by product installation, also represents the default Windows configuration and has commentary that clarifies which keys are configured.) When configured by TERM-INPUT configuration records, the buttons will activate the configured entry regardless of whether the keyboard actually supports the particular key. The Caps Lock, Num Lock, and Scroll Lock keys can be sent to the application, but do not affect the state of the keyboard, that is, do not toggle the corresponding lock state.

- **Move Up and Move Down.** Use these buttons to control the order of the buttons shown in the toolbar. This order is determined by the order of the names in the Name list box. When you choose Move Up, the currently selected name moves up one position in the list. Choosing the Move Down button moves the selected name down one position.
- **Remove.** Use this button to clear the value for the currently selected toolbar button name and remove it from the list.
- **Remove All.** Use this button to clear all toolbar button values for the currently selected program.

## Setting Menu Bar Properties

The Menu Bar Properties tab, illustrated in Figure 14, allows you to identify a list of pulldown menu names and their associated values for the file that was specified using the Select File tab, as described in Selecting a File to Configure (on page 68).

The C$MBar (on page 552) subprogram also can be used to display a menu bar in the RM/COBOL runtime window.

**Figure 14: Menu Bar Properties Tab**


The Menu Bar Properties tab contains the following options:

- **Name.** The value entered in this text box is the string that appears in the menu bar. If the first character is a tilde (~), the name is disabled. An ampersand (&) character causes the next character to be underlined and used as an accelerator.
- **Prompt.** The value entered in this text box is an optional text string that is displayed when the cursor is placed on the menu bar item.

- **String.** The value entered in this text box defines the items in the pulldown menu along with the strings that are returned to the COBOL program when an item is selected. Using the following syntax, it can specify either a value to be returned or additional sub-menu items:

  ```
  pulldownname["prompt"]=menu
  ```

  *pulldownname* is the string that appears in the menu bar.

  *prompt* is an optional text string that is displayed on the status bar when the cursor is placed on the menu bar item specified by *pulldownname*.

  *menu* defines the items in the pulldown menu along with the strings that are returned to the COBOL program when an item is selected. The syntax for *menu* is shown as follows:

  ```
  menu          -> [(]items[)]
  items         -> item name=[string | (menu)][,items]
  item name     -> pulldownname["menu prompt"]
  string        -> string to be sent (see the Toolbar Properties tab)
  ```

  If the first character of *pulldownname* is a tilde (~), the *menu* is disabled. An ampersand (&) in *pulldownname* causes the next character to be underlined and used as an accelerator.

- **Move Up and Move Down.** Use these buttons to control the order of the pulldown menu names shown in the menu bar. This order is determined by the order of the names in the Name list box. When you choose Move Up, the currently selected name moves up one position in the list. Choosing the Move Down button moves the selected name down one position.

- **Remove.** Use this button to clear the value for the currently selected pulldown menu name and remove it from the list.

- **Remove All.** Use this button to clear all pulldown menu values for the currently selected program.

## Setting Pop-Up Menu Properties

The Pop-up Menu Properties tab, illustrated in Figure 15, allows you to identify a list of pop-up menu names and their associated values that will be displayed when right-clicking the mouse button on an RM/COBOL program in the client area of the screen.

The C$RBMenu (on page 556) subprogram also can be used to display a pop-up menu in the RM/COBOL runtime window when the right mouse button is pressed.

**Note** If you are using RM/Panels, a pop-up menu defined by RM/Panels will override a pop-up menu defined by setting mouse menu properties.

**Figure 15: Pop-up Menu Properties Tab**


The Pop-up Menu Properties tab contains the following options:

- **Name.** The value entered in this text box is the string that appears in the pop-up menu. If the first character is a tilde (~), the name is disabled. An ampersand (&) causes the next character to be underlined and used as an accelerator.
- **Prompt.** The value entered in this text box is an optional text string that is displayed when the cursor is placed on the pop-up menu item.
- **String.** The value entered in this text box defines the items in the pop-up menu along with the strings that are returned to the COBOL program when an item is selected. It can specify either a value to be returned or additional sub-menu items by using the following syntax:

  ```
  pop-upname["prompt"]=menu
  ```

  *pop-upname* is the string that appears in the pop-up menu.

  *prompt* is an optional text string that is displayed on the status bar when the cursor is placed on the pop-up menu item specified by *pop-upname*.

*menu* defines the items in the pop-up menu along with the strings that are returned to the COBOL program when an item is selected. The syntax for *menu* is shown as follows:

```
menu          -> [(]items[)]
items         -> item name=[string | (menu)][,items]
item name     -> pop-upname["menu prompt"]
string        -> string to be sent (see the Toolbar Properties tab)
```

If the first character of *pop-upname* is a tilde (~), the *menu* is disabled. An ampersand (&) in *pop-upname* causes the next character to be underlined and used as an accelerator.

- **Move Up and Move Down.** Use these buttons to control the order of the names shown in the pop-up menu. This order is determined by the order of the names in the Name list box. When you choose Move Up, the currently selected name moves up one position in the list. Choosing the Move Down button moves the selected name down one position.
- **Remove.** Use this button to clear the value for the currently selected pop-up menu name and remove it from the list.
- **Remove All.** Use this button to clear all pop-up menu values for the currently selected program.

# Toolbar Editor

RM/COBOL provides a default toolbar in the file, **rmtbar.vrf**. This toolbar is the default value specified in the Icon File property (on page 75). The buttons provided in the default toolbar are documented in Table 12. A bitmap editor (**rmtbedit.exe**), provided with your RM/COBOL development system, allows you to create or edit the buttons on the toolbar.

**Table 12: Default rmtbar.vrf File Button Icons**

| Button | Description |
|---|---|
| 1 – 39 | Numbers 1 through 39 (useful for menu picks) |
| A – Z | Letters A through Z (useful for menu picks) |
| AF1 – AF23 | Alternate Function keys 1 through 23 |
| AP | Accounts Payable |
| AR | Accounts Receivable |
| BREAK | Hammer smashing object (Break key) |
| CF1 – CF23 | Control Function keys 1 through 23 |
| COMPANION | Two buddies (Companion for RM/COBOL) |
| DISK | Hard disk drive |
| DISKETTE | Floppy disk |
| DOWN | Down Arrow key |
| END | Curtains closing (End key) |

**Table 12: Default rmtbar.vrf File Button Icons (*Cont.*)**

| Button | Description |
|---|---|
| ENTER | Enter key |
| ESCAPE | Escape key |
| EXIT | Door with exit sign |
| F1 – F23 | Function keys 1 through 23 |
| FILE | File cabinet |
| GF1 – GF23 | Alternate Graphics Function keys 1 through 23 (AltGr) |
| GL | General Ledger |
| GO | GO sign |
| GRAPH | Three-dimensional graph |
| GREEN | Green traffic light |
| HELP | Question mark |
| HOME | Little house (Home key) |
| INFO | Italic lowercase i |
| LEFT | Left Arrow key |
| LINELEFT | Left Arrow key pointing at margin bar (Tab left) |
| LINERIGHT | Right Arrow key pointing at margin bar (Tab right) |
| MAIL | Bundle of letters |
| MENU | Menu |
| PAGEDOWN | Down Arrow key pointing at margin bar |
| PAGEUP | Up Arrow key pointing at margin bar |
| PHONE | Telephone |
| PR | Payroll |
| PRINTER | Printer |
| RED | Red traffic light |
| REPORT | Text on computer paper |
| RIGHT | Right Arrow key |
| SAFE | Archive (Safe) |
| SEARCH | Flashlight |
| SF1 – SF23 | Shift Function keys 1 through 23 |
| SGF1 – SGF23 | Shift Alternate Graphics Function keys 1 through 23 |
| STOP | Stop sign |
| TERMINAL | Display and keyboard (Data terminal or PC) |
| UP | Up Arrow key |
| WRITE | Pencil writing on paper |
| YELLOW | Yellow traffic light |
| YIELD | Yield sign |

## Running the Toolbar Editor

To run the Toolbar editor, choose the Toolbar Editor icon. The application presents you with a menu bar. Under the File menu, you can choose a command to open a toolbar file or create a new one. A Resource dialog box then displays the bitmap buttons available in the toolbar file.

**Note** The file created by the Toolbar editor is a resource file that is composed of bitmap buttons, each of which has a name. It is that name that you reference in the Toolbar Properties tab when defining character actions, as described in Setting Toolbar Properties (on page 88).

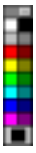
When the Resource dialog box is active, a Resource menu is available. You can edit, delete, copy, and save the bitmap buttons presented in the Resource dialog box. Opening or creating a bitmap Resource dialog box opens a bitmap editor.

## Editing a Bitmap

When you start the bitmap editor, you are in draw mode. When you move the cursor into the editor's grid area, it changes to a pen. You can use the left and right mouse buttons to modify your bitmap. Each button can hold in memory a color that you choose from the color palette. For example, if you click red with the left mouse button and blue with the right mouse button, these colors are stored until you click on another color. By default, when you start the bitmap editor, the left button is black and the right button is gray until you change the color.

In the bottom portion of the color palette, the center square contains the mouse's left button color and the background color is in the mouse's right button color. For example, the color palette in Figure 16 shows the center square to be black (indicating that the color stored in the left mouse button is black) and the background is gray (indicating that the color stored in the right mouse button is gray).

**Figure 16: Color Palette Showing Right and Left Mouse Colors**



Buttons are shown in a pair of frames. The first frame represents the up image of the button. The second frame in the sequence represents the down image of the button.

## Testing the Bitmap

To test a button, choose the Bitmap | Test Button command from the menu bar.

A dialog box appears that shows the bitmap as a button.

## Transferring the Image Up

The bitmap that you create can be duplicated to the down image of the button. Select Bitmap | Transfer Up Image command from the menu bar. At the prompt, choose the Yes button to transfer the image or choose the No button to terminate the transfer.

## Importing and Exporting Bitmaps

You may import a bitmap by choosing the Resource | Import command from the menu bar. This command opens the Import Bitmap dialog box. Enter the name of the **.bmp** file you want to import and choose the OK button.

You may export a bitmap by choosing the Resource | Export command from the menu bar. This command opens the Save Bitmap As dialog. Enter the name of the file you want to export and choose the OK button.

# Character Set Considerations for Windows

This section describes character set considerations for using RM/COBOL under the Windows operating system, including the following topics:



These considerations result from Windows having both an OEM codepage for MS-DOS and an ANSI codepage for Windows. RM/COBOL has historical roots in MS-DOS and, thus, in the OEM codepage, which has resulted in issues caused by the dominance of Windows and its preference for the ANSI codepage.

## Codepages on Windows

Windows has two system codepages: the ANSI codepage and the OEM codepage. A codepage defines a mapping of character code points (often called bytes) to a set of character glyphs. The lower half of all Windows-supported ANSI and OEM codepages, code points 000 – 127 (0x00 – 0x7F), always match each other exactly because they represent the same ASCII character set. The upper half of Windows ANSI and OEM codepages, code points 128 – 255 (0x80 – 0xFF) can differ significantly in the characters that particular code points represent. If you know that your programs do not use code points from the upper half of the codepage, that is, your programs only use and accept ASCII characters, these character set considerations do not affect you. However, if your program does expect to use characters from the upper half of the codepage, that is, extended characters, you need to understand these character set considerations as further described here.

**Note** The acronym “ANSI” actually stands for American National Standards Institute. In RM/COBOL documentation, “ANSI” is usually used with its appropriate meaning. For example, “ANSI COBOL” refers to an implementation of COBOL that follows the American National Standard for the COBOL language and

"ANSI ACCEPT/DISPLAY" refers to the American National Standard Institute's definition of the COBOL ACCEPT and DISPLAY verbs. Such uses have nothing to do with character sets or codepages. Microsoft originally designed the Windows character set following an ANSI standard character set, but then deviated from that standard in their actual implementation. Microsoft documentation, however, continued to use "ANSI" to designate the Windows character set as opposed to the MS-DOS OEM character sets used before Windows. Thus, the term "ANSI codepage" or "ANSI character set" is misleading, but must be used to aid in relating this discussion to Microsoft documentation of character sets on Windows.

Most Windows internal functions interpret code points as being from the ANSI codepage. RM/COBOL was developed and in use much earlier than Windows, so data files written by RM/COBOL under MS-DOS have long existed with OEM code points stored in the files, including files that contain COBOL source programs. Rather than make customers convert their source and data files when Windows was introduced, RM/COBOL continued to consider character data—in files and in memory—as being from the OEM codepage. Thus, conversions from OEM to ANSI or ANSI to OEM take place on RM/COBOL for Windows in the following principal cases:

- When the RM/COBOL runtime system makes calls to Windows functions requiring ANSI code points, the runtime system converts the code points from the OEM codepage to their corresponding code points in the ANSI codepage.
- Most screen and printer fonts have a default script (also called a character set) that interprets code points as being from the ANSI codepage. Thus, when displaying or printing character data to such fonts, the RM/COBOL runtime system converts the in-memory code points from the OEM codepage to the ANSI codepage. (Fonts can support multiple scripts, but the RM/COBOL system currently uses only the default script for a font.)
- Windows delivers data entered from the keyboard to the runtime system with code points from the ANSI codepage. Accordingly, the RM/COBOL runtime system converts the keyed data to the corresponding code points in the OEM codepage to keep the in-memory data consistently OEM. (Note that extended characters can be keyed only when the TERM-ATTR configuration record (on page 327) specifies the keyword DATA-CHARACTERS with a value that allows characters with a code point greater than 126 to be treated as input data characters; otherwise, only ASCII code points 32 – 126 are considered to be valid input data characters.)

Now that Windows has been the dominant operating system for such a long time, customers who use extended characters are having difficulties with the assumption that RM/COBOL character data is from the OEM codepage. They use Windows editors that produce source program files using code points from the upper half of the ANSI codepage. Nonnumeric literal values containing these non-ASCII characters display as expected in the editor, but do not display or print as expected at runtime. This is because the RM/COBOL runtime system assumes that they are code points in the OEM codepage and converts them to the corresponding code points in the ANSI codepage. Since the code points were already from the ANSI codepage, this conversion scrambles the code points in the upper half instead of producing the desired code points. As a result, the extended characters are displayed or printed incorrectly. Also, data entered from the keyboard often undergoes two conversions, one from ANSI to OEM on being keyed, and then from OEM to ANSI on being displayed or printed to a font with a default script that is not OEM/DOS. Since among the extended characters of the two codepages there is not always a matching character, these conversions prevent some characters that can be keyed from displaying or printing as intended by the person entering the characters. The

conversion from ANSI to OEM may substitute a close match such as “Y” (LATIN CAPITAL LETTER Y) for “Ÿ” (LATIN CAPITAL LETTER Y WITH DIAERESIS) or, if there is no close match, a character such as “?” (QUESTION MARK) or “_” (LOW LINE or SPACING UNDERSCORE), which then remains the same when converted from OEM to ANSI since the replacement characters are in the lower common half of the character set. That is, the original ANSI character keyed is not recovered despite the conversion back to ANSI.

## RM/COBOL for ANSI Codepage on Windows

RM/COBOL version 9 provides direct support for using the ANSI codepage in order to assist customers desiring to develop new applications in ANSI mode. Support for the OEM mode is maintained in version 9 and should be used for existing applications.

**WARNING** Great care should be taken to avoid mixing ANSI and OEM code points in any one application or set of application data files, since there is no computable means of undoing the mixing; a human would need to review all the character data to undo the mixed set of code points. If desired, an application can be converted from OEM to ANSI or ANSI to OEM, but the entire application and its entire set of data files must be converted to avoid mixing ANSI and OEM code points in the same application. If two or more applications share a set of data files, all the applications must be converted at the same time.

When RM/COBOL version 9 is installed, it defaults to OEM mode, as was the case before version 9. The command-line option **/cs_ansi** may be specified before the program name to enable ANSI mode. If ANSI mode will be your preferred mode, the **runcobol_ansi.exe** file, installed into the installation directory at install time, may be copied over the **runcobol.exe** file. The compiler can be switched to ANSI mode in a similar manner so that data displayed or printed by the compiler will interpret the code points in the source program correctly. CodeWatch also supports setting the project mode to OEM or ANSI, and the CodeWatch command-line program, **rmcw.exe**, supports the /cs_ansi and /cs_oem command-line options. (Further information about the support for ANSI or OEM native character sets is provided in the *CodeWatch User’s Guide*.) A utility is provided to accomplish switching between a default of OEM and ANSI; the RMSETNCS utility is described beginning on page 101.

In ANSI mode, the compiler, runtime system and CodeWatch development environment assume that code points represent characters from the Windows system ANSI codepage. Thus, a data conversion for character data is required only in that rare situation where a display or printer font is chosen that has a default script of OEM/DOS. In such a situation, the ANSI code points are converted to their corresponding OEM code points before the data is displayed or printed. Also, no conversion is required for keyboard input in ANSI mode since Windows delivers the characters as code points from the ANSI codepage.

**Note** The compiler running in ANSI console mode will not display characters correctly in the console window when the default raster fonts for console windows are used. Use the Console Window Properties dialog box to change the console window font to a True Type font, such as Lucida Console, so that the characters will display correctly.

The C$GetNativeCharset (on page 541) library routine has been provided so that a COBOL program can determine at runtime which character set, ANSI or OEM, is in use as the native character set. The runtime call back table, described in the *CodeBridge User’s Guide*, has also been extended to contain a pNativeCharset

pointer so that non-COBOL programs can determine the native character set used by the calling COBOL program. Note that any single run unit can have only one native character set for the entire duration of that run unit. The native character set for the run unit is established when the run unit is started.

CodeBridge version 9 has been enhanced to allow the native character set of the non-COBOL character data to be declared ANSI or OEM. This information is used in conjunction with the known native character set of the COBOL run unit to provide the appropriate translations for nonnumeric data passed between the COBOL and non-COBOL programs. If the non-COBOL character data is not declared to be from the ANSI or OEM codepage, then no conversion is done. In this case, the non-COBOL character data must either match the native character set of the COBOL run unit or the COBOL program must handle any necessary translations using the C$ConvertAnsiToOem (on page 536) or C$ConvertOemToAnsi (on page 537) library subprograms.

## Installation Character Set Considerations on Windows

When RM/COBOL version 9 is installed on Windows, two client files are installed for starting a COBOL run unit: **runcobol_oem.exe** and **runcobol_ansi.exe**. These two clients differ only in their default native character set, as indicated by their names. The **runcobol_oem.exe** client is also copied to **runcobol.exe** during installation. Thus, the default native character set after installation is OEM when the **runcobol** command is used to start the runtime system. The **runcobol_ansi.exe** file can be copied to the **runcobol.exe** file to change the default native character set to ANSI.

Either client can be started with the **/cs_oem** or **/cs_ansi** command-line option before the main program file name to force the native character set for that run unit to OEM or ANSI, respectively. Alternatively, the **runcobol_oem.exe** or **runcobol_ansi.exe** client may be used to start the run unit.

Similarly for a development system, four client files are installed for starting a COBOL compilation: **rmcobolc_oem.exe**, **rmcobolg_oem.exe**, **rmcobolc_ansi.exe**, and **rmcobolg_ansi.exe**. These correspond, respectively, to the console and GUI compiler clients with a default native character set of OEM, and the console and GUI compiler clients with a default native character set of ANSI.

## *RMSETNCS Utility*

A utility named **rmsetncs.exe** is provided during installation to allow easy switching between the ANSI and OEM default clients, and, for a development system, between the console and GUI compilers. The utility also modifies the CodeWatch INI file **rmcw.ini** in the Windows directory so that new projects will default to the same character set mode as the **runcobol** and **rmcobol** commands.

The RMSETNCS command line is as follows:

```
RMSETNCS  charset-spec  [compiler-mode]
```

where,

```
charset-spec:
       /cs_ansi to select the ANSI character set
       /cs_oem to select the OEM character set

compiler-mode:
       /console to select the console-mode compiler
       /GUI to select the GUI mode compiler
```

The command-line options are case-insensitive per Windows conventions. The options can be specified in either order if both are specified. The charset-spec option is required, but compiler-mode is optional and will default to **/console**. Hyphens can be used instead of slashes to introduce the options, if desired.

The RMSETNCS utility must be run in an RM/COBOL installation folder and assumes that the execution folder is the folder to be modified. That is, the folder to be modified is the folder containing the **rmsetncs.exe** file, which is not necessarily the current directory. For example, when the command is executed with a pathname specified preceding the command, the pathname specifies the installation folder to be modified. Successful execution results in a display of the following lines in a development installation folder for the given command line:

```
[C:\Liant\test\cw1] rmsetncs /cs_ansi /gui
Modifying folder C:\Liant\test\cw1\ --
   setting character set to ANSI;
   setting compiler mode to GUI.
Runtime client runcobol_ANSI.exe copied to runcobol.exe.
Compiler client rmcobolg_ANSI.exe copied to rmcobol.exe.
CodeWatch INI file rmcw.ini file modified.
RMSETNCS modified folder C:\Liant\test\cw1\ successfully
   for a development system.
```

For a runtime-only installation folder, that is, one without a compiler client, the following output would be produced for the given command line:

```
[C:\Liant\test\cw1] rmsetncs /cs_ansi
Modifying folder C:\Liant\test\cw1\ --
   setting character set to ANSI;
   setting compiler mode to Console.
Runtime client runcobol_ANSI.exe copied to runcobol.exe.
Installation path does not contain compiler client rmcobolc_ANSI.exe.
RMSETNCS modified folder C:\Liant\test\cw1\ successfully
   for a runtime-only system (compiler client not found).
```

The runtime client must exist in the execution folder. If it does not, output similar to the following will occur for the given command line:

```
[C:\Liant\test\cw1] rmsetncs /cs_ansi
Modifying folder C:\Liant\test\cw1\ --
   setting character set to ANSI;
   setting compiler mode to Console.
Installation path does not contain runtime client runcobol_ANSI.exe.
RMSETNCS terminated with error.  Be sure utility was run in
   installation folder.
```

The RMSETNCS utility sets the return code (ERRORLEVEL) to zero if successful and one if unsuccessful. The results of running the utility can be checked using the following commands:

`runcobol /showcharset` (for runtime-only or development system)
`rmcobol /showcharset` (for a development system)

Running any runtime or compiler client with just the **/showcharset** command-line option will cause the client to display its native default character set. This is useful when the client has been renamed and it is desired to verify the default native character set. The native character set actually in use is shown in the banner when verbose banners are requested, either with the **–v** runtime command option or the RM_VERBOSE_BANNER=Y environment variable setting. (The native character set actually in use may differ from the default native character set for a client if the /cs_ansi or /cs_oem command-line option has been specified.)

## Related Character Set Configuration on Windows

Several properties and configuration keywords allow modification of how RM/COBOL handles the ANSI and OEM conversions. These are described briefly below, along with how they relate to whether the native character set is ANSI or OEM.

- The Font CharSet OEM property (on page 74) specifies those display font scripts that are considered to be OEM/DOS and thus whether a conversion does not occur when the native character set is OEM; otherwise, a conversion from OEM to ANSI is done as required. If the native character set is ANSI, this property is ignored and a conversion from ANSI to OEM occurs only when the display font script is OEM/DOS.
- The Full OEM To ANSI Conversions property (on page 75) causes additional conversions from OEM to ANSI to occur when the native character set is OEM. These conversions are ones that were missed in earlier implementations of the runtime system for Windows. This property has no effect when the native character set is ANSI, since no OEM to ANSI conversions are needed in this case.
- The Printer Font CharSet OEM property (on page 80) specifies those printer font scripts that are considered to be OEM/DOS and thus whether a conversion does not occur when the native character set is OEM; otherwise, a conversion from OEM to ANSI is done as required. If the native character set is ANSI, this property is ignored and a conversion from ANSI to OEM occurs only when the printer font script is OEM/DOS.

- The ALLOW-EXTENDED-CHARACTERS-IN-FILENAMES keyword in the RUN-FILES-ATTR configuration record (on page 316) determines whether extended characters are allowed in filenames passed from the runtime system to Windows file management functions. If extended characters are allowed, this keyword can further specify whether the characters should be interpreted as ANSI or OEM code points. This keyword should generally be set to the value ANSI when the native character set is ANSI and extended characters are used in filenames. Similarly, it should be set to the value OEM when the native character set is OEM and extended characters are used in filenames.
- The DATA-CHARACTERS keyword in the TERM-ATTR configuration record (on page 327) determines if keyboard input can include extended characters. By default, extended characters cannot be entered from the keyboard.
- The EURO-CODEPOINT-ANSI keyword in the INTERNATIONALIZATION configuration record (on page 309) specifies the code point in the ANSI codepage to use to represent the euro symbol. This can be used to preserve the euro symbol when converting between ANSI and OEM codepages that may not have a euro symbol defined.
- The EURO-CODEPOINT-OEM keyword in the INTERNATIONALIZATION configuration record specifies the code point in the OEM codepage to use to represent the euro symbol. This can be used to preserve the euro symbol when converting between ANSI and OEM codepages that may not have a euro symbol defined.

# Other System Considerations for Windows

This section describes special system considerations for using RM/COBOL under the Windows operating system.

## Memory Available for a COBOL Run Unit on Windows

The memory available for a run unit depends on the configuration of your PC. If the total memory required by a run unit exceeds the amount of available memory, runtime system errors will occur. These errors indicate an inability to obtain enough memory to perform a desired operation. This is unlikely to occur under Windows because 32-bit Windows provides virtual memory. However, it is still possible to use segmentation and subprograms to manage the dynamic memory requirements of very large run units.

## Portable Line Draw Characters

The GRAPHICS keyword of the ACCEPT and/or DISPLAY CONTROL phrase translates the characters described in Table 22: System-Specific Line Draw Characters, described in the topic ACCEPT and DISPLAY Phrases (on page 195), to system-specific line draw characters. Characters that are not listed in this table are output unchanged.

It is not required that the current font contain line draw characters because the runtime system dynamically creates these characters as required.

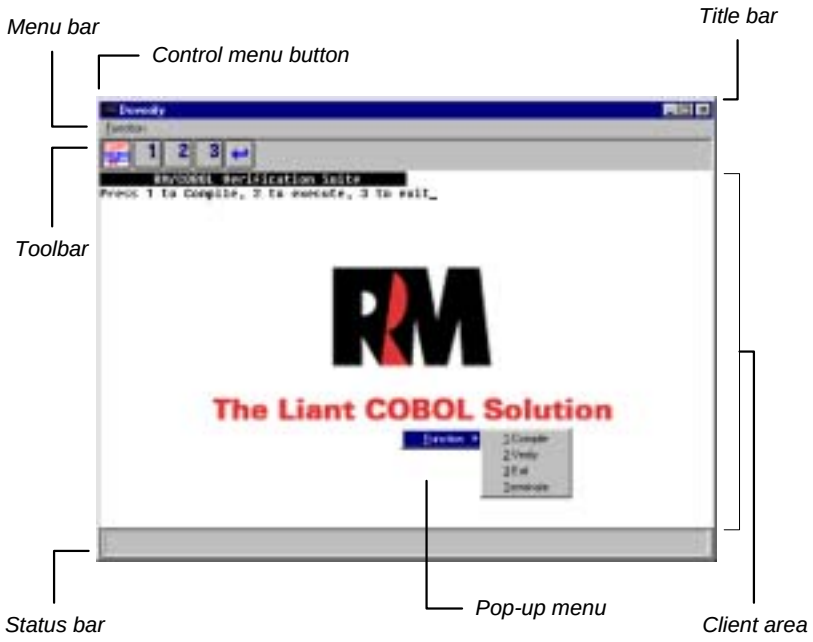
## Blinking Attribute

The blinking attribute is not supported in the Windows environment. See the RUN-ATTR configuration record (on page 313).

## Runtime System Screen

Figure 17 illustrates a sample screen of an RM/COBOL program running under Windows.

**Figure 17: Sample Screen of an RM/COBOL Program Running Under Windows**




The runtime system screen is a typical Windows operating system window with the following areas:

- **Client area.** Used by the RM/COBOL program input and output.
- **Menu bar.** Configurable by the developer. Menu bar can be different for each program. COBOL programs can also display a menu bar by using the C$MBar (on page 552) subprogram.
- **Status bar.** Displays prompt text when the user moves the mouse in the client area, through a menu pick or over a toolbar button. It is configurable by the developer. Status bar can be different for each program. COBOL programs can also display text in the status bar by using the C$SBar (on page 559) subprogram. It can be turned on or off by the user.
- **Control menu button.** Opens the Control menu.
- **Title bar.** Identifies the program-name currently running the COBOL program and displays the Minimize, Maximize, and Close buttons. It is configurable by the developer. Title bar can be different for each program. COBOL programs can also display a title by using the C$Title (on page 575) subprogram.

- **Toolbar.** Configurable by the developer. Toolbar can be different for each program. It can be turned on or off by the user. COBOL programs can also display a toolbar by using the C$TBar (on page 572) subprogram.
- **Pop-up menu.** Configurable by the developer. Pop-up menu can be different for each program. RM/COBOL programs can also change the contents of a pop-up menu by using the C$RBMenu (on page 556) subprogram.

### *Cursor Types*

Under default conditions, there are three types of cursors, each of which indicates a different edit mode during ACCEPT operations.

_ The underscore cursor indicates that standard overtype mode is active.

¦ The full-height cursor indicates that you have typed to the end of the field and that the TAB phrase has been specified in the ACCEPT statement. A backspace key or field termination key is the only valid keystroke in this mode.

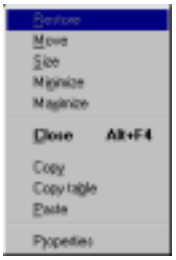
■ The half-height cursor indicates that insert mode is active.

In versions of RM/COBOL prior to 7.5, the cursors were drawn by the RM/COBOL runtime system. In versions 7.5 and higher, the runtime uses the Windows cursor, which is a blinking cursor where the rate at which the cursor blinks is controlled by the Keyboard settings in the Windows Control Panel. The shapes of the three cursors can be configured using three properties in the RM/COBOL Windows registry file: Cursor Overtype property (on page 72), Cursor Insert property (on page 73), and Cursor Full Field property (on page 73). For more information, see the discussion of these properties in Setting Control Properties (on page 70).

## Control Menu Icon

The upper-left corner of the title bar has a button that enables the Control menu (sometimes referred to as System menu). Although the Control menu is standard in the Windows operating system, RM/COBOL for Windows has added functions to this menu. Figure 18 illustrates the RM/COBOL for Windows Control menu.

**Figure 18: RM/COBOL for Windows Control Menu**




The Restore, Move, Size, Minimize, Maximize, and Close commands are standard Control menu functions for the Windows operating system. (For more information, see the Microsoft Windows documentation that accompanied the operating system.) The Copy, Copy table, Paste, and Properties commands have been added to the

Control menu by RM/COBOL for Windows. Each of these commands is described in the following sections.

### *Copy*

Choosing the Copy command from the Control menu copies the text selected in the client area of the RM/COBOL runtime window to the Windows Clipboard. To select text, hold down the mouse button and drag the mouse to the target area. Double-clicking the mouse button selects text in the manner described in the Mark Alphanumeric property (on page 77).

### *Copy table*

Choosing the Copy table command from the Control menu copies the text selected in the client area of the RM/COBOL runtime window to the Windows Clipboard, and also replaces multiple spaces with a tab. This feature is useful in copying a table of numbers to a spreadsheet, since spreadsheets require that number fields be separated by the tab character.

### *Paste*

Choosing the Paste command from the Control menu copies the text in the Windows Clipboard to the currently running RM/COBOL program through the COBOL ACCEPT statement. If more data is pasted than can be accepted by the ACCEPT command, the data is buffered.

### *Properties*

Choosing the Properties command from the Control menu opens the Properties dialog box, which is illustrated in Figure 10 on page 70.

## Return Code Message Box

When **runcobol.exe** terminates with an exit code other than 0, a Return Code message box appears displaying the status code, as shown in Figure 19. For more information, see program exit codes (on page 186). If a COBOL error occurred, that error message is displayed as well. The Show Return Code Dialog property (on page 81) can be used to suppress the display of this message box.

The message box contains two command buttons. The OK button dismisses the message box and closes the application. The Cancel button dismisses the message box only. The application window remains open until you select the Close option from the Control menu. To close the message box, you can click the Close button in the upper-right corner of the window.

**Figure 19: Return Code Message Box**

## CALL “SYSTEM”

When using the SYSTEM (on page 578) non-COBOL subprogram (CALL “SYSTEM”) with DOS programs and batch files, you can customize how these programs run by modifying the MS-DOS Prompt properties. This can be done by right-clicking the mouse on the MS-DOS Prompt icon and selecting Properties from the pop-up menu.

## Performance

For increased file system performance in single-user mode, set the RUN-FILES-ATTR configuration record option to FORCE-USER-MODE=SINGLE (see page 318).

## Using Large Files on Windows

RM/COBOL supports files larger than 2 gigabytes (GB), but not on all versions of Windows and not on all Windows file systems. In addition, even if a particular version of Windows and a particular version of the Windows file system allow local files larger than 2 GB, this does not guarantee that all other machines in a peer-to-peer network can successfully access the large file. The following information describes the conditions under which applications can count on large file support in various Windows environments.

### *Windows File Systems Considerations*

Microsoft provides several different Windows file systems.

Windows 95 operating systems prior to the release of Windows 95 OEM Service Release 2 (OSR2), version number 4.00.950B, support only the File Allocation Table (FAT) file system, which limits files to no more than 2 GB. The Windows 9x class of operating systems (excluding Windows 95 without OSR2) included an updated version of the File Allocation Table file system, called FAT32. This updated file system allows support for files larger than 2 GB, but not larger than 4 GB. Windows 98 and Windows Me support both the FAT (2 GB) and the FAT32 (4 GB) file system.

Although the FAT32 file system supports local files up to 4 GB, Liant Software has determined that Windows 95 does not support access to files larger than 2 GB from remote clients. Attempts to create files larger than 2 GB on a Windows 95 FAT32 file system and to access the file from another machine may result in a hung client when the RM/COBOL runtime attempts the WRITE operation that would cause the file to grow past 2 GB. Everything will work correctly until the attempt to exceed the 2 GB boundary.

While Windows NT-class operating systems do not support the FAT32 file system, they do support the NTFS file system, which allows multiple terabyte (TB) files.

In addition to these file systems considerations for the different versions of the Windows operating system, there are also other variants of the Windows operating systems. In particular, there have been several Service Pack updates for Windows NT 4.0. Liant Software recommends that Windows NT 4.0 Servers be upgraded to at least Service Pack 6. Microsoft generally provides downloadable updates for system modules from their web site between updates.

## Large File Locking Issues

Very large files, defined as RM/COBOL indexed files larger than 2 GB, and RM/COBOL relative and sequential files larger than 1 GB, require the use of the LARGE-FILE-LOCK-LIMIT keyword (see page 318) of the RUN-FILES-ATTR configuration record to specify a lock limit larger than 2 GB. The Define Indexed File (rmdefinx) utility (on page 594) may be used to set the Large File Lock Limit for version 3 indexed files. The lock limit may not be set to more than 4 GB unless the RM/COBOL runtime is running on a Windows NT-class operating system and the file resides on an NTFS file system.

For indexed files, the block size and the value of the lock limit determine how large the indexed file can be. For example, with a 4 GB lock limit, a block size of 1024 will allow a 3.2 GB indexed file and a block size of 4096 will allow a 3.7 GB indexed file. For relative and sequential files, the file size may be no more than one half of the lock limit. Thus, a sequential file may be no more than 2 GB when the lock limit is 4 GB.

Using very large files also requires that the Windows system support region locking at the value specified by the Large File Lock Limit. All Windows systems appear to be able to lock at 4 GB (above 4 GB in the case of a Windows NT-class operating system), but remote access to very large files requires that the network redirector (on the client machine) and the File and Printer Sharing Network Service (on the server machine) also support such locks.

Liant Software has determined that the “File and printer sharing for NetWare Networks” network service on the Windows 9x class of operating systems does not support locking above 2 GB. Applications that require very large files should use the “File and printer sharing for Microsoft Networks” network service on the machine on which the very large file resides (that is, the server, not the client). For further information about the problem with “File and printer sharing for NetWare Networks” service and information about how to switch to “File and printer sharing for Microsoft Networks” service, see Appendix K: *Troubleshooting RM/COBOL* (on page 655).

## Test Programs Available

In order to help the RM/COBOL applications developer who needs to use files larger than 2 GB in a Windows environment, Liant Software has developed some simple C programs which attempt to answer the question of how various Windows systems react to the use of very large files. These programs and any additional information discovered after the release of this product may be found on the Liant Software web site at http://www.liant.com.

Because the Windows environment is very complex with regard to the use of very large files, Liant strongly recommends that the applications developer use these test programs to determine whether it is possible to use very large files in the required Windows environment. Failure to do this testing may result in unfortunate surprises (for example, when the file grows larger than 2 GB) long after the application has been deployed at a customer site. Periodically, Liant will add additional information to the web site. If your application requires very large files, continue to check the web site often for updates.

It is also possible to use the RM/COBOL runtime system to write a test indexed file of the desired size to verify that your application will not have problems with a specific Windows environment. This technique is particularly useful when running in a Windows peer-to-peer environment.

# Environment Variables for Windows

An environment variable is an operating system feature that allows a value to be equated with a name. Table 13 lists those environment variables that are used by RM/COBOL on Windows.

**Note** In addition to the environment variables listed in the following table, RM/COBOL uses environment variables to map generic file access names, as explained in File Access Names on Windows (on page 63).

**Table 13: Environment Variables for Windows**

| Environment Variable | Usage |
|---|---|
| COMSPEC | SYSTEM subprogram (on page 578). |
| GROUP | C$GetSysInfo subprogram (on page 545). |
| GROUPID | C$GetSysInfo subprogram (on page 545). |
| NAME | C$GetSysInfo subprogram (on page 545). |
| PATH | Locating files (see page 61). |
| PRINTER | Printer support (see page 227). |
| RMPATH | Locating files (see page 61). |
| RM_DEVELOPMENT_MODE | C$SetDevelopmentMode subprogram (on page 567). |
| RM_DYNAMIC_LIBRARY_TRACE | Tracing support module loads (see page 432). |
| RM_LOAD_WOW_CLIENT | Loading the WOW Extensions support module, rpcpluswow.dll. |
| RM_LIBRARY_SUBDIR | Locating optional support modules (on page 431). |
| RM_VERBOSE_BANNER | Compile command messages (on page 166) and runcobol banner message (on page 396). |
| RM_Y2K | COMPILER-OPTIONS ALLOW-DATE-TIME-OVERRIDE (see page 287). |
| RUNPATH | Locating files (see page 61). |
| STATION | C$GetSysInfo subprogram (on page 545). |
| TEMP or TMP | Temporary files (on page 239). |
| TZ | Standard C TimeZone variable. |
| USER | C$GetSysInfo subprogram (on page 545). |
| USERID | C$GetSysInfo subprogram (on page 545). |

# Chapter 4: System Considerations for Btrieve

This chapter describes special considerations for using RM/COBOL to access Btrieve files. Btrieve files are an alternative indexed file format to the RM/COBOL indexed file format. Btrieve files can reside on the local machine, in which case they are accessed via client-based Btrieve, or they can reside on a remote machine, in which case they are accessed via server-based Btrieve. The RM/COBOL-to-Btrieve Adapter program (**rmbtrv32**) provides the communication between the RM/COBOL runtime and Btrieve runtime, translating COBOL requests to Btrieve requests.

## RM/COBOL-to-Btrieve Adapter Program Concepts

The RM/COBOL-to-Btrieve Adapter program, **rmbtrv32**, improves performance by providing a mechanism to reduce the overhead required to transmit requests for records in an indexed file across a local area network (LAN).

The goal of the **rmbtrv32** program is to use the local area network for passing general requests to other machines and for receiving completed requests back from the other machines. As a result, significant increases may occur in the performance of the application program, the cost-effectiveness of the local area network, and the productivity of the user.

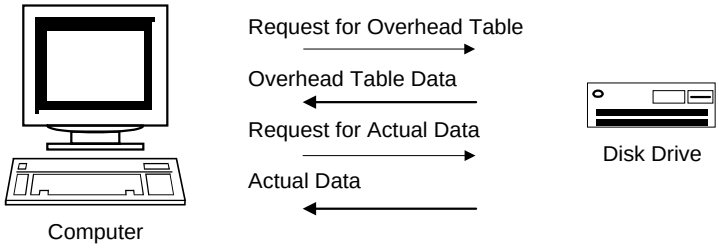
**Note** See RM/COBOL versus Btrieve Indexed File Performance (on page 117) for a situation in which the performance of Btrieve index files may not exceed that of RM/COBOL indexed files.

### Indexed Files

The application program can request a specific record of information in an indexed file. The location of the specified record within the indexed file is determined by means of an identifier known as a key. Indexed files use a much more efficient method of locating the record than simply searching through all the records in the file until the requested record is found. Instead, indexed files build overhead tables into the file that are similar to indexes in a book. These overhead tables enable the indexed files to quickly look up the desired location and then read the desired data. Figure 20 illustrates this process on a single-user system.

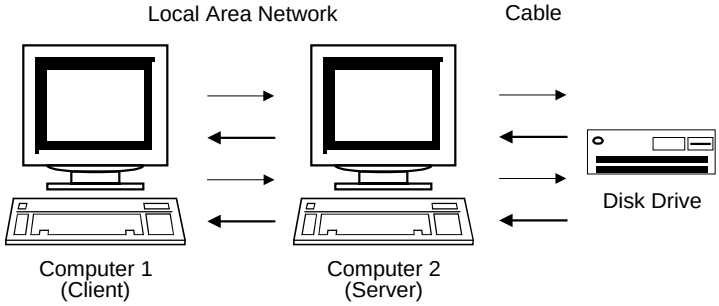
**Note** In Figure 20, Figure 21, and Figure 22, each line represents a separate event that happens at a separate time. The lighter lines represent a small transfer of information, and the heavier lines represent a large transfer.

**Figure 20: Indexed File Requests on a Single-User System**




When this process happens over a network, the situation is very similar, as shown in Figure 21.

**Figure 21: Indexed File Requests on a Local Area Network**




In Figure 21, Computer 2 acts as a conduit, called a server, through which the requests of Computer 1, called a client, are routed. (The server routes requests for more than one client computer, which is an advantage of local area networks.) A more effective way to route requests, however, is shown in Figure 22.

**Figure 22: Indexed File Requests on a Local Area Network by the Btrieve MicroKernel Database Engine (MKDE)**

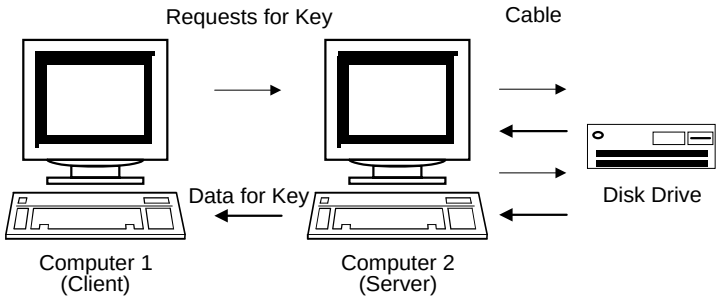



Figure 22 illustrates the way in which a Btrieve requester (running on the client, Computer 1) and a Btrieve MicroKernel Database Engine (running on the server, Computer 2), makes the processing of messages even more efficient. (Note that the Btrieve MicroKernel Database Engine is a key external component of the **rmbtrv32**

program.) Although the interactions between Computer 2 and the disk drive are the same as shown in Figure 21, the interactions between Computer 1 and Computer 2 are significantly different. Instead of Computer 1 giving Computer 2 many small instructions to carry out, Computer 1 now gives Computer 2 a single, general request. Computer 2 searches the overhead table for the indexed files to locate the desired record and then returns only the requested record.

There are several advantages to this method, but the following two are the most significant:

1. The overall operation may be quicker because the number of transfers between the two computers is reduced.
2. Because there are fewer transfers between the Computer 1 and Computer 2, the local area network can use the time that it is not performing transfers between the two computers to make transfers between other computers on the network. It allows the network to handle more computers, which makes it more cost effective.

# Required Software Components

The following components, described in the following sections, are required when using RM/COBOL to access Btrieve files:

- Novell NetWare version 3.11 or later
- Btrieve MicroKernel Database Engine (MKDE) for NetWare Server, version 6.15 or later
- Btrieve Requester for 32-bit Windows
- RM/COBOL compiler (development system) for Windows
- RM/COBOL runtime system for Windows
- RM/COBOL-to-Btrieve Adapter program for Windows

**Note** NetWare products are available from Novell, Incorporated. Btrieve products are available from Pervasive Software Inc. (formerly Btrieve Technologies Inc.).

## Novell NetWare

NetWare is the software that communicates between computers on the local area network. These NetWare products are responsible for handling the actual hardware connections, recovering from transmission errors detected by the hardware, and routing the messages from one program executing on one computer to another program executing on another computer.

NetWare augments the operating system by providing access to files on file servers.

## Btrieve MicroKernel Database Engine (MKDE)

The MKDE component consists of two types. The first type, a client-based Btrieve MKDE, provides access to files that are located on the same machine as the application program. The second type, NetWare Btrieve MKDE, provides access to files that are located on a remote machine in a multi-user environment.

The NetWare Btrieve MKDE is a record management system similar to the indexed files built into the RM/COBOL runtime system. Because the NetWare Btrieve MKDE is not built into the RM/COBOL runtime system, it can run on a separate computer using NetWare, thus providing access to files in the manner illustrated in Figure 22 on page 112).

There are also versions of the Btrieve MKDE that run on other types of networks and on a single machine (client-based Btrieve MKDE), without network support. The client-based Btrieve MKDE, however, no longer has the speed advantage over the RM/COBOL file management system, since both systems have the same access to the disk drive.

## Btrieve Requester for 32-Bit Windows

The 32-bit Windows requester, a dynamic-link library (DLL) program, runs on the client computer and communicates with either the server-based or the client-based Btrieve MKDE.

## RM/COBOL Compiler for Windows

The RM/COBOL compiler (development system) is a GSA-certified high implementation of the American National Standard COBOL X3.23-1985 with extensions and support for most optional features of the language.

## RM/COBOL Runtime System for Windows

The RM/COBOL runtime system executes the application program and carries out its instructions. The runtime system has an internal file management system that accepts input from the user, processes data, produces data in the form of output to the user, and, most importantly, generates requests for records to be written to and read from files.

The runtime system has been designed so that any existing RM/COBOL application may be run in many different environments without changes either to the source of the program or to the actual executable object. Furthermore, any existing RM/COBOL runtime system that executes on Windows can also use the RM/COBOL-to-Btrieve Adapter (**rmbtrv32**) program.

## RM/COBOL-to-Btrieve Adapter Program for Windows

The **rmbtrv32** program acts as an interpreter between either of the two types of Btrieve MKDEs, which are described in Btrieve MicroKernel Database Engine (MKDE) (on page 114), and COBOL application programs. In order to understand how this transparent interface is achieved, it is necessary to briefly describe the

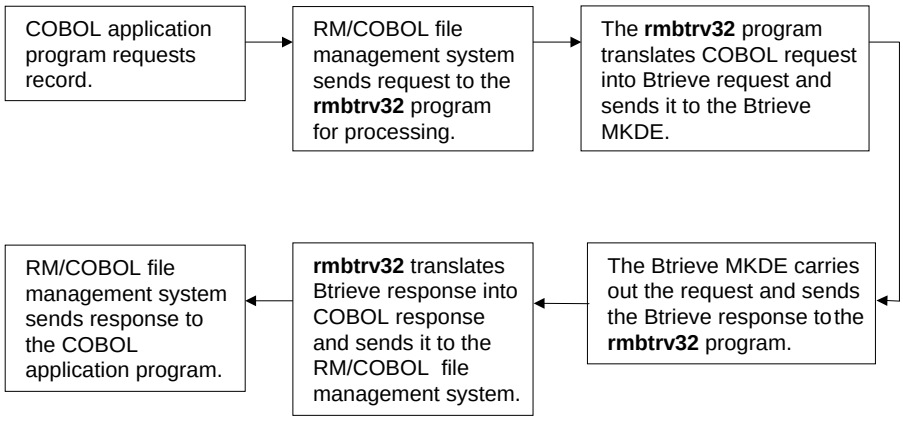
different ways in which the Btrieve MKDE and the COBOL language provide access to indexed files.

The Btrieve MKDE lets an application program access records stored in indexed files, and provides the necessary functions for storing, retrieving, and updating the information. The Btrieve MKDE's method of accessing indexed files is an efficient system that provides significant increases in functionality to the user in certain cases. However, because the Btrieve MKDE does not use COBOL language features that provide access to indexed files, a COBOL application program cannot communicate directly with the Btrieve MKDE.

A COBOL application program uses American National Standard COBOL 1985 language features, such as OPEN, READ, WRITE, REWRITE, and CLOSE, to access indexed files. The RM/COBOL runtime system contains a file management system that provides the runtime system with support for these features. The RM/COBOL runtime system communicates with the file management system by means of requests and responses that are called messages. These messages are processed outside of the file management system by any one of a variety of external file access methods.

The **rmbtrv32** program, in effect, is one such external file access method for the RM/COBOL runtime system. **rmbtrv32** receives messages from the RM/COBOL file management system. Then, acting as an application program for the Btrieve MKDE, **rmbtrv32** translates the messages into Btrieve requests, enabling the Btrieve MKDE to carry out the action originally requested by the COBOL application program. The Btrieve MKDE performs the action either on the user's computer system or acts with NetWare on a remote system using the local area network. (The drive letter in the pathname of the file indicates the machine on which the file resides.) After the Btrieve MKDE has completed the requests, **rmbtrv32** constructs an appropriate response message, which is sent to the RM/COBOL file management system, and, finally, back to the COBOL application program. Figure 23 illustrates this process.

**Figure 23: RM/COBOL-to-Btrieve Adapter Program (rmbtrv32) Acting as an External File Access Method**




See also RM/COBOL-to-Btrieve Adapter Program Options (on page 117).

# Configuration for Btrieve

The installation and configuration of client-based Btrieve (also called Workstation Btrieve) for 32-bit Windows are fully described in the appropriate Btrieve installation and operation manual supplied by Pervasive Software with your Btrieve system. The client-based Btrieve is the MicroKernel Database Engine (MKDE) that is used to access local files (that is, Btrieve files residing on the computer where the RM/COBOL runtime system is run). A number of configuration settings can be modified using the Btrieve Setup utility. After configuring Btrieve, use the Btrieve File Manager utility (or other Btrieve software) to verify that Btrieve is working properly before using Btrieve with RM/COBOL.

Similarly, the installation and configuration of server-based Btrieve (for NetWare or for a Windows NT-class Server) are fully described in the appropriate Btrieve installation and operation manual that was supplied by Pervasive Software with your Btrieve system. These manuals also describe the installation and configuration of the requesters used to communicate with server-based Btrieve. The server-based Btrieve is the MKDE that is used to access remote files (that is, Btrieve files residing on the NetWare or on a Windows NT-class Server). A number of configuration settings for both the MKDE and the requesters can be modified by using the appropriate Btrieve Setup utility.

The *Btrieve Programmer's Guide*, supplied by Pervasive Software with your Btrieve Developer Kit, is an excellent source of information for help in setting the Btrieve configuration options properly. In addition, several books on Btrieve are available commercially, and the *Btrieve Developer's Journal* is published quarterly by Smithware, Inc.

## System Considerations for Btrieve Files

The RM/COBOL-to-Btrieve Adapter program (**rmbtrv32**) creates Btrieve files when necessary or if requested. Btrieve files created by **rmbtrv32** have a computed page size based on one of the following methods that produces the largest value:

1. The size of the block requested by the COBOL application.
2. The size necessary for the length of the longest key, times eight.
3. The size of the largest record requested by the application, plus eight times the number of linked duplicate keys, plus six (for overhead information), plus four if the file specifies variable-length records (again for overhead information). For more information, see Variable-Length Records (on page 129).

Furthermore, if the record size is greater than the maximum page size and the keys of the file all fit into that maximum, the **rmbtrv32** program creates a variable-length file. (The Btrieve MKDE restricts the fixed-length part of records to less than the page size.)

Finally, **rmbtrv32** creates the file with the following characteristics:

- Data compression
- Blank truncation
- Five-percent, free-space threshold
- No page preallocation

To create Btrieve files with characteristics other than those previously listed, use the Btrieve File Manager utility, the filename, and the Btrieve description-file that contains the characteristics for the new file. For more information, see the chapter about using the File Manager utility in the appropriate Btrieve installation and operation manual. Characteristics established using the Btrieve File Manager utility could have a direct impact on performance, including the following:

- The page preallocation value can be used to reserve pages for use by the file. This has the advantage of ensuring, in advance, that the file has the disk space it needs. It can also improve performance by concentrating the location of the file on the disk media (assuming that the disk space is not already fragmented).
- The free-space threshold value can be set to 10, 20, or 30 percent to allow for growth of variable-length records.
- Keys can be created that are binary or have any of the extended key types.
- Null keys can be created.
- More keys can be defined than can be used by the COBOL program. These keys must be defined either at starting locations that are different from the COBOL keys or after the COBOL key description for the same location. Such keys can have any Btrieve attribute and can be split.

## RM/COBOL versus Btrieve Indexed File Performance

In general, when used across the network, Btrieve indexed files have better performance than RM/COBOL indexed files because less network activity has to occur to access a record.

However, this may not be true when a COBOL program opens an indexed file WITH LOCK. In this case, the COBOL program then has exclusive access to that file. This has an important consequence for RM/COBOL indexed files. In this case, the RM/COBOL runtime system knows that no other user is able to change the indexed file overhead tables on the server, and it keeps the overhead tables on the local machine. This results in fewer requests across the network for the overhead tables and may result in better performance than the same program using Btrieve indexed files.

This effect is most pronounced when the indexed file is being read sequentially (for example, producing a report).

# RM/COBOL-to-Btrieve Adapter Program Options

The RM/COBOL-to-Btrieve Adapter program, **rmbtrv32**, has options that are specified on the EXTERNAL-ACCESS-METHOD configuration record (on page 308) or on the RUN-INDEX-FILES configuration record (on page 320) in the RM/COBOL configuration file. These configuration file options, described in the following sections, give **rmbtrv32** information that the Btrieve MKDE requires, but which is not contained in RM/COBOL file management system messages.

**Note** Typically when configuring the Btrieve MKDE, it is often sufficient to specify only the “Largest Compressed Record Size” Btrieve configuration option if you are

using compression (see the appropriate Btrieve installation and operation manual for more details).

## EXTERNAL-ACCESS-METHOD Configuration Record Options

Most of the information that the **rmbtrv32** program needs to operate can be obtained through requests received from the RM/COBOL file management system. However, when **rmbtrv32** needs information required by the Btrieve MKDE, which the RM/COBOL file management system cannot supply, it is possible to provide this information directly to **rmbtrv32** with options in the EXTERNAL-ACCESS-METHOD configuration record.

These options are as follows:

- B (**rmbtrv32** Btrieve MKDE page size) option
- Create option
- D (duplicates) option
- I (initial display) option
- L (lock) option
- M (mode) option
- O (owner) option
- P (**rmbtrv32** page size) option
- T (diagnostic trace filename) option

These options are described in the following sections.

**Note** The create option is specified by the CREATE-FILES keyword in the EXTERNAL-ACCESS-METHOD configuration record. The other options (B, D, I, L, M, O, P, and T) are specified by the OPTIONS keyword in the EXTERNAL-ACCESS-METHOD configuration record (on page 308).

### *B (rmbtrv32 Btrieve MKDE Page Size) Option*

This option is obsolete and should not be specified. The “Maximum Page Size” is no longer a configurable parameter of the Btrieve engine, which always assumes the Btrieve limit of 4096 bytes. If this value were inadvertently specified as an amount smaller than 4096, **rmbtrv32** may create a Btrieve file with variable-length records when such records would not be needed.

### *Create Option*

The create option, for creating a new file, has the following values:

- Yes Create new files as Btrieve indexed files (the default).
- No Do not create new files.

See the description of the CREATE-FILES keyword in the EXTERNAL-ACCESS-METHOD configuration record (on page 308).

The create option is the determinant parameter supplied to the **rmbtrv32** program, because it determines the system that will be responsible for creating a new indexed file. Depending on the value specified in this parameter, the new file can be created by **rmbtrv32**, by another external file access method, or by the RM/COBOL file management system. In order to understand how this process works, it is helpful to know more about the way in which the RM/COBOL file management system searches for a file.

Before an application program creates a file, the RM/COBOL file management system first tries to locate an existing file having the same name as the one specified in the create attempt. The file management system searches the current directory first, and then all the other directories located in the environment variable, RUNPATH. See Directory Search Sequences on Windows (on page 61) for more information on setting the RUNPATH variable.

In addition to the **rmbtrv32** program, other external file access methods can be running on the computer or network at the same time. In searching for a file, the RM/COBOL file management system also communicates with all other known external file access methods.

The search for the filename occurs in the following sequence:

1. Any external file access methods currently running (including **rmbtrv32**) search the current directory.
2. The RM/COBOL file management system searches the current directory.
3. The external file access methods search the first directory in the RUNPATH list.
4. The RM/COBOL file management system searches the first directory in the RUNPATH list.

The search continues until all pertinent directories have been checked. If a file having the same name as the one specified in the create attempt is found, it will be opened. If such a file cannot be found, and the application program wants to create one, then a designated external file access method can create the file.

The **rmbtrv32** program create option value is a yes or no indicator that specifies whether you want **rmbtrv32** to create any new indexed files as Btrieve files. Regardless of the value specified, any new file is created in the first directory possible, usually the current directory. Valid values are Yes and No. The default value is Yes.

A value of Yes causes any new indexed files to be created as Btrieve files:

```
EXTERNAL-ACCESS-METHOD NAME=RMBTRV32 CREATE-FILES=YES
```

A value of No causes **rmbtrv32** not to create the file and enables another external file access method or the RM/COBOL file management system to create new indexed files:

```
EXTERNAL-ACCESS-METHOD NAME=RMBTRV32 CREATE-FILES=NO
```

## D (Duplicates) Option

The duplicates option is used to specify whether linked or repeating duplicatable keys are used for files created by **rmbtrv32**.

The duplicates option has the following values:

- L Create linked duplicatable keys. Linked duplicates mean that only one copy of the duplicated key value is stored in index pages. The data records with the duplicated key value are linked together with pointers in a doubly linked list.
- R Create repeating duplicatable keys. Repeating duplicates mean that the duplicated key value is repeated in the index pages for each data record with that value. The data records are not linked together. Using repeating duplicates uses more space in index pages, but saves space in data pages and also helps avoid position lost errors when files are shared.

The default value is L. Refer to the *Btrieve Programmer's Guide* for more information.

The following example tells **rmbtrv32** to create files with repeating duplicatable keys:

```
EXTERNAL-ACCESS-METHOD NAME=RMBTRV32 OPTIONS='D=R'
```

## I (Initial Display) Option

The initial display option is used to specify whether **rmbtrv32** should display an initial message box when it is first invoked.

The initial display option has the following values:

- Y (Yes) Display the message box. The message box shows the **rmbtrv32** version number and the OPTIONS parameter string that was passed to it from the EXTERNAL-ACCESS-METHOD configuration record. The user must click the OK button to acknowledge and continue. This option is most useful the first time the user attempts to use **rmbtrv32** with RM/COBOL and Btrieve.

  **Note** I=Y should not be used in a production environment.

- N (No) Do not display the message box.

The default value is N.

### Example

The following example tells **rmbtrv32** to display the informative message box:

```
EXTERNAL-ACCESS-METHOD NAME=RMBTRV32 OPTIONS='I=Y'
```

## L (Lock) Option

The lock option is used to specify the manner in which the **rmbtrv32** program is to handle the WITH LOCK phrase on OPEN statements.

The lock option has the following values:

- I Ignore the WITH LOCK phrase. Use the Btrieve MKDE open mode indicated with the M (mode) option (see page 121).
- D Deny the WITH LOCK phrase.
- A Accept the WITH LOCK phrase. If OPEN WITH LOCK is requested by the application, ignore the open mode indicated with the M (mode) option, described on page 121.

The default value is A.

### Examples

The following example tells **rmbtrv32** to ignore the WITH LOCK phrase on OPEN statements:

```
EXTERNAL-ACCESS-METHOD NAME=RMBTRV32 OPTIONS='L=I'
```

The following example tells **rmbtrv32** to deny the WITH LOCK phrase on OPEN statements:

```
EXTERNAL-ACCESS-METHOD NAME=RMBTRV32 OPTIONS='L=D'
```

The following example tells **rmbtrv32** to accept the WITH LOCK phrase on OPEN statements:

```
EXTERNAL-ACCESS-METHOD NAME=RMBTRV32 OPTIONS='L=A'
```

## M (Mode) Option

The mode option is used to specify a value to **rmbtrv32** at the time a Btrieve file is opened. The following values are used only if the file is not OPENed WITH LOCK. The mode option has the following values:

- N Normal
- A Accelerated
- R Read-only
- V Verify
- E Exclusive

The default value is N.

**Note** The ability of **rmbtrv32** to specify a mode value is dependent on whether the application program requests the WITH LOCK phrase on OPEN statements. See the L (lock) option on page 121 for more information.

### Examples

In normal mode, the Btrieve MKDE behaves as it normally does with its recovery option enabled, allowing update requests and performing normal writes to the disk drive. The following example specifies a value of normal when the file is opened:

```
EXTERNAL-ACCESS-METHOD NAME=RMBTRV32 OPTIONS='M=N'
```

In accelerated mode, the data recovery capability of the Btrieve MKDE is disabled to increase the speed at which records are updated. The following example specifies a value of accelerated when the file is opened:

```
EXTERNAL-ACCESS-METHOD NAME=RMBTRV32 OPTIONS='M=A'
```

In read-only mode, no updates can be performed. The following example specifies a value of read-only when the file is opened:

```
EXTERNAL-ACCESS-METHOD NAME=RMBTRV32 OPTIONS='M=R'
```

Verify mode is now disregarded and the MKDE assumes normal mode instead.

In exclusive mode, the user has exclusive access to the file until the user closes it. This is the same as specifying EXCLUSIVE or WITH LOCK on the OPEN statement in the COBOL program.

```
EXTERNAL-ACCESS-METHOD NAME=RMBTRV32 OPTIONS='M=E'
```

## O (Owner) Option

The owner option specifies the “owner” ID (actually a security password) for new files and open requests for existing files. The value is a string of up to a maximum of eight characters delimited by a trailing space. The value cannot contain spaces. The following example specifies an owner ID of YELLOW:

```
EXTERNAL-ACCESS-METHOD NAME=RMBTRV32 OPTIONS='O=YELLOW'
```

## P (rmbtrv32 Page Size) Option

The **rmbtrv32** page size option is the default minimum page size for the files created by **rmbtrv32**. Btrieve files are physically accessed in fixed-length pieces called pages. When **rmbtrv32** creates a new file, the Btrieve MKDE requires the specification of a page size. The size of a page is determined from either the page size option or a computation based on the size of the record. For more information, see Variable-Length Records (on page 129). A larger page size transfers more data in a single disk request, requires more time to transfer, and requires more memory to contain the pages. A smaller page size allows more blocks in memory for a fixed amount of memory, but requires more time to randomly access a record by increasing the tree depth of each index for the file.

If specified, the value must be a multiple of 512 in the range of 512 to 4096, inclusive. When creating a file, the page size used will be the smallest multiple of 512 sufficient to hold the file overhead, eight keys, the fixed part of the record, or, if specified, the default page size, whichever is greater.

The following example sets the value of the page size option to 1024:

```
EXTERNAL-ACCESS-METHOD NAME=RMBTRV32 OPTIONS='P=1024'
```

## *T (Diagnostic Trace Filename) Option*

This diagnostic trace filename option is used to specify the pathname of a file to which **rmbtrv32** will write a trace of open requests. This feature is used when there is a problem with a Btrieve file not being successfully opened by a COBOL program. It is not to be used in a production environment, because it degrades performance and the trace file can become quite large, which might exhaust disk space. To turn on the trace feature, edit the RM/COBOL configuration file for the COBOL program in question and add a T=*trace-file-name* parameter to the OPTIONS keyword in the EXTERNAL-ACCESS-METHOD configuration record (on page 308).

For example, the following record writes trace information to the file **c:\test\trace.fil**:

```
EXTERNAL-ACCESS-METHOD NAME=RMBTRV32
OPTIONS='T=C:\test\trace.fil'
```

The trace file contains a “Begin open” and “End open” pair of records for every open request that **rmbtrv32** receives. This includes all opens that **runcobol** does for files, such as the COBOL program file, as well as every OPEN statement executed by the COBOL program. The “End open” line shows the COBOL status code returned to the RM/COBOL file management system. Between the “Begin” and “End” lines, zero or more “BTRV Create” or “BTRV Open” lines show the full pathname of the file, the exact Btrieve status code returned by the name of the file, and the exact Btrieve status code returned by the Windows Btrieve DLLs. The following is a sample trace file:

```
Trace Initialized
Begin open, Not indexed
End open, Code=35
Begin open, Not indexed
End open, Code=35
Begin open, Not indexed
End open, Code=35
Begin open, Flags=0x4900 (file must exist)
 UFN=INX1
  BTRV Open status 0 on file C:\TEST\INX1
End open, Code=0
Begin open, Flags=0xe100 (file must exist)
 UFN=INX2
  BTRV Open status 12 on file C:\TEST\INX2
End open, Code=35
Begin open, Flags=0xe000
 UFN=INX2
  BTRV Open status 12 on file C:\TEST\INX2
  BTRV Create status 0 on file C:\TEST\INX2
End open, Code=0
```

When you are finished diagnosing the problem, be sure to edit the configuration file again and remove the T=*trace-file-name* parameter from the OPTIONS keyword in the EXTERNAL-ACCESS-METHOD configuration record.

## RUN-INDEX-FILES Configuration Record Options

In addition to the options specified on the EXTERNAL-ACCESS-METHOD, two RUN-INDEX-FILES keywords have meaning for **rmbtrv32**: DATA-COMPRESSION and BLOCK-SIZE. For more information, see RUN-INDEX-FILES configuration record (on page 320).

Specifying DATA-COMPRESSION=NO causes **rmbtrv32** to create uncompressed Btrieve files. The default is to create compressed Btrieve files. (Note that Btrieve does not support key compression.)

Specifying BLOCK-SIZE=*nnnn* causes **rmbtrv32** to create files with a page size of *nnnn*. **rmbtrv32** first computes the minimum allowable page size for the file based on the record size, number of key segments, type of duplicates, and so forth. It then uses the first value greater than or equal to the computed minimum value in the following order:

1. From the BLOCK CONTAINS clause in the program's file description entry.
2. From the P=<page size> option parameter on the OPTIONS keyword in the EXTERNAL-ACCESS-METHOD configuration record (on page 308).
3. From the RUN-INDEX-FILES BLOCK-SIZE=<size> configuration record.

If none of these three values is present or acceptable, **rmbtrv32** uses the computed minimum value.

### Example

The following example represents a typical command line invoking **runcobol** using **rmbtrv32**:

```
runcobol userprog x=config.cfg
```

where the **config.cfg** file contains the following records:

```
RUN-INDEX-FILES DATA-COMPRESSION=NO
EXTERNAL-ACCESS-METHOD NAME=RMBTRV32
& CREATE-FILES=YES
& OPTIONS='P=1024, D=R, O=XyZzY'
```

The ampersand (&), which begins the third and fourth lines in this example, is the configuration file record continuation character. Note that different RM/COBOL applications can specify different **rmbtrv32** option parameters by using different RM/COBOL configuration files.

# Starting the RM/COBOL-to-Btrieve Adapter Program for Windows

The RM/COBOL-to-Btrieve Adapter program, **rmbtrv32**, and either the client-based Btrieve MKDE or 32-bit Windows Btrieve requester programs are all started automatically. This process is initiated by the user placing the following configuration record in the RM/COBOL configuration file and starting the RM/COBOL runtime system:

```
EXTERNAL-ACCESS-METHOD NAME=RMBTRV32
```

The only requirement is that Windows must be able to locate the various executable files that are required.

**Note** The server-based Btrieve MKDE must be started separately. Refer to the appropriate Btrieve installation and operation manual for information on starting server-based Btrieve.

The **rmbtrv32** program, **rmbtrv32.dll**, is a dynamic-link library (DLL) that can be loaded by the 32-bit Windows version of RM/COBOL. **rmbtrv32.dll** communicates directly with **wbtrv32.dll**, which is the Btrieve interface DLL supplied with your Btrieve system. The **wbtrv32.dll** file is normally installed, along with a number of other DLL, EXE, and other Btrieve files, in a separate Btrieve executable subdirectory.

Since RM/COBOL and Btrieve are separate products supplied by separate vendors, the executable files required by each are normally installed in the separate directory structures. Therefore, the recommended way of ensuring that Windows can locate the files is to place the directory names containing the files into the Windows PATH environment variable. For RM/COBOL, this is the directory containing **rmbtrv32.dll** (and also containing **runcobol.exe**, and so forth). For Btrieve, this is the directory containing **wbtrv32.dll** (and other DLLs and EXEs). Add these two directory names to your Windows PATH (which is often done in the **autoexec.bat** file).

Although it is not the recommended method, Windows will find the executable files if they reside in any combination of the following:

1. The directory that contains the **runcobol.exe** that is started.
2. The current directory.
3. The Windows system directory (normally **c:\windows\system**).
4. The main Windows directory (normally **c:\windows**).
5. Any directory in the PATH environment variable

**Note** Both the Btrieve MKDEs and **rmbtrv32.dll** have keywords that can be passed to them when they are started. If no parameters are specified, the programs use default values.

For information on specifying keywords, see the EXTERNAL-ACCESS-METHOD configuration record (on page 308). For more information on **rmbtrv32** options, see RM/COBOL-to-Btrieve Adapter Program Options (on page 117).

The **rmbtrv32** program, the 32-bit Windows Btrieve requester, and the client-based Btrieve MKDE all terminate automatically when the final RM/COBOL runtime system using them terminates. Server-based Btrieve must be terminated separately;

however, server-based Btrieve normally remains running as long as the server computer remains running.

# RM/COBOL Indexed Files and Btrieve MicroKernel Database Engine (MKDE) Limitations

This section describes the limitations of the Btrieve MicroKernel Database Engine (MKDE), and the way in which these limitations affect RM/COBOL indexed files. Although these two systems perform the same functions, they do not operate in the same manner.

**Note** For more information on the RM/COBOL limits and ranges, see Appendix B: *Limits and Ranges* (on page 399).

## Current Record Position Limitations

A COBOL application program can sequentially read through all the records in an indexed file. The manner in which a requested record is presented to the application program for the READ operation varies, depending on how the file was created. The Btrieve MicroKernel Database Engine (MKDE) behaves according to ANSI COBOL 1985 for simple READ statements.

However, for READ NEXT statements, the behavior of the Btrieve MKDE can vary from ANSI COBOL 1985. According to ANSI COBOL 1985, the determination of the next record to be read is not affected by subsequent non-READ operations. As long as the COBOL application program does not perform any non-READ operations to the indexed file, the Btrieve MKDE behaves according to ANSI COBOL 1985. If non-READ operations are performed to the file, however, the Btrieve MKDE defines the next record as being the one after the non-READ operation.

The RM/COBOL-to-Btrieve Adapter program, **rmbtrv32**, compensates for this variation by remembering the location of the record that was last read, and the surrounding records, in order to set the position indicator back to the correct place following the non-READ operation. This compensation works completely when a single-user is accessing the file, but can fail in a shared file environment.

In a Btrieve shared file environment, **rmbtrv32** can become lost when reading records via a key containing duplicate key values. If the COBOL application program performs a non-READ operation between a READ and a READ NEXT statement, and an application program running on the same or another computer deletes the current record and the records around it (and all these records contain duplicate key values), then **rmbtrv32** becomes lost and returns an error message 46, 02 to the application program. See Input/Output Errors for more information.

This position-lost problem can be avoided when the shared Btrieve file is accessed via a version 6.1 or later Btrieve MKDE. **rmbtrv32** sets the new No Currency Change (NCC) option on Insert and Update operations so that Btrieve will not change the current record position. In addition, the user can specify the use of repeating duplicatable keys (see the D (Duplicates) option on page 120 of this user's guide and the discussion of Linked versus Repeating Duplicatable keys in the *Btrieve Programmer's Guide*). Using both of these features avoids the position lost problem and retrieves the correct record.

## File Position Indicator Limitations

The file position indicator specifies the next record to access within a file during certain sequences of input-output operations.

If the COBOL program executes a START LESS THAN statement and there are multiple records in the file that contain duplicate keys (for example, multiple records having the same key value that satisfy the START LESS THAN condition), then the file position indicator will be positioned to the last record in the sequence of duplicate key values. The same result occurs if you execute a START LESS THAN OR EQUAL statement where the equal condition is not met.

If no new records containing duplicates for a key value are added to the file, then **rmbtrv32** behaves identically to the RM/COBOL file management system for the succeeding READ NEXT or READ PREVIOUS statements. The RM/COBOL file management system does not move the file position indicator from the record originally located by the START statement. This position is the record returned for succeeding READ NEXT or READ PREVIOUS statements.

The Btrieve MKDE does not allow **rmbtrv32** to emulate this behavior if new records are added that contain duplicates for a key value. **rmbtrv32** moves the file position indicator to the last record added at the time of the succeeding READ NEXT or READ PREVIOUS statement.

**Note** Once the READ statement has been executed, the position is known, and the RM/COBOL file management system and the Btrieve MKDE again behave the same.

## Permission Error Detection Limitations

When you attempt to open an RM/COBOL indexed file and **rmbtrv32** is active, **rmbtrv32** may open the file before the RM/COBOL file management system opens the file. If the indexed file is already opened by the RM/COBOL file management system on another computer, the Btrieve MKDE returns a Permission Error to **rmbtrv32** instead of a Not a Btrieve File error. **rmbtrv32** assumes that the file is an already-opened RM/COBOL indexed file and reports an Invalid Organization error to the file management system, which then attempts to open the file. If the file is an RM/COBOL indexed file, the open succeeds. If the problem was one with permissions, then the RM/COBOL file management system encounters it also and returns the correct error code.

## Using Existing Btrieve Files with RM/COBOL

RM/COBOL and ANSI COBOL 1985 define some limitations on indexed files that are not imposed by the Btrieve MKDE.

**rmbtrv32** creates new Btrieve files that are compatible with the COBOL concept of indexed files. Existing Btrieve files can be used also, providing they have the following characteristics:

- The primary key cannot have a null value.
- Alternate keys can be modified, can use either the native or alternate collating sequence (ACS), can be binary, and can have a null value.
- Keys cannot have the following Btrieve key flags: descending, supplemental, and any-segment null. Keys must use ACS number zero, if any.

- Keys do not have to be created in the file in any particular order. However, within the file, there must be at least one key residing at the correct position for each COBOL key. That key must be of the correct length, contain the correct duplicates flag, and cannot contain any of the restrictions on keys as described above. Furthermore, this key must be defined within the Btrieve file before any other keys that start at the same position. Subsequent keys may have forbidden characteristics.
- There can be more keys in the Btrieve file than in the COBOL description, and they can have characteristics that are not legal for COBOL keys. However, they must either have a starting position that does not match the starting position for any COBOL key, or they must occur in the Btrieve definition after the COBOL key description for that position.
- Within the record, there should not be any multiple-byte integer data fields. **rmbtrv32** will not reject any files with fields of this type. Because of byte ordering, however, there are no COBOL data types that can directly manipulate the integer data in the field.

If an OPEN OUTPUT is performed on an existing Btrieve file, all characteristics of the original file are preserved. This includes any compression (or lack of it) and any extra keys. The file is simply made empty.

## Btrieve MicroKernel Database Engine (MKDE) Limitations Affecting RM/COBOL Applications

The Btrieve MKDE has limitations that may affect existing COBOL applications:

- Version 5 Btrieve files have a maximum record size of 55296 bytes. Version 6 Btrieve files support record sizes of 64 KB or more using “chunk” operations. **rmbtrv32** does not use any Btrieve “chunk” operations; therefore, the maximum record size is limited by the communication environment in which Btrieve runs. When accessing server-based Btrieve (remote files), the maximum record size is 57000 bytes. When accessing client-based Btrieve (local files), the maximum record size is 64512 bytes. The largest possible record size without using variable-length records is 4088 bytes. RM/COBOL files have a maximum record size of 65280 bytes.
- Btrieve files have a minimum record length of four bytes. RM/COBOL files have a minimum record length of one byte. **rmbtrv32** supports files whose record size is less than four bytes by using a zero-filled, four-byte record.
- Btrieve files must have all keys located within the first 4096 bytes of a record. RM/COBOL files may have keys located anywhere within the record.
- Btrieve files have a limit of 119 key segments. RM/COBOL files have a limit of 255 key segments.
- Btrieve files have a maximum key size of 255 bytes. The RM/COBOL runtime system (including **rmbtrv32**), however, supports a maximum key size of 254 bytes.

## Variable-Length Records

RM/COBOL will support variable-length records using the Btrieve MKDE's variable-length record files. The size of the Btrieve data page will be either the minimum record length or the maximum Btrieve MKDE page size, whichever is smaller.

For more details, refer to the discussion of variable-length records, logical and physical record lengths, and page sizes in the *Btrieve Programmer's Guide*.

## Key Placement

The Btrieve MKDE restricts placement of keys within the first data page of a record. If a file has variable-length records, the keys must fit within the minimum record length of the file or the maximum Btrieve MKDE page size, whichever is smaller.

## Automatic Creation of Variable-Length Record Files

If a COBOL program creates a file with a record size greater than the maximum Btrieve page size, and the keys of that file fit within the maximum Btrieve page size, then the file will be created with a record size equal to the maximum Btrieve page size, with the remainder of the record in the variable-length portion of the Btrieve record. (The Btrieve MKDE allows the portion of the record past the fixed length to be considerably longer.)

## Verification of Maximum Record and Block Length

Btrieve files do not have a mechanism for storing the maximum record length and maximum block length for a file. If a Btrieve file is opened with a maximum length for its RECORD CONTAINS or BLOCK CONTAINS clause that does not match the maximum length at the time the file was created, the mismatch will not be detected.

## Support for RM/COBOL Internal Data Formats

The Btrieve MKDE internally stores integers in Intel binary integer format, with the most significant byte at the highest address and the least significant byte at the lowest address. Therefore, applications that access Btrieve files written outside of RM/COBOL cannot directly access the following three RM/COBOL internal data formats since they store numbers in the opposite manner (as binary integers with the most significant byte at the lowest address and the least significant byte at the highest address):

- BINARY data
- COMPUTATIONAL-1 data
- COMPUTATIONAL-4 data

For more information about RM/COBOL internal data formats, refer to Appendix C: *Internal Data Formats* (on page 403).

## Support for Btrieve Internal Data Formats

RM/COBOL programs can directly access the following Btrieve internal data formats:

- Decimal
- Money
- Numeric Signed ASCII
- Numeric Signed Trailing Separate
- String

All other Btrieve internal data formats can be interpreted by an RM/COBOL program on a byte-by-byte basis. For more information about Btrieve internal data formats, refer to "Extended Key Types" in the *Btrieve Programmer's Guide*.

## Input/Output Errors in Btrieve

Input/output errors that you would expect to occur for an RM/COBOL indexed file may not occur for a Btrieve file. Because of its file structure and organization, information in Btrieve files is stored differently than in RM/COBOL indexed files, or it is not recorded at all. Thus, the RM/COBOL runtime system is unable to check or verify certain values in these files.

For example, the error message 39, 01, which normally occurs if an error is encountered when the runtime system is trying to open an RM/COBOL indexed file, may not occur if the file is a Btrieve file. Appendix A: *Runtime Messages* (on page 357) provides more information on this and other specific input/output error codes where this behavior can occur.

# Chapter 5: System Verification

A suite of verification programs is provided with RM/COBOL. These programs ensure that your delivered diskettes are undamaged and that you have installed the required software correctly.

## System Verification for UNIX

To invoke compilation and execution of the verification suite, enter:

```
doverify
```

For runtime-only systems, the compilation step is ignored.

The verification suite is designed to be self-explanatory. Follow the prompts on the screen for instructions on selecting and running individual tests.

**Note 1** If a problem occurs with the display features of the verification suite, make sure you have properly set and exported the environment variable TERM for your terminal type. If you have done this and a problem exists, verify that your terminal type has an entry in the system terminal database (termcap or terminfo) and check the accuracy of the cursor motion sequence. This can be accomplished by running the system visual editor (vi).

**Note 2** If any of the menu selections within the terminal configuration test work incorrectly, refer to Chapter 8: *RM/COBOL Features* (on page 187) for details on the terminal attributes required by the runtime system for complete ACCEPT and DISPLAY functionality.

## Single-User Tests

Six sets of tests are provided in the verification suite for single-user versions. They are as follows:

1. **Terminal Configuration Test.** This consists of individual tests that verify the functionality of ACCEPT and DISPLAY statements and defaults, screen editing functions and color functions (for appropriately configured terminals).
2. **File System Test.** This tests the sequential, relative, and indexed file system. It reads and writes records to each of the three file types.
3. **Nucleus Test.** This tests the modules of the RM/COBOL nucleus.
4. **Printer Test.** This tests that the printer configuration is correct and that communication from RM/COBOL to the printer is successful. If no printer is attached, preserve the test result in the **prntst.out** file by entering:

   ```
   PRINTER=prntst.out; export PRINTER
   ```

   before running this test.
5. **Sort-Merge Test.** This tests the sort-merge function of RM/COBOL.
6. **Pop-Up Window Manager Test.** This tests the RM/COBOL Pop-Up Window Manager feature. The program displays a self-explanatory menu that allows you to test the various features of the Pop-Up Window Manager system.

## Multi-User Test

An additional verification program is provided with an appropriately licensed, multi-user version of RM/COBOL. This test ensures that RM/COBOL is interacting correctly with the file protection mechanisms employed by your system.

The program **pacetest** needs to be run simultaneously from at least two terminals that use the RM/COBOL runtime system on the computer. This test creates and uses two indexed files: **pinx** and **pinxfl**. **pinx** is a file that contains a single control record, which contains the next available record in **pinxfl**. **pinxfl** is the working data file.

**pacetest** reads the control record, saves it, increments the control record, and writes it back to **pinx**. The original number read is used as the primary key for the record written to the growing **pinxfl** file. This scenario is repeated 100 times for each user running the test. With two users running, 200 records would be written to **pinxfl**. The directions for running **pacetest** are as follows:

1. Invoke **pacetest** at each terminal as follows:

   ```
   runcobol pacetest
   ```

2. At one terminal only, choose function 1 to create the initial files. Wait for this operation to complete.
3. At each terminal, choose function 2 to do simultaneous write operations. You will be required to enter a unique station ID (1, 2, and so forth). Then, all stations should try to press the Enter key simultaneously.
4. At any of the terminals, choose function 3 to read the file. Make sure all the records are shown in order and are all there (100 records are written by each station).

If **pacetest** encounters any errors while reading the file, it will sound an alarm and display an error message next to the record in question. Should this occur, double-check your software installation. If everything appears to be set up correctly and you are still having problems, contact Liant technical support services.

# System Verification for Windows

To invoke compilation and execution of the verification suite, choose the icon named doverify. The **doverify** program allows the user to select compilation and/or execution. For runtime-only systems, the program detects that the compiler is not present and informs the user. The user may still select the execution option.

The verification suite is designed to be self-explanatory. Follow the prompts on the screen for instructions on selecting and running individual tests.

## Single-User Tests

Six sets of tests are provided in the verification suite for single-user versions. They are as follows:

1. **Terminal Configuration Test.** This consists of individual tests that verify the functionality of ACCEPT and DISPLAY statements and defaults, screen editing functions and color functions (for appropriately configured terminals).
2. **File System Test.** This tests the sequential, relative and indexed file system. It reads and writes records to each of the three file types.
3. **Nucleus Test.** This tests the modules of the RM/COBOL nucleus.
4. **Printer Test.** This tests that the printer configuration is correct and that communication from RM/COBOL to the printer is successful. If no printer is attached, preserve the test result in the **prntst.out** file by setting a synonym before running this test. Using the Synonyms Properties tab, type PRINTER in the Name text box and type **prntst.out** in the Value text box. For more information, see Setting Synonym Properties (on page 85).
5. **Sort-Merge Test.** This tests the sort-merge function of RM/COBOL.
6. **The Pop-Up Window Manager Test.** This program displays a self-explanatory menu that allows you to test the various features of the Pop-Up Window Manager.

## Multi-User Test

An additional verification program is provided with an appropriately licensed, multi-user version of RM/COBOL. This test ensures that RM/COBOL is interacting correctly in a network environment.

The program **pacetest** needs to be run simultaneously from each computer that uses the RM/COBOL runtime system on the network. This test creates and uses two indexed files: **pinx** and **pinxfl**. **pinx** is a file that contains a single control record, which contains the next available record in **pinxfl**. **pinxfl** is the working data file.

**pacetest** reads the control record, saves it, increments the control record, and rewrites it back to **pinx**. The original number read is used as the primary key for the record written to the growing **pinxfl** file. This scenario is repeated 100 times for each user running the test. With two users running, 200 records would be written to **pinxfl**. The directions for running **pacetest** are as follows:

1. Compile **pacetest** by choosing the RMCOBOL icon and selecting **pacetest.cbl** as the source file.
2. Invoke **pacetest** at each computer by choosing the RUNCOBOL icon and selecting **pacetest.cob** as the object file.
3. At one computer only, choose function 1 to create the initial files. Wait for this to complete.
4. At each computer, choose function 2 to do simultaneous write operations. You will be required to enter a unique station ID (1, 2, and so forth). Then, all stations should try to press the Enter key simultaneously.
5. At any of the computers, choose function 3 to read the file. Make sure all the records are shown in order and are all there (100 records are written by each station).

If **pacetest** encounters any errors while reading the file, it will sound an alarm and display an error message next to the record in question. Should this occur, double-check your network installation. If everything appears to be set up correctly and you are still having problems, contact Liant technical support services.

# Chapter 6: Compiling

RM/COBOL programs are compiled with a single pass of the RM/COBOL compiler. Specifically, the compiler performs the following actions on the contents of the source programs:

- Verifies syntactic accuracy.
- Creates object programs for execution with the RM/COBOL runtime system. Liant's use of this technique provides compactness and machine-independence.
- Creates program listings, the contents of which are chosen by entering the appropriate Compile Command options.

This chapter presents complete information about the RM/COBOL compiler.

## Compilation Process

Once invoked, the compiler makes one pass through the specified source file. During this pass, both object files and listing files are generated. The RM/COBOL compiler is invoked when you enter the Compile Command, **rmcobol**. The object file contains the machine-independent object code, executed at runtime, for the RM/COBOL program. The listing file contains a source image, which may be printed at the end of each compilation. Using the available Compile Command options, you can alter, augment, and suppress portions of the information contained in the listing.

RM/COBOL provides standard COBOL subprogram structure, but no intermediate linkage process stands between program compilation and execution. It is also possible to define sections of code within your program as overlay segments to the fixed permanent segment, as explained in the discussion of segmentation in Chapter 5: *Procedure Division* of the *RM/COBOL Language Reference Manual*.

**Note** By default, on Windows the RM/COBOL GUI compiler window disappears immediately when a successful compilation completes. If you want the window to remain visible, set the Persistent property (on page 78) to True or use the console mode compiler.

# System Files

RM/COBOL takes its input from a source file, and creates an object file and a listing file.

## Source Files

RM/COBOL source files contain the RM/COBOL source code. Source lines are made up of variable-length records. Source text is ASCII, with either a line feed (LF) character or a carriage return (CR) and line feed (LF) character paired as the line separator. Embedded tab characters are expanded to one or more spaces, according to the default tab column position, which is every fourth column, starting with column 8 and ending with column 72, or according to the value of the TAB-STOPS keyword (see page 326) in the RUN-SEQ-FILES configuration record.

## Object Files

An object file is created on disk as a purely binary file. Its filename is identical to the filename of the source file, with the filename extension **.cob** or **.COB** or the extension specified in the EXTENSION-NAMES configuration record (on page 308).

You can direct the object file to a directory other than the one on which the source file resides. To do this, use the O Compile Command Option. The object file may be suppressed by the use of the N Option.

## Listing Files

The contents of RM/COBOL listings are detailed in the topic Listing (on page 152). Listings can be directed to a disk file, the printer, the screen, or any combination thereof, depending on the options selected in the Compile Command. Listing files are given the filename of the source program, with the filename extension **.lst** or **.LST** or the extension specified in the EXTENSION-NAMES configuration record (on page 308). The listing file is a printer file and, therefore, may be configured using the PRINT-ATTR configuration record (on page 311).

# Libraries

A source file can contain more than one source program. Files containing a sequence of two or more programs are referred to in this manual as libraries. With libraries, the generated object file contains a distinct object module for each source program in the source file, excluding contained programs. The object for a contained program is considered part of the object of the program that contains it. The listing file contains a complete listing of each source program in the source file.

Each noncontained program in a source file or library is compiled strictly independent of the other programs: there need be no relationship between them. However, this capability to concatenate multiple source programs into a single library is used most effectively when there is some logical relationship among the programs. This might be a main program and the called subprograms, or all the

programs that include a specific copy file or group of copy files. In the latter case, recompilation of all the source programs affected by a change in one of the copy files can be accomplished with a single invocation of the Compile Command.

**Note** RM/COBOL versions 1 and 2 did not require END PROGRAM headers to separate a sequence of source programs. Versions 3 and later support nested programs, which make END PROGRAM headers necessary. If you have a source file with a sequence of programs and no END PROGRAM headers, you can either add the headers or specify the Z=2 Compile Command Option (described on 149).

Use the Combine Program (rmpgmcom) utility (on page 583) to combine multiple object files into a single library when the source modules are contained in separate files.

# Compile Command

Use the Compile Command (**rmcobol**) to request program compilation and to specify options.

Under UNIX, the Compile Command is entered at a shell prompt. After typing the desired command and options, press Enter to begin compilation.

Under Windows, the Compile Command can be entered in the Command line text box of the Create Shortcut dialog box. For instructions, see Creating a Windows Shortcut (on page 58). Choose the RMCOBOL icon to begin compilation. Programs also may be executed by dragging the **.cbl** source file to the RMCOBOL object or by double-clicking the source file.

The format of the Compile Command is as follows:

```
rmcobol filename [[(]  [[~]option] ...  [)comment]]
```

*filename* is the name of the source file to be compiled. It may be any valid operating system pathname, and may by partially or fully qualified. Specifying an extension is optional, but that extension must not be the same as the object file extension (**.cob** or **.COB** unless configured otherwise). If you do not enter a filename extension with the pathname, the compiler begins its search for the source file by looking first for the file exactly as specified. If it cannot find such a file, it looks for a file with the supplied name and an extension **.cbl**. If the file is still not found when running under UNIX, it looks for a file with an extension of **.CBL**. For all attempts to open the source file, if neither a drive designator nor a directory path is specified, the directory search sequence is used. If a directory path is specified, a directory search sequence may be used if configured properly. See the discussions of Directory Search Sequences on UNIX (on page 24) and Directory Search Sequences on Windows (on page 61), and the EXPANDED-PATH-SEARCH keyword (on page 317) in the RUN-FILES-ATTR configuration record.

~ (tilde) can be used as a negation character. Its purpose is to negate the presence of attributes in a COMPILER-OPTIONS configuration record (on page 287). Its use is described in Compile Command Options.

*option* specifies the RM/COBOL compiler options, described in the next section. Spaces or commas must separate options. Options may be entered in either

uppercase or lowercase letters. If an option is repeated in a command, the last occurrence of the option is used. Each option may be preceded by a hyphen. If any option is preceded by a hyphen, then a leading hyphen must precede all options. When a value is assigned to an option, the equal sign is optional if leading hyphens are used.

*comment* is used to annotate the command. The comment is ignored by the compiler and has no effect on the compilation. The left parenthesis is always optional. The right parenthesis is a required separator if comments are entered. Under UNIX, the parenthesis must be preceded with a backslash (\) character in order to be protected from the shell.

Up to 54 characters of the *filename* specified in the Compile Command are copied into the "Source file:" field of the listing header. Up to 110 characters of *options* and *comment* from the Compile Command are copied into the "Options:" field within the listing header. The options will also include options specified in the registry (on Windows) or resource files (on UNIX). Thus, this information is reproduced at the top of each listing page. See Figure 24 on page 153 for an example of a listing header.

In addition, the RM/COBOL for Windows compiler also supports the following command-line options, which do not follow the command format described earlier in this section:

- Three OLE server registration commands. These options are described in Compiler Registration (on page 54).

```
rmcobol /regserver
rmcobol /unregserver
rmcobol /showserver
```

- Three character-set commands. These options are described in Character Set Considerations for Windows (on page 97).

```
rmcobol /cs_ansi
rmcobol /cs_oem
rmcobol /showcharset
```

## Batch Compilation on Windows

For Windows, the RM/COBOL compiler can be run as a console application using the **rmcobolc** command or as a GUI application using the **rmcobolg** command. Copying or renaming either of these executables to **rmcobol** can be done to choose the default method of compilation. The two compilers support the same Compile Command options and produce the same results. The console application compiler runs in a console window (an MS-DOS Prompt window or "DOS box" on Windows 9x-class operating systems). The GUI compiler runs in a standard graphical Windows window.

The console application is particularly useful for batch compilations using a command script, for example, in a batch command file. The GUI compiler can also be used for batch compilations, but in this case, the Windows start command with the wait option should be used as follows:

```
start /wait rmcobolg [options]
```

This command causes the script to wait until the compilation completes before executing the next command in the script; otherwise, the next command in the script is executed in parallel, which can lead to problems such as script errors if the next command attempts to access files produced by the compilation or too many parallel compilations. When using the GUI compiler in batch mode, it is often desirable to set the Main Window Type property to Hidden for compiling so that GUI windows are not flashed on the display screen as each file is compiled. For more information, see the Main Window Type property (on page 77).

## Multiple File Compilation on Windows

On Windows, both the console and GUI mode compilers support specifying a question mark ("?") for the *filename* Compile Command parameter. In this case, a File Open dialog box is displayed for selecting the file or files to be compiled. Multiple files may be selected by using the Ctrl or Shift keys in the standard Windows manner for multiple selections. Compilation stops after all the files are compiled or when any single compilation returns a non-zero exit code. Each compilation uses the same Compile Command options that were specified with the "?" for the *filename*. For example, the Compile Command:

```
rmcobol ? L A X
```

would display the file open dialog box and then compile all the selected files with a listing file (the L Option) and the list file will contain an allocation map (the A Option) and a cross reference listing (the X Option). The "?" is not quoted in the Compile Command because quotes cause the question mark to be interpreted as a wildcard character, as described in the following paragraph.

In addition to supporting multiple file selection, the Compile Command on Windows supports wildcard characters in the *filename* parameter. The wildcard characters are asterisk ("*") and question mark ("?"). An asterisk represents zero or more of any character whereas the question mark represents zero or one of any character. Hidden files, system files, offline files, directories, and reparse points are ignored. For example, the Compile Command:

```
rmcobol *.cbl Y=3 L
```

compiles all regular files in the current directory that have an extension of **.cbl**. The Y=3 Compile Command Option is set for each of the compilations. Compilations stop when all the indicated files have been compiled or when any single compilation returns a non-zero exit code. As another example, the Compile Command:

```
rmcobol \\server\src\???.cbl L
```

compiles all regular files in the directory **\\server\src** that have filenames one to three characters and an extension of **.cbl**. The L Compile Command Option is set for each compilation. More than one asterisk can be used. For example, the Compile Command

```
rmcobol *ar*.cbl L X
```

compiles all regular files in the current directory that have "ar" somewhere in the filename and an extension of **.cbl**. The L and X Compile Command Options are set for each compilation. A single question mark ("?" without the quotes) is interpreted as meaning that a File Open dialog box should be displayed instead of a wildcard character. If the question mark is quoted or is part of a pathname, it is interpreted as a wildcard character. For example, either of the Compile Commands:

```
rmcobol "?" L
rmcobol .\? L
```

compile all regular files that have a single character name in the current directory. A File Open dialog box is not displayed in either of these cases. Quotes are required if the filename contains spaces, regardless of whether wildcard characters are used or not used. The wildcard characters are only permitted in the final edgename of the filename. For example, the Compile Commands:

```
rmcobol ?\test.cbl
rmcobol *\test.cbl
```

will cause an open error because the path portion of the filename contains a wildcard character. The open error will occur regardless of the existence of a file named "**test.cbl**" in a subdirectory of the current directory. If the list of files that match a filename containing wildcard characters is empty, the compiler attempts to open the given filename. Since Windows prohibits the wildcard characters in filenames, this will normally result in an open error. See "Open error for file *pathname*" on page 173 for information about the open error message that is displayed. (The expansion of wildcard characters is accomplished using the Windows FindFirstFile and FindNextFile functions.)

When multiple files are compiled either by selecting multiple files after specifying a "?" for the filename on the command line or by using wildcard characters in the filename on the command line, the multiple files are compiled as if the user had entered a sequence of command lines with the selected filenames and the same set of Compile Command options specified in the original command line. The "?" or filename with wildcard characters in it is not used as a registry key for looking up properties set for a particular program. Instead, for each selected program, the properties set for that program are used. For information on setting default properties and program-specific properties in the registry, see Setting Properties (on page 68).

## Compile Command Options

Compile Command options can be specified in the following three ways:

1. They can be placed into the registry (on Windows) or the resource files (on UNIX). In the registry, the Command Line Options property (on page 72) provides command-line options for the compiler when Compiler is selected on the Select File tab of the RM/COBOL Properties dialog box. In resource files, the Options keyword, which is described in Command-Line Options (on page 29), provides command-line options for the compiler in the global resource file **/etc/default/rmcobolrc** and the local resource file **~/.rmcobolrc**.
2. They can be specified in the Compile Command itself.
3. They can be placed into a configuration file, which is processed by the RM/COBOL compiler when the configuration file is automatically located, attached, or specified with a configuration command-line option. For information on configuration files, see Automatic and Attached Configuration Files (on page 282) or Configuration Compile Command Options (on page 142). For a discussion of the compiler options that can be configured, see the COMPILER-OPTIONS configuration record (on page 287).

Options are processed in the order given above, but options specified in the configuration do not override options specified in the resultant set of command-line options as determined from items 1 and 2 above. This means that options specified in a Compile Command will take precedence over conflicting or contradictory options specified by the registry or resource files (item 1) or configuration (item 3). The configured options, together with the options that appear in a Compile Command, apply to every source program in the source file (or, on Windows, files) specified in that Compile Command.

You can override specific options in a configuration file by negating the option in the Compile Command. To do this, enter a tilde (~) and the option in the Compile Command. For example, the following configuration file, possibly named **config.cfg**:

```
COMPILER-OPTIONS     FLAGGING=HIGH,COM2,OBSOLETE
&          OBJECT-VERSION=7
&          LISTING-PATHNAME=LISTINGS
```

directs RM/COBOL to flag HIGH, COM2 and OBSOLETE language elements, to restrict the object version level to 7, and to write the listing file to the directory named LISTINGS.

For a particular compilation, you may want to suppress some or all configured options. For example, to suppress the flagging of COM2 elements and the creation of the listing file (here, assuming the program-name is PAYROLL), enter the following Compile Command:

```
rmcobol payroll G=config.cfg F=~COM2 ~L
```

This negates the flagging of COM2 elements and suppresses the creation of the listing file (L option) for the compilation. The next time you use this configuration file in a compilation, the configured options will be in effect again.

To disable all flagging, and to write the listing to the current directory, enter the following Compile Command:

```
rmcobol payroll G=config.cfg ~F, L=.
```

This negates the flagging of HIGH, COM2 and OBSOLETE elements, and writes the listing to the current directory instead of to LISTINGS as specified in the configuration file.

A negated option calls up the default value for that option; that is, it behaves exactly as if no option were configured.

There are six groups of Compile Command options:

1. Configuration (see the following topic)
2. Data Item (on page 143)
3. File Type (on page 144)
4. Listing (on page 144)
5. Object Program (on page 147)
6. Source Program (on page 149)

## Configuration Compile Command Options

The following options designate a file to be used as the complete compiler configuration or as a supplement to it and allow suppression of the compiler banner message.

**G** Use the G Option to designate a file to be used as the compiler configuration. If the G Option is specified, any attached or automatic configuration is ignored (not processed). The G Option has the following format:

**G**=*pathname*

See the discussion of the COMPILER-OPTIONS configuration record (on page 287). See also the H Compile Command Option.

By default, a configuration file is not designated.

**H** Use the H Option to designate a file as a supplement to the compiler configuration. The specified file is processed after any attached or automatic configuration and after any file specified in the G Option, but before any other command-line options are processed. The H Option has the following format:

**H**=*pathname*

If no configuration exists (either attached or automatic, or specified with the G Compile Command Option), the specified file serves as the complete configuration. For more information, see Automatic and Attached Configuration Files (on page 282).

By default, a supplemental file is not designated.

**K** Use the K Option to suppress the banner message and the terminal error listing. This is useful when you are running under batch files or shell scripts.

By default, this information is displayed on the standard output device.

**W** Use the W Option to specify the amount of memory (in kilobytes) that the compiler should use for its internal table storage. The W Option has the following format:

**W**=*n*

where, *n* is a decimal number from 32 to 16384.

The default value is 1024 kilobytes (1024 KB) and is generally sufficient for a 20,000 – 40,000 line source program. A program with 135,000 source lines compiles at top speed with w=3072. The compiler will adjust the workspace size automatically as needed, but with a performance penalty. The compilation listing summary has information about the maximum amount of memory required for compilation, as described in Summary Listing (on page 162). This information can be used to choose an appropriate value for the W option.

## *Data Item Compile Command Options*

The following compiler options direct the compiler to assume a certain usage for data items.

**S** Use the S Option to direct the compiler to assume a separate sign when the SIGN clause is not specified for a DISPLAY usage, signed numeric data item (that is, for a data item whose PICTURE character-string clause begins with S). The S Option also allows a BLANK WHEN ZERO clause to be specified in the data description entry of a signed numeric data item for compatibility with RM/COBOL 2.*n*. In such cases, a trailing fixed insertion plus symbol (+) is assumed for the PICTURE character-string.

**Note** This option should be used only when compiling existing source programs written with an earlier version of RM/COBOL, and then only with caution. The use of this option creates inconsistencies between RM/COBOL and ANSI COBOL 1974 and 1985.

The default is to assume a trailing combined (zoned) sign unless the SIGN clause is present and to disallow the BLANK WHEN ZERO clause for signed numeric data items. For more information about trailing combined (zoned) signs, see Table 38: Nonnumeric Data in Nonnumeric Data (on page 405).

**U** Use the U Option to direct the compiler to assume an alternative usage for data items described as COMP or COMPUTATIONAL. The U Option has the following format:

**U**[=**B**|**D**|**P**]

The U Option specified alone or as U=B directs the compiler to assume BINARY usage for data items described as COMP or COMPUTATIONAL. This option causes COMP data items to be compatible with IBM OS/VS COBOL COMP data items and may result in improved computational speed at runtime.

The U=D Option directs the compiler to assume DISPLAY usage for items described as COMP or COMPUTATIONAL.

The U=P Option directs the compiler to assume PACKED-DECIMAL usage for items described as COMP or COMPUTATIONAL.

The U[=B] and 2 Options are mutually exclusive; they may not appear in the same Compile Command.

The default is to assume unpacked decimal format for data items described as COMP or COMPUTATIONAL.

## File Type Compile Command Options

The following compiler options determine whether a sequential file is declared as a binary sequential or a line sequential file.

**B** Use the B Option to define as binary sequential those sequential files not explicitly declared to be line sequential in their file control entries. For more information, see the discussion of file types and structure (on page 222).

**V** Use the V Option to direct that any sequential file not declared to be binary sequential be considered line sequential.

**Note** The B and V Options are mutually exclusive; they may not appear in the same Compile Command. If neither the B nor the V Option is used, the decision on whether the file is binary sequential or line sequential is deferred to program execution. The choice is then controlled by the configured DEFAULT-TYPE keyword (see page 326) in the RUN-SEQ-FILES configuration record.

## Listing Compile Command Options

The following compiler options generate a listing and control the destination and contents of the listing.

**Note** The L, P, and T Options direct the listing to different destinations; any or all of these options may appear in the same Compile Command. If neither the T nor the K Option is selected, an error-only listing is written to standard output.

**A** Use the A Option to direct the compiler to generate the allocation map (on page 155) in the listing.

This is useful during program development for use with the RM/COBOL Interactive Debugger.

The A Option may not be specified if none of the L, P, T, or Y=3 Options are specified or configured.

By default, the allocation map is not created as part of the listing or debugging information in the object file.

**C** Use the C Option to suppress the inclusion of copied text in the listing. Copied text is source text brought into the program as a result of encountering a COPY statement. See the description of the COPY statement (on page 212) and in Chapter 1: *Language Structure* of the *RM/COBOL Language Reference Manual*.

The C Option suppresses only the inclusion of the copied text in the listing; the copied text is always compiled. Even though the C Option is selected, erroneous lines encountered in the copied text during compilation are written to the listing along with the associated diagnostic message.

Text to the right of the COPY statement in the source line that contains that statement appears on a line by itself, immediately following the copied text.

The C Option may not be specified if none of the L, P, or T Options are specified or configured.

The C Option has the following variations (note that it has been extended to allow specification of a numeric value from 0 to 3):

| Option | Action |
|---|---|
| C=0 or ~C | Is equivalent to specifying the negated C Option (~C); that is, copied text and replaced text are not suppressed in the listing. This is also the default behavior if C is not specified. |
| C=1 or C | Is equivalent to specifying the C Option without a numeric value; that is, it specifies suppression of copied text in the listing. |
| C=2 | Specifies suppression of replaced text in the listing. That is, replaced text is not shown as comments in the listing file. |
| C=3 | Specifies suppression of both copied and replaced text in the listing. |

By default, copied text is included in the source listing. Copied text immediately follows the line that contains the COPY statement.

**E** Use the E Option to suppress the inclusion of the source program component in the listing. However, if errors are encountered during compilation, the listing will include the erroneous lines and their associated diagnostic messages.

The E Option may not be specified if none of the L, P, or T Options are specified or configured.

By default, the source program component is included in the listing.

**L** Use the L Option to direct that a listing file be written to disk. The L Option has the following format:

```
L[=pathname]
```

The L Option specified above directs the compiler to write the listing to the default directory.

*pathname* specifies a directory into which the listing file is to be written.

The listing file will always have the same name as the source file; its extension will be the listing file extension (**.lst** or **.LST** unless configured otherwise). On those operating systems that have case-sensitive filenames, the case of the extension will match the case of the first character of the source file's extension, or the first character in the source file's name if there is no extension. If there is no extension and the first character of the source filename is not a letter, then the extension **.lst** will be used. For examples of valid filenames, see Table 1 in File Naming Conventions (on page 15).

The default directory, when *pathname* is not specified, depends on whether the source filename was specified with a drive or directory in its value.

If the source filename was specified with a drive or directory in its value, the default directory is the one containing the source file.

Otherwise, the default directory is determined by using the compiler directory search sequence. If an existing file with the same name as the source file and the listing file extension is found using the compiler directory search sequence, the default directory is the one in which that file is found. If such a file is not found using the compiler directory search sequence, the default directory is the current directory. See the discussions of Directory Search Sequences on UNIX (on page 24) and Directory Search Sequences on Windows (on page 61).

If a file already exists with the specified name and extension in the specified or default directory, it is overwritten.

By default, the listing is not written to disk.

**P** Use the P Option to direct the compiler to write a copy of the listing to the printer.

Without a print spooler, the P Option cannot be used when the printer is busy.

By default, a copy of the listing is not written to the printer. See the discussion of listing (on page 152).

**R** Use the R Option to direct the compiler to generate a sequential number in the first six columns of source records as they appear on the listing. The source file itself is not affected.

If selected, this option numbers records beginning with 1 for each source or copy input file. The number can be helpful when editing the source file. This line number cannot be used with the RM/COBOL Interactive Debugger.

The default is to print the source record exactly as read, including any commentary information present in columns 1 through 6.

**T** Use the T Option to direct the compiler to write a copy of the listing to the standard output device. Generally, the standard output device is the screen, but this can be controlled through redirection.

By default, a copy of the listing is not written to the standard output device. However, the last two lines of the summary listing—as well as all erroneous lines and associated diagnostic messages—are written to the standard output device regardless of the T Option. This display can be suppressed with the K Option (see page 143).

**X** Use the X Option to direct the compiler to generate a cross reference map in the listing. The cross reference map contains an alphabetic list of all user-defined words that appear in the source program. For each user-defined word, the line number of each appearance is listed. Each line number is marked to indicate that the word is being used as a declaration, a source operand or a possible destination operand. (See Figure 33 on page 161 for a sample of the cross reference map.)

The X Option may not be specified if none of the L, P, T, or Y=3 Options are specified or configured.

By default, the cross reference map is not included in the listing or in the debugging information in the object file.

## *Object Program Compile Command Options*

The following compiler options generate or suppress an object program and control the destination and features of the object program.

**M** Use the M Option to direct the compiler to suppress automatic conversions in certain ACCEPT and DISPLAY statements. In Format 1 and 3 ACCEPT statements, this option suppresses automatic input conversion for numeric operands and suppresses right justification for justified operands. For Format 3 DISPLAY statements (DISPLAY *screen-name*), this option suppresses automatic output conversion for numeric fields within the screen description entry.

**Note** This option must be used if Format 1 ACCEPT statements with numeric operands are to be treated in compliance with ANSI COBOL 1985 and 1974.

The default is to provide input conversion for numeric operands of Format 1 and 3 ACCEPT statements, right justification for justified operands of Format 1 and 3 ACCEPT statements, and output conversion for numeric fields of Format 3 DISPLAY statements.

**N** Use the N Option to suppress the generation of an object program.

The default is to generate object code according to the rules for the O Option, described in the following section.

**O** Use the O Option to specify the directory pathname where the object file will be placed. The O Option has the following format:

**O**=*pathname*

where, *pathname* specifies a directory into which the object file is to be written.

The object file will always have the same name as the source file. Its extension will be the object file extension (**.cob** or **.COB** unless configured otherwise). On those operating systems that have case-sensitive filenames, the case of the extension will match the case of the first character of the source file's extension, or the first character in the source file's name if there is no extension. If there is no extension and the first character of the source filename is not a letter, then the extension **.cob** will be used. For examples of valid filenames, see Table 1 in File Naming Conventions (on page 15).

The O and N Options may appear together in a single compilation. For example, the OBJECT-PATHNAME keyword (see page 298) in the COMPILER-OPTIONS configuration record specifies the directory for the object file. Entering the N Option on the Compile Command suppresses the generation of the object file (and as a result negates the OBJECT-PATHNAME keyword in the configuration file).

The default directory depends on whether or not the source filename was specified with a drive or directory in its value.

If the source filename was specified with a drive or directory in its value, the default directory is the one containing the source file.

Otherwise, the default directory is determined by using the directory search sequence. If an existing file with the same name as the source file and the object file extension is found using the compiler directory search sequence, the default directory is the one in which that file is found. If such a file is not found using

the compiler directory search sequence (see the appropriate installation and systems considerations chapter in this user's guide for your specific operating system), the default directory is the current directory.

If a file already exists with the specified name and extension in the specified or default directory, it is overwritten.

**Q** Use the Q Option to direct the compiler to eliminate debugging information from generated object programs. Programs compiled with this option will appear invisible to the Interactive Debugger and Instrumentation. A statement address consisting of an optional segment number and segment offset will be substituted for line numbers in Normal Termination, Error Termination and Traceback runtime system messages. A segment number and segment offset replace line number references when this option is selected.

The Q and Y options are mutually exclusive; that is, they may not appear in the same Compile Command.

**Note** This option may be used to both reduce the memory requirements and increase the execution speed of most programs.

The default is to generate debugging line number information in object programs.

**Y** Use the Y Option to direct the compiler to output debugging information in the object file. The Y Option has the following variations:

| Option | Action |
|---|---|
| Y=0 or ~Y | Omits the symbol and debug line table from the object program file. This is also the default behavior if Y is not specified. |
| Y=1 or Y | Places the symbol table but not the debug line table in the object file. When the symbol table is included in the object program file, the source program data-names and index-names may be used in Debug commands during execution. For more information, see Chapter 9: *Debugging* (on page 245). |
| Y=2 | Places both the symbol table and the debug line table in the object file. The line table is used by CodeWatch to display the source program. |
| Y=3 | Same as Y=2, except that the debug line table also includes allocation map and cross-reference information if the A and/or X options are also specified. This information can then be viewed within CodeWatch, but may lead to large object program files. |

Object program files created with Y=2 and Y=3 are fully compatible with all versions of the RM/COBOL runtime (note that previous versions will ignore these tables). This option does increase the size of the object program files, but has no effect on runtime performance or memory requirements.

The Y and Q options are mutually exclusive; that is, they may not appear in the same Compile Command.

**Note** A new option in the Combine Program (rmpgmcom) utility (on page 583), STRIP, may be used to remove symbol table and debug line table information from object files that were created with Y=1 or Y=2. For source code security,

object program files that contain line table information should be reduced in size with this option or recompiled without the Y option before they are redistributed.

By default, the symbol table is omitted from the object file.

**Z** Use the Z Option to indicate the highest allowed object version of the generated code. The Z Option has the following format:

```
Z=version
```

where, *version* must be an integer in the range 7 through 12. For compilers licensed for evaluation or educational purposes or for a limited time (ValidThru date specified in the compiler license), the minimum *version* value is 9.

Statements that require a higher object version level than the value specified will be flagged in error. See the Compile Command Messages (on page 166) and the description of the COMPUTATIONAL-VERSION keyword on page 291 for the COMPILER-OPTIONS configuration record. This option forces the generation of code accepted by earlier versions of the RM/COBOL runtime system.

Appendix H: *Object Versions* (on page 617) lists the changes between object versions.

The default is to use the current object version number (12) as the limit, but the generated object version is the minimum necessary for any given source program.

**7** Use the 7 Option to specify the semantic rules under which the program is to be compiled.

7 specifies that the source program is to be compiled with ANSI COBOL 1974 semantics. ANSI COBOL 1974 semantics affect the I-O status values, PERFORM . . . VARYING statements, ALPHABETIC class conditions, and alphabetic-edited data items. A more specific discussion of these semantic differences can be obtained by contacting Liant technical support services.

The 7 Option is implied if the 2 Option is specified.

The default is to compile the source program using ANSI COBOL 1985 semantics.

## Source Program Compile Command Options

The following compiler options affect the analysis of the source program and cause flagging of certain source features.

**D** Use the D Option to direct RM/COBOL to compile all source programs as if the WITH DEBUGGING MODE clause appeared in each compiled program. This option causes all source lines with the letter D in the indicator area to be compiled as if they had a space in the indicator area.

This option is independent of the RM/COBOL Interactive Debugger, described in Chapter 9: *Debugging* (on page 245).

The default is to treat source lines with the letter D in the indicator area as commentary information unless the WITH DEBUGGING MODE clause is specified in the source program.

**F** Use the F Option to direct the compiler to flag occurrences of these language elements:

| | |
|---|---|
| COM1 | INTERMEDIATE |
| COM2 | OBSOLETE |
| EXTENSION | SEG1 |
| HIGH | SEG2 |

The F Option has the following format:

```
[F=(keyword-list)]
[F=keyword       ]
```

where,

*keyword-list* specifies multiple elements to be flagged. Enclose the list in parentheses, and if the *keyword-list* contains more than one item, separate them with a space or comma. If leading hyphens are being used, the parentheses are optional. You can negate an individual keyword by preceding it with a tilde (~).

*keyword* specifies a single element to be flagged.

The names of elements can be abbreviated, as long as they remain unique. If the abbreviation is not unique, the keyword that occurs first alphabetically is chosen. For example, C, CO and COM are valid abbreviations of COM1 but not of COM2.

Certain keywords cause more than one element of the language to be flagged:

1. Selecting INTERMEDIATE flags both HIGH and INTERMEDIATE elements.
2. Selecting COM1 flags both COM1 and COM2 elements.
3. Selecting SEG1 flags both SEG1 and SEG2 elements.

See Appendix I: *Extension, Obsolete, and Subset Language Elements* (on page 629) for a complete list of elements flagged.

By default, no elements of the language are flagged.

**2** Use the 2 Option to direct the compiler to accept source programs created for the RM/COBOL (74) 2.*n* compiler.

If the programs were compiled (or designed to be compiled) without the RM/COBOL (74) 2.*n* compiler ANSI Option, the separate sign (S) and line sequential (V) Options (described on pages 143 and 144, respectively) may also need to be selected.

The 2 Option removes certain words from the list of RM/COBOL reserved words. The removed words are those that are RM/COBOL additions to RM/COBOL (74) 2.*n*; thus, all words used in the earlier version as user-defined words are still valid. Note carefully that if RM/COBOL language features are added to the program, the 2 Option can no longer be used, and the program must be changed accordingly. There is also a technique for removing individual words from the list of reserved words. See the discussion of the COMPILER-OPTIONS configuration record (on page 287).

The 2 Option directs that COMP-3 data items always be signed, irrespective of the presence or absence of an S in the associated PICTURE character-string.

The 2 Option directs that COMP-1 data items behave as in RM/COBOL (74) 2.*n*. This causes the number of digits in the PICTURE character-string describing a COMP-1 item to be ignored in three situations: when the item is the receiving item in a MOVE statement, in an arithmetic statement that specifies ON SIZE ERROR, and in an ACCEPT statement that specifies, explicitly or implicitly, input conversion. In these situations, the COMP-1 item may contain any value in the range –32768 through 32767.

The 2 Option directs that OPEN EXTEND create a new file when the file is not present, even when OPTIONAL was not specified in the file control entry.

The 2 Option directs that equality and inequality relation conditions, where the subject and object are similar signed packed-decimal (COMP-3 or PACKED-DECIMAL usage) or signed unpacked-decimal (COMP usage) operands, should not be optimized to use string comparison operations. The string comparison optimization prevents detection of equality when the only difference between the subject and object of the relation results from the change in positive sign convention for such items.

The 2 Option directs that the size of index data items be two bytes in length.

The 2 Option directs that the implied EXIT PROGRAM required by ANSI COBOL 1985 at the end of the Procedure Division be omitted. RM/COBOL (74) 2.*n* had only an implied STOP RUN at the end of the Procedure Division.

The 2 and U[=B] Options are mutually exclusive; they may not appear in the same Compile Command.

The 2 Option implies the 7 Option.

The default is to recognize all RM/COBOL reserved words, treat COMP-3 data items without an S in their PICTURE character-string as unsigned data items, treat COMP-1 data items the same as two-byte COMP-4 data items, return a file not present error for OPEN EXTEND of a nonexistent file not described with the OPTIONAL phrase in its file control entry, use the string comparison optimization for conditional relations of similar signed COMP-3 and COMP data items, use a size of four bytes for index data items, and include the implied EXIT PROGRAM at the end of the Procedure Division.

# Sample Compile Commands

Here are examples of valid and invalid RM/COBOL Compile Commands.

## Valid Compile Commands

```
rmcobol payroll.con P, V R
```

This command compiles the program named **payroll.con**; it directs the listing to the system printer (the P Option); declares all sequential files not defined as binary sequential in the source program to be line sequential files (the V Option); and sequentially numbers the printed listing, starting with 1 for each copy level, in the first six columns of the listing (the R Option).

```
rmcobol demo.prg (D,L=COBOL,S X) 3RD COMPILE
```

This command compiles the program **demo.prg**; the program is compiled as if the WITH DEBUGGING clause were present (the D Option); the listing is written to the directory named COBOL (the L Option); a separate sign is assumed in the absence of a SIGN clause (the S Option); and the cross reference map is generated (the X Option). A comment—3RD COMPILE—is reproduced in the listing header, but is ignored by the compiler.

**Note** Under UNIX, the parenthesis must be preceded with a backslash (\) character in order to be protected from the shell.

### Invalid Compile Command

```
rmcobol payroll.cob B V
```

Here, the extension to the filename (**.cob**) is illegal, since .cob is the default extension for the object file. The B and V Options are entered together: B treats all sequential files not specified as either binary sequential or line sequential in the file control entry as binary sequential, but V treats all such files as line sequential.

## Listing

Depending on the options specified in the Compile Command, the compiler generates a detailed listing. The T Option (see page 146) directs the listing to standard output. The listing can be directed to the printer with the P Option (see page 146) and to a file with the L Option (see page 145). All three of these options—or any combination thereof—may be specified. However, keep in mind that in certain circumstances the listing may contain lines as long as 132 characters. If the device to which the listing is sent cannot accommodate lines of that width, characters at the right end of the long lines may be truncated or wrapped.

**Note** Error lines are always listed to standard output unless suppressed by the K Option (see page 143).

The components of the listing (in order of appearance) are as follows:

1. **Program listing**, which contains the source image of the program.
2. **Allocation map**, which defines and locates each identifier used in the program.
3. **Called program summary**, which lists the names of all programs called or canceled by the program being compiled.
4. **Cross reference listing**, which lists the names of all identifiers used in the program, along with the source line numbers at which they are declared and used.
5. **Summary listing**, which provides status information on the compilation itself.

When the listing is written to a printer (either because the P Option is selected or because a disk file that was generated as a result of the L Option is printed), each component starts a new page.

## Program Listing

At the top of each page of the program listing, a header appears, a sample of which appears in Figure 24.

**Figure 24: Program Listing Header**

```
RM/COBOL (Version 9.0n.00) for operating-system 03/15/2004 08:52:03 Page 1
Source file: ALLOCMAP        Options: L A X
```

**Note** The date and time formats are configurable. For more information, see the discussion of the COMPILER-OPTIONS configuration record (on page 287).

Each page of the program listing also contains a subheader, illustrated in Figure 25.

**Figure 25: Program Listing Subheader**

```
LINE   DEBUG     PG/LN -A 1
B..+....2....+....3....+....4....+....5....+....6....7..IDENTFCN
```

This subheader sets a scale against which material on each page can be measured. The column of numbers under the “LINE” heading contains sequential line numbers assigned by the compiler to each line read from the source file or from a copy file; these line numbers are used in the cross reference listing and in Debug. The numbers under the “DEBUG” heading are used with the Interactive Debugger or for interpreting error messages when the compiler Q Option is used; this column is used only when listing the Procedure Division. The remaining headings locate the regions of the source line images: the internal six-column line number field, area A, area B, the main body of the source image (subdivided into ten-column subregions) and the eight-column identification field.

If the R Option was present in the Compile Command, the program listing contains a compiler-generated line number in the PG/LN column.

The subheading is sometimes useful in determining when the text of a source line has inadvertently extended beyond column 72, and is therefore not seen by the compiler.

The program listing itself contains the sequential line number, statement address, copy level indicator (described in the next paragraph) and the source record. If errors were detected during compilation, the appropriate error message diagnostic appears. See Error Marker and Diagnostics (on page 164).

The copy level indicator is a character-string of the following form:

```
+ n +
```

where, *n* is a decimal digit in the range 1 through 9. The copy level indicator appears between the sentence address (DEBUG heading) and source record in the listing whenever the source record has been copied at level *n*.

A sample of a program listing is shown in Figure 26 on page 155.

Statement addresses are listed in decimal notation. For overlay segments, the segment number is printed as part of the statement address. A slash separates the segment number from the offset within the segment. For example:

```
50/000100
```

refers to location 100 within segment 50. Segment numbers and the slash are suppressed for the fixed permanent segment.

The generation of the program listing may be suppressed by specifying the E Option (see page 145) in the Compile Command. Copied source text can be suppressed with the C Option (see page 137). Error messages (if any) and their associated undermarks and source text are not suppressed, even when the C or E Option has been selected.

The copy level indicator has been expanded into a source indicator by varying the brackets around the copy nesting level number *n*. Copy nesting level number 0 is the original source file that is being compiled. A source indicator of +0+ is never included in the listing, but the new source indicators may be used with copy nesting level number 0 because of the REPLACE statement. The additional source indicators have the following meanings:

| Source Indicator | Meaning |
|---|---|
| <*n*> | The source text was replaced at copy nesting level *n* because of a REPLACE statement or the REPLACING phrase of a COPY statement. Such source text is listed on a comment line in the listing, even though the original line that contained the source text was not a comment line. |
| >*n*< | The source line was inserted at copy nesting level *n* by a REPLACE statement or the REPLACING phrase of a COPY statement. |
| [*n*] | The source line was modified by a REPLACE statement, the REPLACING phrase of a COPY statement, or by relocation of source text that occurred on the same line as all or part of a COPY statement (the compiler relocated the source text so that it would be compiled after the copied file). |
| {*n*} | The source text was relocated to a new line because it followed a matching replacement key of a REPLACE statement or of the REPLACING phrase of a COPY statement, or it occurred on the same line as all or part of a COPY statement. When possible for replacements, such source text is merged with the last line of the inserted replacement text. Therefore, this source indicator only occurs when the merge is not possible. That is, when the merge is possible, the source indicator >*n*< is printed for the last line of the inserted replacement text that includes the merged text-words. (Source text that is split from a COPY statement is not merged with the last line of the copied file.) |

For easy reference, a summary of the source indicator meanings is included in the summary listing portion of the listing file when the source indicator is used in the listing, as described in Summary Listing (on page 162).

**Figure 26: Sample Program Listing**

```
    1               IDENTIFICATION DIVISION.
    2               PROGRAM-ID.    ALLOCMAP.
0   3               ENVIRONMENT DIVISION.                                    0
    4               CONFIGURATION SECTION.
    5               SOURCE-COMPUTER. IBM-PC-XT.
    6               OBJECT-COMPUTER. IBM-PC-XT,
0   7                                    PROGRAM COLLATING SEQUENCE EBCDIC-CODE.  0
    8               SPECIAL-NAMES.
    9                    SWITCH-1 IS REPORT-MODE,
   10                        ON STATUS IS REPORT-LIST,
0  11                        OFF STATUS IS REPORT-NOLIST;                     0
   12                   SWITCH-3 IS DISPLAY-MODE,
   13                        ON STATUS IS DISPLAY-LIST,
   14                        OFF STATUS IS DISPLAY-NOLIST;
0  15                   CO1 IS TOP-OF-FORM;                                  0
   16                   CO5 IS AMOUNT-LINE;
   17                   CONSOLE IS PC-DISPLAY;
   18                   SYSIN IS STANDARD-IN;
0  19                   SYSIN IS STANDARD-OUT;                               0
   20                   ALPHABET ASCII-1 IS STANDARD-1;
   21                   ALPHABET ASCII-2 IS STANDARD-2;
   22                   ALPHABET NATIVE-1 IS NATIVE;
0  23                   ALPHABET EBCDIC-CODE IS EBCDIC;                      0
   24                   ALPHABET BACKWARDS IS "ZYXWVUTSRQPONMLKJIHGFEDCBA";
   25                   SYMBOLIC CHARACTERS QUESTION-MARK, ASTERISK ARE 64, 43;
   26                   CLASS PUNCTUATION IS ";", ",", ".", "!", "?".
0  27               INPUT-OUTPUT SECTION.                                    0
   28               FILE-CONTROL.
   29                   SELECT REPORT-FILE1   ASSIGN TO PRINTER;
   30                                         ORGANIZATION IS SEQUENTIAL;
0  31                                         ACCESS IS SEQUENTIAL.            0
   32                   SELECT LOOKUP-FILE1   ASSIGN TO DISC;
   33                                         ORGANIZATION IS RELATIVE;
   34                                         ACCESS IS SEQUENTIAL.
0                          .                                                  0
                           .
                           .
  151               PROCEDURE DIVISION USING ARG1-GROUP, ARG2-GROUP.
0 152   000002     A.                                                         0
  153   000005         CALL "CHRRTN" USING NW5-MDATE, NW5-MTIME.
  154   000016         CALL MATHRTN USING NBS-1, NBU-1, NCS-1, NCU-1,
  155                     NLC-1, NPS-1.
0 156   000035     STOP RUN.                                                  0
  157               END PROGRAM ALLOCMAP.
```

## Allocation Map

The allocation map provides information on each user-defined word from the source program, listed in the order declared. The type of user-defined word (described in the following section) determines the allocation map format. The allocation map is generated in the listing when the A Option is specified in the Compile Command (see page 144) or the LISTING-ATTRIBUTES keyword (see page 295) is configured with the ALLOCATION-MAP value.

## *Alphabet-Names, Symbolic-Characters, Mnemonic-Names, and Class-Names*

User-defined words declared in the SPECIAL-NAMES paragraph are listed in the allocation map with the following information:

1. **Association**, which is the value for a figurative or symbolic-character; the code-name for an alphabet-name; the switch-name for a mnemonic-name or condition-name associated with a switch-name; the channel-name for a mnemonic-name associated with a channel-name; the low-volume-I-O-name for a mnemonic-name associated with a low-volume-I-O-name; or blank for a class-name. The value of a figurative or symbolic-character is listed as the hexadecimal value in the native character set. If that value represents a printable character, the printable character is listed in quotation marks.
2. **Status**, which is On or Off for a condition-name associated with a switch-name. The letters PCS appear with an alphabet-name declared as the program collating sequence. Otherwise, the column is blank.
3. **Type**, which indicates whether the user-defined word is: an alphabet-name; a mnemonic-name associated with a switch-name; a condition-name; a mnemonic-name associated with a channel-name; a mnemonic-name associated with a low-volume-I-O-name; a class-name; or a symbolic-character.
4. **Name**, which is the actual user-defined word declared with the indicated attributes or the figurative constant LOW-VALUE or HIGH-VALUE. These particular figurative constants are listed since their value depends on the program collating sequence declared in the source program.

Figure 27 is an example of this part of the allocation map.

**Figure 27: Allocation Map (Part 1 of 5)**

```
Special-Names
  Association              Status  Type                  Name
  X"00"                            Figurative constant   LOW-VALUE
  X"FF"                            Figurative constant   HIGH-VALUE
  SWITCH-1                         Switch-name           REPORT-MODE
  SWITCH-1                   On    Condition-name        REPORT-LIST
  SWITCH-1                   Off   Condition-name        REPORT-NOLIST
  SWITCH-3                         Switch-name           REPORT-MODE
  SWITCH-3                   On    Condition-name        REPORT-LIST
  SWITCH-3                   Off   Condition-name        REPORT-NOLIST
  C01                              Channel-name          TOP-OF-FORM
  C05                              Channel-name          AMOUNT-LINE
  CONSOLE                          Low-volume-I-O-name   PC-DISPLAY
  SYSIN                            Low-volume-I-O-name   STANDARD-IN
  SYSOUT                           Low-volume-I-O-name   STANDARD-OUT
  STANDARD-1                       Alphabet-name         ASCII-1
  STANDARD-2                       Alphabet-name         ASCII-2
  NATIVE                           Alphabet-name         NATIVE-1
  EBCDIC                     PCS   Alphabet-name         EBCDIC-CODE
  Literal                          Alphabet-name         BACKWARDS
  X"3F" = "?"                      Symbolic-character    QUESTION-MARK
  X"2A" = "*"                      Symbolic-character    ASTERISK
                                   Class-name            PUNCTUATION
```

*alphabet-names, symbolic-characters, mnemonic-names, and class-names*

## *Split Key Names*

User-defined words, declared as part of a RECORD KEY clause in an indexed file control entry of the Environment Division that defines a split key, are listed in the allocation map with the following information:

1. **File-Name** is the name of the file from the indexed file control entry.
2. **Key-Number** specifies the number of the key that has a split key defined. A value of zero indicates the prime record key. Alternate keys are numbered from 1 to 254.
3. **Type** indicates that the entry is a split-key-name.
4. **Name** is the name associated with the split key.

Figure 28 illustrates a section of the allocation map for a file that defines split keys for the primary key and the second alternate key in the file control entry.

**Figure 28: Allocation Map (Part 2 of 5)**

```
Split Key Names for program SPLITKEY
File-Name              Key-Number   Type                Name
FILE-1                          0   Split-key-name      KEY-1
                                2   Split-key-name      KEY-2
```

*split-key-names*

## *Data-Names, Condition-Names, File-Names and Cd-Names*

User-defined words declared in the Data Division, other than index-names, are listed in the allocation map with the following information:

1. **Address**, which is the decimal address for data-names. The “Address” column is blank for file-names, cd-names and condition-names.

   For data items declared with the external attribute in the File Section or Working-Storage Section, the compiler-generated external number is printed on a line preceding the file or level 01 item description.

   For data-names declared in the Linkage Section, each level 01 or 77 item is preceded by an indication of how it is addressable:

   - If it is listed in the USING phrase of the Procedure Division header, “U*n*:” and “Using argument *n*” are printed to indicate the formal argument umber is *n* within the USING argument list.
   - If it is listed in the GIVING (RETURNING) phrase of the Procedure Division header, “G:” and “Giving argument” are printed to indicate that the item is the formal GIVING argument.
   - If it is a based linkage record and is not a formal argument, “B*n*:” and “Based linkage record *n*” are printed to indicate that the compiler assigned based linkage record number is *n*.
   - If none of the preceding descriptions apply, “Not addressable” is printed to indicate that the Linkage Section data item is not available to the program.

2. **Size**, which is the decimal number of character positions required to store the value of a data-name, or the maximum block size—in characters or records—for a file-name declared with a non-zero block size. The “Size” column is blank for cd-names and condition-names.

3. **Debug**, which contains an abbreviated type indicator used in the Interactive Debugger to describe the format of the data item. The “Debug” column contains “Fixed” or “Variable” for file-names to indicate that records of the file are fixed or variable length, respectively. The “Debug” column is blank for cd-names and condition-names.

   **Note** These first three columns (Address, Size, and Debug) are used with the Interactive Debugger to display and modify the values of data-names. See Chapter 9: *Debugging* (on page 245).

4. **Order**, which indicates the number of subscripts required when referencing the data-name or condition-name. The “Order” column is blank for data-names not requiring subscripting and also for file-names and cd-names. When one or more subscripts are required, the order is indicated with a decimal number enclosed in parentheses.

5. **Type**, which is a brief description of the item associated with the user-defined word. For files, the organization and access are listed, in that order, separated by a slash.

6. **Name**, which is the actual user-defined word declared with the listed attributes. The name is indented one column to the right for each increase in level-number.

Figure 29 is an example of this part of the allocation map.

**Figure 29: Allocation Map (Part 3 of 5)**

```
File Section for program ALLOCMAP
 Address    Size     Debug    Order     Type                Name
0                    Variable          File Seq/Seq        REPORT-FILE1      0
       8      80     ANS               Alphanumeric         REPORT-RECORD-1
       8      40     ANS               Alphanumeric         REPORT-RECORD-2
              .
0             .                                                             0
              .
Working-Storage Section for program ALLOCMAP
  Address   Size     Debug    Order     Type                Name
0     532   112      GRP               Group               G1                0
      532     8      ABS               Alphabetic           ABS-1
      540     8      ANSE              Alphanumeric edited  ANSE-1
      548     8      ABS               Alphabetic, just     ABSR-1
0             .                                                             0
              .
              .
Linkage Section for program ALLOCMAP
0 Address   Size     Debug    Order     Type                Name              0
   U1:                                 Using argument 1
        0     44     GRP               Group               ARG-GROUP
        0      4     NSU               Numeric unsigned     ARG-COUNT
0       4      8     ANS      (1)      Alphanumeric         ARG-AREA          0
              .
              .
              .
0 Communication Section for program ALLOCMAP                                  0
  Address   Size     Debug    Order     Type                Name
                                       Cd for Input        NET-WORK-1
      734     12     ANS               Alphanumeric         NW1-SYM-Q
0     746     12     ANS               Alphanumeric         NW1-SQ1           0
      758     12     ANS               Alphanumeric         NW1-SQ2
      770     12     ANS               Alphanumeric         NW1-SQ3
      782      6     NSU               Numeric unsigned     NW1-MDATE
```

*data-names, condition-names, file-names and cd-names*

## *Index-Names*

User-defined words declared as index-names in the Data Division are listed in the allocation map with the following information:

1. **Address**, which is the decimal address of the index-name within the index-name table for the program. For index-names with the external attribute, the compiler-generated external number is printed on a line preceding the index-name description.
2. **Span**, which is the decimal number of character positions of the table entry associated with the index-name. This value is needed in order to convert index-name values to occurrence number values and vice versa.
3. **Debug**, which contains an abbreviated type indicator used in the Interactive Debugger to describe the format of the index-name. For all index-names, the type indicator is IXN.

   **Note** These first three columns (Address, Span, and Debug) are used with the Interactive Debugger to display and modify the values of index-names. See Chapter 9: *Debugging* (on page 245).
4. **Type**, which is a description of the item associated with the user-defined word. It is Index-name for all index-names.
5. **Name**, which is the actual user-defined word declared as the index-name.

Figure 30 is an example of this part of the allocation map.

**Figure 30: Allocation Map (Part 4 of 5)**

```
Index-names for program ALLOCMAP
   Address       Span      Debug        Type          Name
0          0          5        IXN        Index-name    G2-I1          0
           4          5        IXN        Index-name    G2-I2
           8         13        IXN        Index-name    NW3-I1
          12         13        IXN        Index-name    NW3-I2
```

*index-names*

### *Constant-Names*

User-defined words declared as constant-names in the Data Division are listed in the allocation map with the following information:

1. **Constant Value**, which is the value associated with the constant-name. If the constant-name value was specified with a constant-expression, then the result value is shown. Otherwise, the literal associated with the constant-name is shown.
2. **Type**, which is a brief description of the type of the value associated with the constant-name. If the constant-name value was specified with a constant-expression, then the type is always Numeric unsigned. Otherwise, the type is the type of the literal specified as the value for the constant-name.
3. **Name**, which is the actual user-defined word declared as the constant-name.

Figure 31 is an example of this part of the allocation map.

**Figure 31: Allocation Map (Part 5 of 5)**

```
Constant-names for program ALLOCMAP
   Constant Value                  Type                Name
0  2                               Numeric unsigned    TWO                0
   "STRING1"                       Alphanumeric        STRING1
   QUOTE (QUOTES)                  Alphanumeric        MY-QUOTES
   -256.357                        Numeric signed      CONSTANT1
0  X"454647"                       Alphanumeric        HEX1               0
   ALL "ABC"                       Alphanumeric        STRING2
   ZERO (ZEROS, ZEROES)            Numeric unsigned    MY-ZEROS
   SPACE (SPACES)                  Alphabetic          MY-SPACES
```

*constant-names*

## Called Program Summary

The called program summary lists the names of all called and canceled programs and the using count associated with each. Figure 32 illustrates this listing.

**Figure 32: Called Program Summary**

```
Called Program Summary
Program-name required          Using count
0  MATHRTN                             6                              0
   "CHARRTN"                           2
```

The program-name appears without quotation marks for dynamic (identifier) references and inside quotation marks for static (literal) references. The “Using count” field lists the maximum number of arguments used in any CALL reference to the listed literal or identifier.

## Cross Reference Listing

The cross reference alphabetically lists all user-defined words used in the program, and provides the line number of each declaration, source and possible destination reference. The line number is enclosed in slashes if the reference is a declaration or in asterisks if the reference is a possible receiving item. The line number is not marked for sending items. Procedure-names specified as the first operand of an ALTER statement and data-names that are specified as receiving operands of Procedure Division statements are considered destination references and are thus marked with asterisks in the cross reference listing. The cross reference is generated in the listing when the X Option is specified in the Compile Command (see page 146) or the LISTING-ATTRIBUTES keyword (see page 295) is configured with the CROSS-REFERENCE value. Figure 33 illustrates the cross reference listing.

**Note** The method used to mark possible destination references with surrounding asterisks errs on the conservative side, particularly in arithmetic statements. The compiler marks the second operand of an arithmetic statement as a possible destination even though it may be followed by the GIVING phrase, which causes the second operand to be only a sending item. The operands in the USING phrase of a CALL statement are always considered to be possible destination references unless they are subject to a BY CONTENT phrase.

**Figure 33: Cross Reference Listing**

```
Cross reference       /Declaration/     *Destination*
A                 /0152/
0 ABSE-1          /0082/                                    0
  ABSR-1          /0083/
  ABS-1           /0081/
  AMOUNT-LINE     /0016/
0 ANSE-1          /0085                                     0
  ANSR-1          /0086/
  ANS-1           /0084/
  ARG1-AREA       /0113/
0 ARG1-COUNT      /0112/                                    0
  ARG1-GROUP      /0111/  0151
  ARG2-AREA       /0116/
  ARG2-COUNT      /0115/
0 ARG2-GROUP      /0114/  0151                              0
  ARG3-AREA       /0119/
  ARG3-COUNT      /0118/
  ARG3-GROUP      /0117/
0 ASCII-1         /0020/                                    0
  ASCII-2         /0021/
  ASTERISK        /0025/
  BACKWARDS       /0024/
0 DB1-DATA        /0070/                                    0
  DB1-KEY          0047  /0069/
```

## Summary Listing

The summary listing shows the sizes of the regions of the generated object program, the maximum compilation memory used, and other summary information about the entire source program. Figure 34 illustrates this listing.

**Figure 34: Summary Listing**

```
Program Summary Statistics
Read only size:                    266 (X"0000010A") bytes
Read/write size:                   532 (X"00000214") bytes
Overlayable segment size:            0 (X"00000000") bytes
Total generated object size:       798 (X"0000031E") bytes
Maximum EXTERNAL size:              88 (X"00000058") bytes
Total EXTERNAL size:                92 (X"0000005C") bytes
Source program used 4489 (7%) of 65534 available identifiers
(T1C limit).
Source program used 33004 (6%) of 588800 available user-defined
word space (T2B limit).
Maximum compilation memory used was 487K bytes (2 presses and 0
increases required).
+n+ Source was copied from copy file at copy nesting level n
(level 0 indicator is suppressed).
<n> Source was replaced at copy nesting level n because of REPLACE
or REPLACING.
>n< Source was inserted by REPLACE or REPLACING.
[n] Source was modified by REPLACE, REPLACING, or split of text
following a COPY statement.
(n) Source was split from a previous line with a replacement
match or COPY statement.
Errors: 1, Warnings: 0, Lines: 157 for program ALLOCMAP
Previous diagnostic message occurred at line 151.
Object version level = 3
Options in effect:
A - Allocation map listing
L – Listing file
X - Cross reference listing
```

The line labeled “Read only size” lists the size of that region of the object program that contains values that do not change during program execution. It consists primarily of the instructions generated for the resident (or fixed) portion of the Procedure Division, representations of the literals mentioned in the Procedure Division, and descriptors of the operands referred to in the Procedure Division.

The line labeled “Read/write size” lists the size of that region of the object program that contains values that might change during the course of execution. It consists primarily of a current record area and a control block for each of the files specified, an area for the Working-Storage Section and other internal control information.

The line labeled “Overlayable segment size” lists the size of the region of the object program that is reserved for the independent and fixed overlayable segments of the Procedure Division. Its length is the length of the longest independent or fixed overlayable segment. All such segments are loaded into this common region on an as-needed basis.

The line labeled “Total generated object size” lists the sum of the preceding values, and is therefore the amount of memory needed to load the object program. It is not the total size needed to execute that program. To execute the program there must be memory available to accommodate not only the total size (as shown on the fourth line), but also the operating system, the runtime system, any external data items, and the I/O buffers. Although you have no control over the size of the operating system

or runtime system, you can exercise some control over the memory requirement for the I/O buffers by use of the RESERVE and BLOCK CONTAINS clauses, described in detail in File Types and Structure (on page 222).

The line labeled “Maximum EXTERNAL size” indicates the size of the single largest record area with the external attribute declared in the source program. This number is useful because the maximum allowed value varies depending on the environment in which the program is run. For more information on these limitations, see Memory Available for a COBOL Run UNIT on UNIX (on page 32) and Memory Available for a COBOL Run UNIT on Windows (on page 103).

The line labeled “Total EXTERNAL size” indicates the sum of the sizes of all record areas with the external attribute declared in the source program. This number provides information needed in estimating the runtime system memory requirements of the program, but is not a direct measure since the memory requirements depend on the use of matching external records in other programs of the run unit.

**Note** The two lines regarding EXTERNAL size are omitted in the listing file when the program does not specify the EXTERNAL clause for any item.

The line labeled “Source program used ... of 65534 available identifiers ...” indicates the amount of the identifier table limit consumed. Identifiers are the individual items (classes, symbolic-characters, data items, conditions, and so forth) declared in the program. Each data item and condition defined in the program requires its own identifier entry even if the data-name or condition-name for the data item or condition is the same, since qualification can be used to distinguish between the data items or conditions. The T1C in the message refers to the compiler limit listed in Table 14 beginning on page 169.

The line labeled “Source program used ... of 588800 available user-defined word space ...” indicates the amount of the user-defined word space consumed. User-defined words are the unique spellings of words used as alphabet-names, cd-names, class-names, condition-names, data-names, file-names, index-names, key-names, mnemonic-names, paragraph-names, section-names, and symbolic-characters in the source program. Any particular spelling consumes space only once in the user-defined word table. The T2B in the message refers to the compiler limit listed in Table 14 beginning on page 169. The limit of 98133 shown in that table assumes 30-character names, which use six words each in the user-defined word space. If names averaged 24-characters in length (5 words average use of word space), the limit would be 117760 names.

The line labeled “Maximum compilation memory ...” indicates the amount of memory required to compile the source program. Setting the workspace size for the compiler to a value at least this size or slightly larger results in the best compilation speed with the minimum amount of memory consumption. The workspace size can be set using the W Compile Command Option (see page 143) or the WORKSPACE-SIZE keyword (see page 304) of the COMPILER-OPTIONS configuration record. The number of presses indicates how many times the compiler attempted to recover unused memory. Minimizing the number of presses by increasing the workspace size provides improved compilation speed. If the number of presses is zero, then the compilation speed cannot be improved by increasing the workspace size. The number of increases indicates the number of times the compiler had to request more memory because the original workspace size was too small.

The line labeled “Source indicators ...” and the lines indented under this header provide a summary of the source indicators used in columns 16- 8 of the listing. Only those explanation lines for source indicators actually used in the program listing are included in the summary. If no source indicators were used in the program

listing, then the header line is not printed in the summary listing. For further details on source indicators, see page 154.

The lines labeled “Errors: . . .” and “Previous diagnostic message . . .” summarize the number of diagnostic messages issued during compilation and the location of the last diagnostic message, respectively.

The line labeled “Object version level” indicates the object version level of the object program associated with the program being compiled. For complete information on the object version levels accepted by RM/COBOL, see Appendix H: *Object Versions* (on page 617).

The line labeled “Options in effect” and the lines that follow list the options selected for the compilation. The listed options may have been specified in the Compile Command (on page 137) or be part of a configuration file, as discussed in the COMPILER-OPTIONS configuration record (on page 287). All command-line options are listed, as well as some configuration options important to understanding the generated object program, such as BINARY-ALLOCATION in the COMPILER-OPTIONS configuration record; if no options were specified, these lines will not appear.

## Error Marker and Diagnostics

Violations of syntactical or semantic rules are detected during the compiler’s pass through the source program. If an error is detected, it is undermarked by a dollar sign. Figure 35 illustrates the RM/COBOL diagnostic message format.

**Figure 35: Error Marker and Diagnostics**

```
     1            IDENTIFICATION DIVISION.
     2            PROGRAM-ID.  ALLOCMAP.
0    3            ENVIRONMENT DIVISION                                    0
     4            CONFIGURATION SECTION.
                  $
*****   1) 0319: E Period space separator expected.
0    5            SOURCE-COMPUTER. RMCOBOL.                               0
     6            OBJECT-COMPUTER. same.
                                   $
*****   1) 0382: E Computer-name must be user-defined word instead of
0                   reserved word. (scan suppressed).                      0
*****Previous diagnostic message occurred at line 4.
     7                   PROGRAM COLLATING SEQUENCE EBCDIC-CODE.
                         $
0    *****   1) 0005: I Scan resumed.                                       0
*****Previous diagnostic message occurred at line 6.
     8            SPECIAL-NAMES.
     9                SWITCH-1 IS REPORT-MODE,
```

The first number on the line following the line with the undermark refers to the undermark number. Multiple errors on the same line are numbered in ascending order, reading left to right. The next number is the error number. This corresponds to the appropriate message listed in Appendix B: *Compiler Messages* of the *RM/COBOL Language Reference Manual*.

Following the error number is a single letter that indicates the severity of the error. There are three classes:

1. **I** indicates the message is informational only.
2. **E** indicates a severe error.
3. **W** indicates a warning.

## Error Recovery

The RM/COBOL compiler may display a recovery message along with the error diagnostic. This recovery message is generated if—as often happens—a compilation error interrupts scanning. In this case, the source text is ignored until the compiler finds a recovery point. This minimizes the amount of code you need to examine if an error occurs. See Figure 36 for an illustration.

**Figure 36: Error Recovery Display**

```
   10                  ON STATUS IS REPORT-LIST,
   11                  OFF STATUS IS REPORT-NOLIST;
0  12                  C21 IS TOP-OF-FORM;                              0
                           $
*****  1) 0088: E Wrong code-name in ALPHABET clause. (scan suppressed).
*****Previous diagnostic message occurred at line 7.
0  13                  CONSOLE IS CRT-DISPLAY;                          0
   14                PROCEDURE DIVISION.
                             $
*****  1) 0005: I Scan resumed.
```

The undermark indicates that the compiler did not recognize the alphabet code-name given.

When the compiler encounters an error, it first attempts to make an assumption about what was actually meant. When it can do so, it continues compiling from the point of error, without displaying the “(scan suppressed)” portion of the message.

If it cannot do so, the compiler suppresses scanning until it finds a point where it can begin again. In this case, an undermark indicates where it restarted scanning, and the informational “Scan resumed” message is written. No source text between the undermark associated with the “(scan suppressed)” message and the “Scan resumed” message is compiled. This may result in data-names being undefined if the message occurs in the Data Division.

The diagnostic information described previously is always contained in the listing regardless of the setting of the compiler options. If the L, P, and T Options are all absent (meaning that the listing is not being written to any device), the diagnostic information is written to the standard output device.

## Error Threading

RM/COBOL provides error-threading facilities. By reading the "Previous diagnostic message occurred at line" message, you can trace back through every error encountered during compilation. This message may also appear after the summary listing, to point to the last error in the program.

Compilation always proceeds to the end of the program regardless of the number of errors found, unless an error causes abnormal termination. Global errors, such as undefined paragraph names and illegal control transfers, are listed at the end of the listing file allocation map.

# Compile Command Messages

The banner appears when you first invoke the compiler:

```
RM/COBOL Compiler - Version 9.0n.nn for operating system
Copyright © 1985-2005 by Liant Software Corp.  All rights reserved.
Configured Options: option list

Registration Number:  xx-nnnn-nnnnn-nnnn
```

The third line of the compiler banner appears only when options have been specified in a configuration file or in the Compile Command. Options displayed as a single character appear first. If flagging is configured, the configured keywords appear next; long keywords are abbreviated. If an object pathname or a listing pathname is configured, it appears in the form O=*pathname* or L=*pathname*. If the object version level number is configured, it appears in the form Z=*nn*.

A verbose banner has additional information about the product and environment in which it is running. The verbose banner is obtained for the compiler by setting the environment variable RM_VERBOSE_BANNER to a value that begins with "Y" or "y". The verbose banner adds the following lines to the banner:

```
RM/COBOL: User user-name running on machine machine-name (system-name)
RM/COBOL: Native character set: ncs (Codepage: cp-number)
```

The lines in the verbose banner are not suppressed by the K Runtime Command Option (see page 180).

You may produce a list of the support modules loaded by the RM/COBOL compiler by defining the environment variable RM_DYNAMIC_LIBRARY_TRACE. The listing will indicate which modules are present, such as the Automatic Configuration File module or the Enterprise CodeBench server support module. This information is most helpful when attempting to diagnose a problem with support modules. When the environment variable RM_VERBOSE_BANNER is set to "Y" or "y", the list of support modules is also produced, regardless of the setting of the RM_DYNAMIC_LIBRARY_TRACE environment variable.

If you enter an invalid Compile Command, you will see:

```
Usage:    RMCOBOL name [options]
Options:  [(] [A] [B] [C[=0-3] [D] [E] [F=(keyword list)] [G=cfgfile1]
   [H=cfgfile2] [K] [L[=path]] [M] [N] [O=path] [P] [Q] [R] [S] [T]
   [U[=B|D|P]] [V] [W=workspace] [X] [Y[=0-3] [Z=version] [2] [7]
   [)comments]
```

In addition, the following messages may be displayed:

```
Command line error: file name is missing from command line.

Conflict error: COMPUTATIONAL-VERSION conflicts with OBJECT-VERSION.

Conflict error: option Q conflicts with Y.

Conflict error: option U conflicts with 2.

Conflict error: option V conflicts with B.

Mismatch error: options A and X require option L, P, T, or Y=3.

Mismatch error: option C requires option L, P, or T.

Mismatch error: option E requires option L, P, or T.

Syntax error: flag keyword is incorrect.

Syntax error: option characters must be followed by space or comma.

Syntax error: option C specifies an incorrect numeric value.

Syntax error: option O requires path specification.

Syntax error: option G or H requires path specification.

Syntax error: option U specifies invalid type character.

Syntax error: option Z requires numeric version specification.

Syntax error: symbol n is incorrect option letter.

Syntax error: option W requires numeric workspace specification.

Syntax error: option U describes incorrect type character.

Syntax error: option Y specifies an incorrect numeric debug level.

Version error: value must be greater than 6 and less than or equal to 12.

Version error: value must be greater than 8 for current compiler license.

Error invoking unauthorized copy of compiler.
```

# Compiler Status Messages

The RM/COBOL compiler displays messages that tell you it has completed normally, or that it has terminated abnormally.

One of these messages—Compilation complete—appears every time the compilation is finished, regardless of the number of errors that appear. This message has the following form:

```
Compilation complete–Programs: p, Errors: e, Warnings: w
```

where,

*p* is the number of programs in the source file, excluding contained programs.

*e* is the number of errors found.

*w* is the number of warnings issued.

The other messages are displayed under specific circumstances. They are listed in Appendix B: *Compiler Messages* of the *RM/COBOL Language Reference Manual*.

If a compilation error results in an abnormal termination, a message is displayed in the following general form:

```
Compiler error n:  text
```

where,

*n* is a compiler error number.

*text* is any of the first sentences in the following definitions.

The numbers and their definitions are listed in Table 14.

In addition to these errors, unnumbered error messages may appear as a result of configuration or I/O errors. These unnumbered error messages are described beginning on page 173.

**Table 14: Abnormal Termination Messages**

| Error Number | Message Text | | |
|---|---|---|---|
| **1** | **Compiler limit exceeded, T*nn message*.** | | |
| | The program has exceeded an internal compiler limit. This can be remedied by dividing the program into a main program with multiple subprograms. The table number and table usage are included in the message to provide additional information to help keep the program in conformance with compiler limits. If this error continues to occur even in a small program, it suggests an internal compiler malfunction. Provide a source copy and the table number as it appears in this message to Liant technical support services.<br>The values of *nn* are listed as table numbers, and the values for *message* are listed as table usage in the following table. Limits are provided only where meaningful; all compiler tables are listed since error number 2 also displays this information. | | |
| | **Table Number** | **Table Usage** | **Limit** |
| | T00 | Source (input source records, contiguous comments) | 28000 |
| | T01 | AliasID (aliased identifiers) | 65534 |
| | T02 | Alter (ALTER statements) | 65534 |
| | T03 | BackPatchPsect (object back patches) | |
| | T04 | Cfd (COBOL file definers) | 65534 |
| | T05 | Code (object code buffer) | |
| | T06 | Condition (condition-names) | 65534 |
| | T07 | Corresponding (CORRESPONDING items for MOVE, ADD, or SUBTRACT) | |
| | T08 | CrossRef (cross reference entries) | 290000 |
| | T09 | DataParameter (forward data references, for example, FILE STATUS) | 65534 |
| | T0A | DataRecord (DATA RECORDS clause references) | |
| | T0B | DeclarativeRefError (declarative reference errors) | |
| | T0C | DeferredScript (deferred subscripting in Screen Section) | 65534 |
| | T0D | DimensionTemp (table dimensions in subscripting) | |
| | T0E | Dsect (data descriptions for data references) | 65534 |
| | T0F | ErrorID (identifier errors discovered after the definition) | |
| | T10 | ErrorMessage (diagnostic messages for current line) | |
| | T11 | ErrorProcedure (procedure errors) | |
| | T12 | Error (diagnostic message entries) | |
| | T13 | ErrorTemp (diagnostic message temporaries) | |
| | T14 | Exit (stacked internal exit locations) | |
| | T15 | External (external data items or files) | |
| | T16 | Fail (stacked recovery information for parsing errors) | |
| | T17 | FileArea (file areas for SAME [RECORD] AREA clauses) | 65534 |
| | T18 | FileAreaTemp (file area temporaries) | |

**Table 14: Abnormal Termination Messages (*Cont.*)**

| Error Number | Message Text | | |
|---|---|---|---|
| | Table Number | Table Usage | Limit |
| | T19 | Fsect (file references) | 65534 |
| | T1A | Global (global data items or files) | 65534 |
| | T1B | Group (group data items stack for a record) | |
| | T1C | ID (identifier definitions) | 65534 |
| | T1D | IndexTempHold (held index temporaries) | |
| | T1E | IndexTemp (index temporaries) | |
| | T1F | InspectTempHold (held INSPECT temporaries) | |
| | T20 | InspectTemp (INSPECT temporaries) | |
| | T21 | IntegerConstant (integer constants) | 65534 |
| | T22 | Jsect (procedure references) | 65534 |
| | T23 | Label (made intra-statement labels) | 65534 |
| | T24 | LiteralCharacter (literal characters) | 65534 |
| | T25 | LiteralRef (literal references) | 65534 |
| | T26 | Literal (literal descriptors) | 65534 |
| | T27 | LiteralTemp (literal temporaries) | |
| | T28 | LiteralValue (literal values) | 65534 |
| | T29 | LocalSymbol (local symbol information for object symbol table) | |
| | T2A | NameLink (user-defined word links) | 98133 |
| | T2B | Name (user-defined words) | 98133 |
| | T2C | NextSentenceLabel (NEXT SENTENCE labels) | |
| | T2D | NumericTemp (numeric temporaries) | 65534 |
| | T2E | Operand (statement operands) | |
| | T2F | PackTemp (character packing temporaries) | |
| | T30 | ParameterText (diagnostic message parameter text) | |
| | T31 | Partial (partial segments) | 65534 |
| | T32 | Perform (PERFORM statements) | 65534 |
| | T33 | PictureTemp (PICTURE character-string temporaries) | |
| | T34 | PointerTemp (pointer temporaries stack) | |
| | T35 | PointerTempHold (pointer temporaries save) | |
| | T36 | Polish (expression evaluation Polish) | |
| | T37 | PolishTemp (expression evaluation Polish temporary) | |
| | T38 | PrecomputeRef (precomputed subscripting or reference modification) | 65534 |
| | T39 | Preset (initial VALUE clause values) | |
| | T3A | ProcedureRef (procedure references) | 65534 |

**Table 14: Abnormal Termination Messages (*Cont.*)**

| Error Number | Message Text | | |
|---|---|---|---|
| | **Table Number** | **Table Usage** | **Limit** |
| | T3B | Procedure (procedure definitions) | 65534 |
| | T3C | ProgramName (program-names) | |
| | T3D | ProgramNest (contained programs) | |
| | T3E | Program (programs referenced by CALL statements) | 65534 |
| | T3F | Qualifier (qualifiers in identifiers) | |
| | T40 | QualifierTemp (qualifier temporaries) | |
| | T41 | RecordKey (record keys) | 65534 |
| | T42 | RecordKeyTemp (record key temporaries) | |
| | T43 | RefMod (reference modifiers) | 65534 |
| | T44 | ReplaceKey (REPLACE statement keys) | 65534 |
| | T45 | ReplaceText (REPLACE statement text) | 65534 |
| | T46 | ReplacingKey (REPLACING phrase keys in COPY statements) | 65534 |
| | T47 | ReplacingText (REPLACING phrase text in COPY statements) | 65534 |
| | T48 | SameSortArea (SAME SORT AREA list) | |
| | T49 | ScreenAttributes (Screen Section data item attributes) | 65534 |
| | T4A | ScreenGroup (Screen Section groups) | |
| | T4B | ScriptPlex (subscripted reference entries) | 65534 |
| | T4C | ScriptRef (subscripted references) | 65534 |
| | T4D | ScriptTemp (subscript temporaries) | |
| | T4E | Segment (Procedure Division segments) | 65534 |
| | T4F | SortMergeBlock (SORT and MERGE statements) | |
| | T50 | SourceTemp (input source character temporaries) | |
| | T51 | SpecialRegister (special register references) | 65534 |
| | T52 | Symbol (user-defined word temporaries) | |
| | T53 | SystemNames (implementor-names) | 65534 |
| | T54 | TableIndex (INDEXED BY phrases of OCCURS clauses) | |
| | T55 | TableKey (KEY phrases of OCCURS clauses) | 65534 |
| | T56 | Table (OCCURS clauses) | 65534 |
| | T57 | UndefinedProcedure (undefined procedure references) | |
| | T58 | UsingID (Procedure Division header USING list) | 256 |
| | T59 | Work (compiler data stack) | 65534 |

**Table 14: Abnormal Termination Messages (*Cont.*)**

| Error Number | Message Text |
|---|---|
| **2** | **Table memory overflow, T*nn message*.** |
| | The program has exceeded the available workspace when adding information to the indicated compiler table. Increase the amount of user space available to the compiler with the W Option, reduce the program size by dividing the program into a main program with multiple subprograms or by using segmentation, or use shorter data-names.<br>The values of *nn* are listed as table numbers, and the values for *message* are listed as table usage in the table provided above for error number 1. Note that the table listed is not necessarily one of the tables causing the problem; it may simply be the table being increased in size when the operating system refuses to provide more memory to the compiler. |
| **3** | **Program data or code overflow.** |
| | The program exceeded an internal compiler limit. The listing file shows whether a data or procedure overflow occurred.<br>One of the object sections has run out of space. Segmenting the program or dividing it into a main program with multiple subprograms may solve a procedure overflow. Reducing the size of data items described in the Data Division may solve a data overflow condition.<br>A program overflow can also occur if the program has too many source lines; that is, a Procedure Division header that begins at line 65536 or higher or more than 65535 lines of code in the Procedure Division and the object version is restricted to less than 12 and the Q Compile Command option is not specified or configured. Object version 12 is required to properly support debugging line numbers in excess of 65535. |
| **4** | **Internal logic error, <*error location information*>** |
| | An internal compiler error has been encountered. If this problem arises, call Liant technical support services for assistance. The <*error location information*> included in this message may help determine the cause of this malfunction and should be recorded for reference. |
| **5** | **Fatal syntax error.** |
| | The compiler has encountered a syntax error from which it cannot recover. Fix the syntax error in the source program and then compile the program again. |
| **6** | **Object file overflow.** |
| | The object file has become too large for the compiler to produce a correct object file. Break the program into two or more smaller programs that communicate using the CALL statement. |
| **7** | **Internal logic error, <*error location information*>** |
| | An internal compiler error has been encountered: an invalid compiler table number (roll) has been referenced. If this problem arises, call Liant technical support services for assistance. The <*error location information*> included in this message may help determine the cause of this malfunction and should be recorded for reference. |
| **8** | **Internal logic error, <*error location information*>** |
| | An internal compiler error has been encountered: an erroneous compiler table entry number (group) has been referenced. If this problem arises, call Liant technical support services for assistance. The <*error location information*> included in this message may help determine the cause of this malfunction and should be recorded for reference. |

**Table 14: Abnormal Termination Messages (*Cont.*)**

| Error Number | Message Text |
|---|---|
| **9** | **Internal logic error, *<error location information>*** |
| | An internal compiler error has been encountered: an erroneous compiler table entry offset (rung) has been referenced. If this problem arises, call Liant technical support services for assistance. The *<error location information>* included in this message may help determine the cause of this malfunction and should be recorded for reference. |
| ***N*** | **Unknown error number.** |
| | The value of *n* was not 1 through 9, inclusive. If this occurs, call Liant technical support services for assistance. |

In the unnumbered error messages described below, *pathname* may be one of the following: a valid pathname, PRINTER if the P Option is used, or the standard output device if the T Option is used.

#### Open error for file *pathname*

This message may occur for program listings or object files. There are a number of reasons this message may appear, including:

- A write-protected file was opened for output.
- The system is out of disk space.
- An invalid pathname was specified.
- A file exists, but RM/COBOL could not locate it because the directory search sequence was not specified or was specified incorrectly.
- The system has reached its limit for the number of files that can be open at one time.

#### Write error for file *pathname*

This message can occur for both listing and object files. Generally, it means space is not available to perform the write operation.

#### Read error for file *pathname*

This message can occur for source and object files. Generally, it indicates a corrupted file. The error will also occur on source files that contain a NULL character. Restore the file from its backup copy, or, for object files, restart the compilation.

# Compiler Configuration Errors

Compiler configuration errors include all errors that occur because of an error in the configuration. The formats are as follows:

```
Error code at record number in location.
Error code in configuration.
```

where, *code* is one of the following:

- The compiler configuration error number listed in Table 15. See also Configuration Errors (on page 284).
- An input/output error, as described in Input/Output Errors (on page 370).

*number* is the logical record in the configuration file where the error occurred. When using *number* to determine which record is in error, count lines combined with their continuation lines as one record, and do not count comment lines or blank lines.

*location* identifies one of the following sources of configuration records:

- Attached configuration or automatic configuration file
- Overriding configuration file
- Supplemental configuration file

Attached configuration or automatic configuration file refers to configuration files either attached to the executable on Windows or located automatically by the automatic configuration support module. For more information, see Automatic and Attached Configuration Files (on page 282). Overriding configuration file refers to a configuration file specified by the G Option (see page 142). Supplemental configuration file refers to a configuration file specified by the H Option (see page 142).

The format with the record number and filename appears if an error is detected during the processing of a configuration record. The text of the configuration record in error follows the message. The other format is used if an error is detected after all configuration records have been processed or if an error is detected without an associated record.

**Table 15: Compiler Configuration Errors**

| Code | Description |
|---|---|
| E002 | An invalid delimiter was found. |
| E004 | A keyword has not been provided where one was expected or the keyword is invalid. |
| E007 | Syntax error. |
| E009 | A value has not been provided where one was expected or the value is invalid. |
| E00B | A logical configuration record exceeds the maximum length. |
| E00C | Token requested to dereserve was not found. |

## Compiler Initialization Errors

If the compiler encounters difficulties initializing the RM/COBOL file management system, the following error message will appear:

```
Error initializing file manager.
```

This error generally occurs because a buffer pool has been configured that is too large to be allocated. See the BUFFER-POOL-SIZE keyword (on page 316) of the RUN-FILES-ATTR configuration record for instructions on changing the buffer pool size.

## Support Module Version Errors

During initialization, the compiler locates and loads various support modules, including the automatic configuration support module. Also, at initialization, the compiler verifies that each support module is the correct version for the runtime system. If a support module is not the correct version, the following message is displayed:

```
RM/COBOL:  module-name version mismatch, expected 9.0n.nn,
   found n.nn.nn.
```

When the previous message is displayed, the compiler terminates with the following message:

```
Error invoking mismatched compiler and support module.
```

# Compiler Exit Codes

Compiler exit codes indicate the status of the compilation (either successful or unsuccessful). These codes and their associated definitions are listed in Table 16.

Under UNIX, the exit code can be interrogated from the shell. See shell (**sh**) in your UNIX documentation for details.

Under Windows, a non-zero exit code is displayed in a message box titled “Return Code”. Selecting the OK button closes the compiler window. The message box also will contain the COBOL error code, if one occurred. Display of the Return Code message box may be disabled by setting the RM/COBOL Windows registry value “Show Return Code Dialog” to False. For more information, see Setting Control Properties (on page 70).

If the compiler was invoked from a COBOL program using the SYSTEM non-COBOL subprogram (CALL “SYSTEM”), the exit code can be retrieved by passing an exit code variable in the USING list. For more information, see the SYSTEM (on page 578) subprogram.

**Table 16: Compiler Exit Codes**

| Code | Description |
|---|---|
| 0 | Normal termination. |
| 249 | Warnings in program. |
| 250 | System initialization error. |
| 251 | Incorrect Compile Command. |
| 252 | Errors in program. |
| 253 | Reserved. |
| 254 | Compilation canceled (by pressing the CTRL and BREAK keys or the system Interrupt key). |
| 255 | Compiler error. |

# Chapter 7: Running

One of the immediate results of compilation is the creation of the object file. Object files contain the object version of the source program. The object version of the program can be executed with the runtime command. To execute the object program, use the RM/COBOL Runtime Command described in the following section.

If your program uses segmentation, the segments are loaded and executed—as they are referenced—by the RM/COBOL runtime system. The runtime system also allocates memory for file buffers, external data items, and called RM/COBOL and non-COBOL subprograms.

This chapter contains information on the RM/COBOL Runtime Command, **runcobol**, and its options, examples of valid and invalid runtime commands, runtime messages, and program exit codes.

## Runtime Command

The RM/COBOL Runtime Command (**runcobol**) loads and executes RM/COBOL programs.

Under UNIX, the Runtime Command is entered at a shell prompt. After typing the desired command and options, press Enter to begin execution.

Under Windows, the Runtime Command can be entered in the Command line text box of the Create Shortcut dialog box. For instructions on creating a shortcut, see Creating a Windows Shortcut (on page 58). Choose the RUNCOBOL icon to begin execution. Programs also may be executed by dragging the **.cob** object file to the RUNCOBOL object or by double-clicking on the object file.

The format of the Runtime Command is as follows:

```
runcobol filename [option] ...
```

*filename* is the name of the main program of the run unit. If the L Option (described in the next paragraph) is not specified, *filename* must be a valid pathname. If a filename extension is not specified, RM/COBOL uses first **.cob**, and then **.COB** unless configured otherwise.

The L Option allows you to invoke execution of a program contained in a library by entering the name of the library only. See the discussion of libraries (on page 136). The rules for invocation of libraries are as follows:

1. If the main program is not in a library, you must enter the appropriately qualified pathname for *filename*.
2. If the main program is in a library, you must enter the L Option and the library name containing the main program. The main program-name specified by *filename* must have been specified in the PROGRAM-ID paragraph of the program.

*option* specifies the available RM/COBOL Runtime Command options, described in the next section. Spaces or commas must separate options. Options may be entered in uppercase or lowercase letters. Each option may be preceded by a hyphen. If any option is preceded by a hyphen, then a leading hyphen must precede all options. When a value is assigned to an option, the equal sign is optional if leading hyphens are used. In general, command-line options are processed from left to right and, for most options, the last value encountered is the one used.

**Note** More than one L Option may be specified without one overriding the other. See the description of the L Option on page 183 for more information.

In addition, the RM/COBOL for Windows runtime system also supports the following command-line options, which do not follow the command format described earlier in this section:

- Three OLE server registration commands. These options are described in Runtime Registration (on page 56).

```
runcobol /regserver
runcobol /unregserver
runcobol /showserver
```

- Three character-set commands. These options are described in Character Set Considerations for Windows (on page 97).

```
runcobol /cs_ansi
runcobol /cs_oem
runcobol /showcharset
```

# Runtime Command Options

Runtime Command options can be specified in the following three ways:

1. They can be placed into the registry (on Windows) or the resource files (on UNIX). In the registry, the Command Line Options property (on page 72) provides command-line options for the runtime when Runtime is selected on the Select File tab of the RM/COBOL Properties dialog box. In resource files, the Options keyword, which is described in Command Line Options (on page 29), provides command-line options for the runtime in the global resource file **/etc/default/runcobolrc** and the local resource file **~/.runcobolrc**.
2. They can be specified in the Runtime Command itself.
3. They can be placed into a configuration file, which is processed by the RM/COBOL runtime when the configuration file is automatically located, attached, or specified with a configuration command-line option. For information on configuration files, see Automatic and Attached Configuration Files (on page 282) or Configuration Runtime Command Options (on page 179). For a discussion of the runtime options that can be configured, see the RUN-OPTION configuration record (on page 322) and the RUN-SORT configuration record (on page 327).

Options are processed in the order given above, but options specified in the configuration do not override options specified in the resultant set of command-line options as determined from items 1 and 2 above. This means that options specified in a Runtime Command will take precedence over conflicting or contradictory options specified by the registry or resource files (step 1) or configuration (step 3).

There are four groups of Runtime Command options:

1. Configuration (see the following topic)
2. Debug and Test (on page 181)
3. Environment (on page 181)
4. Program (on page 182)

## *Configuration Runtime Command Options*

The following options designate a file to be used as the complete runtime configuration or as a supplement to it and allow suppression of the banner and STOP RUN messages.

**C** Use the C Option to designate a file to be used as the runtime configuration. If the C Option is specified, any attached or automatic configuration is ignored (not processed). The C Option has the following format:

```
C=pathname
```

See the discussion of the RUN-OPTION configuration record (on page 322). See also the X Option on page 180.

The default is to use the default configuration options described in Chapter 10: *Configuration* (on page 281).

**K** Use the K Option to suppress the banner message and the STOP RUN message. This option is most useful when running under batch command files or shell scripts.

The default is to display the banner and STOP RUN messages.

**V** Use the V Option to display a verbose banner, including a list of the support modules (shared objects on UNIX and dynamic load libraries on Windows) loaded by the RM/COBOL runtime system. For UNIX, this list will indicate which Terminal Interface support module is being used and which other optional modules are present, if any. For both UNIX and Windows, the list will include any non-COBOL modules loaded because of the L Runtime Command Option (see page 183). The list indicates the full pathname of the support module, so the location of the loaded file can be determined by examining the list. This option is most useful when attempting to diagnose a problem with support modules. For more information, see Appendix D: *Support Modules (Non-COBOL Add-Ons)* on page 429 in this user's guide and the *CodeBridge* manual.

Alternatively, the RM_DYNAMIC_LIBRARY_TRACE environment variable may be defined (with any value) or the V keyword of the RUN-OPTION configuration record may be set to DISPLAY if you wish to see the just the list of support modules. The RM_VERBOSE_BANNER environment variable may be defined with a value that begins with "Y" or "y" to obtain the complete verbose banner, including the list of support modules.

The default is not to display the verbose banner or the list of support modules loaded.

**X** Use the X Option to designate a file as a supplement to the runtime configuration. The specified file is processed after any attached or automatic configuration and after any file specified in the C Option, but before any other command-line options are processed. The X Option has the following format:

```
X=pathname
```

If no configuration exists (either attached or automatic, or specified with the C Option, described previously in this section), the specified file serves as the complete configuration. For more information, see Automatic and Attached Configuration Files (on page 282).

## Debug and Test Runtime Command Options

The following options invoke the RM/COBOL Interactive Debugger and collect program instrumentation data.

**D** Use the D Option to invoke the RM/COBOL Interactive Debugger (called Debug). Complete details on program debugging are contained in Chapter 9: *Debugging* (on page 245).

By default, the Interactive Debugger is not invoked.

**I** Use the I Option to collect RM/COBOL program instrumentation data. Complete details on program instrumentation are contained in Chapter 11: *Instrumentation* (on page 351).

By default, instrumentation data is not collected.

## Environment Runtime Command Options

The following options specify the runtime environment.

**B** Use the B Option to specify a maximum buffer size for use with the ACCEPT and DISPLAY statements. The B Option has the following format:

**B**=*n*

The maximum buffer size is 65280 characters. The default size is 264 characters.

For information about ACCEPT and DISPLAY buffers, see page 45.

**F** Use the F Option to specify a fill character value. The fill character value is used to initialize read-write memory allocated for the run unit. Working-Storage data items that do not specify a VALUE clause in their data description entry will be filled with this character value at program load time. The F Option has the following format:

**F=**<*fill-char*>

where, <*fill-char*> can be a single character, a decimal number from 00 to 255, or a hexadecimal number from 0x00 to 0xFF. The single character digits 0 through 9 represent the ASCII digit characters (“0” – “9”, 0x30 – 0x39, or decimal 48 – 57), so if they are meant as numeric code points, a leading 0 is required to make them more than a single character. Quoting the fill character value is allowed, but has no effect on whether the value is interpreted as a number or a character.

The default is to use a space character to fill read-write memory allocated for the run unit.

**M** Use the M Option to direct that level 2 ANSI semantics are to be used for Format 1 ACCEPT and DISPLAY statements.

The default is to use level 1 ANSI semantics in these situations (see the discussion that begins on page 45).

**S** Use the S Option to set (or reset) the initial value of switches in the RM/COBOL program. The S Option has the following format:

**S=***n ... n*

Switches are numbered left to right from 1 to 8, without trailing zeroes. Each *n* indicates a switch value: 0 indicates OFF and 1 indicates ON.

The default is to initialize all switches to OFF at the start of the run unit.

**T** Use the T Option to specify the amount of memory (*n* bytes) to be used for a sort operation. The T Option has the following format:

**T**=*n*

There are a number of reasons to use the T Option in association with a sort operation:

- To increase the amount of memory available for the sort operation, thereby increasing the efficiency of the sort operation.
- To reduce the default memory allocation. This provides more room for loading other data or called subprograms into memory during an input procedure.

If no SORT or MERGE statement is used in the run unit, using a value of 0 will allow the runtime system to allocate the memory generally used to contain the sort-merge logic for other purposes.

The default is 256000 bytes. The maximum allowed value is 2147483647 bytes.

## *Program Runtime Command Options*

The following options define an argument to be passed to the main program and the object libraries to be used for the run unit.

**A** Use the A Option to pass an argument to the main program. The A Option has the following format:

**A=**[*delim*]*string*[*delim*]

where, *string* is an alphanumeric series of characters.

The delimiter character specified for *delim* may be either `'` or `"`.

The delimiter character chosen as the opening delimiter must be used as the closing delimiter as well. The closing delimiter must be followed by a space or comma if another option follows the A Option. The delimiter character used cannot appear as part of *string*.

If *string* contains no spaces, delimiter characters are not required.

Under UNIX, it is safer to delimit *string* using single quotation marks `'...'` because characters in the argument might otherwise be meaningful to the shell (sh) command interpreter.

To use the string assigned to the A option, you must have a Linkage Section for the main program with the following form:

```
01 MAIN-PARAMETER.
   02 PARAMETER-LENGTH  PIC S9(4) BINARY (2).
   02 PARAMETER-TEXT.
      03 PARAMETER-CHAR PIC X OCCURS 0 TO 100 TIMES
               DEPENDING ON PARAMETER-LENGTH.
```

The Procedure Division header should have the following form:

```
PROCEDURE DIVISION USING MAIN-PARAMETER.
```

The variable PARAMETER-LENGTH contains the number of characters between delimiter characters. PARAMETER-TEXT contains a copy of the characters between the delimiter characters. If no parameter is passed with the A Option and the main program describes a parameter as shown above, the value of PARAMETER-LENGTH will be zero. When this is the case, PARAMETER-TEXT should not be referenced. In all cases, no part of MAIN-PARAMETER should be modified.

The number of characters between the delimiter characters cannot exceed 100.

The following is an example program using the A Option.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  CPASS.
*  To see a command-line argument string passed to a
*  COBOL main program, run this program as follows:
*     runcobol cpass A="string"
 DATA DIVISION.
 LINKAGE SECTION.
 01  APARAM.
     02  APARAM-SIZE    PIC S9(4) BINARY (2).
     02  APARAM-STRING.
         03 FILLER      PIC X OCCURS 0 TO 100 TIMES
                         DEPENDING ON APARAM-SIZE.
 PROCEDURE DIVISION USING APARAM.
 0010-BEGIN.
     DISPLAY APARAM-STRING(1:) LINE 22 ERASE.
     STOP RUN.
 END PROGRAM CPASS.
```

**L** Use the L Option to designate RM/COBOL object or non-COBOL subprogram libraries. The L Option has the following format:

**L=***pathname*

These libraries allow more than one program to be contained within a file. RM/COBOL imposes no limitation on the number of times the L Option may appear in a single Runtime Command. Multiple uses of the L Option are processed cumulatively from left to right as they are encountered on the command line. For additional information on how the libraries specified by the L Option are searched, see the discussion of Subprogram Loading (on page 214). Further information on RM/COBOL libraries may be found in the topic Libraries (on page 136). You can also learn more about non-COBOL libraries in

the appropriate appendixes in the *CodeBridge* manual on the non-COBOL subprogram internals for Windows and UNIX.

**Q** Use the Q Option to indicate that the program is being scheduled by the Message Control System (MCS) to process a message. The Q Option has the following format:

```
Q=[delim]string[delim]
```

where, *string* is an alphanumeric series of characters.

*delim* may be either of the delimiter characters double quote (") or single quote ('). The delimiter character chosen as the opening delimiter must be used as the closing delimiter as well. The closing delimiter must be followed by a space or comma if another option follows the Q Option. The delimiter character cannot appear as part of *string*. If *string* contains no spaces, delimiter characters are not required. Under UNIX, it is safer to delimit *string* using the single quotation (') delimiter because characters in the argument might otherwise be meaningful to the shell (sh) command interpreter.

The value of *string* is moved into the SYMBOLIC QUEUE, SYMBOLIC SUB-QUEUE-1, SYMBOLIC SUB-QUEUE-2, and SYMBOLIC SUB-QUEUE-3 fields (4 * 12 characters each = 48 characters total) of a CD FOR INITIAL INPUT or in the SYMBOLIC TERMINAL field (12 characters) of a CD FOR INITIAL I-O. When the indicated fields in the CD FOR INITIAL record area are not spaces, it indicates that the program was scheduled by the MCS to process a message. Thus, the Q option is intended for use in a script used by the MCS to schedule a run unit to process a message.

When the main program does not contain a CD FOR INITIAL, the Q Option, if specified, is ignored.

When the Q Option is omitted and the main program contains a CD FOR INITIAL, the specified fields of the initial CD contain spaces. This indicates that the program was not scheduled by the MCS to process a message.

See the rules for the communication description entry in the *RM/COBOL Language Reference Manual* for additional information about a CD FOR INITIAL.

# Sample Runtime Commands

Following are examples of valid and invalid RM/COBOL Runtime Commands.

## Valid Runtime Commands

```
runcobol payroll B=500,K
```

This command executes the program named **payroll.cob**. It establishes a maximum buffer size of 500 bytes for ACCEPT and DISPLAY statements (the B Option) and suppresses banner and STOP RUN messages (the K Option).

```
runcobol FIRSTPRG L=lib1\library.cob,D
```

This command executes the program FIRSTPRG contained in the RM/COBOL library named **lib1\library.cob**.

It informs the runtime system of the name of the library (the L Option) that contains the programs available, and invokes the Interactive Debugger (the D Option).

## Invalid Runtime Commands

```
runcobol payroll.sal A='PRINT-RUN"
```

Here, the A Option is invalid, since the opening and closing delimiters are not identical.

```
runcobol lib1\library D T
```

In this example, the library used in the valid example cannot be executed by this command, assuming the library contains more than one program. Also, the T Option is specified without an associated value.

# Runtime Messages

Messages of different classes may appear on the screen during program execution. The message types are defined in the following paragraphs.

## Diagnostic Messages

Diagnostic messages indicate either that an internal RM/COBOL error occurred or that an I/O error occurred that was not handled by an appropriate USE procedure (see the description of the USE statement in Chapter 5: *Procedure Division* of the *RM/COBOL Language Reference Manual*). If the D Option (see page 181) was entered in the Runtime Command and one of these errors occurs, the Interactive Debugger will be entered to allow examination of program data values. Otherwise, control will return to the operating system.

## Execution Messages

Execution messages report the status of the runtime system, or problems within the RM/COBOL program that prevent successful execution.

These messages result from normal program termination, the presence of a STOP *literal* statement, or an incorrectly entered option.

Diagnostic and execution error messages are detailed in Appendix A: *Runtime Messages* (on page 357).

# Program Exit Codes

An appropriate exit code is returned to the operating system when program execution ends. The exit code may indicate that execution was successful or unsuccessful. Users may move (or otherwise assign) any exit code value in the range 0 through 255 to the implicitly defined RETURN-CODE special register. The program exit codes are listed and defined in Table 17.

Under UNIX, the exit code can be interrogated from the shell. See shell (sh) in your UNIX documentation for details.

Under Windows, a non-zero exit code is displayed in a message box titled “Return Code”. Choosing the OK button closes the runtime window. The message box also will contain the COBOL error code, if one occurred. Display of the Return Code message box may be disabled by setting the RM/COBOL Windows registry value Show Return Code Dialog property (on page 81) to False. For more information, see Setting Control Properties (on page 70).

If the runtime system was invoked from a COBOL program using the SYSTEM non-COBOL subprogram (CALL “SYSTEM”), the exit code can be retrieved by passing an exit code variable in the USING list. For more information, see the SYSTEM (on page 578) subprogram.

**Note** User-defined termination codes in the range 249 – 255 will be ambiguous if a runtime system error occurs.

**Table 17: Program Exit Codes**

| Code | Description |
|---|---|
| 0 | Normal termination. |
| 249 | Internal library subprogram called with incorrect parameters. |
| 250 | System initialization error. |
| 251 | Incorrect Runtime Command. |
| 252 | Program load failure. |
| 253 | Program error. |
| 254 | Run unit canceled (by pressing the Ctrl and Break keys or the system Interrupt key). |
| 255 | I/O error. |

# Chapter 8: RM/COBOL Features

This chapter offers operating system-specific information on the use of RM/COBOL statements and on RM/COBOL file types and structure. It is assumed the reader is familiar with RM/COBOL statements.

## ACCEPT and DISPLAY Statements

Specific characteristics of your operating system affect the following aspects of RM/COBOL ACCEPT and DISPLAY statement usage:

- The initial contents of a screen field.
- The use of certain edit keys with the ACCEPT statement.
- The use of phrases with ACCEPT and DISPLAY statements.
- Redirection and piping of standard input and standard output. For more information, see Chapter 2: *Installation and System Considerations for UNIX* (on page 17).

### Initial Contents of a Screen Field

Depending on the current configuration and the phrases specified in the ACCEPT statement, the initial contents of a screen field may be the following:

- Unchanged but treated as if the field contained all spaces. This is the default if neither the PROMPT nor UPDATE phrase is specified, and if the ACCEPT-FIELD-FROM-SCREEN keyword (as described on page 313) of the RUN-ATTR record is not specified or is set to NO in the configuration file.
- Unchanged if neither the PROMPT nor UPDATE phrase is specified and the ACCEPT-FIELD-FROM-SCREEN keyword of the RUN-ATTR record is set to YES in the configuration file.
- Filled with prompt characters if the PROMPT phrase is specified in the ACCEPT statement.
- Filled with the current value of the associated ACCEPT operand if the UPDATE phrase is specified in the ACCEPT statement.

- Filled with the literal characters specified with the MASK keyword of the CONTROL phrase, if that CONTROL phrase is specified. If UPDATE is also specified, or the ACCEPT-FIELD-FROM-SCREEN keyword of the RUN-ATTR configuration record is set to YES, then the input character positions specified in the mask are replaced by the contents of the ACCEPT operand or the current contents of the screen field, respectively.

You can then modify the contents of the screen field. Except for literal characters specified with the MASK keyword of the CONTROL phrase, all positions of that field can be modified until a field termination key is pressed. This modification of displayed data is called field editing.

## Defined Keys

The following sections list and explain the specially defined screen field editing keys and the keys that generate field termination key codes.

Before these keys can function as described under UNIX, you must associate them with the definition in the termcap or terminfo database, as described in Terminal Input and Output (on page 33), and detailed in Chapter 10: *Configuration* (on page 281). For example, the Left Arrow key might be associated with the **k1** termcap attribute.

## Edit Keys

Table 18 describes the keys used to manipulate the cursor during field editing.

**Table 18: Edit Keys**

| Key | CONTROL Phrase | Action |
|---|---|---|
| Left Arrow | Default | Moves the cursor left one character without affecting any input characters. If the cursor is already at the leftmost character in the screen field, a beep sounds. |
| | MASK | Same as above; however, the cursor skips over literal characters that were specified in the mask. |
| Right Arrow | Default | Moves the cursor right one character without affecting any input characters. If the cursor is already at the rightmost character in the screen field, a beep sounds. |
| | MASK | Same as above; however, the cursor skips over literal characters that were specified in the mask. |
| Backspace | Default | Moves the cursor left one character, and deletes the input character in that position. All characters to the right of the deleted characters are shifted to the left. The prompt character (or a space if the PROMPT phrase was not specified) is used to pad the screen field on the right. If the cursor is already at the leftmost character in the screen field, a beep sounds. |
| | MASK | Same as above; however, if the character to the left of the cursor is a literal character, the cursor is moved left until another input character is encountered, and that character is deleted without altering any subsequent input characters. |
| Delete Character | Default | Deletes the input character at the cursor position. All screen field characters to the right of the cursor are shifted to the left. The cursor remains stationary. The prompt character (either as specified in the PROMPT phrase, or spaces if the PROMPT phrase was not specified) is used to pad the screen field on the right. If the cursor is positioned at the right margin when this key is pressed, and no characters are deleted, a beep sounds. |
| | MASK | Same as above; however, only input characters up to the next literal character to the right are shifted to the left. |
| Erase Entire | Default | Places the cursor at the leftmost field position, and fills all input positions with the prompt character, or spaces if the PROMPT phrase was not specified. Note that the Erase Entire key is not a field terminator. |
| | MASK | Same as above; however, literal characters in the mask are not overwritten. |

**Table 18: Edit Keys *(Cont.)***

| Key | CONTROL Phrase | Action |
|---|---|---|
| Erase Remainder | Default | Without moving the cursor, fills all input positions from the current cursor position to the rightmost position of the screen field with the prompt character, or spaces if the PROMPT phrase was not specified. Note that the Erase Remainder key is also a field termination key. |
| | MASK | Same as above; however, literal characters in the mask are not overwritten. |
| Insert Character | Default | Initializes insert mode. Subsequent keystrokes insert characters at the cursor position. Screen field characters to the right of the cursor are shifted further to the right to accommodate the inserted characters. If an attempt is made to shift any character except for a space or a prompt character (if the PROMPT phrase was specified) beyond the rightmost input position of the screen field, a beep sounds.<br>Insert mode is canceled when you press a field termination key or any screen field editing key other than Insert Character. |
| | MASK | Same as above; however, an attempt to shift an input character past a literal character specified in the mask is rejected and results in a beep. |

## Keys That Generate Field Termination Codes

Table 19 lists the keys that generate field termination codes and the corresponding codes. The table lists the default termination codes that the runtime system returns for the indicated PC keyboard keys and for the input sequences from the terminfo and termcap interfaces. The keys may be configured to perform different actions and return different codes. For information on these specifications, see the TERM-INTERFACE configuration record (on page 342).

An entry in brackets ([ ]) next to a terminfo or termcap entry identifies an alternate actual input sequence that will generate the same termination code under UNIX. These predefined input sequences are implied by the terminfo and termcap databases, as they have no defined terminfo or termcap name.

**Note** Any key not covered by footnote 1 in Table 19 causes the ON EXCEPTION imperative sequence.

The generic key name is described in the “EXCEPTION and NOT EXCEPTION Phrases” section of the ACCEPT statement in Chapter 6: *Procedure Division Statements* of the *RM/COBOL Language Reference Manual*. For more information, see Table 34: RM/COBOL Generic Exception Keys on page 341.

**Table 19: Keys that Generate Field Termination Codes**

| PC Keyboard Key | Terminfo Name | Termcap Name | Generic Name | Termination Code |
|---|---|---|---|---|
| F1 | kf1 | k1 | Function 1 | 01 |
| F2 | kf2 | k2 | Function 2 | 02 |
| F3 | kf3 | k3 | Function 3 | 03 |
| F4 | kf4 | k4 | Function 4 | 04 |
| F5 | kf5 | k5 | Function 5 | 05 |
| F6 | kf6 | k6 | Function 6 | 06 |
| F7 | kf7 | k7 | Function 7 | 07 |
| F8 | kf8 | k8 | Function 8 | 08 |
| F9 | kf9 | k9 | Function 9 | 09 |
| F10 | kf10 | k; | Function 10 | 10 |
| F11 | kf11 | F1 | Function 11 | 11 |
| F12 | kf12 | F2 | Function 12 | 12 |
| Shift+F1 | kf11 | F1 | Function 11 | 11 |
| Shift+F2 | kf12 | F2 | Function 12 | 12 |
| Shift+F3 | kf13 | F3 | Function 13 | 13 |
| Shift+F4 | kf14 | F4 | Function 14 | 14 |
| Shift+F5 | kf15 | F5 | Function 15 | 15 |
| Shift+F6 | kf16 | F6 | Function 16 | 16 |
| Shift+F7 | kf17 | F7 | Function 17 | 17 |
| Shift+F8 | kf18 | F8 | Function 18 | 18 |
| Shift+F9 | kf19 | F9 | Function 19 | 19 |
| Shift+F10 | kf20 | FA | Function 20 | 20 |
| Ctrl+F1 | kf21 | FB | Function 21 | 21 |
| Ctrl+F2 | kf22 | FC | Function 22 | 22 |
| Ctrl+F3 | kf23 | FD | Function 23 | 23 |
| Ctrl+F4 | kf24 | FE | Function 24 | 24 |
| Ctrl+F5 | kf25 | FF | Function 25 | 25 |
| Ctrl+F6 | kf26 | FG | Function 26 | 26 |
| Ctrl+F7 | kf27 | FH | Function 27 | 27 |
| Ctrl+F8 | kf28 | FI | Function 28 | 28 |

[1] *Causes field termination but does not take the ON EXCEPTION imperative sequence.*

[2] *Performs the Erase Remainder.*

[3] *Normal STTY configuration to terminate the runtime system under UNIX.*

[4] *Terminates the runtime system under Windows. Also, the normal STTY configuration to terminate the runtime system under UNIX.*

**Table 19: Keys that Generate Field Termination Codes *(Cont.)***

| PC Keyboard Key | Terminfo Name | Termcap Name | Generic Name | Termination Code |
|---|---|---|---|---|
| Ctrl+F9 | kf29 | FJ | Function 29 | 29 |
| Ctrl+F10 | kf30 | FK | Function 30 | 30 |
| Ctrl+Shift+F1 | kf31 | FL | Function 31 | 31 |
| Ctrl+Shift+F2 | kf32 | FM | Function 32 | 32 |
| Ctrl+Shift+F3 | kf33 | FN | Function 33 | 33 |
| Ctrl+Shift+F4 | kf34 | FO | Function 34 | 34 |
| Ctrl+Shift+F5 | kf35 | FP | Function 35 | 35 |
| Ctrl+Shift+F6 | kf36 | FQ | Function 36 | 36 |
| Ctrl+Shift+F7 | kf37 | FR | Function 37 | 37 |
| Ctrl+Shift+F8 | kf38 | FS | Function 38 | 38 |
| Ctrl+Shift+F9 | kf39 | FT | Function 39 | 39 |
| Ctrl+Shift+F10 | kf40 | FU | Function 40 | 40 |
| | kf41 | FV | Function 41 | 41 |
| | kf42 | FW | Function 42 | 42 |
| | kf43 | FX | Function 43 | 43 |
| | kf44 | FY | Function 44 | 44 |
| | kf45 | FZ | Function 45 | 45 |
| | kf46 | Fa | Function 46 | 46 |
| | kf47 | Fb | Function 47 | 47 |
| | kf48 | Fc | Function 48 | 48 |
| | kf49 | Fd | Function 49 | 49 |
| | kf50 | Fe | Function 50 | 50 |
| | kf51 | Ff | Function 51 | 51 |
| | kf52 | Fg | Function 52 | 52 |
| | kf53 | Fh | Function 53 | 53 |
| | kf54 | Fi | Function 54 | 54 |
| | kf55 | Fj | Function 55 | 55 |
| | kf56 | Fk | Function 56 | 56 |
| | kf57 | Fl | Function 57 | 57 |
| | kf58 | Fm | Function 58 | 58 |
| | kf59 | Fn | Function 59 | 59 |
| | kf60 | Fo | Function 60 | 60 |
| | kf61 | Fp | Function 61 | 61 |
| | kf62 | Fq | Function 62 | 62 |
| | kf63 | Fr | Function 63 | 63 |
| Enter [Ctrl+M][1] | cr [Ctrl+M] | cr [Ctrl+M] | Enter | 13 |
| Ctrl+Shift+K[1, 2] | | | | 13 |

**Table 19: Keys that Generate Field Termination Codes *(Cont.)***

| PC Keyboard Key | Terminfo Name | Termcap Name | Generic Name | Termination Code |
|---|---|---|---|---|
| →\| | kc3 [Ctrl+I] | K5 [Ctrl+I] | Tab Right | 58 |
| Ctrl+A | | | Function 1 | 01 |
| Ctrl+B | | | Function 2 | 02 |
| Ctrl+C[4] | | | Function 3 | 03 |
| Ctrl+D | | | Function 4 | 04 |
| Ctrl+E | | | Function 5 | 05 |
| Ctrl+F | | | Function 6 | 06 |
| Ctrl+G | | | Function 7 | 07 |
| Ctrl+I | | | Function 9 | 09 (58 on UNIX) |
| Ctrl+J | | | Function 10 | 10 (55 on UNIX) |
| Ctrl+K | | | Function 11 | 11 |
| Ctrl+L | | | Function 12 | 12 |
| Ctrl+N | | | Function 14 | 14 |
| Ctrl+O | | | Function 15 | 15 |
| Ctrl+P | | | Function 16 | 16 |
| Ctrl+Q | | | Function 17 | 17 |
| Ctrl+R | | | Function 18 | 18 |
| Ctrl+S | | | Function 19 | 19 |
| Ctrl+T | | | Function 20 | 20 |
| Ctrl+U | | | Function 21 | 21 |
| Ctrl+V | | | Function 22 | 22 |
| Ctrl+W | | | Function 23 | 23 |
| Ctrl+X | | | Function 24 | 24 |
| Ctrl+Y | | | Function 25 | 25 |
| Ctrl+Z | | | Function 26 | 26 |
| Esc | | | Escape | 27 |
| Ctrl+[ | | | | 27 |
| Ctrl+\[3] | | | | 28 |
| Ctrl+] | | | | 29 |
| Ctrl+6 | | | | 30 |
| Ctrl+ – | | | | 31 |
| Ctrl+Shift+C | kf0 | k0 | Command | 40 |
| Ctrl+Shift+A | ka3 | K3 | Attention | 41 |
| Ctrl+Shift+P | lf0 | l0 | Print | 49 |
| ↑ | kcuu1 | ku | Up Arrow | 52 |
| ↓ | kcud1 | kd | Down Arrow | 53 |
| Home | khome | kh | Home | 54 |

**Table 19: Keys that Generate Field Termination Codes *(Cont.)***

| PC Keyboard Key | Terminfo Name | Termcap Name | Generic Name | Termination Code |
|---|---|---|---|---|
| Ctrl+Shift+N | nel [Ctrl+J] | nw [Ctrl+J] | New Line | 55 |
| \|← | kc1 | K4 | Tab Left | 56 |
| Ctrl+Shift+E | kel | kE | Erase Remainder | 57 |
| Ctrl+Shift+R | kc3 | k5 | Tab Right | 58 |
| Ctrl+Shift+I | kil1 | kA | Insert Line | 59 |
| Ctrl+Shift+D | kdl1 | kL | Delete Line | 61 |
| Ctrl+Shift+S | kb2 | K2 | Send | 64 |
| Ctrl+← | | | | 65 |
| Ctrl+→ | | | | 66 |
| PgUp | kpp | kP | Page Up | 67 |
| PgDn | knp | kN | Page Down | 68 |
| Ctrl+PgUp | | | | 69 |
| Ctrl+PgDn | | | | 70 |
| Ctrl+Shift+1 | | | | 71 |
| Ctrl+Shift+2 | | | | 72 |
| Ctrl+Shift+3 | | | | 73 |
| Ctrl+Shift+4 | | | | 74 |
| Ctrl+Shift+5 | | | | 75 |
| Ctrl+Shift+6 | | | | 76 |
| Ctrl+Shift+7 | | | | 77 |
| Ctrl+Shift+8 | | | | 78 |
| Ctrl+Shift+9 | | | | 79 |
| Ctrl+Shift+0 | | | | 80 |
| Ctrl+Home | | | | 81 |
| End | ka1 | K1 | End | 82 |
| Ctrl+End | khlp | %1 | Help | 83 |
| N/A | krdo | %0 | Redo | 84 |
| Ctrl+Shift+ – | | | | 85 |
| Ctrl+Shift+= | | | | 87 |

Table 20 lists the default semantic actions that the runtime system returns for input sequences from the different terminal interfaces under UNIX.

**Table 20: Default Semantic Actions**

| Semantic Action | Terminfo Name | Termcap Name |
|---|---|---|
| Left Arrow | kcub1 | kl |
| Right Arrow | kcuf1 | kr |
| Backspace | kbs [Ctrl+H] | kb [Ctrl+H] |
| Set RM Insertion | kich1 | kI |
| Delete Character | kdch1 | kD |
| Erase Entire | kclr | kC |
| Erase Remainder[1] | kel | kE |

[1] *This action also generates a termination code.*

# ACCEPT and DISPLAY Phrases

The CONTROL, ERASE, HIGH, LOW, OFF, and REVERSE phrases affect the use of color attributes with the ACCEPT and DISPLAY statements. The SIZE phrase used with the ACCEPT and DISPLAY statements affects the size of the screen field. The TIME phrase is used to "time-out" the execution of a pending ACCEPT statement. These phrases are defined in the following paragraphs.

## *CONTROL Phrase*

Some of the system dependencies that apply to the CONTROL phrase value concern color-capable terminals. Systems with monochrome terminals ignore color information contained in the CONTROL phrase value. (See the appropriate manufacturer's manual for information on configuring your system with color capability.)

Under UNIX, color requests are processed only if the terminal does not require an attribute byte and if one of the following conditions is met:

1. The terminfo database contains the set_foreground and set_background string sequences. (The back_color_erase and orig_pairs string sequences are not required.) The termcap database contains the **Sb** (set current background color) and **St** (set current foreground color) sequences.
2. A configuration record is present to force the use of ISO Set Graphics Rendition (SGR) sequences when the terminfo information is not available.

The method a terminal uses to process SGR color sequences will vary from one manufacturer to another. When color sequences are sent to monochrome terminals, they are ignored, processed as shades of gray, or represented as characters on the screen. Color sequences sent to color-capable terminals may or may not conflict with other attributes sent to the terminal. For example, sending a color sequence followed by a blink sequence may result in the loss of the color request. RM/COBOL always sends color sequences after all other requested attributes. This prevents areas of the terminal screen from appearing without the desired color. You will need to refer to the terminal manufacturer and the UNIX terminfo documentation in order to determine the sequences necessary to access color

capabilities. For information on the color options, see the TERM-ATTR configuration record (on page 327).

RM/COBOL provides eight system-dependent keywords in the CONTROL phrase that affect an ACCEPT or DISPLAY field: FCOLOR, BCOLOR, GRAPHICS, MASK[1], PASS-THRU[1], PROMPT, REPAINT-SCREEN[1], and SCREEN-COLUMNS[1].

1. **FCOLOR = *color-name***

   When FCOLOR is present, *color-name* specifies the foreground color of the ACCEPT or DISPLAY field. This name is then used as the default value for subsequent ACCEPT and DISPLAY statements in the program.

   See the discussion of the HIGH, LOW and OFF phrases in the following section for information concerning high-intensity colors.

   The initial default for *color-name* is white.

   **Note** Under Windows, the default colors are determined by the Use Windows Colors property (on page 85).

2. **BCOLOR = *color-name***

   When BCOLOR is present, *color-name* specifies the background color of the ACCEPT or DISPLAY field. This value is then used as the default value for subsequent ACCEPT and DISPLAY statements in the program.

   The initial default for *color-name* is black.

   Table 21 contains a list of all the possible names for *color-name*. The left column contains the valid color name. The right column shows the color that appears when high intensity is specified (the default intensity).

   **Note** Under Windows, the default colors are determined by the Use Windows Colors property (on page 85).

**Table 21: Valid COBOL Color Names**

| Valid Color Names | High-Intensity Color Values (Defaults) |
|---|---|
| Black | Gray |
| Blue | Light Blue |
| Green | Light Green |
| Cyan | Light Cyan |
| Red | Light Red |
| Magenta | Light Magenta |
| Brown | Yellow |
| White | High-Intensity White |

[1] These keywords are supported only under RM/COBOL for UNIX.

3. **GRAPHICS**

The GRAPHICS keyword causes the characters in Table 22 to be translated to portable, system-specific line draw characters. Characters that are not listed in Table 22 are output unchanged.

**Table 22: System-Specific Line Draw Characters**

| Description | Single-Line Character | Double-Line Character |
|---|---|---|
| lower-right corner | j(Ù) | J(¼) |
| upper-right corner | k(¿) | K(») |
| upper-left corner | l(Ú) | L(É) |
| lower-left corner | m(À) | M(È) |
| plus | n(Å) | N(Î) |
| horizontal line | q(Ä) | Q(Í) |
| left tee | t(Ã) | T(Ì) |
| right tee | u(´) | U(¹) |
| bottom tee | v(Á) | V(Ê) |
| top tee | w(Â) | W(Ë) |
| vertical line | x(³) | X(º) |

If the requested line draw characters are not available, the runtime system uses the best available characters. If double-line characters are requested and only single-line characters are available, they are used. If no line draw characters are available, then plus-characters, vertical bars, and dashes are used.

For details on how the runtime system determines whether line draw characters are available for a given terminal under UNIX, see page 43.

Here is a sample program that demonstrates how boxes are drawn:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  GRAPHXMP.
PROCEDURE DIVISION.
GRAPHXMP.
    DISPLAY " ", LINE 5 POSITION 1 ERASE.
* Single-line graphics
    DISPLAY "lqqqqwqqqqk", CONTROL "HIGH, GRAPHICS".
    DISPLAY "x    x    x", CONTROL "HIGH, GRAPHICS".
    DISPLAY "tqqqqnqqqqu", CONTROL "HIGH, GRAPHICS".
    DISPLAY "x    x    x", CONTROL "HIGH, GRAPHICS".
    DISPLAY "mqqqqvqqqqj", CONTROL "HIGH, GRAPHICS".
    DISPLAY " ".
* Double-line graphics
    DISPLAY "LQQQQWQQQQK", CONTROL "HIGH, GRAPHICS".
    DISPLAY "X    X    X", CONTROL "HIGH, GRAPHICS".
    DISPLAY "TQQQQNQQQQU", CONTROL "HIGH, GRAPHICS".
    DISPLAY "X    X    X", CONTROL "HIGH, GRAPHICS".
    DISPLAY "MQQQQVQQQQJ", CONTROL "HIGH, GRAPHICS".
END PROGRAM GRAPHXMP.
```

4. **MASK**

A new keyword, MASK, has been added to the CONTROL phrase in ACCEPT and DISPLAY statements. Use the following format:

```
MASK = mask
```

**Note** The MASK keyword is ignored when standard input or standard output is redirected. This keyword is supported only under UNIX.

The MASK keyword in the CONTROL phrase causes a literal mask to be edited into the ACCEPT or DISPLAY screen field. Literal mask characters are inserted into the operand as it is transferred to the screen field if UPDATE is specified, or overlaid onto the screen field if ACCEPT-FIELD-FROM-SCREEN is in effect.

In all cases, the size of the mask determines the size of the actual ACCEPT or DISPLAY screen field. The optional SIZE phrase, or the size of the actual operand, is used only to limit the number of data characters that may be edited and entered into the ACCEPT screen field, or edited into the DISPLAY screen field prior to the screen operation.

The mask is specified in the CONTROL phrase as a literal string with no embedded spaces. If the mask specifies more input positions than are contained in the ACCEPT/DISPLAY operand, then excess mask input positions are replaced by literal spaces. The mask is limited to a total of 80 characters, including escape characters. When a screen field is edited, literal characters specified with the MASK keyword cannot be modified.

Table 23 lists the characters and character sequences that have special meanings in the MASK keyword in a CONTROL phrase string. All other characters are treated as literal characters.

**Table 23: Characters Used with the MASK Keyword of a CONTROL Phrase**

| Character | Meaning |
|---|---|
| X | Specifies an input/output position. Characters will be accepted wherever an uppercase “X” appears in the mask. DBCS characters can be entered only into two, adjacent input positions. |
| _ | Specifies a literal space. |
| \ | Forces the following character to be treated as a literal character. The backslash character is the escape character. |
| \X | Specifies a literal “X”. |
| _ | Specifies a literal underscore. |
| \, | Specifies a literal comma. |
| \= | Specifies a literal equal sign. |
| \\ | Specifies a literal backslash. |

**Note 1** The preceding characters are case-sensitive. For example, “x” is not the same as “X”.

**Note 2** If a mask character overlays part of a double-byte (DBCS) character, the entire character is replaced by blanks.

When an ACCEPT operation that specifies the MASK keyword in a CONTROL phrase is processed, the RM/COBOL runtime takes the following actions:

a. If the operation is ACCEPT with UPDATE, characters from the operand are copied (from left to right) into mask input positions. Mask literal characters are skipped. If the operand is exhausted while there are still remaining input positions, such positions are changed to literal spaces. If the mask is exhausted before the operand, the remainder of the operand is ignored. The SIZE phrase, if specified, limits the size of the operand, not the size of the mask.

   The size of the screen field is then set to the size of the mask, including trailing literal characters.

   If PROMPT is also specified, trailing input positions that are initialized with spaces are replaced with the prompt character.

b. If ACCEPT without UPDATE is specified, and ACCEPT-FIELD-FROM-SCREEN is not in effect, mask input positions are initialized with spaces or with the prompt character, if PROMPT is specified. The number of mask input positions is still restricted, based on the SIZE phrase or the size of the operand.

c. If ACCEPT without UPDATE is specified, and ACCEPT-FIELD-FROM-SCREEN is in effect, mask input positions are initialized from the current screen field. In this case, there is a one-for-one correspondence between mask characters and screen characters; that is, the mask is overlaid rather than inserted into the screen field. If PROMPT is specified, trailing input positions that are initialized with spaces are replaced with the prompt character.

When the ACCEPT is terminated, the input field is scanned from left to right. Characters appearing in input positions only are copied into the ACCEPT operand. The operand is then processed by the CONVERT and UPPER phrases as if a regular ACCEPT operation had been performed.

Table 24 lists keywords and phrases that, when specified in ACCEPT and/or DISPLAY statements, have an effect on masked input processing.

**Table 24: Effect of Certain Keywords and Phrases on Masked Input Processing**

| Keyword or Phrase | Effect |
|---|---|
| CURSOR | The CURSOR phrase in an ACCEPT statement specifies the input position, rather than the field position where the cursor is placed. It returns the input position occupied by the cursor when the ACCEPT statement is terminated. |
| GRAPHICS | The GRAPHICS keyword in the CONTROL phrase of an ACCEPT or a DISPLAY statement translates mask characters and input characters. |
| HIGH, LOW, HIGHLIGHT, LOWLIGHT | The presence of these phrases in an ACCEPT or a DISPLAY statement causes literal mask characters and input characters to be displayed at the specified intensity. |
| OFF, SECURE | When the OFF (SECURE) phrase is specified in an ACCEPT statement, literal mask characters are displayed, while input characters are not displayed. |
| PROMPT | When the PROMPT phrase is specified in an ACCEPT statement, trailing input positions are filled with the specified prompt character. |
| SIZE | When the SIZE phrase is specified in an ACCEPT or a DISPLAY statement, the size of the ACCEPT and DISPLAY operand is limited, but there is no effect on the screen field size. |
| TAB | If the TAB keyword is not specified in the CONTROL phrase of an ACCEPT statement (or the TAB phrase is not specified in an ACCEPT statement), field termination occurs when the cursor leaves the last input position, which may be followed by literal characters. |

5. **PASS-THRU**

   The ability to write escape sequences (such as pass-through printing) to the terminal with DISPLAY statements requires an additional keyword in the CONTROL phrase. The keyword, PASS-THRU, indicates that all data specified in the corresponding DISPLAY statement is to be written directly to the unit and not recorded in the in-memory image of the screen. Thus, if the DISPLAY statement causes the screen to change, the runtime system will have no knowledge of the change, and subsequent DISPLAY statements may cause confusion for the terminal operator.

   This ability also can be used automatically by specifying a TERM-ATTR PASS-THRU-ESCAPE configuration file record (see page 330). If used, any DISPLAY statements beginning with one of the escape characters will behave as if PASS-THRU were specified in the statement.

   **Note** This keyword is supported only under UNIX.

6. **PROMPT = *prompt-char***

   The PROMPT keyword causes ACCEPT statements to accept data with fill characters in positions from which data is to be accepted. Optionally, the PROMPT keyword may specify *prompt-char*, which causes the ACCEPT operation to use a prompt character different from the system default. *prompt-char* must be a single, literal character.

For example:

```
ACCEPT FOO CONTROL "PROMPT=*,TAB".
```

where, asterisk (*) is the *prompt-char*.

7. **REPAINT-SCREEN**

   The REPAINT-SCREEN keyword causes the entire screen to be refreshed from the runtime system's in-memory screen image. Any characters that were written directly to the screen, such as from C routines or DISPLAY statements with the PASS-THRU keyword (which are not recorded in the in-memory screen image), are replaced by the last value written to that location by regular DISPLAY statements. This provides the ability to clean up the screen without manually having to redraw the entire display. REPAINT-SCREEN may be used in both ACCEPT and DISPLAY statements. It is also callable from C subprograms contained in optional support modules. See the "Runtime Functions for Support Modules" topic in Appendix H: *Non-COBOL Subprogram Internals for UNIX* in the *CodeBridge User's Guide*.

   **Note** This keyword is supported only under UNIX.

8. **SCREEN-COLUMNS = *screen-width***

   The SCREEN-COLUMNS keyword instructs the runtime system to change the current display state of the user's terminal to accommodate the requested screen size. Screen-width values of 80 and 132 are currently supported.

   Changing the terminal state produces a new, blank screen of the requested screen width. All characters and windows on the original display are erased. In order to maintain valid user-defined window control blocks, programs using pop-up windows must close all pop-up windows before changing the screen size.

   Most terminals support varying screen dimensions through normal and wide terminfo and termcap entries. These normally correspond to 80 and 132 columns, respectively. When a screen dimension change is requested, the runtime system switches the TERM environment variable to the appropriate value and then sends reset or initialization strings that change the terminal's state. For terminfo, the strings are defined with the capabilities **rs1**, **rs2**, and **rs3**. For termcap, the strings are defined with the capabilities **r1**, **r2**, and **r3**. If these termcap capabilities are not defined, the runtime system attempts to use the capability **is**. If these strings are not set correctly, the terminal may be changed to an unpredictable state.

   Most UNIX systems append a "–w" to terminal descriptions to indicate a terminal's wide screen mode. For example, the wyse60 terminal description for wide displays is normally referred to as wyse60–w. Because not all UNIX systems follow this standard, the COBOL runtime allows users to use the RMTERM80 and RMTERM132 environment variables. If both variables are set, the runtime system changes the TERM environment variable to the appropriate name, as specified in RMTERM80 or RMTERM132. For example, some systems append "w" to wide terminal descriptions. RMTERM132 can be used to ensure proper behavior by setting it as RMTERM132=wyse60w.

   **Note** This keyword is supported only under UNIX.

### ERASE Phrase

All valid ERASE options (that is, ERASE, ERASE EOL, and ERASE EOS) erase the screen with the specified background color, if possible. Under UNIX, if the back_color_erase termcap or terminfo capability is set to false, or the appropriate termcap or terminfo capability to perform the specified ERASE operation is not available, blanks will be used to perform the operation.

### HIGH Phrase

HIGH specifies that the foreground color be the corresponding high-intensity color listed in Table 21 on page 196.

Under UNIX, when the HIGH phrase is present, the termcap capabilities used to set the attributes of the terminal are **nM**, **nB**, **nR** or **nS**. The terminfo capabilities are **sgr0**, **blink**, **rev**, or **sgr**. The capability used is determined by the BLINK and REVERSE phrases, and by the definition of termcap or terminfo capabilities in the terminal database.

When used with a color monitor under UNIX, the HIGH phrase specifies that the foreground color be the high-intensity color from Table 21 that corresponds to the foreground color name. If the REVERSE phrase is also present in the statement, it takes precedence over the HIGH phrase. That is, any reversal of colors takes place before the intensity is determined.

### LOW Phrase

LOW specifies that the foreground color be the default foreground color unless overridden with the FCOLOR keyword.

When the LOW phrase is present under UNIX, the termcap capabilities used to set the attributes of the terminal are **aL**, **aB**, **aR**, or **aS**. The terminfo capabilities are **dim** or **sgr**. The capability used is determined by the BLINK and REVERSE phrases, and by the definition of termcap or terminfo capabilities in the terminal database. The BLINK and REVERSE phrases are not supported by a terminfo runtime system unless the **sgr** capability is available.

If the REVERSE phrase is also present in the statement, it takes precedence over the LOW phrase, that is, any reversal of colors takes place before the intensity is determined.

### OFF Phrase

OFF specifies that the background color be used for the foreground color. During field editing for ACCEPT operations, the cursor is moved as specified, but without character echoing.

If the REVERSE phrase is also present in the statement, it takes precedence over the OFF phrase; that is, any reversal of colors takes place before the background color is determined.

SECURE is a synonym for OFF.

### REVERSE Phrase

When the REVERSE phrase is present, the specified (or default) foreground color is used as the background color, and the background color is used as the foreground color. The REVERSE phrase is processed before the HIGH, LOW, and OFF phrases.

### SIZE Phrase

The SIZE phrase is used to specify the size of an ACCEPT or DISPLAY field. The runtime system imposes the following restrictions and limitations on the value of the SIZE phrase:

1. The size of an ACCEPT or DISPLAY field must not exceed the number of characters that can appear on the screen at one time, minus the column of the first character of the data item.
2. The ACCEPT or DISPLAY field must not exceed the size of the associated buffer: the default is 264. See the discussion of the B Runtime Command Option in the Environment Runtime Command Options (on page 181).
3. Fields that extend beyond the physical right margin of the screen wrap around to the next line.
4. Fields that extend beyond the last line of the screen cause the screen to scroll one line.

### TIME Phrase

The BEFORE TIME phrase is used to "time-out" the execution of a pending ACCEPT statement. The value of *literal-8* or *identifier-8* in the BEFORE TIME phrase represents the time-out value in hundredths of seconds. The time-out value is limited to 23 hours, 59 minutes, 59.99 seconds (or 8,639,999). A value greater than 8,639,999 and less than or equal to 4,294,967,295 ($2^{32} – 1$) is set to 8,639,999.

A time-out value of 0 indicates that the ACCEPT operation should terminate immediately if there is no character waiting. A time-out value greater than 4,294,967,295 (a PIC 9(10) data item set to a value of 9999999999 is recommended) indicates that the BEFORE TIME phrase is being overridden and the ACCEPT statement will behave as if the BEFORE TIME phrase were not specified.

When the ACCEPT statement is executed, a target time is calculated as the sum of the current time and the time-out value. The time-out operation runs until the target time is reached or a key is pressed. Once a key has been pressed, the time-out function is disabled.

If the target time is reached before a key has been pressed, the ACCEPT statement is terminated. A termination code of 99 is returned in *identifier-9* if the ON EXCEPTION phrase is specified.

The BEFORE TIME phrase is intended for terminal input. The phrase is not available if input is redirected.

Under UNIX, if the CHARACTER-TIMEOUT keyword (see page 327) of the TERM-ATTR configuration record has a value, it will affect the BEFORE TIME phrase of the ACCEPT statement. The actual time-out value will be the first integral multiple of the CHARACTER-TIMEOUT value that is greater than or equal to the value specified in the ACCEPT BEFORE TIME phrase.

# Pop-Up Windows

A COBOL program can create one or more pop-up windows on the terminal output device. A pop-up window (referred to hereinafter as a window) is a temporary subscreen within the terminal screen to which all terminal output is directed. The rules concerning placement of data and default video attributes that apply to full screen input/output also apply to the window (including wrapping and scrolling). Thus, the window performs just like a full screen, except that a window is usually smaller.

A window is used for terminal input/output from the time it is created until the window is removed by the COBOL program or another window is created. When a window is removed, the contents that occupied the window area before it was created are restored, and the previous window again becomes the active subscreen. All current defaults that are associated with the newly restored window, such as the current video attributes, the current line, and the current position, are restored.

**Note** Only information written to the screen by the RM/COBOL runtime system can be restored to the screen in the event that it is covered by a window that is later removed.

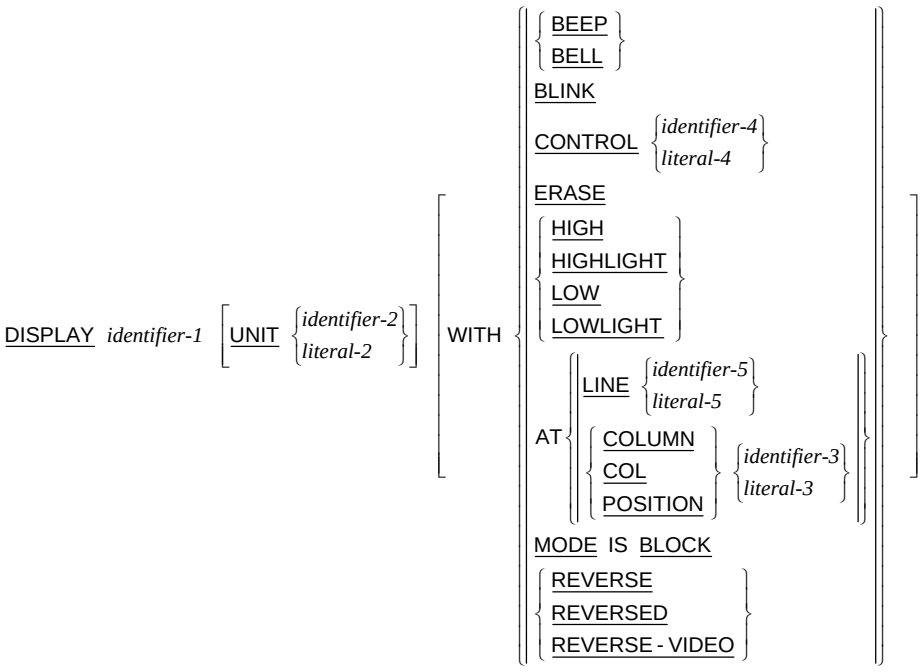
For examples on using the RM/COBOL Pop-Up Window Manager, see your installation directory and examine the following programs:

- **wintest.cbl**
- **winattrb.cbl**
- **winbordr.cbl**
- **wincolor.cbl**
- **winreltv.cbl**
- **winstat.cbl**
- **wintitle.cbl**

## Creating Pop-Up Windows

A window is created by a Format 2 DISPLAY statement containing the WINDOW-CREATE keyword in its CONTROL phrase. See the description of the DISPLAY statement (terminal I-O) in Chapter 6: *Procedure Division Statements* of the *RM/COBOL Language Reference Manual*. The general format of a DISPLAY statement used to create a window is shown below.

**Note** The format shown is a subset of the Format 2 DISPLAY Terminal I-O statement because some options of the complete statement are not applicable to window creation.




*identifier-1* specifies the window control block for the window creation. For more information, see Control Phrase (on page 195) and Pop-Up Window Control Block (on page 208).

## BEEP Phrase

The presence of the BEEP phrase in the DISPLAY statement causes the audio alarm signal to occur at the creation of the window. If the BEEP phrase is omitted, no signal is given.

BELL is a synonym for BEEP.

## BLINK Phrase

The presence of the BLINK phrase causes the border, title, and fill characters of the window to appear in a blinking mode. If the BLINK phrase is not specified, the border, title, and fill characters appear in a nonblinking mode.

**Note** The blinking attribute is not available under Windows.

## CONTROL Phrase

A DISPLAY statement with a CONTROL phrase containing the WINDOW-CREATE keyword (see discussion of the CONTROL phrase of the DISPLAY statement in Chapter 6: *Procedure Division Statements* of the *RM/COBOL Language Reference Manual*) causes *identifier-1* to be treated as a window control block, and this data item must have the structure described in Pop-Up Window Control Block (on page 208). The window is created according to the specifications given in the window control block. The window control block provided must not be that of an active window.

The FCOLOR and BCOLOR keywords can be used to set the colors of the border characters, title characters, and fill characters of the window being created. FCOLOR specifies the foreground color of each character, and BCOLOR defines the background color. FCOLOR and BCOLOR also establish the initial default colors for ACCEPT and DISPLAY statements performed while the window is active. See Table 21 on page 196 for valid color names. If FCOLOR and BCOLOR are not specified when creating a window, the default colors (if any) in effect when the window creation is requested are carried over to the new window.

Additional keywords that may be specified in the CONTROL phrase and that affect the creation of the window include: HIGH, LOW, BLINK, NO BLINK, REVERSE, NO REVERSE, ERASE, NO ERASE, BEEP, and NO BEEP. The meanings of these keywords when they appear in the value of the CONTROL phrase operand are the same as the corresponding phrases that may be written as static options of the DISPLAY statement, with the addition of the negative forms to allow suppression of statically declared options. The window creation effects of the static phrases and the corresponding CONTROL phrase keywords are described in the following paragraphs.

### ERASE Phrase

The presence of the ERASE phrase causes the window area to be erased upon creation of the window.

**Note** Using the EOS or EOL reserved words with ERASE causes the ERASE phrase to be ignored.

### HIGH and LOW Phrases

The presence of the HIGH or LOW phrase causes the border, title, and fill characters of the window to be painted at the specified intensity. When HIGH or LOW is not specified, the default intensity is HIGH.

HIGHLIGHT is a synonym for HIGH and LOWLIGHT is a synonym for LOW.

**Note** Under Windows, the HIGH and LOW phrases do not affect the border or the title of the window.

### LINE and POSITION Phrases

The window is painted on the screen with LINE 1 and POSITION 1 of the window positioned at the LINE and POSITION specified in the DISPLAY statement creating the window. For further discussion of the placement of the window, see Defining the Location of the Pop-Up Window (on page 209). LINE 1, POSITION 1 of the window is limited to the boundaries of the screen.

If requested, the border occupies the lines immediately above and below the window, and the columns immediately to the right and to the left of the window. If a title is requested, it will be painted within the top or bottom border. If a title is requested and a border is not requested, the title will occupy the line either immediately above or immediately below the window.

If the LINE or POSITION phrase is omitted from the DISPLAY statement, the line and position values for the window are determined in the same manner as the line and position values in a non-window Format 2 DISPLAY statement, except that the ERASE phrase and the window dimensions are not considered (see the section

"Determining Line and Position" for the DISPLAY statement in Chapter 6: *Procedure Division Statements* of the *RM/COBOL Language Reference Manual*).

COLUMN and COL are synonyms for POSITION.

### *REVERSE Phrase*

The presence of the REVERSE phrase causes the border, title, and fill characters of the window to appear in a reverse video mode. If the REVERSE phrase is not specified, the border, title, and fill characters appear in the normal video mode.

REVERSED and REVERSE-VIDEO are synonyms for REVERSE.

**Note** Under Windows, the REVERSE phrase does not affect the border or the title of the window.

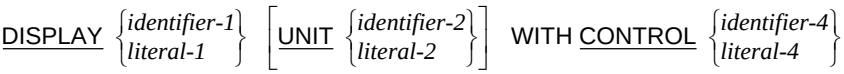
### *UNIT Phrase*

The UNIT phrase, if specified, must be written first. The other phrases may be written in any order. If not running under UNIX, the value of *identifier-2* or *literal-2* in the UNIT phrase is ignored. Under UNIX, the value of *identifier-2* or *literal-2* specifies the terminal upon which the window is to be created. If the UNIT phrase is omitted, the terminal that started the run unit is used.

## Removing a Pop-Up Window

A window is removed by a Format 2 DISPLAY statement (see the description of the "DISPLAY Statement" in Chapter 6 of the *RM/COBOL Language Reference Manual*) containing the WINDOW-REMOVE keyword in its CONTROL phrase. The general format of a DISPLAY statement used to remove a window is shown below.

**Note** The format shown is a subset of the Format 2 DISPLAY statement because some options of the complete statement are not applicable to window removal.




*identifier-1* specifies the window control block for the window creation. For more information, see Control Phrase (on page 195) and Pop-Up Window Control Block (on page 208).

### *CONTROL Phrase*

A DISPLAY statement with a CONTROL phrase containing the WINDOW-REMOVE keyword (see discussion of the CONTROL phrase of the DISPLAY statement in Chapter 6 of the *RM/COBOL Language Reference Manual*) causes *identifier-1* to be treated as a window control block, and causes the active window to be removed. The window control block should be the same one used to create the window; it must not be that of a different active window.

### *UNIT Phrase*

The UNIT phrase, if specified, must be written first. The other phrases may be written in any order. If not running under UNIX, the value of *identifier-2* or *literal-2* in the UNIT phrase is ignored. Under UNIX, the value of *identifier-2* or *literal-2* specifies the terminal upon which the window is to be created. If the UNIT phrase is omitted, the terminal that started the run unit is used.

## Pop-Up Window Control Block

The following is an example of a window control block in a COBOL program. The order of the fields, the PICTURE character-string, and the declared usage in the example are fixed parameters that cannot be changed when defining a window control block. The data-names and contents of the fields in the example are variable and thus may be changed.

```
01 WINDOW-CONTROL-BLOCK.
   03 WCB-HANDLE                  PIC 999 BINARY(2)
                                    VALUE 0.
   03 WCB-NUM-ROWS                PIC 999 BINARY(2).
   03 WCB-NUM-COLS                PIC 999 BINARY(2).
   03 WCB-LOCATION-REFERENCE      PIC X.
      88 WCB-SCREEN-RELATIVE        VALUE "S".
      88 WCB-WINDOW-RELATIVE        VALUE "W".
   03 WCB-BORDER-SWITCH           PIC X.
      88 WCB-BORDER-ON              VALUE "Y" FALSE "N".
   03 WCB-BORDER-TYPE             PIC 9.
      88 WCB-BORDER-WCB-CHAR        VALUE 0.
      88 WCB-BORDER-PLUS-MINUS-BAR  VALUE 1.
      88 WCB-BORDER-LINE-DRAW       VALUE 2.
      88 WCB-BORDER-DBL-LINE-DRAW   VALUE 3.
   03 WCB-BORDER-CHAR             PIC X.
   03 WCB-FILL-SWITCH             PIC X.
      88 WCB-FILL-ON                VALUE "Y" FALSE "N".
   03 WCB-FILL-CHAR               PIC X.
   03 WCB-TITLE-LOCATION          PIC X.
      88 WCB-TITLE-TOP              VALUE "T".
      88 WCB-TITLE-BOTTOM           VALUE "B".
   03 WCB-TITLE-JUSTIFICATION     PIC X.
      88 WCB-TITLE-CENTER           VALUE "C".
      88 WCB-TITLE-LEFT             VALUE "L".
      88 WCB-TITLE-RIGHT            VALUE "R".
   03 WCB-TITLE-LENGTH            PIC 999 BINARY(2).
      88 WCB-TITLE-LENGTH-COMPUTE   VALUE 0.
   03 WCB-TITLE                   PIC X(40).
```

## Identifying the Pop-Up Window

The field WCB-HANDLE is initialized by the WINDOW-CREATE DISPLAY operation to contain a value that identifies the window. This field must be set to zero before the WINDOW-CREATE operation. The value that was placed in this field following the WINDOW-CREATE operation must be in the WCB-HANDLE field when the WINDOW-REMOVE DISPLAY operation is performed to remove the window. The only other value that is allowed is zero, which removes the active window.

**Note** The use of zero is allowed for compatibility with previous versions of the Pop-Up Window Manager, but it is strongly discouraged.

## Defining the Size of the Pop-Up Window

The parameters WCB-NUM-ROWS and WCB-NUM-COLS define the number of rows and columns available for ACCEPT and DISPLAY statements within the window. If the window is to have a border, it will occupy two additional rows and two additional columns on the screen. If the window has a title but does not have a border, the window occupies one additional line on the screen.

## Defining the Location of the Pop-Up Window

The WCB-LOCATION-REFERENCE parameter determines whether the LINE and POSITION in the DISPLAY statement used to create the window describes a location relative to the physical screen or the active window. A value of S indicates the location is relative to the physical screen. A value of W indicates the location is relative to the active window.

The created window is limited to the boundaries of the screen, not to the boundaries of the active window.

## Defining the Border of the Pop-Up Window

The parameter WCB-BORDER-SWITCH determines whether a border should be painted around the window. A value of Y indicates that a border is to be painted. A value of N indicates that a border is not to be painted.

The parameter WCB-BORDER-TYPE determines what characters are used to make up the border. A value of 0 indicates that the character specified by WCB-BORDER-CHAR is used to paint the border. A value of 1 indicates that the plus sign, hyphen, and vertical bar characters (+, –, |) are used to paint the border. A value of 2 in the WCB-BORDER-TYPE field indicates that graphic line draw characters are used to paint the border. A value of 3 in the WCB-BORDER-TYPE field indicates that graphic double-line line draw characters are used to paint the border. If the terminal does not support line draw graphics characters, the border will be drawn using normal characters (+, –, |). This field is ignored unless WCB-BORDER-SWITCH has a value of Y.

The WCB-BORDER-CHAR parameter determines the character to be used in building the border around the window. This field is ignored unless WCB-BORDER-SWITCH has a value of Y and WCB-BORDER-TYPE has a value of 0.

**Note** The WCB-BORDER-CHAR and WCB-BORDER-TYPE parameters are ignored under Windows.

### Initializing the Pop-Up Window Area

WCB-FILL-SWITCH determines whether the window should be filled with the character defined by WCB-FILL-CHAR when it is created. If the window is not filled, then the contents in the defined window area remain untouched until modified by a subsequent ACCEPT or DISPLAY statement in the window. A value of Y indicates that the window area be filled with the defined character. A value of Y will also cause an ERASE phrase to be ignored. A value of N indicates that the window area is to be left unchanged.

The parameter WCB-FILL-CHAR determines the character to be used to fill the window area. This field is ignored unless WCB-FILL-SWITCH has a value of Y.

### Defining the Location of the Title of the Pop-Up Window

The parameter WCB-TITLE-LOCATION determines whether the text in WCB-TITLE should be placed within the location of the top border or bottom border. A value of T indicates that the title is to be painted at the top of the window. A value of B indicates that the title is to be painted at the bottom of the window. This field is ignored if WCB-TITLE-LENGTH has a value of zero and WCB-TITLE is filled with spaces.

The WCB-TITLE-JUSTIFICATION parameter determines whether the text of the title should be centered, left-justified, or right-justified in its location at the top or bottom of the window. A value of C indicates that the title should be centered; L indicates that it should be left justified; and R indicates that the window should be right justified. This field is ignored if WCB-TITLE-LENGTH has a value of zero and WCB-TITLE is filled with spaces.

**Note** The WCB-TITLE-JUSTIFICATION and WCB-TITLE-LOCATION parameters are ignored under Windows.

### Defining the Title of the Pop-Up Window

The length of the title is defined by the value of the WCB-TITLE-LENGTH parameter. If the value of this field is non-zero, WCB-TITLE-LENGTH indicates the number of characters, beginning with the first character of the WCB-TITLE field, that are to be used as the title of the window. If the value of this field is zero, the title string is made up of all characters between the first character of the WCB-TITLE field to the last non-blank character in the field.

The WCB-TITLE parameter defines the text to be placed in the title of the window. This field may be any length sufficient to contain the desired title, and must be the last data item in the window control block. The length of 40 specified in the example is an arbitrary value.

If the WCB-TITLE field contains all spaces, regardless of whether WCB-TITLE-LENGTH is set to a zero or non-zero value, the pop-up window does not have a title.

## Pop-Up Window Operation Status

The COBOL program can obtain the status of a window operation immediately after a request is made to create or remove a window. The ACCEPT FROM EXCEPTION STATUS statement places the status in the field designated by *identifier-1*. The general format of the ACCEPT FROM EXCEPTION STATUS statement is shown below.

ACCEPT *identifier-1* FROM EXCEPTION STATUS [ END - ACCEPT ]

The information requested is transferred from EXCEPTION STATUS to *identifier-1* according to the rules of the MOVE statement (see the discussion of the MOVE statement in Chapter 6: *Procedure Division Statements* of the *RM/COBOL Language Reference Manual*). EXCEPTION STATUS is an implicitly defined data item that behaves as if it had been defined in the Data Division as an unsigned, three-digit, numeric integer data item.

Table 25 is a list of the error codes and values returned to the COBOL program after a request is made to create or remove a window.

**Table 25: Pop-Up Window Error Codes**

| Code | Description |
|---|---|
| 0 | Operation successfully completed. |
| 301[1] | Window border or title does not fit on screen. |
| 302[1] | Title is too long for window or specified title length is longer than the title field in the window control block. |
| 303[1] | Requested window will not fit on screen. |
| 304 | No windows are active. |
| 305 | Window manager is not available. |
| 306 | Out of memory. |
| 307 | Too many windows. |
| 308 | Buffer I/O error. |
| 309 | Requested REMOVE-WINDOW for inactive WCB. |
| 310 | CREATE-WINDOW requested with active WCB. |
| 311 | Invalid parameter in WCB. |

[1] *These error codes are reported differently under Windows than under UNIX, or they are not reported.*

# COPY Statement

Use the COPY statement to copy RM/COBOL source text from a specified file into the source program. The text copied may have been created outside RM/COBOL, either through a text editor or through some other process. The file is copied at the point at which the COPY statement appears in the program; the file logically replaces the COPY statement.

A copied file may in turn contain COPY statements, up to the limits of either five (the typical case) or nine (if the last statement in the copy file is a COPY statement).

See the discussion of the COPY statement in Chapter 1: *Language Structure* of the *RM/COBOL Language Reference Manual*, for a description of the syntax and field definitions.

Filenames that are not reserved words and that are made up of valid RM/COBOL alphabetic, numeric and special characters do not need to be enclosed in quotation marks within the COPY statement. Filenames that are reserved words or contain characters other than the standard set must be specified as a nonnumeric literal (that is, they must be enclosed in quotation marks). A period followed by a space terminates the COPY statement (that is, it is not considered part of a text-name or a library-name).

If a library-name is specified in the COPY statement, it is treated as a pathname for the filename.

If you do not enter a filename extension with the filename, the compiler assumes an extension of **.cbl**. If it cannot find such a file, it then looks for a file with the supplied name with the extension **.CBL**. The assumed extension can be changed with the EXTENSION-NAMES configuration record (on page 308). For all attempts to open the copied file, if a directory path or a drive letter is not specified, the directory search sequence is used to try to locate the file. See the discussion of search sequences in Directory Search Sequences on UNIX and Directory Search Sequences on Windows.

Here are three examples of valid COPY statements.

```
IDENTIFICATION DIVISION.
COPY STDID.
```

The preceding COPY statement copies the file **stdid.cbl** from the path specified by RMPATH.

```
ENVIRONMENT DIVISION.
COPY "cobol".
```

The preceding COPY statement copies the file **cobol.cbl** from the path specified by RMPATH (see the discussion of RMPATH in the appropriate installation and system considerations chapter in this user's guide for your specific operating system).

```
COPY data1 OF lib1.
```

The preceding COPY statement copies the file **lib1/data1.cbl** relative to the current directory.

# STOP RUN Numeric Statement and RETURN- CODE Special Register

When a run unit terminates, a numeric value—termed an exit code—is returned to the operating system. The exit code is made up of the low-order eight bits of the binary RETURN-CODE special register.

At the start of the run unit, the RETURN-CODE special register is initialized to zero. The program may change the RETURN-CODE special register value by using it as the destination of a MOVE statement or arithmetic verb, or by using the "STOP RUN numeric" form of the STOP RUN statement. Certain program exit codes (on page 186) are used by the runtime system to indicate error conditions. Use of these values should be avoided in the program.

# CALL and CANCEL Statements

The CALL statement transfers control to a contained RM/COBOL subprogram, to an external RM/COBOL subprogram, or to a non-COBOL subprogram. Called subprograms may themselves call other subprograms during the course of execution.

There are certain requirements that must be observed before RM/COBOL subprograms or non-COBOL subprograms can be called:

1. The RM/COBOL subprogram must have been compiled.
2. Under UNIX, the non-COBOL subprogram must be contained in an optional support module (shared object). The optional support module may be built using CodeBridge, Liant Software Corporation's cross-language call system, or it may be built using the method described in Appendix H: *Non-COBOL Subprogram Internals for UNIX* in the *CodeBridge* manual. In either case, however, the support module must conform to the rules set forth in the CodeBridge appendix.

   Under Windows, the non-COBOL subprogram must be contained in a dynamic-link library (DLL). The dynamic-link library may be built using CodeBridge, Liant Software Corporation's cross-language call system, or it may be built using the method described in Appendix G: *Non-COBOL Subprogram Internals for Windows* in the *CodeBridge* manual. In either case, however, the support module must conform to the rules set forth in the CodeBridge appendix.
3. The name specified in the CALL statement must be complete enough to search for and locate the program.

A called subprogram is loaded and is in its initial state in the following instances: the first time it is called in the run unit; the first time it is called after execution of a CANCEL statement identifying the program that directly or indirectly contains the subprogram; every time the subprogram is called if it possesses the initial attribute; and the first time the subprogram is called after the execution of a CALL statement identifying a program that possesses the initial attribute and that directly or indirectly contains the subprogram. In all other entries into the called subprogram, the state of that program is unchanged from its state when last exited.

Called RM/COBOL subprograms remain in memory until implicitly or explicitly canceled. Optional support modules are loaded during RM/COBOL runtime system initialization and remain loaded until the runtime system terminates. A called

subprogram is implicitly canceled in only two cases. A called subprogram with the initial attribute (that is, one with the INITIAL clause specified in the PROGRAM-ID paragraph) is implicitly canceled whenever it executes an EXIT PROGRAM statement, and is therefore in its initial state every time it is called. All programs associated with a run unit are implicitly canceled when the run unit terminates. In all other cases, an explicit CANCEL statement identifying the program is required in order to cancel it. Use of the CANCEL statement to cancel a C subprogram sets the initial flag to zero on the next entry into the subprogram, but has no effect on the values of the external and static variables used in the C subprogram. An explicit CANCEL of a non-COBOL subprogram in a support module, but neither of the two implicit cancels, causes the runtime system to call the **RM_AddOnCancelNonCOBOLProgram** special entry point when that entry point is defined in the support module that defines the non-COBOL subprogram. For complete details about special entry points in support modules, see the "Special Entry Points for Support Modules" topic in Appendix G: *Non-COBOL Subprogram Internals for Windows* and Appendix H: *Non-COBOL Subprogram Internals for UNIX* of the *CodeBridge* manual.

## Subprogram Loading

When a CALL statement is executed, the program-name (the value of *identifier-1* or *literal-1* as defined in the discussion of the CALL statement in Chapter 1: *Language Structure* of the *RM/COBOL Language Reference Manual*) determines the subprogram to which control is transferred. This is done in the following order:

1. **If the program-name matches the value of program-name or literal as specified in the PROGRAM-ID paragraph** of a program directly contained in the calling program or of a program possessing the common attribute that is directly contained in a program that directly or indirectly contains the calling program, control is transferred to that program.

2. **If a program has been loaded and not canceled**, and is called again by the same program-name, control is transferred to that program.

3. **If the program-name matches the value of *program-name* or *literal*** (see the *RM/COBOL Language Reference Manual*) **as specified in the PROGRAM-ID paragraph of a program in an RM/COBOL program library**, the program is loaded and control transferred to it. Remember, at the point of loading, the program is in its initial state. In the same manner, if the program-name matches a called name literal in a non-COBOL subprogram library (see the appropriate appendixes in the *CodeBridge* manual for information on the non-COBOL subprogram internals for Windows and UNIX), control transfers to the subprogram associated with the called name literal.

   Libraries—both RM/COBOL and non-COBOL—are searched in the order specified, from left to right, by one or more L Runtime Command Options, as described in Program Runtime Command Options (on page 182). Other non-COBOL libraries (optional support modules), automatically loaded from either the runtime execution directory or the **rmcobolso** (on UNIX) or **RmAutoLd** (on Windows) subdirectory of the execution directory, are searched next, as described in Appendix D: *Support Modules (Non-COBOL Add-Ons)* on page 429. The first program-name matched is the only one considered during program loading. Because libraries specified on the command line are searched before libraries loaded automatically, it is possible for a developer to test a new optional support module while other users are running an application in live "production mode."

If the RPC (Remote Procedure Calls) optional support module is present, RM/COBOL programs specified to be executed remotely by the RPC server override programs contained in either RM/COBOL or non-COBOL libraries, including automatically loaded optional support modules. RM/COBOL programs executed remotely using the CALL "REMOTEPROGRAM" capability of RPC do not override programs contained in libraries.

4. **If the program-name matches the name of a subprogram in the RM/COBOL subprogram library**, control transfers to that subprogram. See Appendix F: *Subprogram Library* (on page 527) for a description of the RM/COBOL subprogram library.

5. **If the program-name contains no extension**, a default filename extension of first **.cob** and then **.COB** is added to the name before beginning the search for a valid program file. If such a file exists and contains only one object program, the program is loaded and control is transferred to that program, regardless of the name in its PROGRAM-ID paragraph. The default extension can be changed with the EXTENSION-NAMES configuration record (on page 308).

6. **Under Windows, if the program-name specifies a filename extension .dll**—or if it specifies no filename extension at all—the filename extension **.dll** is used with the filename before starting the search for a valid non-COBOL subprogram file. If such a file exists, it is loaded and control is transferred to that program, as described in the *Non-COBOL Subprogram Internals for Windows* appendix of the *CodeBridge* manual.

7. **If the program-name does not specify a filename extension**, or if the program-name specifies a filename extension other than **.dll** (under Windows), the program-name is used to search for a valid RM/COBOL program file. If such a file exists, and contains only one object program, the program is loaded and control is transferred to it, regardless of the name in its PROGRAM-ID paragraph.

8. **If the program-name cannot be found**, an exception condition occurs. This condition may be detected by the calling program with the ON EXCEPTION or ON OVERFLOW phrase. If the calling statement does not contain the ON EXCEPTION or ON OVERFLOW phrase, execution ends.

The use of contained programs and program libraries eliminates Step 5 through Step 7. In the case of RM/COBOL program libraries, the I/O overhead of searching for the program file is minimized.

Steps 5 through 7 search the environment for a match with the name used in each step. If an environment variable name matches the name in one of those steps, the value of the environment variable replaces that name in that step for locating the file. Steps 5 through 7 also use the RUNPATH directory search sequence, as described in either Directory Search Sequences on UNIX (on page 24) or Directory Search Sequences on Windows (on page 61).

RM/COBOL for Windows searches for dynamic-link libraries (DLLs) specified without a drive or path specification in the following order:

1. The directory from which the application executable was loaded; for example, the directory containing **runcobol.exe**.
2. The current working directory.
3. The Windows system directory on Windows 9x-class operating systems; for example, **c:\windows\system**. On Windows NT-class operating systems, first, the 32-bit Windows system directory (for example, **c:\windows\system32**); second, the 16-bit Windows system directory (for example, **c:\windows\system**).
4. The Windows directory; for example, **c:\windows**.
5. The directories listed in the PATH environment variable.
6. The directories listed in the RUNPATH environment variable.

RM/COBOL for UNIX does not search for shared object files except for names specified in the L Runtime Command Option, as described in Program Runtime Command Options (on page 182). When specified in the L Runtime Command Option, the UNIX runtime searches for shared objects (**.so** files) specified without a path specification in the following order:

1. The execution directory; for example, the directory containing **runcobol**, which is typically, **/usr/bin**.
2. The current working directory.
3. The directory search sequence used by the UNIX dynamic-load open library system function (dlopen on many UNIX systems). On some UNIX systems, this may be influenced by an environment variable, such as LD_LIBRARY_PATH. Consult your UNIX system documentation for information on the search sequence used on your system.
4. The directories listed in the RUNPATH environment variable.

## Argument Considerations

RM/COBOL allows, as a nonstandard extension to COBOL, passing literals as argument values from a calling program to a called program in the USING phrase of the CALL statement. Prior to version 7.5, if the literal was not subject to a BY CONTENT phrase, the compiler generated code to pass the literal by reference (for compatibility with older versions of RM/COBOL prior to the addition of the BY REFERENCE and BY CONTENT phrases). When a literal is passed by reference and the called program modifies the corresponding Linkage Section data item, the literal value is modified in the calling program. Since the compiler shares literal references among multiple uses of the same value, a changed literal value can cause unexpected behavior and failures in the calling program. Thus, the version 7.5 or later compiler has been changed to generate code to pass all literals specified in the USING phrase of the CALL statement as if BY CONTENT were specified. When passed by content, a temporary copy of the literal is passed to the called program. A new COMPILER-OPTIONS configuration record keyword, SUPPRESS-LITERAL-BY-CONTENT (see page 299), has been added to override this new behavior.

## External Objects

A source program may use the EXTERNAL clause to declare three types of external objects:

1. **Data records**, named by record-names
2. **File connectors**, named by file-names
3. **Indexes**, named by index-names

See the discussion of the EXTERNAL clause in Chapter 4: *Data Division* of the *RM/COBOL Language Reference Manual* for more details.

During execution of a run unit, the runtime system maintains a list of external objects. The list is established as being empty when the run unit begins. When an object program is loaded, the names of external objects it declares are checked for a match—of both name and type—against the list of external objects. If both name and type match, the declared external objects and existing external objects are considered references to the same object. Only the first 30 characters of the name are used in this matching operation. The declared object is then checked to determine whether it matches the description of the external object. A mismatch in the description terminates execution and displays an error message. If either name or type does not match, the declared external object is allocated and added to the list. If there is not enough memory to load the object, execution ends and an error message is displayed.

The determination of matching descriptions depends on the type of object file:

1. **Data Records.** The record-name for both objects must be described with the same number of character positions.
2. **File Connectors.** The file-name for both objects must be described as follows:
   - The file organization specified in the ORGANIZATION clause, that is, SEQUENTIAL, RELATIVE, or INDEXED, must be the same.
     - If the organization is sequential, both objects must agree on the presence or absence of LINAGE and PADDING CHARACTER.
     - If the organization is relative, both objects must declare the same external data item as the relative key data item, or must omit specification of a relative key data item.
     - If the organization is indexed, both objects must declare the same number of record keys at the same positions in the record, and must agree on the presence or absence of a COLLATING SEQUENCE clause. Both objects must have the same COLLATING SEQUENCE clause, if the clause is present. In addition, if split keys are present, both objects must have the same split key clauses.
   - The access mode specified in the ACCESS MODE clause must be the same.
   - The presence or absence of the OPTIONAL phrase in the SELECT clause of the file control entry must be the same.
   - The number of input-output areas specified in the RESERVE clause must be the same.
   - The alphabet specified in the CODE-SET clause of the file control entry or the file description entry (RM/COBOL allows the CODE-SET clause in either entry) must be the same.

- The BLOCK CONTAINS clause in the file description entry must specify the same minimum and maximum values, and must agree on whether these are expressed in CHARACTERS or RECORDS.
- The RECORD clause in the file description entry must specify the same minimum and maximum record size.
- The LABEL RECORDS clause in the file description entry must specify the same property of OMITTED or STANDARD.

3. **Indexes.** The index-name for both objects must be associated with the same external data record. Both index-names must be associated with table items that span the same number of character positions.

**Note** It is recommended that source programs use COPY statements to copy a common definition of an external object in order to avoid mismatched external object descriptions.

Once an external object is added to the runtime-maintained list, it remains in existence until the end of the run unit. Execution of a CANCEL statement identifying a program that describes an external object does not affect the allocation, contents or state of the external object. For external file connectors left in an open mode, the runtime system closes the file when the run unit terminates but not when a program describing the file is canceled.

# Composite Date and Time

Beginning with version 7.0 of RM/COBOL, the ACCEPT statement supports the CENTURY-DATE, CENTURY-DAY, and DATE-AND-TIME options (for more information, see the *RM/COBOL Language Reference Manual*). These options provide a four-digit year in one operation without using the C$CENTURY (on page 534) subprogram. In addition, the DATE-AND-TIME option provides the date and time in a single operation that is guaranteed to be consistent.

Prior to version 7.0, RM/COBOL followed the standard COBOL, which has separate statements to obtain the date and the time. This may cause the composite date and time to be inaccurate for times near midnight. For example, the following two ACCEPT statements will obtain a date and time that is nearly a full day earlier than correct if executed such that midnight occurs between the two statements:

```
ACCEPT CURRENT-DATE FROM DATE.

ACCEPT CURRENT-TIME FROM TIME.
```

COBOL developers have long been aware of such date/time problems and many have already solved it by checking to see whether the time crossed midnight while fetching the date and time and, if necessary, fetching the date and time again.

# DELETE FILE Operation

Under UNIX, the DELETE FILE operation will fail if the user does not have write permission for both the file to be deleted and the directory containing the file.

# File Sharing

RM/COBOL supports shared environments[1], which allow files to be shared by two or more users. This includes allowing two or more users to have a file open simultaneously and apply updates to that file. The FORCE-USER-MODE keyword (see page 318) of the RUN-FILES-ATTR configuration record can force files not to be shared.

The WITH LOCK phrase may be used on the OPEN statement to restrict the use of a file by other users during the period the file is open. When used on an OPEN I-O, OUTPUT or EXTEND statement, the WITH LOCK phrase prevents other RM/COBOL users from opening the file. When used on an OPEN INPUT statement, the WITH LOCK phrase prevents other RM/COBOL users from opening the file I-O, OUTPUT or EXTEND. When the WITH LOCK phrase is used, file performance is improved by eliminating the overhead of locking records and permitting the buffering of file data in program memory.

**Note** On UNIX systems in which record locking is implemented through the **fcntl()** system call, the file must be available with read/write access to enforce file locking. If the file is not available with read/write access, the file is opened but file locking is not enforced.

When the WITH LOCK phrase is absent, file access permits sharing by other users. The WITH LOCK phrase is ignored in single-user configurations. Table 26 illustrates the sharing permitted between applications in shared environments.

In a shared environment, a sequential file is considered shared if the WITH LOCK phrase is omitted, even for OPEN OUTPUT and OPEN EXTEND. This permits other users to OPEN EXTEND the same file and write records at the end of file.

If the EXCLUSIVE phrase is specified in the OPEN statement or in the applicable LOCK MODE clause, the same behavior, as described above for the WITH LOCK phrase, applies.

**Note** RM/COBOL version 5.3 and later runtime systems do not lock program files that are being executed. Although this characteristic improves performance, under certain circumstances it can allow the compiler to recompile a program that is being executed at the time.

[1] Shared environments apply to appropriately licensed users only.

**Table 26: Sharing Permissions**

| Desired Open Mode | Current Open Mode | | | | | |
|---|---|---|---|---|---|---|
| | **Input** | **Input/ Output** | **Output** | **Extend** | **Input[1]** | **I-O[1] Output Extend** |
| **Input** | ■ | ■ | ■ | ■ | ■ | ▨ |
| **Input/ Output** | ■ | ■ | ■ | ■ | ▨ | ▨ |
| **Output** | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |
| **Extend** | ■ | ■ | ■ | ■ | ▨ | ▨ |
| **Input[1]** | ■ | ▨ | ▨ | ▨ | ■ | ▨ |
| **I-O[1] Output Extend** | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

■ *Open granted.*

▨ *Open denied with I/O error 93,02 or 90,05.*

[1] *WITH LOCK.*

# File Buffering

Buffering of the data in files can significantly increase the speed of accessing a file by keeping frequently accessed data in memory buffers instead of reading the data from disk every time it is needed. RM/COBOL maintains a pool of memory from which it takes the buffers for all files. When the memory in this pool is exhausted, the memory for buffers, which have not been used recently, will be taken from the file that had been using them and given to the file that needs them. This accommodates applications that open large numbers of files, but concentrates on only a few files at once.

Sequential and relative files must be opened WITH LOCK in order to make use of more than one buffer from the buffer pool.

The amount of memory in the buffer pool can be controlled by use of the BUFFER-POOL-SIZE keyword (see page 316) of the RUN-FILES-ATTR configuration record. Increasing the default may improve the performance of the I/O of the application file. Decreasing the default value can increase the amount of program memory available.

The minimum size of the buffer pool must be adequate for the block sizes of the files opened by the application. For more information, see the description of the BLOCK CONTAINS clause (indexed file description entry) on page 233 for indexed files.

# Very Large File Support

Very large files are defined as RM/COBOL indexed files larger than 2 gigabytes (GB) and RM/COBOL relative and sequential files larger than 1 GB. The RM/COBOL runtime system allows RM/COBOL files to have file sizes of 2 GB and larger when running under operating systems that support very large files. For information about the operating systems that support large files, see Using Large File on UNIX (on page 47) and Using Large Files on Windows (on page 107).

Support for large files is provided by the LARGE-FILE-LOCK-LIMIT keyword (see page 318) of the RUN-FILES-ATTR configuration record. In order to use this new limit on relative or sequential files, you must use the USE-LARGE-FILE-LOCK-LIMIT keyword in a RUN-REL-FILES or RUN-SEQ-FILES configuration record. In order to use this new limit on indexed files, you must either use an indexed file version level 3 (on page 244), or use an indexed file version level 4 (on page 244) and the USE-LARGE-FILE-LOCK-LIMIT keyword (see page 322) in a RUN-INDEX-FILES configuration record.

# File Types and Structure

There are three types of files supported by RM/COBOL. Each file type and structure is most useful for specific functions. This section describes each of the following file types:

1. Sequential files (see the following topic)
2. Relative files (on page 228)
3. Indexed files (on page 230)

## Sequential Files

Sequential files are organized such that records are always read or written serially. In other words, during a specific I/O operation, the first record read is always the first record in the file, and the next record written is always written after the last record currently in the file. RM/COBOL has two types of sequential files:

1. **Line Sequential Files.** Line sequential files should contain only ASCII text data. (In other words, they are equivalent to standard source files.) Each logical record within line sequential files is variable in length and ends with a line feed or carriage return/line feed pair.

   If the ASCII control codes, that is, carriage return (CR), line feed (LF), form feed (FF) or SUB, are present in a record, the record cannot be written. When reading a file under UNIX, the LF, CR, FF, LF CR, CR LF, and FF CR sequences terminate a record. Under Windows, the CR LF sequence terminates a record and leading and trailing LF and FF sequences are ignored. SUB terminates the file and tab (HT) causes one or more spaces to be inserted according to the default tab column positions, which are every four columns, starting with column 8 and ending with column 72.

   The device-name in the ASSIGN clause of the file control entry determines the treatment of spaces in a line sequential record. If the device-name is DISC, DISK, or RANDOM, trailing spaces are preserved when a line sequential record is written. The record length returned when the record is read is the length of the record when it was written. If the device-name is CASSETTE, INPUT-OUTPUT, MAGNETIC-TAPE, OUTPUT, PRINT or PRINTER, all trailing spaces are removed when a record is written to the file. If the device-name is CARD-READER, CARD-PUNCH, CASSETTE, INPUT, INPUT-OUTPUT or MAGNETIC-TAPE, records read are padded with spaces to the maximum record length, and the record length returned is always the maximum record length.

   A file that is assigned to either of the device-names PRINT or PRINTER, for which the LINAGE clause is used, or for which the ADVANCING phrase of the WRITE statement is used, is always treated as a line sequential file. In this case, ASCII control codes are allowed.

   Each logical record within line sequential files has a maximum record length of 65280 bytes.

2. **Binary Sequential Files.** Whereas line sequential files contain ASCII text data, binary sequential files may contain any type of data. Binary sequential files may be fixed length or variable length.

   A fixed-length binary sequential file is one for which only one record description exists or all record descriptions describe the same number of characters, and for which no record description contains the OCCURS . . . DEPENDING ON clause, or for which the RECORD clause specifies fixed-length records. Such files may have a maximum record length of 65280 characters. Fixed-length binary sequential files are recorded by RM/COBOL without any additional structure; the byte count implied by the record length provides data transparency.

   A variable-length binary sequential file does not satisfy the record length requirements for fixed length. The data is encapsulated in an eight-byte overhead to provide data transparency. The maximum record size for variable-length binary sequential files is 65280 characters.

   **Note** The compiler listing allocation map indicates in the "Debug" column whether a file has been described with fixed- or variable-length records. This is described on page 158 and illustrated in Figure 29: Allocation Map (Part 3 of 5).

## RECORD Clause (Sequential File Description Entry)

The RECORD clause specifies the minimum and maximum lengths of records in a sequential file. The minimum and maximum record lengths are not recorded with the file; however, a READ statement encountering a record whose length is less than the minimum record length receives an error. Also, an OPEN EXTEND for a fixed-length binary sequential file succeeds only if the total file size at the time of the OPEN is a multiple of the maximum record length of the file.

## BLOCK CONTAINS Clause (Sequential File Description Entry)

In a single-user environment, sequential disk files are physically read and written in fixed length pieces called blocks. In a shared environment, sequential files are read and written in blocks only when the WITH LOCK phrase is specified. When the WITH LOCK phrase is omitted, the records of the file are read or written individually (without blocking) from the operating system.

The size of a block is determined by the BLOCK CONTAINS clause in the RM/COBOL program. A larger block size causes more data to be transferred in a single request, requires more time to affect the data transfer, reduces the total number of disk accesses, and requires more memory from the buffer pool. Blocking in this fashion may increase performance; however, because blocking may defer the physical writing of a block to disk until well after the WRITE statement that first places data in the block, errors (which can include loss of data) caused by that physical write may not be reported until a subsequent WRITE or CLOSE statement for the file is executed.

The file block size is not considered a fixed attribute of a sequential file; different programs may specify different block sizes for the same file.

The runtime system uses the following algorithm to determine the block size when opening a sequential disk file:

1. If no BLOCK CONTAINS clause is present, the block size is 4096 characters. The default block size may be changed with a RUN-SEQ-FILES configuration record (on page 326).
2. If a BLOCK CONTAINS *nnn* CHARACTERS clause is present, the block size is the specified number of characters.
3. If a BLOCK CONTAINS *nnn* RECORDS clause is present, the block size is the specified number of records multiplied by the sum of the maximum record length and the record overhead.

The maximum allowed block size on most systems is 65535 characters.

## LINAGE Clause (Sequential File Description Entry)

When a file described with the LINAGE clause is opened for output, it is assumed the output device is already positioned to the first line of the first logical page body. This is the operator's responsibility. The program should be written to allow the operator an opportunity to adjust the forms in the printer (or any other output device) as required.

As an alternative, the LINAGE-INITIAL-FORM-POSITION keyword (see page 311) in the PRINT-ATTR configuration record may be set to the value TOP-OF-FORM. In this case, which is particularly useful for page printers, it is assumed that the output device is positioned at the top of the form. When the first record is written to the file, the record will be written after advancing over the top margin on the first logical page.

The logical pages of a file described with the LINAGE clause are normally written contiguously with no additional spacing provided between pages. The LINAGE-PAGES-PER-PHYSICAL-PAGE keyword (see page 312) in the PRINT-ATTR configuration record may be used to cause physical page breaks, such as form feed characters, to be written to the file.

## RESERVE Clause (Sequential File Control Entry)

The RESERVE clause is ignored. Buffer memory is automatically managed based on the amount of activity of a particular file. See File Buffering (on page 221).

## CODE-SET Clause (Sequential File Control Entry or File Description Entry)

The CODE-SET clause determines the character set used to represent the sequential file. For example, if the CODE-SET is EBCDIC, all records written to the file are translated from ASCII to EBCDIC. The CODE-SET is not considered a fixed attribute of the sequential file. Different programs may specify different character sets for the same file.

### REVERSED Phrase (OPEN Statement)

The REVERSED phrase is not supported. If specified, it is ignored.

### WITH NO LOCK Phrase (READ Statement)

If a READ statement without the WITH NO LOCK phrase fails because the record is locked, the contents of the record area are undefined and the file position indicator is unchanged. A subsequent READ behaves as if the failing READ statement had never been issued.

A READ statement with the WITH NO LOCK phrase may be used to read a record without regard to the lock status of the record. If an OPEN INPUT statement opened the file, the WITH NO LOCK phrase is assumed on all READ statements on the file.

If the file control entry does not contain a FILE STATUS clause or there is no USE declarative procedure defined for a file, record lock status is not reported to the program. Instead, the runtime system waits for the record to become unlocked. By using the FATAL-RECORD-LOCK-TIMEOUT keyword (see page 317) in the RUN-FILES-ATTR configuration record, the runtime system can be made to return a fatal error instead of waiting indefinitely. If the record is locked using a different file descriptor in the same run unit, then the runtime system never waits. Instead, to prevent a deadlock situation, it returns an error indicating that the record is locked.

If the file control entry does contain a FILE STATUS clause and there is a USE declarative procedure defined for a file, the record lock status is normally reported to the program immediately by calling the USE procedure. By using the USE-PROCEDURE-RECORD-LOCK-TIMEOUT keyword (see page 320) in the RUN-FILES-ATTR configuration record, the runtime system can be told how long to wait before calling the USE procedure. If the record is unlocked during this time, the USE procedure is not called.

### ADVANCING ZERO LINES Phrase (WRITE Statement)

As explained in the discussion on the WRITE statement in Chapter 6: *Procedure Division Statements* of the *RM/COBOL Language Reference Manual*, WRITE statements acting on sequential files allow overprinting on a line for systems whose physical devices support this feature. However, some line printers are designed to advance one line after each line is printed. In such a case, the ADVANCING ZERO LINES phrase is treated as an ADVANCING 1 LINE phrase.

### ADVANCING mnemonic-name Phrase (WRITE Statement)

RM/COBOL allows the WRITE . . . ADVANCING *mnemonic-name* statement when *mnemonic-name* is associated with a channel-name from the SPECIAL-NAMES paragraph. No standard way exists to communicate channel-slewing information. Because of this, the runtime system interprets C01 as if page were specified; it interprets any other channel as if 1 LINE were specified.

## REEL and UNIT Phrases (CLOSE Statement)

The REEL and UNIT phrases are not supported. If specified, they are ignored.

## WITH NO REWIND Phrase (CLOSE Statement)

A print file is released to the operating system at run unit termination or when a CLOSE statement (without the WITH NO REWIND phrase) is issued.

The WITH NO REWIND phrase may be used to prevent the release of a print file to the operating system. This feature may be used to prevent undesirable side effects such as banner pages and form feeds provided by the operating system when the print file is released.

A subsequent OPEN statement (typically, OPEN EXTEND) must be issued before the program can again successfully access the print file.

A print file is a line sequential file that has any or all of the following RM/COBOL source program features:

1. ASSIGN TO PRINT or ASSIGN TO PRINTER clause in the file control entry for the file.
2. LINAGE clause in the file description entry for the file.
3. ADVANCING phrase in a WRITE statement for the file.

For all other file types, the WITH NO REWIND phrase is ignored.

## Device Support

Files that are opened on devices are treated as read-only (INPUT) or write-only (OUTPUT or EXTEND) sequential files. A program may open the same device more than once within the same run unit. Those devices that are opened with the same mode (read-only or write-only) share the same file handle and the same buffer. Those devices that are opened with different modes use different file handles and have different buffers.

At most, two buffers are allocated to each device; one when opened for read-only and one when opened for write-only. These buffers are dedicated to the device and do not come from the buffer pool.

Because of the non-portability of applying locks to devices, locks are never applied to device files. Thus, files opened on devices WITH LOCK do not guarantee exclusive access to the file.

## Printer Support

Under UNIX, printer support is provided either through the lp or the lpr spooler. Under Windows, RM/COBOL provides printer support using the Windows printer devices.

Under UNIX, the RM/COBOL runtime system creates a write-only pipe to the print spooler and sends all print records to it. The pipe is closed and the output is allowed to print only when the RM/COBOL program issues a CLOSE statement to the file (except for CLOSE WITH NO REWIND, as explained previously).

Under Windows, when a file is opened with the name PRINTER, the output is sent to the default system printer configured under the Windows operating system. When a file is opened with the name PRINTER*x*, where *x* is a single-digit number, output is sent to the device connected to LPT*x*.

The device is opened for writing only and is closed only when the RM/COBOL program issues a CLOSE statement to the file (except for CLOSE WITH NO REWIND, as explained previously).

The destination for files named PRINTER can be changed by doing any of the following:

- Assigning an environment variable PRINTER, as discussed in File Access Names on UNIX (on page 25) and File Access Names on Windows (on page 63).
- Defining a synonym in the Windows registry, as detailed in Setting Synonym Properties (on page 85).
- Configuring a keyword in the DEFINE-DEVICE configuration record (on page 304).

## Tape Support

Under UNIX, RM/COBOL provides tape support through direct access to the tape devices. Given the default configuration of RM/COBOL for UNIX, as described in Chapter 10, when a file is opened with the name TAPE, the RM/COBOL runtime system opens the tape device and writes or reads to it as directed by the RM/COBOL program. The tape device is closed when the RM/COBOL program issues a close to the file except for CLOSE WITH NO REWIND. Since the operating system does not provide a mechanism to write a tape mark to a tape without closing the device (which may rewind it), the RM/COBOL runtime system only simulates the close as it does for the printer device.

Most systems allow multiple files to be placed on a tape by specifying that the tape is not to be rewound on OPEN and CLOSE. This must be specified at the OPEN of the file by using the correct name for the operation. For example, rtpyy may mean rewind on OPEN and CLOSE while rtpyn may mean rewind on OPEN but not on CLOSE. (Actual names vary between implementations. Consult your system administrator for the actual names.) If it is desired to place multiple files on a single tape, it will be necessary to use DEFINE-DEVICE configuration records to name the different options desired and use the names within the RM/COBOL program appropriately.

All sequential record formats are supported on the tape: line, binary fixed, and binary variable-length records. For transfer of information to other RM/COBOL programs, any method may be used. For transferring information to other programs executing on other equipment, fixed-length binary records are most compatible.

Tape files are read or written in 512-byte blocks by default. If the COBOL program specifies a nonzero maximum block size in the BLOCK CONTAINS clause of the file description entry for the file, that block size is used instead of 512. Variable length blocks and unblocked files are not supported on tape devices.

The destination for files named TAPE can be changed either by assigning an environment variable TAPE, as explained in the discussion of File Access Names on UNIX (on page 25), or by using a DEFINE-DEVICE configuration record (on page 304).

### Named Pipe Support

Under UNIX, named pipes are treated as sequential devices. They are allowed to be opened as INPUT, OUTPUT or EXTEND, but never I-O. Also, as with the other devices, OPEN WITH LOCK does not guarantee exclusive access to the pipe.

Named pipes are special FIFO (first in first out) files that are created with an **mknod** command that specifies the **p** option after the name. For example:

```
mknod MyNamedPipe01 p
```

makes a named pipe with the name **MyNamedPipe01**. Super-user privileges may be required to run the **mknod** command, but not necessarily when the **p** option is specified. A named pipe may be deleted with the **rm** command. Two run units running at the same time may communicate through the pipe by each using a file access name that refers to the pipe by its name (including the directory path, if necessary).

## Relative Files

Relative files are ordered collections of records, each of which is directly addressable by use of a relative record number; this number is a non-zero integer corresponding to the ordinal position of a record within the file. A record within a relative file acquires its record number when it is written to the file with a WRITE statement.

If the access mode of the file at the time the record is written is sequential, the record number is assigned by the runtime system, and is one greater than the last record number assigned (or 1 if no records have been written to the file). If the access mode is random or dynamic, specify the record number before execution of the WRITE statement. To do this, move (or otherwise assign) the value to the data item declared to be the relative key for the file.

Record numbers do not necessarily correspond to actual records in a file. For instance, a record number may never have been given a corresponding data record, or some numbers may correspond to data records that have been deleted.

Relative files must be assigned to a disk device, since the records can be accessed randomly.

The maximum record length is 65280 bytes. Records can be variable length. No restrictions are placed on the value of individual bytes within the record.

Records in a relative file are written to disk in a fixed-length area four bytes longer than the length of the longest record declared for the file. Estimate the total disk space requirement by multiplying the maximum record length plus four by the anticipated number of records.

## RECORD Clause (Relative File Description Entry)

The RECORD clause specifies the minimum and maximum record lengths of records in a relative file. The maximum record length is a fixed attribute of the file, and is validated against the file size during an OPEN statement.

The minimum record length is not recorded with the file; however, a READ statement encountering a record whose length is less than the minimum record length will receive an error. A REWRITE statement may change the record length of a record in a relative file.

## BLOCK CONTAINS Clause (Relative File Description Entry)

In a single-user environment, relative files are physically read and written in fixed-length pieces called blocks. In a shared environment, relative files are read and written in blocks only when the WITH LOCK phrase is specified. When the WITH LOCK phrase is omitted, the records of the file are read or written individually (without blocking) from the operating system.

The size of a block is determined by the BLOCK CONTAINS clause in the RM/COBOL program. A larger block size causes more data to be transferred in a single request, requires more time to affect the data transfer and requires more memory from the buffer pool. A smaller block size allows more blocks in memory for a fixed amount of memory. Blocking may increase performance; however, because blocking may defer the physical writing of a block to disk until well after the WRITE statement that first places data in the block, errors (including loss of data) caused by that physical write may not be reported until a subsequent statement for the file is executed.

The file block size is not considered a fixed attribute of a relative file. Different programs may specify different block sizes for the same file.

The runtime system uses the following algorithm to determine the block size when opening a relative file:

1. If no BLOCK CONTAINS clause is present, the block size is typically 4096 characters if the file is opened for DYNAMIC or SEQUENTIAL access. The default block size may be changed with a RUN-REL-FILES configuration record (on page 325).
2. If a BLOCK CONTAINS *nnn* CHARACTERS clause is present, the block size is the specified number of characters.
3. If a BLOCK CONTAINS *nnn* RECORDS clause is present, the block size is the specified number of records multiplied by the sum of the maximum record size, plus four. The maximum allowed block size on most systems is 65489 characters.

## RESERVE Clause (Relative File Control Entry)

The RESERVE clause is ignored. Allocating buffer memory is automatically managed based on the amount of activity of a particular file. See File Buffering (on page 221).

### CODE-SET Clause (Relative File Control Entry or File Description Entry)

The CODE-SET clause determines the character set used to represent the relative file on disk. For example, if the CODE-SET is EBCDIC, all records written to the file will be translated from ASCII to EBCDIC. The CODE-SET is not considered a fixed attribute of the relative file. Different programs may specify different character sets for the same file.

### WITH NO LOCK Phrase (READ Statement)

A READ statement with the NEXT phrase and without the WITH NO LOCK phrase that fails because the record to which the file position indicator points is locked, does not modify the file position indicator and the contents of the record area are undefined. The record is not read and the program should not depend on the contents of the record area being unchanged. A subsequent READ statement behaves as if the failing READ statement had never been issued.

A READ statement with the WITH NO LOCK phrase may be used to read a record without regard to the lock status of the record. If an OPEN INPUT statement opened the file, the WITH NO LOCK phrase is assumed on all READ statements on the file.

If the file control entry does not contain a FILE STATUS clause or there is no USE declarative procedure defined for a file, record lock status is not reported to the program. Instead, the runtime system waits for the record to become unlocked. By using the FATAL-RECORD-LOCK-TIMEOUT keyword (see page 317) in the RUN-FILES-ATTR configuration record, the runtime system can be made to return a fatal error instead of waiting indefinitely. If the record is locked using a different file descriptor in the same run unit, then the runtime system never waits. Instead, to prevent a deadlock situation, it returns an error indicating that the record is locked.

If the file control entry does contain a FILE STATUS clause and there is a USE declarative procedure defined for a file, the record lock status is normally reported to the program immediately by calling the USE procedure. By using the USE-PROCEDURE-RECORD-LOCK-TIMEOUT keyword (see page 320) in the RUN-FILES-ATTR configuration record, the runtime system can be told how long to wait before calling the USE procedure. If the record is unlocked during this time, the USE procedure is not called.

## Indexed Files

Indexed organization files contain data and one or more indexes or keys. These indexes are used to locate data in the file by identifying the key value and the location in the file of the corresponding record.

Every record in an indexed file contains a prime record key and may contain a number of alternate record keys.

## Data Compression

Each record of the file may be represented in a compressed or uncompressed data format. Data record compression replaces multiple occurrences of space, zero and null or repeated characters with a single compression character. Uncompressed data records contain the data written by the program, with no compression characters. Data record compression usually improves the performance of indexed files by reducing the file size and allowing more information to be read in a single physical transfer. When an indexed file is created by the runtime system, data record compression is enabled. This default may be changed as described in the DATA-COMPRESSION keyword (see page 231) of the RUN-INDEX-FILES configuration record. Whether data record compression is enabled for a particular file may be established with the Define Indexed File (rmdefinx) utility (on page 594).

RM/COBOL indexed files allow index keys to be compressed or uncompressed. Key compression replaces the leading characters of a key that equal the preceding key, and any trailing space characters of a key, with compression characters. This usually reduces the amount of disk space occupied by a file, especially a file in which many keys contain trailing spaces (such as names and addresses). When an indexed file is created by the runtime system, key compression is enabled. This default may be changed as described in the KEY-COMPRESSION keyword (see page 321) of the RUN-INDEX-FILES configuration record. Whether key compression is enabled for a particular file may be established with the **rmdefinx**.

## Data Recoverability

RM/COBOL provides a choice of data recovery strategies for indexed files. A data recovery strategy is a tradeoff between deferring the writing of data to disk in order to improve the performance of file modification operations, and forcing the writing of data to disk in order to guarantee that the data is available for recovery if there is a system failure. The data recovery strategy for a particular indexed file is determined when the file is first opened OUTPUT or when the file is predefined. The file may be predefined using the Define Indexed File (rmdefinx) utility (on page 594). If the file is not predefined, the default recovery strategy is used by the runtime system. This default strategy may be configured as described in the discussion of the RUN-INDEX-FILES configuration record (on page 320). Unlike other attributes associated with an indexed file, the recovery strategy for a file may be changed, thus allowing optimal performance when a file is being built and switching to a higher level of data integrity when a file is updated. Use the Define Indexed File (**rmdefinx**) utility to change the recovery strategy for a file.

The following four options are available to provide the various levels of data recovery strategy:

1. **Force File Closed (FORCE-CLOSED keyword).** The indexed file header contains a count of the number of concurrent run units that have the file open for modification. If the system fails while this count is non-zero, the file index structure must be rebuilt using the Indexed File Recovery (recover1) utility (on page 598). Selecting this option causes the runtime system to increment this count before each DELETE, REWRITE, or WRITE operation and decrement the count at the end of each operation. Otherwise, the count will be incremented when the OPEN statement is executed and decremented when the CLOSE statement is executed.

Selecting this option causes two additional disk transfers for each modify operation, but gives a high probability that a file rebuild will not be required if a system failure occurs.

Selecting this option causes the runtime system to act as if the following three options are also set.

2. **Force Disk (FORCE-DISK keyword).** Your operating system maintains a system disk buffer pool in its memory. Issuing an operating system write request, as described below in the Force Data option, causes data to be written from the buffer maintained by the runtime system to a buffer in this disk buffer pool. The data is not necessarily written to the disk at that time. Thus, the selected data recovery strategy may be defeated.

   Selecting this option causes the runtime system to attempt to force the operating system to actually write its buffer to disk.

3. **Force Data (FORCE-DATA keyword).** Selecting this option causes the runtime system to issue a write request to the operating system when a block containing a data record is modified. Otherwise, such blocks remain in the block buffer pool maintained by the runtime system for the file, and the write request to the operating system is not made until the buffer containing the block is needed for a different block.

   This option is available only when a file is in single-user mode, that is, when the runtime system is in single-user mode or the file is opened WITH LOCK. This option is always selected for files in a shared environment.

4. **Force Index (FORCE-INDEX keyword).** Selecting this option causes the runtime system to issue a write request to the operating system when a block containing index information is modified. Otherwise, such requests are issued only as buffer availability in the block buffer pool maintained by the runtime system dictates. This option is always selected for files in a shared environment.

## *RECORD Clause (Indexed File Description Entry)*

The RECORD clause specifies the minimum and maximum length of records in an indexed file. These record lengths are considered fixed attributes of the file. Any program using an indexed file must specify the minimum and maximum length specified in the program that created the file.

The following algorithm computes the maximum disk space required to represent a record:

1. Assume the record representation disk space is the maximum record size of the file.
2. If data record compression is enabled, increase the record representation space by the ceiling of the maximum record size divided by 127.
3. Increase the record representation space by four times the number of alternate keys that allow duplicates.
4. Add four to the record representation disk space.

RM/COBOL indexed files are restricted both in the maximum record size declared in the program and in the maximum disk space required to represent a record. The maximum record size allowed an indexed file is 65280. The maximum disk space required to represent a record must not exceed 65527. The disk space limitation is the more restrictive, especially when data record compression is enabled.

## BLOCK CONTAINS Clause (Indexed File Description Entry)

RM/COBOL indexed files are physically read and written in fixed-length pieces called blocks. The size of a block is determined by the BLOCK CONTAINS clause in the RM/COBOL program, or with the Define Indexed File (rmdefinx) utility (on page 594). A larger block size transfers more data in a single request, requires more time to affect the data transfer, and requires more memory from the buffer pool. A smaller block size allows more blocks in memory for a fixed amount of memory, but requires more time to randomly access a record by increasing the depth of each index.

The file block size is considered a fixed attribute of the file. The BLOCK CONTAINS clause in a program that uses a file must be identical to the BLOCK CONTAINS clause in the program that created the file.

To access a block of an indexed file, the runtime system must use a piece of the buffer pool memory that is at least as large as the file block size. The indexed file algorithms require that a minimum of three pieces of buffer pool memory be available. If the file block size exceeds 32768, the buffer pool must be at least as large as the sum of 131072 and the block size. If the file block size exceeds 16384, the buffer pool must be at least as large as the sum of 65536 and the block size.

The algorithm used to determine the block size of an indexed file is outlined in the following paragraphs. The algorithm distinguishes specified block size from actual block size. Specified block size is defined by the BLOCK CONTAINS clause, and may be a function of the maximum disk space required to represent a record, defined previously in the RECORD clause description. Actual block size is defined as a function of the specified block size. When the term block size is used by itself elsewhere in this document in reference to an indexed file, it means the computed actual block size.

1. If no BLOCK CONTAINS clause is present, and the file already exists, the current block size is used as the block size.
2. If no BLOCK CONTAINS clause is present, and the file does not already exist, the block size is 512 (under Windows) or the value of BUFSIZ, taken from the C include file <**stdio.h**> (under UNIX). This default may be changed with a RUN-INDEX-FILES configuration record (on page 320).
3. If a BLOCK CONTAINS *nnn* CHARACTERS clause is present, the specified block size is the number of characters.
4. If a BLOCK CONTAINS *nnn* RECORDS clause is present, the specified block size is eight characters more than the specified number of records multiplied by the maximum disk space required to represent a record, as described above.
5. The minimum block size is the smallest value that meets all of the following criteria:
    a. The minimum block size must be at least 256, or 266 if the file version level is greater than 3. For more information on indexed file version numbers, see Indexed File Version Levels (on page 243).
    b. The minimum block size must be at least 292 (302 if the file version level is greater than 3) if an enumerated CODE-SET or COLLATING SEQUENCE is specified.
    c. The minimum block size must be sufficient to contain three index keys. This is approximately the longest key length times 3 plus 46 (plus 10 if the file version level is greater than 3).

d. The minimum block size must be sufficient to contain one data record. This minimum is eight characters (18 characters if the file version level is greater than 3) more than the maximum disk space required to represent a single record, as described above.

e. The minimum block size must be at least the value of the MINIMUM-BLOCK-SIZE keyword (see page 321) of the RUN-INDEX-FILES configuration record. If not specified, the default of this parameter is 1024.

6. If the block size is rounded to a multiple of a "nice" block size, which is controlled by the ROUND-TO-NICE-BLOCK-SIZE keyword (see page 321) of the RUN-INDEX-FILES configuration record, the actual block size chosen is the greater of the minimum block size and specified block size (or the default block size if no BLOCK CONTAINS clause is present), rounded up to a multiple of 512 (under Windows) or the value of BUFSIZ, taken from the C include file <**stdio.h**> (under UNIX). If the block size is not rounded to a multiple of a "nice" block size, the actual block size chosen is the lowest multiple of the specified block size (or the default block size if no BLOCK CONTAINS clause is present) that is not less than the minimum block size.

7. If the computed actual block size does not meet all the following restrictions, the indexed file description is invalid and the file cannot be opened:

   a. The actual block size must not exceed 65489.

   b. If key entries are compressed, the actual block size must not exceed 65499 less four times the length of the longest key. If no alternate record key allows duplicates, the limit is 65515 less four times the length of the longest key.

   c. If key entries are uncompressed, the actual block size must not exceed 65526, less the length of the longest key. If no alternate record key allows duplicates, the limit is 65530, less the length of the longest key.

## *RESERVE Clause (Indexed File Control Entry)*

The RESERVE clause is ignored. Buffer memory is automatically managed based on the amount of activity of a particular file. See File Buffering (on page 221).

## *CODE-SET Clause (Indexed File Control Entry or File Description Entry)*

The CODE-SET clause determines the character set used to represent the indexed file on disk. For example, if the CODE-SET is EBCDIC, all records written to the file will be translated from ASCII to EBCDIC. The CODE-SET specified when an indexed file is created is a permanent attribute of the file, and will be used whenever the file is accessed by a program not specifying a CODE-SET. However, a program may specify a different CODE-SET than that used to create the file, and that code set will be used for the duration of the program.

## COLLATING SEQUENCE Clause (Indexed File Control Entry)

The COLLATING SEQUENCE clause allows a program to determine the collating function used when comparing keys in an indexed file (for example, whether uppercase and lowercase letters are to be treated identically). The collating function specified when an indexed file was created is a fixed attribute of the file. If a program opening an indexed file specifies a COLLATING SEQUENCE clause, the specified collating function must be identical to that specified when the file was created.

## WITH NO LOCK Phrase (READ Statement)

A READ statement with the NEXT phrase and without the WITH NO LOCK phrase that fails because the record is locked does not modify the file position indicator and the contents of the record area are undefined. The record is not read and the program should not depend on the contents of the record area being unchanged. A subsequent READ statement will behave as if the failing READ statement had never been issued.

A READ statement with the WITH NO LOCK phrase may be used to read a record without regard to the lock status of the record. If an OPEN INPUT statement opened the file, the WITH NO LOCK phrase is assumed on all READ statements on the file.

If the file control entry does not contain a FILE STATUS clause or there is no USE declarative procedure defined for a file, record lock status is not reported to the program. Instead, the runtime system waits for the record to become unlocked. By using the FATAL-RECORD-LOCK-TIMEOUT keyword (see page 317) in the RUN-FILES-ATTR configuration record, the runtime system can be made to return a fatal error instead of waiting indefinitely. If the record is locked using a different file descriptor in the same run unit, then the runtime system never waits. Instead, to prevent a deadlock situation, it returns an error indicating that the record is locked.

If the file control entry does contain a FILE STATUS clause and there is a USE declarative procedure defined for a file, the record lock status is normally reported to the program immediately by calling the USE procedure. By using the USE-PROCEDURE-RECORD-LOCK-TIMEOUT keyword (see page 320) in the RUN-FILES-ATTR configuration record, the runtime system can be told how long to wait before calling the USE procedure. If the record is unlocked during this time, the USE procedure is not called.

## File Allocation

As an indexed file grows in size, the runtime system allocates additional blocks to the file. The number of blocks allocated is determined by the allocation increment. The default allocation increment is eight blocks; a different allocation increment may be set with the Define Indexed File (rmdefinx) utility (on page 594). A larger allocation increment may improve performance when writing records to the file, by reducing the number of operating system allocation requests. A smaller allocation increment may yield less wasted space.

Unused blocks in an indexed file are kept on the empty block list. The format of this list is determined by the file version number. For more information, see Indexed File Version Levels (on page 243). Beginning with file version number 2, a new list format and algorithm are used to maintain the empty block list in order to increase performance when adding records to the file. This new algorithm keeps track of the

first unused empty block in the file, which is followed only by other unused empty blocks, and avoids reading those blocks from the disk when they are used. New files are created with a version number of 4, although this default can be changed with the DEFAULT-FILE-VERSION-NUMBER keyword (see page 320) of the RUN-INDEX-FILES configuration record. The version number can also be changed on existing files or set when creating new files with the Define Indexed File (**rmdefinx**) utility.

When a file version number is set to 2, the minimum write version number of the file is also set to 2 in order to prevent previous versions of RM/COBOL from attempting to modify the empty block list. The minimum read version number of the file is not changed. Files with a version number of 2 can be read but not modified by versions of RM/COBOL prior to version 6. If an OPEN OUTPUT is performed on a file with version number of 2 by a runtime system prior to version 6, the file version number and minimum write version number will be reset to 0, and the previous style of the empty block list will be used.

## File Size Estimation

Use the following formulas to set upper and lower bounds on the number of characters that are required. Of course, with record data compression and key compression, the actual disk space required for records and keys varies greatly with the contents of the records.

In these formulas, “ceiling” means rounding up and “floor” means rounding down.

For example:

```
7.145
```

has a ceiling of 8 and a floor of 7.

```
Variables -  A     =  allocation increments in blocks
             B     =  block size in bytes
             Kn    =  length of nth key in bytes
             MaxL  =  maximum record size
             MinL  =  minimum record size
             N     =  number of keys
             Nd    =  number of keys that allow duplicates
             R     =  number of records

Subtotals -  H     =  1
             H     =  H + 2  if code-set is enumerated
             H     =  H + 1  if collating sequence is enumerated

                      If N > floor ((B - 256)/36) then
             H     =  H+ceiling((N-((B-256)/36))/((B-6)/36))

                      If records are compressed:
          MaxD     =  ceiling(R/(floor((B-8)/(MaxL+4+
                      (4*Nd)+ceiling(MaxL/127)))))
          MinD     =  ceiling(R/(floor((B-8)/(ceiling
                      (MinL/65)+4+(4*Nd)))))

                      If records are uncompressed:
          MaxD     =  ceiling(R/(floor((B-8)/
                      (MaxL+4+(4*Nd)))))
          MinD     =  ceiling(R/(floor((B-8)/
                      MinL+4+(4*Nd)))))

                      For each key n, 0 through N-1:
             TOn   =  4
             TOn   =  TOn+4 if key n allows duplicates

                      If key compression is enabled:
          TOn      =  TOn+2
          MaxTEn   =  floor((B-10)/TOn)
          MinTEn   =  floor((B-10)/(2*(TOn+Kn)))
          MaxLEn   =  floor((B-10)/(TOn+1))
          MinLEn   =  floor((B-10)/(2*(TOn+Kn+1)))

                      If key compression is not enabled:
          MaxTEn   =  floor((B-10)/(TOn+Kn))
          MinTEn   =  ceiling(MaxTEn/2)
          MaxLEn   =  floor((B-10)/(TOn+Kn+1))
          MinLEn   =  ceiling(MaxLEn/2)

                      For each key n:
          MaxLBn   =  ceiling(R/MinLEn)
          MinLBn   =  ceiling(R/MaxLEn)
          MaxTHn   =  ceiling(Log10(MaxLBn)/Log10(MinTEn))
          MinTHn   =  ceiling(Log10(MinLBn)/Log10(MaxTEn))
          MaxWn    =  ceiling(MaxLBn/(MinTEn**(MaxTHn-1)))
          MinWn    =  floor(MinLBn/(MaxTEn**(MinTHn-1)))
          MaxTBn   =  1+MaxWn*(((MinTEn**
                      (MaxTHn-1))-1)/(MinTEn-1))
          MinTBn   =  1+MinWn*((MaxTEn**
                      (MinTHn-1))-1)/(MaxTEn-1)

Totals -  MaxFB    =  H+MaxD+Sum(MaxLBn+MaxTBn)
          MinFB    =  H+MinD+Sum(MinLBn+MinTBn)
          MaxSize    =  (A*ceiling(MaxFB/A))*B
          MinSize    =  (A*ceiling(MinFB/A))*B
```

For example, suppose a single key indexed file of 2000 records is written. Every record is 100 bytes long. The prime record key is 15 bytes. Assume a default block size of 512 bytes, allocation increment of 8 blocks, the default compression state, and that key compression and record compression are enabled. No code-set or collating sequence is specified.

```
Variables -  A     =  8
             B     =  512
             K0    =  15
             MaxL  =  100
             MinL  =  100
             N     =  1
             Nd    =  0
             R     =  2000

Subtotals -  H     =  1
          MaxD     =  ceiling(2000/(floor((512-8)/(100+
                       4+(4*0)+ceiling(100/127)))))
                   =  500

             MinD  =  ceiling(2000/(floor((512-8)/
                       (ceiling(100/65)+4+4(4*0)))))
                   =  24
                TO0   =  4
                TO0   =  4+2
                   =  6
       MaxTE0      =  floor((512-10)/6)
                   =  83
       MinTE0      =  floor((512-10)/(2*(6+15)))
                   =  11

       MaxLE0      =  floor((512-10)/(6+1))
                   =  71
       MinLE0      =  floor((512-10)/(2*(6+15+1)))
                   =  11

       MaxLB0      =  ceiling(2000/11)
                   =  182
       MinLB0      =  ceiling(2000/71)
                   =  29

       MaxTH0      =  ceiling(Log10(182)/Log10(11))
                   =  3
       MinTH0      =  ceiling(Log10(29)/Log10(83))
                   =  1

        MaxW0       =  ceiling(182/(11**(3-1)))
                    =  2
        MinW0       =  floor(29/(83**(1-1)))
                    =  29

        MaxTB0      =  1+2*(((11**
                       (3-1))-1)/(11-1))
                    =  25
        MinTB0      =  1+29*((83**
                       (1-1))-1)/(83-1)
                    =  1
Totals  -  MaxFB    1+500+(182+25)
                    =  708
           MinFB    =  1+24+(29+1)
                    =  55
           MaxSize  =  (8*ceiling(708/8))*512
                    =  364544
           MinSize  =  (8*ceiling(55/8))*512
                    =  28672
```

This file will require between 28672 and 364544 bytes, depending on the contents of the records and the order in which the records are written. Since, in most cases, the key and data compression will save much less than all the bytes in the keys and records, the actual file size will probably be nearer to the maximum size than to the minimum size.

# Temporary Files

The sort-merge facility of RM/COBOL makes use of temporary files for its intermediate sort files. These files are given unique names and are placed in the current directory and the current disk drive.

The choice of the directory in which to place the temporary files under Windows may be changed by use of the environment variables TMP or TEMP. If both variables are set, the value of TMP is used. Assign the environment variable a value before executing the RM/COBOL program. Under Windows, a synonym may be set in the Windows registry database. Under UNIX, the TMPDIR environment variable is used.

# Indexed File Performance

RM/COBOL indexed files contain data records and indexes to these data records. The index structure is based on B+ trees, described by D. E. Knuth and others. The index for each record key is maintained as a separate tree in the file. Each tree consists of nodes that contain key values and pointers to records or to other nodes. If an index node can contain N keys and pointers, a file of N or fewer records can be indexed by a single node. A file containing N*N records may be indexed by two levels of nodes, a root node of N entries pointing to N leaf nodes, each leaf node containing N entries pointing to records. Each random access to an indexed file requires that the entire index tree be traversed from the root node through any intermediate nodes to the leaf node and finally to the desired data record. Write operations to a file require adding the data record to the file, adding each key of the record to the appropriate index tree, possibly creating new leaf and intermediate nodes, or even creating a new root node and increasing the height of the index tree.

Since modifications to an indexed file require updating the index tree for each key added, changed, or deleted, one way to improve performance is to reduce the number of record keys. An application may use an index key to rapidly find a particular record. Some applications use record keys simply to sort records for report generation. If a record key is used only to access records in a particular order when generating a report, consider removing that record key and sorting the records with the SORT verb when the report is generated. Not only will this reduce the number of disk operations by using the more efficient sort algorithm, but it will also perform the operations only when the report is requested. When an index key is used to select a subset of the file's records for a report, the same selection can be performed with a SORT INPUT PROCEDURE.

The index tree manipulation algorithms are designed to keep each index node at least half full of entries. When records are written in ascending order by key, the algorithms can completely fill each index node with entries, reducing both the tree height and the size of the file. This improves performance by reducing both the number of disk operations and the required arm movement for the operations

performed. If a file is infrequently modified, it should be built in ascending order by the key most frequently used to access the records.

## In-Memory Buffering

Disk operations can be reduced by keeping file fragments in memory as long as possible. Subsequent accesses to the same information use the copy in memory instead of rereading the information from disk. This technique of in-memory buffering of data is the single most effective method of improving performance.

RM/COBOL files offer two levels of in-memory buffering of data: buffering by the operating system and buffering by the runtime system. A combination of the two seems to yield the best performance. Both UNIX and Windows provide buffering automatically. Additional memory may be dedicated to the buffering of disk data by further increasing the number of operating system buffers, or by increasing the size of the memory pool that the runtime system uses for disk buffers. The size of the memory pool may be increased through use of the BUFFER-POOL-SIZE keyword (see page 316) on the RUN-FILES-ATTR configuration record.

Runtime system disk buffers, like the operating system buffers, are shared among all files opened by an application. Runtime system buffering is especially useful in a network environment. When a file is shared over a network, the operating system is unable to perform any buffering. Every WRITE operation must be sent to the remote machine and every READ operation must receive the data from the remote machine. With runtime system buffering of index files, at the beginning of each operation the runtime system determines which buffers still contain valid data and which contain data that may have been modified by an application on another machine. This significantly reduces the number of remote READ operations required (although all modified data must still be written to the remote machine).

Providing sufficient space in the buffer memory pool can supply the runtime system with enough buffers to work efficiently.

The optimal number of buffers for a particular file depends on the operations being performed and the height of each index tree. The height of the index trees for a particular indexed file may be obtained with the Map Indexed File (rmmapinx) utility (on page 589). If a file is being read randomly by only a single key, the minimal number of buffers that will improve performance is the height of the particular key's index tree, plus one. Fewer buffers than this number require that all index nodes be read for each operation. Increasing the number of buffers above this minimum increases the probability that needed nodes already will be in memory. WRITE operations require that every key be updated. It is desirable that the number of buffers available be greater than the sum of the heights of all trees. Short of this number, buffers equal to the height of the tallest tree plus one should be available. Again, increasing the number of buffers above the minimum increases the probability that required nodes will already be in memory.

Once the desired number of buffers is known, the amount of space to allocate in the buffer pool can be computed by using the following calculations:

| | | | | |
|---|---|---|---|---|
| N | = | Desired Number of Buffers | | |
| BSz | = | Size of Buffer[1] | | |
| NBSeg | = | Number of Buffers per Segment | = | INT(65496/BSz) |
| NSeg | = | Number of Segments | = | INT(N/NBSeg) |
| PSz | = | Pool Size = NSeg*65496 + (N – NSeg*NBSeg)*BSz | | |

If more than one file is expected to be active at the same time, the computed Pool Size (PSz) for the files usually can be added together, especially if the files have the same buffer size. (In fact, if the files have the same buffer size, it is best to add the number of desired buffers together and make the calculation using that number.) However, if the files have different buffer sizes and the difference between the buffer size and the block size of the file having the larger buffer size is greater than the buffer size of another active file, the amount of space required will be less than the sum of the separate buffer pool size calculations for the two files. (Recall that the buffer size is computed from the block size rounded to the next power of two. Not all that space, however, is in use by the buffer; only the block size rounded up to the next multiple of 512. The remainder is available for use by other buffers.)

## Altering the Size of Indexed File Blocks

RM/COBOL indexed files are accessed in fixed size fragments called blocks. The size of these blocks is determined by the application with the BLOCK CONTAINS clause. The default block size is the smallest multiple of the disk sector size sufficient to contain one record or an index node. In general, this default block size is also the optimal choice. Although larger block sizes allow more entries in an index node and thus reduce the height of the index trees, this reduction in height is usually not sufficient to compensate for the reduced number of buffers available. Doubling a file's block size usually reduces the height of the trees by much less than a factor of 2. This means that if the memory available for buffers is just sufficient to contain the height of the tree at the smaller block size, it will no longer be sufficient at the larger block size. If there is insufficient memory even with a block size of only one sector for the desired minimal number of buffers discussed above (for example, the sum of the heights of the index trees when write operations are performed), a larger block size may improve performance. If the application does choose a block size greater than the default, this block size should still be a multiple of the disk sector size; otherwise, whenever a block is written, the operating system must read a sector of the disk, modify the block data in the sector, and rewrite the sector to disk.

## Controlling the Length of Record Keys

The length of record keys affects indexed file performance by affecting the height of the index trees. Each index node consists of as many pairs of keys and pointers as fit in a block. A longer key means fewer entries, and increases the number of disk operations. Key length is especially significant if uncompressed keys are requested. When key compression is enabled for a file, leading characters in an entry that duplicate the preceding entry and trailing space characters are removed. This reduces the node entry to its significant characters, reducing an entry whose key is equal to its preceding entry to only 10 bytes.

---

[1] The size of the buffer (Bsz) is computed by rounding the block size up to the next power of two. If this is less than 512, use 512 instead.

Whether a key allows or prohibits duplicates affects performance primarily as it affects the tree height. Node entries that allow duplicates are four bytes longer than node entries that prohibit duplicates. If the key length were very short, the height of the index tree may increase. Each key that allows duplicates also increases the length of the data record representation by four bytes. This is usually insignificant, but it may increase the size of the file and degrade performance if the records are very short or if a large number of keys allow duplicates. Of course, any file with a large number of keys may be expected to have poorer performance than a file with a relatively small number of keys. If there are fewer buffers available than the sum of the heights of the index trees, keys that prohibit duplicates degrade performance more than keys that allow duplicates. When a write request is made, every key that prohibits duplicates must be checked to see if the record to be written will create an illegal duplicate key value. When insufficient disk buffers are available, this will cause some additional disk operations.

Record length affects indexed file performance by placing a lower limit on the block size, perhaps forcing the block size to be larger than one sector, increasing the transfer time and reducing the number of buffers that will fit in available memory. Large record lengths also affect indexed file performance by increasing the disk space required by the file. RM/COBOL indexed files support variable-length records, using only sufficient disk space to contain the actual record data. Converting fixed-length records to variable-length records, however, is not necessary to improve performance. With data record compression enabled, fixed-length records with trailing space filled fields are almost as small as variable-length records. Of course, larger data records require more disk space. This can reduce the effectiveness of the disk buffers by reducing the percentage of the file that can be kept in memory and thus the probability that a desired file block will already be in memory.

## Correct Data Recovery Strategy

RM/COBOL indexed files support several optional data recovery strategies that determine when and how much data should be written to the operating system or disk when the application executes a file modification statement. Enabling a data recovery strategy that does not defer I/O operations degrades performance. This is especially true if index tree modifications are written in addition to data record modifications, since a single record operation may cause many index modifications. Of course, the user might consider disabling a data recovery strategy to be a performance problem if the machine is turned off without exiting the application and hours of data entry are lost.

## Using Key and Data Compression

RM/COBOL indexed files support key compression, in which leading characters of an index node entry key that match the preceding entry and trailing space characters are removed from the entry. This is usually a performance benefit, by increasing the number of entries in an index node, decreasing the height of the tree, decreasing the size of the file, and decreasing the number of disk operations. An exception occurs when sufficient buffers can be reserved to contain all or almost all of a file’s index trees. This will make the file operations compute bound instead of disk bound. Changing the file to have uncompressed keys may improve performance. Typically, this occurs when running small benchmark programs with all of the memory reserved for disk buffers. Key compression may also degrade performance when numeric keys are used or when the key length is only 1 or 2, by unnecessarily adding two

compression overhead characters to keys, which, in fact, compress by fewer than three characters.

RM/COBOL indexed files support data record compression, in which repeated characters in a data record are replaced by a single compression character. Since the compression algorithm operates on all repeated characters, not just repeated zeros and spaces, this compression almost always results in less disk space for each data record, less disk space for the entire file, less arm movement when reading the file, and fewer disk operations to access the file. It may not be beneficial on very short data records, and data records that are artificially forced to be incompressible. It also yields anomalous results when a file is initially created with records that are almost all spaces, and these records are then rewritten with the actual data.

## Using RM/COBOL Facilities

The WITH LOCK phrase on the OPEN statement may be used to improve the performance of exclusively accessed files in a network environment. When files are shared, at the beginning of each operation the runtime system must lock the file, read the header of the file to determine which buffers no longer contain valid data, and lock the record being modified. The file, and sometimes the record, is unlocked at the end of the operation. If a file should be opened by only one user at a time, then opening the file WITH LOCK avoids the locking and validation overhead on each operation. With a sufficient number of buffers, this can make the difference between four network transactions and zero network transactions to perform an operation.

Some applications may be written to delete and write a record when the record contents are changed, instead of using a REWRITE statement. A REWRITE statement always yields better performance than the DELETE and WRITE statements. This is because there is always at least one key whose value has not been changed, the prime record key. The index trees for unchanged keys need not be updated and, except for the prime record key, need not even be accessed.

## Indexed File Version Levels

Over time, the structure of RM/COBOL indexed files has changed in order to support new features or improved performance. Each indexed file is marked with an indexed file version level number that identifies the structure used for that file, which is established at the time the file is created. The various indexed file version levels are individually described in the following paragraphs. In addition, an explanation of how to change an indexed file from one version level to another, either up or down, is provided.

### *File Version Level 0*

Indexed files with a version level of 0 have the original RM/COBOL indexed file structure.

### *File Version Level 2*

Indexed files with a version level of 2 have a modified structure that can improve performance when adding records to the file. This feature is most noticeable when the file has been opened WITH LOCK. Files with a version level of 2 can be read but not modified by versions of RM/COBOL prior to version 6. Although by

default, files are created with a version level of 2, this can be changed by using the DEFAULT-FILE-VERSION-NUMBER keyword (see page 320) of the RUN-INDEX-FILES configuration record.

### File Version Level 3

Indexed files with a version level of 3 can grow to a larger size than those with a version level of 0 or 2. When creating a file with a version level of 3, the LARGE-FILE-LOCK-LIMIT keyword (see page 318) of the RUN-FILES-ATTR configuration record is used to determine the largest address that can be locked in the file. This value is also stored in the file header, allowing different files to have different lock limit values. Files with a version level of 3 cannot be read or written by prior versions of RM/COBOL. For information on how to change the default indexed file version number to 3 using the DEFAULT-FILE-VERSION-NUMBER keyword (see page 320) of the RUN-INDEX-FILES configuration record.

### File Version Level 4

Indexed files with a version level of 4 have new integrity features, including support for atomic I/O. Files with a version level of 4 cannot be read or written by RM/COBOL versions prior to version 7.5.

In addition, files with a version level of 4 may, like those with a version level of 3, grow to a larger size than those with a version level of 0 or 2. However, when creating a file with a version level of 4, the value of the USE-LARGE-FILE-LOCK-LIMIT keyword (see page 322) of the RUN-INDEX-FILES configuration record determines whether the LARGE-FILE-LOCK-LIMIT keyword (see page 318) or the FILE-LOCK-LIMIT keyword (see page 318) of the RUN-FILES-ATTR configuration record is used to determine the largest address that can be locked in the file. For information on how to change the default indexed file version number to 4 using the DEFAULT-FILE-VERSION-NUMBER keyword (see page 320) of the RUN-INDEX-FILES configuration record.

### Changing the File Version Level

The Define Indexed File (rmdefinx) utility (on page 594) can be used to change the version level of an existing file or to change the lock limit stored in the indexed file header for versions 3 and later. If the version level is changed, it may also be necessary to run the Indexed File Recovery (recover1) utility (on page 598). The Define Indexed File (**rmdefinx**) utility will indicate whether it is necessary to run the Indexed File Recovery (**recover1**) utility. Generally, when changing the version level to or from level 0 or to or from level 4, the Indexed File Recovery (**recover1**) utility must be run.

# Chapter 9: Debugging

RM/COBOL provides an interactive debugging facility, the RM/COBOL Interactive Debugger (called Debug). This chapter details the concepts, structure, and use of Debug.

This chapter describes the following aspects of Debug along with each of the Debug commands:



## Invoking a Program for Debug

In order to execute programs for debugging, enter the D Option in the RM/COBOL Runtime Command (see page 181). Any compiled RM/COBOL program is ready to be run under control of Debug. In general, no special Compile Command options or compilation steps are required. However, if data references are to be specified in the **D** (Display), **M** (Modify), **T** (Trap), and **U** (Untrap) Debug Commands, you need to enter either the A or Y Compile Command Option (you may need to enter both). See pages 144 and 148, respectively, for details on these options.

Any program within the run unit that was compiled with the Q Compile Command Option will not contain any debugging information and will, therefore, appear "invisible" to the Interactive Debugger.

Table 27 summarizes the RM/COBOL Debug commands.

**Table 27: Debug Command Summary**

| Command | Function |
|---|---|
| A (Address Stop) Command (on page 260) | Sets a breakpoint and resumes program execution. |
| B (Breakpoint) Command (on page 261) | Sets a breakpoint or displays all active breakpoints. |
| C (Clear) Command (on page 262) | Clears breakpoints. |
| D (Display) Command (on page 263) | Displays the value of a specified data item. |
| E (End) Command (on page 267) | Ends debugging and resumes standard program execution. |
| L (Line Display) Command (on page 267) | Specifies the line on which Debug displays will appear. |
| M (Modify) Command (on page 268) | Modifies the value of a specified data item. |
| Q (Quit) Command (on page 272) | Terminates program execution and returns control to the operating system. |
| R (Resume) Command (on page 272) | Resumes program execution at a specified location. |
| S (Step) Command (on page 273) | Executes the program to the start of the next statement, paragraph or section, or trace execution. |
| T (Trap) Command (on page 273) | Sets a data trap or displays all active data traps. |
| U (Untrap) Command (on page 277) | Clears data traps. |

# General Debug Concepts

This section highlights some general concepts about Debug.

## Statements

Debug considers section names, paragraph names and procedural statements to be statements for the purpose of setting breakpoints (on page 246), stepping (on page 247), execution counts (on page 247) or program area references (on page 249). Procedural statements are those RM/COBOL statements that begin with a Procedure Division verb (for example, IF, ADD, MOVE, READ, PERFORM, GO, STOP, and so forth).

## Breakpoints

Breakpoints can be set for any statement. When the RM/COBOL runtime system encounters a breakpoint, it stops before it executes the statement at which the breakpoint is set. At this point, using the appropriate Debug commands, you can examine and modify the value of data within the program. Note that line numbers are used to indicate breakpoints. Breakpoints may be set for lines that have no

statement (such as comment lines), but this does not cause a stop in response to the breakpoint. Any number of breakpoints can be set.

## Traps

Traps are used to compare the current value of a data item to its last known value, to see if a change has occurred. Whereas breakpoints stop program execution before the statement at which the breakpoint was set, traps wait until the statement has completed, then compare for a change in value. If such a change occurs, program execution is suspended and the current value of the data item appears. Any number of traps can be set.

## Stepping

When you step through the program, you direct that execution halt before the next statement, paragraph, or section is executed.

## Execution Counts

You can specify the number of times a breakpoint is to be ignored before Debug halts the program. For example, if you set a breakpoint at line 100 and specify an execution count of 15, upon resumption of execution the statement at line 100 will be executed 14 times. When the statement is encountered the 15th time, Debug will halt the program before executing that statement. If the statement is not executed the specified number of times before execution ends, execution will not stop in response to the breakpoint.

The maximum value for an execution count is 65535.

## Line and Intraline Numbers

References to statements are made by a line number, optionally modified by an intraline value. For example:

```
150  MOVE A TO B, MOVE B TO C, MOVE C TO D1
```

There are three procedural statements here: the first—MOVE A TO B—is referenced only by the line number 150. The second statement in the line—MOVE B TO C—is referenced by the line number and the intraline number (or offset): in this case, 150 + 1. And the third statement—MOVE C TO D—is referenced by the line number and its intraline number: 150 + 2.

## Debug Values

All numeric values used by Debug (for instance, to specify an address) are entered and displayed as decimal numbers. For more information, see Data Address Development (on page 249).

## Data Types

Certain Debug commands allow (and in some cases require) you to enter a value for *type*. *type* indicates the type of data item. This makes Debug aware of the internal representation of the data item specified. The value of *type* is generally the value shown for the data item in the Debug column (see page 158) of the allocation map. However, other values may be specified for *type* if you want the data item decoded according to a different representation.

The allowed values for *type* are listed in Table 28.

**Table 28: Valid Data Types**

| Data Type | Meaning | Data Type | Meaning |
|---|---|---|---|
| ABS | Alphabetic | NCU | Numeric unpacked unsigned |
| ABSE | Alphabetic edited | NLC | Numeric display signed (leading combined) |
| ANS | Alphanumeric | NLS | Numeric display signed (leading separate) |
| ANSE | Alphanumeric edited | NPP | Numeric packed unsigned positive |
| GRP | Group | NPS | Numeric packed signed |
| HEX | Hexadecimal | NPU | Numeric packed unsigned |
| IXN | Index-name | NSE | Numeric edited |
| NBS | Numeric binary signed | NSU | Numeric display unsigned |
| NBU | Numeric binary unsigned or index data item | NTC | Numeric display signed (trailing combined) |
| NBSN | Numeric binary signed native | NTS | Numeric display signed (trailing separate) |
| NBUN | Numeric binary unsigned native | PTR | Pointer |
| NCS | Numeric unpacked signed | | |

# Debug References

There are a number of ways to refer to specific lines of code or to specific data items within an RM/COBOL program.

## Program Area References

You can refer to statements by the line number as it exists in your source program, by an intraline number if more than one statement is on a source line, and by a program-name as it appears in the PROGRAM-ID paragraph in the Identification Division.

## Data Item References

A data item can be referred to by its address and size, plus the specific occurrence (subscript) information if defined within an OCCURS clause, plus the argument number if defined within the Linkage Section or the external number if defined with the external attribute. A data item may be referred to by its symbolic name if the symbol table is included in the object program (see the discussion of the Y Compile Command Option that begins on page 148).

# Screen Positions

You can request that Debug screen displays appear on a particular line. This feature is useful when you are debugging interactive programs (those that accept and display information on the screen) because it minimizes the risk of overwriting a program display with Debug commands or messages. For instance, if your screen display uses lines 10 through 25, you can direct that Debug commands and messages be displayed on line 6.

# Data Address Development

Several Debug commands require you to specify particular data items, or specific occurrences of specific data items. Debug provides three ways for you to enter this information:

1. Identifier Format (on page 250)
2. Address-Size Format (on page 252)
3. Alias Format (on page 254)

All three methods can be used during a single Debug session if the conditions required for their use are satisfied. Only one method can be used for an individual Debug command.

## Identifier Format

The identifier format is similar to the source format for identifier specification. This method requires that you specify (or configure) the Y Compile Command Option (see page 148), which outputs the symbol table to the object file for use by Debug. If the symbol table is absent from the object file, this format cannot be used.

The syntax is as follows:

```
name-1  [ { IN | OF }  name-2 ] …  [ script ]  [ refmod ]
              [ ,  { type | { * | & } [ type ] } ]  [ # alias ]
```

*name-1* is a name declared as a constant-name, data-name, or index-name in the Data Division of the current program or a constant-name, data-name, or index-name with the global attribute in a program that contains the current program. *name-2* is a qualifier for *name-1*. References to alphabet-names, cd-names, class-names, condition-names, file-names, mnemonic-names, paragraph-names, program-names, screen-names, section-names and symbolic-characters are not allowed, except that a file-name or cd-name may appear as the final qualifier of a data-name. The sequence of names must form a valid qualified reference to a data item, according to the rules for source program data references. A constant-name may only be specified in the **D** (Display) Command.

*script* is required if the data item referenced by *name-1* is a table element. If it is not, do not specify *script*. The format for *script* is as follows:

```
( integer-1 [ [ ,  ] integer-2 ] …  )
```

*integer-n* is a sequence of one or more decimal digits. Parentheses are required. Either commas or spaces can separate the integers. The number of integers must match the number of OCCURS clauses in the hierarchy of data description entries for the data item referenced by *name-1*. The value of *integer-n* is interpreted as an occurrence number in the same way as a literal subscript in the source program.

*refmod* selects a subfield of the data item in the same manner as a reference-modification in the source program. It has this format:

```
( offset : [ length ] )
```

*offset* is a string of decimal digits whose value ranges from 1 to the length of the data item referenced by *name-1*. The parentheses and the colon following *offset* are required.

*length* is a string of decimal digits whose value ranges from 1 to the remaining length of the data item referenced by *name-1*, after *offset* has been applied. Failure to specify *length* requests the maximum length from *offset* to the end of the data item referenced by *name-1*.

*type* specifies the type of data item referenced. If this parameter is omitted, *type* defaults to the type of the named data item except when a type modifier is specified. (See the list of valid data types in Table 28 on page 248.) The type

value **IXN** may only be used with index-names and, when the named data item is an index-name, the only permissible type value is **IXN**. The type value **PTR** may only be used with pointer data items and, when the named data item is a pointer data item, the only permissible type value is **PTR**, except when a type modifier is specified.

The type modifiers ***** and **&** have the following effect:

- The ***** type modifier indicates an indirect reference, that is, a reference to the data item referenced by a pointer data item value. The data item specified in the command must be a pointer data item (data type **PTR**). In this case, *type*, if specified, indicates the type of the data item referenced by the pointer data item value. If *type* is not specified, the default is hexadecimal. If *refmod* is not specified, the command refers to the entire effective memory area specified by the pointer data item value. The effective memory area specified by a pointer data item value begins with the effective address (*pointer.address* + *pointer.offset*) and ends just before the effective limit address (*pointer.address* + *pointer.length*). If *refmod* is specified for an indirect reference, *refmod* is applied to the indirect reference rather than the pointer data item itself. Thus, *refmod* may be used to specify an offset and length (subfield) within the current effective memory area of an indirect reference.
- The **&** type modifier indicates that the address of the data item specified is the data item referenced by the command. In this case, the type of the operand will always be a pointer and, if *type* is specified, *type* must be **PTR**. The **D** (Display) Command may use the **&** type modifier for any data item type except index-names (data type **IXN**). The **&** type modifier may not be used with a constant-name since a constant-name does not name a data item. The **&** type modifier may be used in the **M** (Modify), **T** (Trap), and **U** (Untrap) Commands only if the referenced data item is a based linkage record because, otherwise, the address is not modifiable.

*alias* is a name you enter to serve as another name for the data operand specification that precedes it. If present, *alias* must follow a pound sign (#). The characters that follow the # must form a valid COBOL word. Only the first three characters of this word are significant. Once specified, *alias* can be used in later Debug commands that use the alias format. For a complete discussion of this format, see Alias Format (on page 254). The three-character alias must be unique. If you assign the same alias to a different data operand, it will no longer refer to the earlier operand. When a command defines an alias and specifies the ***** (indirect through pointer) type modifier, the indirection is resolved at the time the alias is defined. In this case, the alias continues to refer to the data item that that the pointer data item referenced when the alias was defined even if the pointer data item value has subsequently been changed. When a command defines an alias and specifies the **&** (address of data item) type modifier, the address of the data item is resolved each time the alias is specified using an alias format command.

## Address-Size Format

This method requires that you specify the A Compile Command Option (see page 144) to produce an allocation map in the listing file. This map provides the information that must be entered in a Debug command employing this method.

The syntax is as follows:

```
[ base : ]  address  [ + occur-size * occur-num ] …  ,  size ,
                [ type ]  [ # alias ]
```

*base* specifies the base item for formal arguments, based linkage items, and external items as follows:

- For a USING formal argument, *base* is specified as **U** *arg-num*, where *arg-num* specifies the ordinal position of the argument in the USING list of the Procedure Division header provided in the allocation map of the program listing.
- For a GIVING formal argument, *base* is specified as **G**, as shown in the allocation map of the program listing.
- For a based linkage item, *base* is specified as **B** *item-num*, where *item-num* specifies the compiler assigned based linkage item number provided in the allocation map of the program listing.
- For an external item, base is specified as **X** *ext-num*, where *ext-num* specifies the compiler assigned external number provided in the allocation map of the program listing.

**Note** A Linkage Section data item, which is neither a formal argument item nor a based linkage item, is shown as "Not addressable:" in the allocation map of the program listing. Since such items have not been used in the source program, the compiler does not allocate a base pointer item for them and the Interactive Debugger cannot access them.

*address* specifies the decimal data address of the data item to be displayed, modified or monitored. The value is obtained from the data allocation map in the compiler listing.

*occur-size* specifies the size of data items that contain OCCURS clauses in their definitions. If *occur-size* is present, the plus sign (+) must appear as well. When a referenced data item contains an OCCURS clause in its definition, *occur-size* is equal to the value of *size* (defined in this section).

*occur-num* is the occurrence number for data items with OCCURS clauses in their definitions.

*size* specifies the size of the data item to be displayed, modified or monitored. (For **IXN** items, *size* specifies the span of the data item indexed by the index-name.)

*type* specifies the data type of the data item to be displayed, as it appears in the Debug column (see page 158) of the data allocation map (on page 155). (Complete details on the types of data allowed with RM/COBOL are found in Chapter 4: *Data Division* of the *RM/COBOL Language Reference Manual*, and a list of the data types appears in Table 28 on page 248.) If this parameter is omitted, *type* defaults to hexadecimal. The type modifiers ***** (indirect through

pointer) and **&** (address of data item) may not be specified in the address-size format.

*alias* is the name you assign to the data item referenced by the preceding entry. For the full description, see Alias Format (on page 254).

Note that the term:

```
+occur-size*occur-num
```

selects specific occurrences of data items in tables.

Look at the program listing and associated data allocation map in Figure 37. Then look at Figure 38, which shows a developed reference to GRP-2(3).

The effect of this reference is the creation of a developed data address of 946, which appears as the address on the first line of the data display. Subsequent references to GRP-2(3) may appear as 946, 471 instead of 4 + 471*3, 471.

**Note** This developed data address is not the algebraic equivalent of the formula 4 + 471*3. It is the algebraic equivalent of 4 + (471*(3-1)) = 4 + (471*2) = 946.

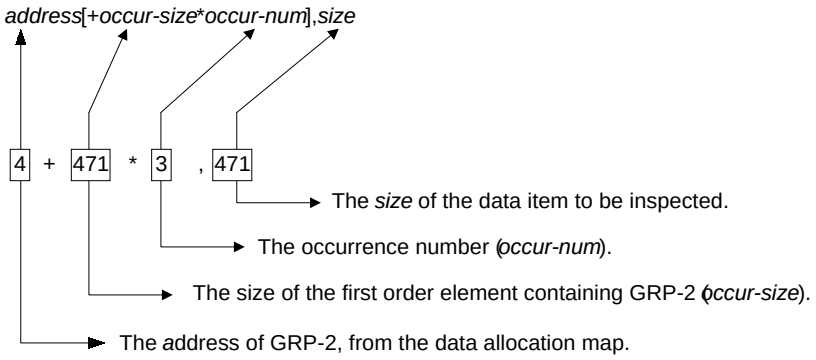
**Figure 37: Data Allocation Map**

```
    8
    9  DATA DIVISION.
   10  WORKING-STORAGE SECTION.
   11  01  GRP-1.
   13      02  GRP-2                      OCCURS 20.
   14          03  FILLER                 PIC X(6).
   15          03  GRP-3                  OCCURS 15.
   16              04  FILLER             PIC X(5).
   17              04  THE-ITEM           PIC 9V9 OCCURS 12.
   18              04  FILLER             PIC X(2).

Working-Storage Section for program ALLOCMAP

Address  Size      Debug Order  Type              Name
      4    9420      GRP     0    Group             GRP-1
      4     471      GRP     1    Group              GRP-2
     10      31      GRP     2    Group                GRP-3
     15       2      NSU     3    Numeric unsigned      THE-ITEM
```

**Figure 38: Developed Data Address**



### Alias Format

The alias format allows you to reference a data item or index that had been assigned an alternate name. Once assigned, the alias applies to that data item until you use it to name another data item or you end the Debug session.

The general format for referring to an item with an alias is as follows:

```
# alias
```

**#** is required.

*alias* must be a valid COBOL word, only the first three characters of which are used. The alias must be previously defined in an identifier or address-size format specification.

## Regaining Control

Debug regains control under the following conditions:

- Breakpoint
- Runtime system error
- Step through a statement, paragraph or section
- Execution of a STOP RUN statement
- Trap

## Debug Command Prompt

The Debug command prompt first appears at the lower-left corner (line 25) of the screen. It looks like this:

```
condition line [ + intraline ] name  C?_
```

*condition* is the name of a condition that has stopped program execution. It may be one or more of the following:

- **BP**, which indicates a breakpoint is present.
- **DT**, which indicates a data trap has occurred.
- **ER**, which indicates a runtime system error has occurred.
- **SR**, which indicates the program has executed a STOP RUN statement.
- **ST**, which indicates program stepping is active.

*line* is the line on which the next statement to be executed begins. The string "Line?" appears if the line number is not known.

*intraline* indicates the next statement to be executed when line contains more than one statement.

*name* is the name of the currently executing program.

C?_ is the prompt and cursor.

Any Debug command may be entered when the prompt appears on the screen. If the last Debug command was an **S** (Step) Command, you can repeat it by pressing Enter in response to the prompt.

# Debug Error Messages

This section lists and defines the error messages that may be generated during debugging.

### Command Error

Indicates that you entered an invalid command character, or a command with inconsistent or invalid parameter values. The Debug command prompt appears again.

This message is displayed if any errors occur in scanning a reference in address-size format. Further, this message is displayed after any of the following specific errors are diagnosed.

### Address too big

Indicates that the address value specified for a data item in a Debug address-size format is not correct.

This means that the end of the data item, calculated by summing the address of the data item, including any subscript calculations, and the length of the data item less one, exceeds the size of the region containing the data item. If the item resides in the program, the end of the data item must not exceed the sum of the lengths of the File Section and the Working-Storage Section. If the item is a Linkage Section item that is, or is subordinate to, a formal argument, the end of the data item must not exceed the length of the corresponding actual argument item. If the item is a Linkage Section item that is, or is subordinate to, a based linkage item, the end of the data item must not exceed the length of the area of memory addressed by the pointer value used to set the base address of the based linkage item. If the item is external, the end of the data item must not exceed the length of the highest level containing external item.

### Colon (:) expected

Indicates that the colon is misplaced or omitted in the reference modification specified with a data item address in a Debug identifier format.

A reference modification specification consists of a left parenthesis, a non-zero integer starting position, a colon, optionally a non-zero integer length, and a right parenthesis. This message is displayed if the left parenthesis is not followed by a

colon, there is no right parenthesis, or the colon follows the right parenthesis. The reference modification specification follows a subscript specification, if present.

### Dword alignment

Indicates that the address specified for an index-name (Debug type **IXN**) is not a multiple of four.

### Extraneous characters

Indicates that the command entered to Debug contains characters that are not expected past the data item specification or modification value.

The **D** (Display) and **T** Trap Commands accept an alias identifier of a data item or a data item specification with an optional alias definition. The **M** (Modify) Command accepts the same parameters as a **D** (Display) Command followed by a modification value. The **U** (Untrap) Command accepts the data item specification or alias identifier of a data item with which a trap is currently associated. No additional characters are accepted.

### Identifier expected

Indicates that the first character of what should be an identifier is not alphabetic, numeric, or hyphen.

An identifier is expected at the start of a Debug reference in an identifier format, following **IN** or **OF** in identifier format, and following the pound sign (**#**), which identifies an alias.

### Identifier not a data item

Indicates that the symbolic identifier specified is a user-defined word in the COBOL source, but is not a data item.

The Debug commands, D (Display), M (Modify), and T (Trap), allow access to program data items. The state of user-defined words, such as condition-names or switch-names, which are not data items, cannot be examined. Examine the source or the allocation map to determine the data item that contains the field to be accessed.

This error message also occurs if a symbolic identifier specifies a Linkage Section data item that is neither a formal argument (USING or GIVING) nor a based linkage item. Since such items have not been used in the source program, the compiler does not allocate a base pointer item for them and the Interactive Debugger cannot access them.

### Identifier refers to constant

Indicates that in a command other than the **D** (Display) Command, the symbolic identifier specified refers to a constant-name or is the constant address (**&** type modifier specified) for a data item that is not a based linkage data item. Constant-names and constant addresses may not be specified in the **M** (Modify), **T** (Trap), or **U** (Untrap) Commands.

### IN or OF expected

Indicates that the symbolic identifier in a Debug identifier format specification of a data item is followed by something other than a qualification specification.

If a symbolic name is not followed by the special characters left parenthesis, comma, or pound sign, it must be followed by **IN** or **OF** and the symbolic name of a higher level data item.

### Incorrect number of subscripts

Indicates that too few subscripts are provided with a data item in a Debug identifier format.

### Index Error

Indicates that you requested the display of an index-name (Debug type **IXN**) and Debug discovered an inconsistency in the internal representation of the index-name.

This message generally indicates that the address or size value of the index-name was entered incorrectly.

### Index-name inconsistency

Indicates that the symbolic identifier conflicts with the type value specified in the command. One of the following conflicts has occurred:

- The symbolic identifier refers to an index-name, but a type value other than **IXN** was specified. Index-names may not use a type value other than **IXN**.
- The symbolic identifier does not refer to an index-name, but the type value **IXN** was specified. Only index-names may be specified with the type value **IXN**.
- The symbolic identifier refers to an index-name and a *refmod* field was specified. Reference modification of an index-name is not allowed.
- The type modifier **&** (address of data item) was specified with a symbolic identifier that refers to an index-name (data type **IXN**). The type modifier **&** may not be specified for an index-name.

### Left parenthesis expected

Indicates that the data item in a Debug identifier format refers to a table element and no subscript identifying a particular element is entered.

### Length too big

Indicates that the sum of the starting position value and the length, specified in the reference modification of a data item in a Debug identifier format, exceeds the declared length of the data item.

A reference modification specification consists of a left parenthesis, a non-zero integer starting position, a colon, optionally a non-zero integer length, and a right parenthesis. The ending position, that is, the sum of the starting position minus 1 and the length, must not exceed the length of the symbolic data item. If the symbolic data item is a variable-length group, the ending position must not exceed the maximum length of the group.

### Name undefined

Indicates that a symbolic identifier entered in a Debug identifier format specification of a data item does not exist in the COBOL program.

The name appears undefined when debugging a contained program if the name is defined in a containing program but is not described with the GLOBAL attribute. When debugging a containing program, names in contained programs appear undefined. This message also appears if a name provided as a qualifier is not defined anywhere in the separately compiled program.

### No such based linkage item

Indicates that you specified a based linkage item number for a data item in the Debug address-size format incorrectly. This means that the integer between the letter **B** and the following colon does not correspond to a based linkage item.

### No such external

Indicates that you specified an external number for a data item in the Debug address-size format incorrectly. This means that the integer between the letter **X** and the following colon does not correspond to an external group item.

### No such parameter

Indicates that a USING argument value for a data item in the Debug address-size format is specified incorrectly.

This means that the integer between the letter **U** and the following colon does not correspond to a Linkage Section data item included in the USING list of the Procedure Division or that the calling program did not pass a value in the corresponding position of its CALL statement.

### Non-zero integer expected

Indicates that a subscript, reference modification starting position, or reference modification length in a Debug identifier format specification of a data item is zero or is not numeric.

### Not enough free memory to continue

Indicates that Debug cannot allocate memory.

Debug allocates memory to remember names associated with an alias and to contain the symbol table of the current program.

### Offset too big

Indicates that the starting position value entered in a reference modification, specified with a data item address in a Debug identifier format, exceeds the declared length of the data item.

### Pointer inconsistency

Indicates that the symbolic identifier conflicts with the type value specified in the command. One of the following conflicts has occurred:

- The symbolic identifier refers to a pointer data item, but a type value other than **PTR** was specified without a type modifier. Pointer data items may not use a type value other than **PTR**.
- The symbolic identifier does not refer to a pointer data item, but the type value **PTR** was specified. Only pointer data items may be specified with the type value **PTR**.
- The symbolic identifier refers to a pointer data item without the type modifier ***** (indirect through pointer) and a *refmod* field was specified. Reference modification of a pointer data item is not allowed.
- The type modifier ***** (indirect through pointer) was specified with a symbolic identifier that is not a pointer data item. The type modifier ***** may only be used with pointer data items.
- The type modifier **&** (address of data item) was specified with a type value other than **PTR**. The type modifier **&** always results in a pointer value and thus cannot specify a type value other than **PTR**.
- The type modifier **&** (address of data item ) was specified with a symbolic identifier that refers to a constant-name. The type modifier **&** may not be specified with a constant-name.
- For the **M** (Modify) Command, the = value modifier was followed by a symbolic identifier that does not refer to a pointer data item or reference modification was specified for the pointer.

### Pointer memory access violation

Indicates that the ***** (indirect through pointer) type modifier was specified with a pointer data item that has a null value or a value that points to memory that cannot be read by the program. For the **M** (Modify) Command, this may mean that the program cannot write to the memory area referenced by the pointer value.

### Program not compiled with Y option

Indicates that a data item is entered in a Debug identifier format to a **D** (Display), **M** (Modify), or **T** (Trap) Command, but no symbol table is available.

### Qualifier undefined

Indicates that an alias identifier is undefined or that the symbolic qualifier provided after an **IN** or **OF** in a Debug item address in an identifier format is not a higher level data item of the preceding identifier.

Check the qualification specification with the program source or the allocation map of the compilation listing.

### Qualification ambiguous

Indicates that the symbolic qualifiers provided after an **IN** or **OF** in a Debug item address in an identifier format do not uniquely identify a data item.

Qualification allows unique specification of data items with the same name. The qualification identifiers must be the names of lower numbered group items or the name of the COBOL filename if the data item is in the File Section. Check the qualification specification with the program source or the allocation map of the compilation listing. Reenter the command with additional qualification items to uniquely identify the desired data item.

### Right parenthesis expected

Indicates that a subscript or reference modification field, specified with a data item address in a Debug identifier format, is missing its closing right parenthesis or that too many subscripts are provided.

### Subscript value too big

Indicates that a subscript exceeds 65535 or that the offset corresponding to the specified subscripts exceeds $2^{32}$.

# A (Address Stop) Command

Use the **A** Command to set a breakpoint and to resume program execution from the current location. When the specified breakpoint is reached and Debug regains control, the breakpoint is cleared and has no further effect.

The syntax for the **A** Command is as follows:

```
A  [ line [ + intraline ]  [ ,  [ prog-name ]  [ ,  [ count ] ] ] ]
```

*line* indicates the line number containing the statement at which the breakpoint is to be set. *line* always refers to the first statement of the line. If this parameter is omitted, no breakpoint is set before execution is resumed.

*intraline* refers to a specific statement within the program line. For example, 1 indicates the first intraline statement (or the second actual statement), 2 indicates the second intraline statement, and so forth. If this parameter is omitted, the first statement on the line is assumed.

*prog-name* provides for explicit program qualification during debugging. The value of *prog-name* must be a program-name from the PROGRAM-ID paragraph of the Identification Division. If this parameter is omitted, it is assumed the reference is to the currently executing program. If this parameter is omitted and *count* is specified, there must be two commas before *count*.

*count* is an execution count. Debug allows the statement to execute a number of times equal to *count* minus one, then honors and clears the breakpoint immediately before the next execution (which now equals *count*) of the statement. Debug then regains control. The maximum value for *count* is 65535. If this parameter is omitted, *count* defaults to 1.

When you use the **A** Command, keep in mind that the specified breakpoint remains in effect until it is honored. In other words, if execution halts and Debug regains control at a statement other than that specified in the **A** Command, the breakpoint set by the **A** Command remains in effect. It can be cleared by the **C** Command and displayed by the **B** Command.

# B (Breakpoint) Command

Use the **B** Command to display all currently set breakpoints or to set breakpoints at specific procedural statements. Note that—unlike the **A** Command—a breakpoint set by this command is not cleared once the conditions have been satisfied. To clear breakpoints set with a **B** Command, you must enter a **C** Command.

The command syntax is nearly identical to that used with the **A** Command:

```
B  [ line [ + intraline ]  [ ,  [ prog-name ]  [ ,  [ count ] ] ] ]
```

*line* indicates the line number containing the statement at which the breakpoint is to be set. *line* always refers to the first statement of the line. If this parameter is omitted, no breakpoint is set. Instead, all currently set breakpoints are displayed.

*intraline* refers to a specific statement within the program line. For example, 1 indicates the first intraline statement (or the second actual statement), 2 indicates the second intraline statement, and so forth. If this parameter is omitted, the first statement on the line is assumed.

*prog-name* provides for explicit program qualification during debugging. The value of *prog-name* must be a program-name from the PROGRAM-ID paragraph of the Identification Division. If this parameter is omitted, it is assumed the reference is to the currently executing program. If this parameter is omitted and *count* is specified, there must be two commas before *count*.

*count* is an execution count. Debug allows the statement to execute a number of times equal to *count* minus one, then honors and clears the breakpoint immediately before the next execution (which now equals *count*) of the statement. Debug then regains control. The maximum value for *count* is 65535. If this parameter is omitted, *count* defaults to 1. Because the **B** Command does not clear breakpoints after responding to them (as does the **A** Command), it remains in effect for all subsequent occurrences.

For example, entering:

```
B 150+2, PAY-TAX, 5
```

sets a breakpoint at the third statement (second intraline statement) at line 150 in the program PAY-TAX. When execution resumes, the breakpoint is ignored four times and honored the fifth time. Program control then returns to Debug.

The format of the breakpoint display (when *line* is omitted) is as follows for each active breakpoint:

```
line [ + intraline ] prog-name count
```

A count of zero is equivalent to a count of one.

# C (Clear) Command

Use the **C** Command to clear any breakpoints that have been set with the **A** or **B** Commands.

The syntax for the **C** Command is as follows:

```
C  [ line [ + intraline ]  [ ,  [ prog-name ] ] ]
```

*line* is the line number containing the statement at which the breakpoint to be cleared is set. If no line number is specified, all currently active breakpoints are removed.

*intraline* refers to a specific statement within the program line. For example, 1 indicates the first intraline statement (or the second actual statement), 2 indicates the second intraline statement, and so forth. If this parameter is omitted, the first statement on the line is assumed.

*prog-name* provides for explicit program qualification during debugging. The value of *prog-name* must be a program-name from the PROGRAM-ID paragraph of the Identification Division. If this parameter is omitted, it is assumed the reference is to the currently executing program. If a line is specified on which no breakpoint exists, the command is in error.

For example, entering:

```
C 100+2
```

clears the breakpoint set at the second intraline statement of line 100 in the currently executing program.

# D (Display) Command

Use the **D** Command to display on the screen the value of a specified data item.

## Identifier Format

The syntax for the **D** Command with the identifier format (on page 250) is as follows:

```
D  name-1  [ { IN | OF }  name-2 ] …  [ script ]  [ refmod ]
             [ ,  { type | { * | & } [ type ] } ]  [ # alias ]
```

*name-1* is a name declared as a constant-name, data-name, or index-name for the literal value, data item, or index to be displayed. *name-2* is a qualifier for *name-1*. Qualification is required if the name is not unique. The named data item or index must be described in the Data Division of the current program or be described with the GLOBAL attribute in a program that contains the current program.

*script* specifies subscripting and is required if the data item referenced by *name-1* is a table element. (See page 250 for a complete description.) If the data item is not a table element, do not specify *script*. The format for *script* is as follows:

```
( integer-1 [ [ ,  ] integer-2 ] …  )
```

*refmod* specifies a subfield of the data item. (See page 250 for a complete description.) It has this format:

```
( offset : [ length ] )
```

*type* specifies the data type to be used for displaying the named data item or index. If this parameter is omitted, *type* defaults to the type of the item specified except when a type modifier is specified. The type value **IXN** may only be used with index-names and, when the named data item is an index-name, the only permissible type value is **IXN**. The type value **PTR** may only be used with pointer data items and, when the named data item is a pointer data item, the only permissible type value is **PTR**, except when a type modifier is specified.

The type modifiers ***** and **&** have the following effect:

- The ***** type modifier indicates that the data item to be displayed is the data item determined by an indirect reference, that is, a reference to the data item referenced by a pointer data item value. The data item specified in the command must be a pointer data item (data type **PTR**). In this case, *type*, if specified, indicates the type of the item referenced by the pointer data item value. If *type* is not specified, the default is hexadecimal. If *refmod* is not specified for an indirect reference, the entire effective memory area specified by the pointer data item value is displayed. The effective memory

area specified by a pointer data item value begins with the effective address (*pointer.address + pointer.offset*) and ends just before the effective limit address (*pointer.address + pointer.length*). If *refmod* is specified for an indirect reference, *refmod* is applied to the indirect reference rather than the pointer data item itself. Thus, *refmod* may be used to display a subfield within the current effective memory area of an indirect reference.

- The **&** type modifier indicates that the address of the data item specified is to be displayed as a pointer value (data type **PTR**). Since the result type will always be a pointer, if *type* is specified, *type* must be **PTR**. The **&** type modifier may be used with a data item of any data type except an index-name (data type **IXN**). The **&** type modifier may not be used with a constant-name since a constant-name does not name a data item. The address of the data item will be displayed as three sets of sixteen hexadecimal digits; the first set is the base address, the second set is the offset (SET pointer UP/DOWN), and the third set is the length of the memory area covered by the pointer.

*alias* is a name you enter to serve as another name for the data operand specification that precedes it. For a complete description, see Alias Format. If present, *alias* must follow a pound sign (**#**). The characters that follow the **#** must form a valid COBOL word. Only the first three characters of this word are significant. When a command defines an alias and specifies the ***** (indirect through pointer) type modifier, the indirection is resolved at the time the alias is defined. In this case, the alias continues to refer to the data item that the pointer data item referenced when the alias was defined even if the pointer data item value has subsequently been changed. When a command defines an alias and specifies the **&** (address of data item) type modifier, the address of the data item is resolved each time the alias is specified using an alias format command.

For example, entering:

```
D MONTH-NAME(11)(1:3)
```

directs Debug to display the first three bytes of the 11th element in the table MONTH-NAME. Debug then displays the following:

```
140 ANS NOV
```

This shows the data address of 140, the type of data as alphanumeric, and the value as NOV.

## Address-Size Format

The syntax for the **D** Command with the address-size format (on page 252) is as follows:

```
D  [ base : ]  address  [ + occur-size * occur-num ] …  ,  size ,
                  [ type ]  [ # alias ]
```

*base* specifies the base item for formal arguments, based linkage items, and external items as follows:

- For a USING formal argument, *base* is specified as **U** *arg-num*, where *arg-num* specifies the ordinal position of the argument in the USING list of the Procedure Division header provided in the allocation map of the program listing.
- For a GIVING formal argument, *base* is specified as **G**, as shown in the allocation map of the program listing.
- For a based linkage item, *base* is specified as **B** *item-num*, where *item-num* specifies the compiler assigned based linkage item number provided in the allocation map of the program listing.
- For an external item, base is specified as **X** *ext-num*, where *ext-num* specifies the compiler assigned external number provided in the allocation map of the program listing.

**Note** A Linkage Section data item, which is neither a formal argument item nor a based linkage item, is shown as "Not addressable:" in the allocation map of the program listing. Since such items have not been used in the source program, the compiler does not allocate a base pointer item for them and the Interactive Debugger cannot access them.

*address* specifies the address of the data item to be displayed. This is based on the value obtained from the data allocation map. For more information on addresses used with Debug, see address-size format (on page 252).

*occur-size* specifies the size of data items that contain OCCURS clauses in their definitions.

*occur-num* specifies the occurrence number for data items that contain OCCURS clauses in their definitions.

*size* specifies the size of the data item to be displayed. If type is **IXN**, this is the value that appears in the Span column of the data allocation map.

*type* specifies the data type of the data item to be displayed. If this parameter is omitted, *type* defaults to hexadecimal. The type modifiers ***** (indirect through pointer) and **&** (address of data item) may not be specified in the address-size format.

*alias* must form a valid COBOL word, only the first three characters of which are valid. For a complete description, see Alias Format (on page 254). Once specified, the alias can be used to refer to the operand to which it is assigned.

The specified data item appears in the following format:

```
address type value …
```

*address* is the developed data address.

*type* is the type of data item to be displayed.

*value* is the value of the data in the format specified by type in the **D** Command.

For example, entering:

```
D 13 + 2*7,2,NSU
```

directs Debug to display the data item located at decimal data address 25.

Debug then displays:

```
25 NSU 15
```

This shows that at the developed data address of (decimal) 25 is a value of 15, of type numeric string unsigned.

If the L (Line Display) Command (on page 267) is used to specify a line number of the monitor, the display appears one line at a time, at the line specified in the **L** Command. If the display does not fit on one line, press the Enter key to see the next line. To return to the Debug command prompt after the last line of data appears, press Enter.

If the **L** Command was not used, the display begins at the next line and scrolls when the bottom of the screen is reached.

## Alias Format

The syntax for the **D** Command with the alias format for specifying a data item or index reference is as follows:

```
D  # alias
```

**#** is required.

*alias* must form a valid COBOL word, only the first three characters of which are valid. For a complete description, see Alias Format (on page 254). The alias must have been previously defined in an identifier or address-size format specification.

# E (End) Command

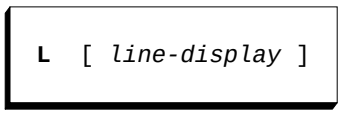
Use the **E** Command to leave Debug. The currently executing program runs until completion.

The syntax of the **E** Command is as follows:


# L (Line Display) Command

Use the **L** Command to specify a line on the monitor screen at which command input echoes and Debug responses are to be displayed.

The syntax of the **L** Command is as follows:




*line-display* designates a line number on the monitor and may be in the range 0 through the number of lines on the screen.

This command is useful when you are debugging programs that have a variety of interactive ACCEPT and DISPLAY statements. By selecting a specific line for Debug displays, you can reduce or avoid conflicts between lines produced by Debug and lines produced by the program.

If *line-display* is omitted from the command, or *line-display* equals 0, the screen resumes its standard mode of operation (scrolling).

# M (Modify) Command

Use the **M** Command to change the value of a specified data item.

## Identifier Format

The syntax for the **M** Command with the identifier format (on page 250) is as follows:

```
M  name-1  [ { IN | OF }  name-2 ] …  [ script ]  [ refmod ]
           [ ,  { type | { * | & } [ type ] } ]  [ # alias ] , value
```

*name-1* is a name declared as a data-name or index-name for the data item or index to be modified. *name-2* is a qualifier for *name-1*. Qualification is required if the name is not unique. The named data item or index must be described in the Data Division of the current program or be described with the GLOBAL attribute in a program that contains the current program.

*script* specifies subscripting and is required if the data item referenced by *name-1* is a table element. (See page 250 for a complete description.) If the data item is not a table element, do not specify *script*. The format for *script* is as follows:

```
( integer-1 [ [ ,  ] integer-2 ] …  )
```

*refmod* specifies a subfield of the data item. (See page 250 for a complete description.) It has this format:

```
( offset : [ length ] )
```

*type* specifies the data type of the item to be modified. If this parameter is omitted, *type* defaults to the type of the item specified except when a type modifier is specified. The type value **IXN** may only be used with index-names and, when the named data item is an index-name, the only permissible type value is **IXN**. The type value **PTR** may only be used with pointer data items and, when the named data item is a pointer data item, the only permissible type value is **PTR**, except when a type modifier is specified.

The type modifiers ***** and **&** have the following effect:

- The ***** type modifier indicates that the data item to modify is determined by an indirect reference, that is, a reference to the data item referenced by a pointer data item value. The data item specified in the command must be a pointer data item (data type **PTR**). In this case, *type*, if specified, indicates the type of the data item referenced by the pointer data item value. If *type* is not specified, the default is hexadecimal. If *refmod* is not specified for an indirect reference, the entire effective memory area specified by the pointer data item value is modified. The effective memory area specified by a pointer data item value begins with the effective address (*pointer.address* +

*pointer.offset*) and ends just before the effective limit address (*pointer.address* + *pointer.length*). If *refmod* is specified for an indirect reference, *refmod* is applied to the indirect reference rather than the pointer data item itself. Thus, *refmod* may be used to modify a subfield within the current effective memory area of an indirect reference.

- The **&** type modifier indicates that the address of the data item specified is to be modified. The data item referenced in the command must be a based linkage record because, otherwise, the address is not modifiable. The data item referenced in the command may be any data type except an index-name (data type **IXN**). When the **&** type modifier is used, the *value* must be a pointer value and, if *type* is specified, *type* must be **PTR**.

*alias* is a name you enter to serve as another name for the data operand specification that precedes it. For a complete description, see Alias Format (on page 254). If present, *alias* must follow a pound sign (**#**). The characters that follow the **#** must form a valid COBOL word. Only the first three characters of this word are significant. When a command defines an alias and specifies the ***** (indirect through pointer) type modifier, the indirection is resolved at the time the alias is defined. In this case, the alias continues to refer to the data item that the pointer data item referenced when the alias was defined even if the pointer data item value has subsequently been changed. When a command defines an alias and specifies the **&** (address of data item) type modifier, the address of the data item is resolved each time the alias is specified using an alias format command.

*value* specifies the new value for the data item or index. The format for specifying the value is described by *type* in the following section, Address-Size Format.

For example, entering:

```
M MONTH-NAME(12), DECEMBER
```

directs Debug to modify the 12th element of the table MONTH-NAME to have the value of DECEMBER.

## Address-Size Format

The syntax for the **M** Command with the address-size format (on page 252) is as follows:

```
M  [ base : ]  address  [ + occur-size * occur-num ] …  ,  size ,
               [ type ]  [ # alias ] ,  value
```

*base* specifies the base item for formal arguments, based linkage items, and external items as follows:

- For a USING formal argument, *base* is specified as **U** *arg-num*, where *arg-num* specifies the ordinal position of the argument in the USING list of the Procedure Division header provided in the allocation map of the program listing.
- For a GIVING formal argument, *base* is specified as **G**, as shown in the allocation map of the program listing.

- For a based linkage item, *base* is specified as **B** *item-num*, where *item-num* specifies the compiler assigned based linkage item number provided in the allocation map of the program listing.
- For an external item, base is specified as **X** *ext-num*, where *ext-num* specifies the compiler assigned external number provided in the allocation map of the program listing.

**Note** A Linkage Section data item, which is neither a formal argument item nor a based linkage item, is shown as "Not addressable:" in the allocation map of the program listing. Since such items have not been used in the source program, the compiler does not allocate a base pointer item for them and the Interactive Debugger cannot access them.

*address* specifies the address of the data item to be modified. This is based on the value obtained from the data allocation map. For more information on addresses used with Debug, see address-size format (on page 252).

*occur-size* specifies the size of data items that contain OCCURS clauses in their definitions.

*occur-num* specifies the occurrence number for data items that contain OCCURS clauses in their definitions.

*size* specifies the size of the data item to be modified. If type is **IXN**, this is the value that appears in the Span column of the data allocation map.

*type* specifies the type of data item referenced. If this parameter is omitted, *type* defaults to hexadecimal. The type modifiers ***** (indirect through pointer) and **&** (address of data item) may not be specified in the address-size format.

*alias* is a name you enter to serve as another name for the data operand specification that precedes it. For a complete description, see Alias Format (on page 254). If present, *alias* must follow a pound sign (#). The characters that follow the # must form a valid COBOL word. Only the first three characters of this word are significant.

*value* is the value of the data in the format specified by type in the **M** Command line.

If *type* is **HEX**—or omitted in the address-size format—*value* is entered as a string of hexadecimal digits. This hex value is stored in the specified data item and is left justified with zero fill or truncation on the right. The hexadecimal value must contain an even number of digits.

If *type* is one of the nonnumeric types **ANS**, **ANSE**, **ABS**, **ABSE**, **GRP**, or **NSE**, *value* is stored in the specified data item and is left justified with blank fill or truncation on the right. Note that no editing is performed during this operation.

If *type* is one of the numeric types **NBS**, **NBU**, **NBSN**, **NBUN**, **NCS**, **NCU**, **NLC**, **NLS**, **NPP**, **NPS**, **NPU**, **NSU**, **NTC**, or **NTS**, *value* is converted to a signed integer according to the rules for a MOVE from a numeric edited sending item to a numeric destination item (see the "MOVE Statement" section in Chapter 6: *Procedure Division Statements* of the *RM/COBOL Language Reference Manual* for more information).

If *type* is **IXN**, *value* is converted to a signed integer occurrence number. This number is then converted to the internal index-name representation based on the value of *size*.

If *type* is **PTR**, *value* must be a pointer value. The pointer value 0 is equivalent to NULL (NULLS). For pointer values other than 0, a pointer value is forty-eight hex digits, where the first sixteen digits specify the base address, the middle sixteen digits specify the offset from the base address, and the last sixteen digits specify the length of the memory area. Embedded spaces are allowed and ignored. Leading zeroes must be specified. If the program was compiled with the Y Compile Command Option, then a pointer *value* may also be specified with either of the value modifiers = or **=&** as follows:

```
{ = | =& } name-3  [ { IN | OF } name-4 ] …  [ script ]  [ refmod ]
```

If the = value modifier is specified (without the **&**), *name-3* must refer to a pointer data item and *refmod* is not allowed. The current value of the referenced pointer data item is used for *value*. This is equivalent to the COBOL statement:

```
SET name-1 [OF name-2] …  TO name-3 [OF name-4] …
```

If the **=&** value modifier is specified, *value* is composed from the address of the data item named by *name-3*, an offset of zero, and the length of the data item named by *name-3*. This is equivalent to the COBOL statement:

```
SET name-1 [OF name-2] …  TO
          ADDRESS OF name-3 [OF name-4] …
```

**Note** The Interactive Debugger attempts to validate a pointer *value* when specified, but the validation results may not be conclusive. It is the user's responsibility to take care when modifying pointer data items or based linkage base addresses to ensure correctness. One easy method of correctly modifying a pointer value is to display the desired pointer value using the **D** Command and then using copy/paste to paste the value into the *value* field of an **M** Command. Another method is to use one of the = or **=&** value modifiers described above.

In all other cases, the resulting integer is stored in the data item as if the item had no assumed decimal point. If conversion results in a noninteger, an error message is displayed and the specified data item remains unaltered.

For example, entering:

```
M 13+2*7,2,NSU,0
```

directs Debug to modify the data item located at decimal data address 25. The **NSU** data item will have a new value of 0.

## Alias Format

The syntax for the **M** Command with the alias format for specifying a data item or index reference is as follows:

```
M  # alias ,  value
```

**#** is required.

*alias* must form a valid COBOL word, only the first three characters of which are valid. For a complete description, see Alias Format (on page 254). The alias

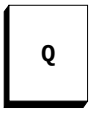
must have been previously defined in an identifier or address-size format reference to the desired item (for example, in a **D** (Display) Command).

*value* specifies the new value for the data item or index. The format for specifying the value is described by *type* in the Address-Size Format section on page 269.

# Q (Quit) Command

Use the **Q** Command to stop program execution. This command terminates the program as if a STOP RUN statement were executed.

The syntax of the **Q** Command is as follows:

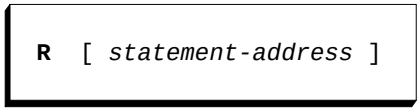



When the **Q** Command is executed, open files are closed and control returns to the operating system.

# R (Resume) Command

Use the **R** Command to specify that program execution resume at the current location, or at another location specified in the command.

The syntax of the **R** Command is as follows:




*statement-address* specifies the specific program address for the sentence at which execution is to resume. If *statement-address* is not specified, execution resumes at the current location. *statement-address* appears in the Debug column at the left of the program listing, and should be entered as printed. *statement-address* is not a line number.

An error condition or stop run condition (that is, the command prompt contains ER or SR) forces the **R** Command to disallow *statement-address*. The **R** Command may be used in its simple form (that is, without an accompanying *statement-address*) to allow Debug to trace back through the program units of a run unit, but the run unit may not be restarted when an error or stop run condition occurs.

**Note** The **R** Command used with a *statement-address* resets the program counter. If an improper *statement-address* is specified, Debug displays an error. The program counter remains invalid until another **R** Command with a valid *statement-address* is used. An **R** Command with no *statement-address* at this time causes Debug to display an error.

# S (Step) Command

Use the **S** Command to specify that program execution occur one step at a time. With Debug, you can step through a statement, a paragraph, or an entire section.

The syntax of the **S** Command is as follows:

```
S  [ P | S ]  [ count ]
```

P specifies that paragraphs are to be stepped through.

S specifies that sections are to be stepped through. If neither P nor S is present, statements are stepped through.

*count* specifies the number of statements, paragraphs or sections that are to be executed before execution suspends. The maximum value for *count* is 65535. A value of zero is treated as 1. The default is 1.

Specifying *count* greater than 1 causes Debug to trace the statements, paragraphs or sections executed while in stepping mode. The format of the trace message is as follows:

```
TR line [+intraline] name
```

*line* is the line number on which the statement begins.

*intraline* is the specific statement within a line.

*name* is the name of the program as it appears in the PROGRAM-ID paragraph of the Identification Division.

For example, entering:

```
SS 10
```

directs Debug to execute 9 sections, produce 9 trace messages, and then halt before executing the 10th.

# T (Trap) Command

Use the **T** Command to monitor the value of a specified data item, and to suspend execution whenever a change in that value occurs. The **T** Command can also be used to display all currently set traps. If you enter only the command keyword **T**, all currently active data traps appear.

A data trap set with the **T** Command can be removed with the **U** (Untrap) Command. A data trap set on a Linkage Section data item is removed automatically when the program exits. A data trap set on a File Section, Working-Storage Section or Screen Section data item is removed automatically when the separately compiled program is canceled. A data trap set on an external data item will continue until the run unit ends.

## Identifier Format

The syntax for the **T** Command with the identifier format (on page 250) is as follows:

```
T  name-1  [ { IN | OF }  name-2 ] …  [ script ]  [ refmod ]
             [ ,  { type | { * | & } [ type ] } ]  [ # alias ]
```

*name-1* is a name declared as a data-name or index-name for the data item or index to be monitored. *name-2* is a qualifier for *name-1*. Qualification is required if the name is not unique. The named data item or index must be described in the Data Division of the current program or be described with the GLOBAL attribute in a program that contains the current program.

*script* specifies subscripting and is required if the data item referenced by *name* is a table element. (See page 250 for a complete description.) If it is not a table element, do not specify *script*. The format for *script* is as follows:

```
( integer-1 [ [ ,  ] integer-2 ] …  )
```

*refmod* specifies a subfield of the data item. (See page 250 for a complete description.) It has this format:

```
( offset : [ length ] )
```

*type* specifies the data type to be used in displaying the monitored data item or index when a change in value occurs. If this parameter is omitted, *type* defaults to the type of the item specified except when a type modifier is specified. The type value **IXN** may only be used with index-names and, when the named data item is an index-name, the only permissible type value is **IXN**. The type value **PTR** may only be used with pointer data items and, when the named data item is a pointer data item, the only permissible type value is **PTR**, except when a type modifier is specified.

The type modifiers ***** and **&** have the following effect:

- The ***** type modifier indicates that the data item to start monitoring is determined by an indirect reference, that is, a reference to the data item referenced by a pointer data item value. The data item specified in the command must be a pointer data item (data type **PTR**). In this case, *type*, if specified, indicates the type of the item referenced by the pointer data item value. If *type* is not specified, the default is hexadecimal. If *refmod* is not specified for an indirect reference, the entire effective memory area specified by the pointer data item value is monitored. The effective memory area specified by a pointer data item value begins with the effective address (*pointer.address* + *pointer.offset*) and ends just before the effective limit address (*pointer.address* + *pointer.length*). If *refmod* is specified for an indirect reference, *refmod* is applied to the indirect reference rather than the pointer data item itself. Thus, *refmod* may be used to monitor a subfield within the current effective memory area of an indirect reference. The indirect reference is resolved at the time the trap is set with the **T** (Trap)

Command and subsequent changes to the pointer data item used to set the trap do not change the data item that is being monitored by the trap. To monitor changes in the pointer data item itself, do not use the ***** type modifier.

- The **&** type modifier indicates that the base address of the data item specified is to be monitored. The data item referenced in the command must be a based linkage record because, otherwise, the address is not modifiable. The data item referenced in the command may be any data type except an index-name (data type **IXN**). When the **&** type modifier is used, the value to be monitored is a pointer value and, if *type* is specified, *type* must be **PTR**. A trap set on a based linkage data item, without the **&** type modifier, is resolved at the time the trap is set and subsequent changes to the base address of the based linkage item do not change the data item that is being monitored. To monitor changes in the base address of a based linkage item, use the **&** type modifier with the based linkage record data-name.

*alias* is a name you enter to serve as another name for the data operand specification that precedes it. For a complete description, see Alias Format. If present, *alias* must follow a pound sign (**#**). The characters that follow the **#** must form a valid COBOL word. Only the first three characters of this word are significant. When a command defines an alias and specifies the ***** (indirect through pointer) type modifier, the indirection is resolved at the time the alias is defined. In this case, the alias continues to refer to the data item that the pointer data item referenced when the alias was defined even if the pointer data item value has subsequently been changed. When a command defines an alias and specifies the **&** (address of data item) type modifier, the address of the data item is resolved each time the alias is specified using an alias format command.

For example, entering:

```
T MONTH-NAME(12)
```

directs Debug to suspend execution whenever the value of the 12th element in the table MONTH-NAME changes.

## Address-Size Format

The syntax for the **T** Command with the address-size format (on page 252) is as follows:

```
T  [ base : ]  address  [ + occur-size * occur-num ] …  ,  size ,
                 [ type ]  [ # alias ]
```

*base* specifies the base item for formal arguments, based linkage items, and external items as follows:

- For a USING formal argument, *base* is specified as **U** *arg-num*, where *arg-num* specifies the ordinal position of the argument in the USING list of the Procedure Division header provided in the allocation map of the program listing.
- For a GIVING formal argument, *base* is specified as **G**, as shown in the allocation map of the program listing.

- For a based linkage item, *base* is specified as **B** *item-num*, where *item-num* specifies the compiler assigned based linkage item number provided in the allocation map of the program listing.
- For an external item, base is specified as **X** *ext-num*, where *ext-num* specifies the compiler assigned external number provided in the allocation map of the program listing.

**Note** A Linkage Section data item, which is neither a formal argument item nor a based linkage item, is shown as "Not addressable:" in the allocation map of the program listing. Since such items have not been used in the source program, the compiler does not allocate a base pointer item for them and the Interactive Debugger cannot access them.

*address* specifies the address of the data item to be monitored. This is based on the value obtained from the data allocation map. For more information on addresses used with Debug, see address-size format (on page 252).

*occur-size* specifies the size of data items that contain OCCURS clauses in their definitions.

*occur-num* specifies the occurrence number for data items that contain OCCURS clauses in their definitions.

*size* specifies the size of the data item to be monitored.

*type* specifies the type of data item referenced. If this parameter is omitted, *type* defaults to hexadecimal. The type modifiers ***** (indirect through pointer) and **&** (address of data item) may not be specified in the address-size format.

*alias* is a name you enter to serve as another name for the data operand specification that precedes it. For a complete description, see Alias Format (on page 254). If present, *alias* must follow a pound sign (**#**). The characters that follow the **#** must form a valid COBOL word. Only the first three characters of this word are significant.

Before RM/COBOL executes a statement, it examines the contents of the specified data item with the value the data item had at the point program execution last resumed. If a change has not occurred, execution proceeds to the next statement. If a change has occurred, execution is suspended and the contents of the data item appear according to the rules set down in the discussion of the D (Display) Command (on page 263). The trap is updated, and remains in effect until a **U** Command is executed.

## Alias Format

The syntax for the **T** Command with the alias format for specifying a data item or index reference is as follows:

```
T  [ # alias ]
```

**#** is required.

*alias* must form a valid COBOL word, only the first three characters of which are valid. For a complete description, see Alias Format (on page 254). The alias must have been previously defined in an identifier or address-size format reference to the desired item (for example, in a **D** (Display) Command).

# U (Untrap) Command

Use the **U** Command to clear a single active data trap, or all currently active data traps.

## Identifier Format

The syntax for the **U** Command with the identifier format (on page 250) is as follows:

```
U  name-1  [ { IN | OF }  name-2 ] …  [ script ]  [ refmod ]
             [ ,  { type | { * | & } [ type ] } ]
```

*name-1* is a name declared as a data-name or index-name for the data item or index whose data trap is to be removed. *name-2* is a qualifier for *name-1*. Qualification is required if the name is not unique. The named data item or index must be described in the Data Division of the current program or be described with the global attribute in a program that contains the current program.

*script* specifies subscripting and is required if the data item referenced by name is a table element. (See page 250 for a complete description.) If it is not a table element, do not specify *script*. The format for *script* is as follows:

```
( integer-1 [ [ ,  ] integer-2 ] …  )
```

*refmod* specifies a subfield of the data item. (See page 250 for a complete description.) It has this format:

```
( offset : [ length ] )
```

*type* specifies the data type of the monitored data item that is to be removed from the monitored item list. If this parameter is omitted, *type* defaults to the type of the item specified except when a type modifier is specified. The type value **IXN** may only be used with index-names and, when the named data item is an index-name, the only permissible type value is **IXN**. The type value **PTR** may only be used with pointer data items and, when the named data item is a pointer data item, the only permissible type value is **PTR**, except when a type modifier is specified.

The type modifiers ***** and **&** have the following effect:

- The ***** type modifier indicates that the data item to discontinue monitoring is determined by an indirect reference, that is, a reference to the data item referenced by a pointer data item value. The data item specified in the command must be a pointer data item (data type **PTR**). In this case, *type*, if specified, indicates the type of the item referenced by the pointer data item value. If *type* is not specified, the default is hexadecimal. If *refmod* is not specified for an indirect reference, the entire effective memory area

specified by the pointer data item value is the data reference to discontinue monitoring. The effective memory area specified by a pointer data item value begins with the effective address (*pointer.address + pointer.offset*) and ends just before the effective limit address (*pointer.address + pointer.length*). If *refmod* is specified for an indirect reference, *refmod* is applied to the indirect reference rather than the pointer data item itself. Thus, *refmod* may be used to discontinue monitoring a subfield within the current effective memory area of an indirect reference.

- The **&** type modifier indicates that the data item to discontinue monitoring is the address of the data item specified. The data item referenced in the command must be a based linkage record because, otherwise, the address is not modifiable. The data item referenced in the command may be any data type except an index-name (data type **IXN**). When the **&** type modifier is used, the monitored item is a pointer data item and, if *type* is specified, *type* must be **PTR**.

For example, entering:

```
U MONTH-NAME(12)
```

clears the trap on the 12th element in the table MONTH-NAME.

## Address-Size Format

The syntax for the **U** Command with the address-size format (on page 252) is as follows:

```
U  [ [ base : ] address [ + occur-size * occur-num ] …  ]
```

*base* specifies the base item for formal arguments, based linkage items, and external items as follows:

- For a USING formal argument, *base* is specified as **U** *arg-num*, where *arg-num* specifies the ordinal position of the argument in the USING list of the Procedure Division header provided in the allocation map of the program listing.
- For a GIVING formal argument, *base* is specified as **G**, as shown in the allocation map of the program listing.
- For a based linkage item, *base* is specified as **B** *item-num*, where *item-num* specifies the compiler assigned based linkage item number provided in the allocation map of the program listing.
- For an external item, base is specified as **X** *ext-num*, where *ext-num* specifies the compiler assigned external number provided in the allocation map of the program listing.

**Note** A Linkage Section data item, which is neither a formal argument item nor a based linkage item, is shown as “Not addressable:” in the allocation map of the program listing. Since such items have not been used in the source program, the compiler does not allocate a base pointer item for them and the Interactive Debugger cannot access them.

*address* specifies the address of the data item for which a trap is active. This is based on the value obtained from the data allocation map. For more information on addresses used with Debug, see address-size format (on page 252). If *address* is not specified, all currently active data traps are cleared.

*occur-size* specifies the size of data items that contain OCCURS clauses in their definitions.

*occur-num* specifies the occurrence number for data items that contain OCCURS clauses in their definitions.

If you enter only **U**, all currently activated traps are cleared.

For example:

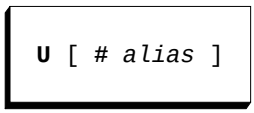
```
U 13+2*7
```

clears the trap on the data item located at decimal data address 25.

If a specific data trap does not exist, the command is in error.

## Alias Format

The syntax for the **U** Command with the alias format for specifying a data item or index reference is as follows:




**#** is required.

*alias* must form a valid COBOL word, only the first three characters of which are used. For a complete description, see Alias Format (on page 254). The *alias* must have been previously defined in an identifier or address-size format reference to the desired item (for example, in the **T** (Trap) Command which set the trap).

# Chapter 10: Configuration

Configuration determines such actions as screen displays, indexed file characteristics, default operational modes, the method by which the terminal is to be accessed, and general terminal characteristics.

Configuration is altered by writing a specific set of configuration directives into a configuration record. The complete set of configuration records is then written to a configuration file.

This chapter details the configuration file structure, configuration error format, configuration record types, and terminal configuration record types.

## Configuration File Structure

Configuration directives are contained in the configuration file. The configuration file is a line sequential file, and it can be created with any convenient text editor. The configuration file can then be specified in the Compile Command (using the G and H Options, as described on page 142) or in the Runtime Command (using the C and X Options, as described on pages 179 and 180, respectively).

Configuration records can appear in any order within the configuration file, except where noted otherwise. The first field in each record identifies the type of record or value being defined. The format of the remainder of the record depends on the type of record. Except for specific character sequences, keywords and parameters in the records are case-insensitive; uppercase and lowercase letters are equivalent.

The records are free field; that is, individual fields need not start in any predetermined column. The general syntax is as follows:

```
record-type keyword=value [,value]
```

*record-type* is the identifier of one of the configuration records detailed in this chapter. For many of the record-type identifiers, singular and plural forms of the record-type identifier are considered to define the same configuration record type. For example, either RUN-OPTION or RUN-OPTIONS may be used to specify the runtime options configuration. The alternative forms of the record-type identifiers are shown in parentheses in Table 29 on page 285.

*keyword* is the name of the keyword specification being described. It must be followed by an equal sign. Optional spaces following the equal sign are allowed.

*value*, depending on the keyword, may be either a string or a number. Value strings that contain a space, equal sign, or comma must be quoted with either the double quote (") or single quote (') character, and must use the same beginning and ending quote character. Quoted strings that contain the quote character must use a pair of consecutive quote characters to represent one quote character. For some keywords, the value may be either a single-character string or a number. In such cases, if the single-character string is a digit (0 – 9), it must be quoted or it will be considered to be a number. In all other cases, strings may be specified without quotes. Value strings are limited to a maximum of 160 characters. Value numbers may be specified in decimal notation as a string of decimal digits (0 – 9) or in hexadecimal notation as string of hexadecimal digits (0 – 9, A – F) with a leading 0x or trailing h.

Configuration records may vary in length. Configuration records may be continued beyond one record by placing an ampersand (&) character in column 1 of the second and subsequent records. The ampersands are logically replaced with spaces and the records are logically concatenated, ignoring trailing spaces. The maximum total length of a configuration record, including all continued records, is 510 characters.

Comments may be included in the configuration file. Comment text begins with a slash and an asterisk (/*) in columns 1 and 2. Lines that have /* in columns 1 and 2, as well as blank lines, are ignored in their entirety. Lines may have a single tail comment at the end. A tail comment, which begins with a forward slash and an asterisk (/*) and ends with an asterisk and a forward slash (*/), must be the last non-blank item on the line. The tail comment is removed and the rest of the line preceding the tail comment is processed normally. For example:

```
RUN-OPTION K=SUPPRESS  /* this is a tail comment */
```

# Automatic and Attached Configuration Files

UNIX versions of RM/COBOL prior to 7.1 allowed a configuration file to be linked into the runtime, compiler, or recovery utility. Version 7.1 and later of RM/COBOL for UNIX and version 7.5 and later of RM/COBOL for Windows allow a configuration file to be located automatically. This new method is described as an "automatic configuration file." Windows versions of RM/COBOL also allow a configuration file to be attached to the runtime, the compiler, and the recovery utility. This method is described as an "attached configuration file." Both of these methods are described in the following topics.

## Automatic Configuration File

Version 7.1 and later of RM/COBOL for UNIX and version 7.5 and later of RM/COBOL for Windows allow a configuration file to be located automatically by the RM/COBOL runtime system, the compiler, and the recovery utility. If the Automatic Configuration File module**, librmconfig.so** (on UNIX) or **librmcfg.dll** (on Windows), is present in the execution directory for the RM/COBOL component being executed, the module will be loaded and will attempt to locate an automatic

configuration file. The execution directory on UNIX is normally **/usr/bin**. The execution directory on Windows is normally **"C:\Program Files\RMCOBOL"**.

If it is desired to have a configuration file automatically loaded, there should be a file named **runcobol.cfg** (for the runtime system), **rmcobol.cfg** (for the compiler), or **recover1.cfg** (for the recovery utility) in the execution directory. If no file with the appropriate name is present, then there is no automatic configuration file. If the appropriate file is present, the records in the file will be used to configure the component being executed.

The automatic configuration file may be created and maintained with the editor of your choice. Records in the file are identical to those of a non-automatic configuration file. For additional information about the format of a configuration file, see Configuration File Structure (on page 281).

If the RM_DYNAMIC_LIBRARY_TRACE environment variable is defined (or the V Option, described on page 180, is specified on the Runtime Command or the V keyword of the RUN-OPTION configuration record (on page 322), is set to the value DISPLAY) and the Automatic Configuration File module is present in the execution directory, the load message produced by the Automatic Configuration File module indicates whether or not a configuration file has been automatically loaded. Further, if the first line of the configuration file (for example, **runcobol.cfg**) contains a slash and an asterisk (/*) in columns 1 and 2, the remainder of the line will be included with the load message.

An automatic configuration file, when found, is processed prior to any configuration file specified with a command-line option. On Windows, when an automatic configuration file is found, an attached configuration, if any, is ignored.

## Attached Configuration File

The Windows version of RM/COBOL allows configuration files to be physically attached to **rmcobol.exe**, **runcobol.exe**, and **recover1.exe** using the Attach Configuration (rmattach) utility (on page 613). The attached configuration will be processed prior to any configuration file specified with a command-line option, but if automatic configuration finds a configuration file, an attached configuration is ignored.

**Note** The use of an attached configuration file is no longer recommended. Attached configurations cannot be easily determined to be present or reviewed when present, are applicable to Windows only, and may be disallowed by future versions of the Windows operating system for security reasons. Automatic configuration, as described in the preceding section, has none of these problems and is recommended for all future development of RM/COBOL applications.

## Command-Line Configuration Files

In addition to automatic and attached configuration files, the Runtime and Compile Commands have command-line options to specify configuration files when the command is run. Each command has a pair of options, one to override any automatic or attached configuration and another to supplement either the automatic or attached or command-line specified overriding configuration file. For more information, see the “Configuration Options” section in Compile Command options (on page 141) and Runtime Command options (on page 179).

## Configuration Processing Order

All configuration processing occurs after the command-line options have been processed. Where a configurable option corresponds to a command-line option, if the option is specified in both the command line and the configuration, the command-line specified option overrides the configured option for that run of the command unless otherwise specified in this document. For cumulative options, for example, the L Runtime Command Option and the RUN-OPTION configuration record keyword L, all occurrences of the option are accumulated, first from the command line and then from the configuration.

If an overriding configuration file option is not specified on the command line, the automatic or attached configuration is processed first, as described in Automatic and Attached Configuration Files (on page 282).

If an overriding configuration file option is specified on the command line, any attached or automatic configuration file is ignored and the specified configuration file is processed after all the command-line options have been processed.

After completing processing of the automatic, attached, or command-line-specified overriding configuration file (only one of which is processed), if a supplemental configuration file is specified in the command-line options, the specified configuration file is processed.

Other than cumulative configuration options, it is generally true that when an option is specified more than once in the configuration, the last specified setting of the option takes effect unless the option corresponds to a command-line option that has been specified, in which case the command-line option setting is used. Exceptions are noted in the configuration option descriptions in this chapter. This means that, other than for the exceptional cases, the supplemental configuration file can be used to override configuration options specified in the automatic, attached, or command-line-specified overriding configuration file.

# Configuration Errors

If your configuration file contains errors, you will see a message similar to one of the following:

```
Configuration error code at record number in configuration file.

Configuration error code in configuration file.
```

*code* is the error number listed for configuration records, described in Configuration Errors (on page 284).

*number* is the logical record in the configuration file where the error occurred. When using *number* to determine which record is in error, count lines combined with their continuation lines as one record, and do not count comment lines or blank lines.

If the message is of the first format shown, the text of the configuration record in error will follow the message.

The actual error message formats are described in Compiler Configuration Errors (on page 174) for the compiler and in Appendix A: *Runtime Messages* (on page 357) for the runtime system and Indexed File Recovery utility.

# Configuration Records

Table 29 lists and describes the types of configuration records. Configuration options that are not used by the compiler, the runtime system, or the Indexed File Recovery (**recover1**) utility are ignored. Therefore, the same configuration file may be used to configure the compiler, the runtime system, and the Indexed File Recovery program, if appropriate.

**Note** Configuration is never necessary unless you want to change default option settings.

**Table 29: Types of Configuration Records**

| Record Type Identifier | Description | Compiler | Runtime System | Recover1 |
|---|---|---|---|---|
| COMPILER-OPTIONS (COMPILER-OPTION) (see page 287) | Allows default compiler options to be changed. | ✓ | | |
| DEFINE-DEVICE (see page 304) | Associates the file access name with a physical filename of a device or process. | ✓ | ✓ | |
| EXTENSION-NAMES (EXTENSION-NAME) (see page 308) | Defines the character-strings to be used for filename extensions. | ✓ | ✓ | |
| EXTERNAL-ACCESS-METHOD (see page 308) | Describes the access to external file access methods, such as Btrieve, and RM/InfoExpress, from the RM/COBOL file management system. | ✓ | ✓ | |
| INTERNATIONALIZATION (see page 309) | Specifies information necessary for internationalization. | ✓ | ✓ | |
| PRINT-ATTR (see page 311) | Describes printer characteristics. | ✓ | ✓ | |
| RUN-ATTR (see page 313) | Describes general runtime characteristics. | | ✓ | |
| RUN-FILES-ATTR (RUN-FILE-ATTR) (see page 316) | Describes the characteristics common to all file organizations. | ✓ | ✓ | ✓ |

[1] *Terminal configuration records are never necessary because terminal independence is provided by terminfo and termcap. Terminal configuration is still provided, however, to allow extensions to the basic capabilities provided by terminfo and termcap in a manner that will not conflict with other applications on your system.*

✓ *Indicates that the configuration record is used.*

**Table 29: Types of Configuration Records *(Cont.)***

| Record Type | Description | Compiler | Runtime System | Recover1 |
|---|---|---|---|---|
| RUN-INDEX-FILES (RUN-INDEX-FILE) (see page 320) | Describes indexed file characteristics. | | ✓ | |
| RUN-OPTION (RUN-OPTIONS) (see page 322) | Describes default runtime option values. | | ✓ | |
| RUN-REL-FILES (RUN-REL-FILE) (see page 325) | Describes relative file characteristics. | ✓ | ✓ | |
| RUN-SEQ-FILES (RUN-SEQ-FILE) (see page 326) | Describes sequential file characteristics. | ✓ | ✓ | ✓ |
| RUN-SORT (see page 327) | Describes SORT-MERGE characteristics. | | ✓ | |
| TERM-ATTR [1] (see page 327) | Describes terminal characteristics. | | ✓ | ✓ |
| TERM-INPUT [1] (see page 332) | Defines incoming character sequences. | | ✓ | ✓ |
| TERM-INTERFACE [1] (see page 342) | Specifies the format for the other terminal configuration records, as well as the interface to be used for screen I/O. | | ✓ | ✓ |
| TERM-UNIT [1] (UNIX Only) (see page 342) | Associates the unit number of ACCEPT and DISPLAY statements with the actual device on the system. | | ✓ | |

# COMPILER-OPTIONS Configuration Record

The COMPILER-OPTIONS configuration record identifier is followed by one or more keywords. If the keyword is allowed to have a value, it is followed by an equal sign (=) and the value. The COMPILER-OPTIONS record allows options to be configured. Configured options can be overridden by options entered in the Compile Command line. See the discussion of Compile Command options (on page 141).

Certain options require that another option be present before the compiler can proceed. For example, it is not valid to specify a cross-reference if the program listing is not being generated. Other options may conflict with each other. For instance, the option to treat COMPUTATIONAL operands as binary is not valid if the program is being compiled in RM/COBOL (74) version 2.*n*-compatible mode. These conditions are not checked when the configuration file is processed. Rather, they are deferred until the command-line options have also been processed.

The options in the COMPILER-OPTIONS record are processed in the order they appear in the configuration file. New options are simply added to old options. If a LISTING-PATHNAME or OBJECT-PATHNAME occurs more than once, the last occurrence of each is used. The possible keywords for the COMPILER-OPTIONS record are as follows:

1. **ACCEPT-SUPPRESS-CONVERSION.** This keyword suppresses automatic input conversion for Format 1 and 3 ACCEPT statements with numeric operands. If the value is set to YES, conversion is suppressed. If the value is set to NO, conversion is performed. The default value for this keyword is NO.

   This keyword corresponds to the compiler M Option (see page 147).

2. **ALLOW-DATE-TIME-OVERRIDE.** This keyword assists in testing for Year 2000 (Y2K) and other date/time problems by allowing parts of an application to be tested without changing the actual date and time on the computer. An initial date and time can be set prior to starting the RM/COBOL runtime system. The value of ALLOW-DATE-TIME-OVERRIDE is set to YES, and the value of the RM_Y2K environment variable is set with the desired date and time using the following format:

   ```
   RM_Y2K=YYYY,MM,DD,hh,mm,ss
   ```

   For example, RM_Y2K=1999,12,31,23,59,50 sets the initial runtime date and time to December 31, 1999 at 23:59:50 (ten seconds before the year 2000 begins). The time and date then advance normally from this initial value for the life of the runtime. The default value for this keyword is NO.

   The following points should be taken into consideration when using this keyword:

   - If the main program does not allow date/time override or if there is any error in the RM_Y2K environment variable value, no error message is generated and the runtime system continues with the actual machine date and time.
   - Only future time is allowed; the user cannot set time to the past.
   - Only the main program's option determines whether the RM_Y2K environment variable is scanned.
   - Once set at the beginning of the runtime, time advances normally for all programs called by that runtime system.

- Any new runtime system that is invoked will begin at the date and time set in the RM_Y2K environment variable at that instant. This could cause application errors. It is best to test multiple runtime systems by changing the actual date and time on a test machine.
- Writing records that contain future dates and times could damage production files. Users should test copies of their application files when using this feature.
- This feature can also be used to test date and time events, for example, end-of-day, end-of-month, end-of-quarter, end-of-year, and daylight-savings-time-change.
- Most systems do not understand time beyond 2038,1,18,21,14,7. Attempts to use an override date and time close to or beyond this value may lead to unpredictable results because the time cannot advance past this value.

For more information on obtaining composite date and time values, see Composite Date and Time (on page 218).

3. **BINARY-ALLOCATION.** This keyword allows configuration of the allocation sizes allowed for a binary numeric data item. The value may be specified in any of the following ways:

   BINARY-ALLOCATION=RM specifies the traditional RM/COBOL allocation scheme of two, four, eight, or sixteen bytes. This is the default configuration.

   BINARY-ALLOCATION=RM1 specifies that one- and two-digit binary data items will be allocated as a single byte. For three to thirty digits, traditional RM/COBOL allocation of two, four, eight, or sixteen bytes will be used.

   BINARY-ALLOCATION=MF-RM specifies that the minimum number of bytes will be allocated for binary numeric data items consistent with the PICTURE character-string. This is the traditional Micro Focus COBOL binary allocation when the IBMCOMP directive is not specified, but modified to use the minimum number of bytes needed for digit counts of nineteen through thirty. The number of bytes allocated is described in Table 30.

**Table 30: MF-RM Binary Allocation**

| Digits in PICTURE character-string | | Bytes of memory allocated |
|---|---|---|
| **Signed: S9(*n*)** | **Unsigned: 9(*n*)** | |
| 1-2 | 1-2 | 1 |
| 3-4 | 3-4 | 2 |
| 5-6 | 5-7 | 3 |
| 7-9 | 8-9 | 4 |
| 10-11 | 10-12 | 5 |
| 12-14 | 13-14 | 6 |
| 15-16 | 15-16 | 7 |
| 17-18 | 17-19 | 8 |
| 19-21 | 20-21 | 9 |
| 22-23 | 22-24 | 10 |
| 24-26 | 25-26 | 11 |
| 27-28 | 27-28 | 12 |
| 29-30 | 29-30 | 13 |

BINARY-ALLOCATION=CUSTOM=*integer-list* specifies a user-selected custom binary allocation configuration, where *integer-list* is a comma-separated list of integers. Each integer in *integer-list* is 1 through 16 and specifies an allowed allocation size in bytes. Allocation sizes not listed in *integer-list* will not be used. For a custom configuration, the compiler will allocate the minimum number of bytes allowed by the custom configuration that supports the number of digits described in the PICTURE character-string for a numeric binary data item. See Table 30 for the minimum number of bytes necessary for a given number of digits.

The RM, RM1, and MF-RM values for the BINARY-ALLOCATION keyword have the following relationships to the CUSTOM=*integer-list* value:

- BINARY-ALLOCATION=RM is equivalent to BINARY-ALLOCATION=CUSTOM=2,4,8,16 when BINARY-ALLOCATION-SIGNED=YES. When BINARY-ALLOCATION-SIGNED=NO (the default), the only difference is that 19-digit unsigned binary is allocated with 16 bytes of storage for RM mode and 8 bytes of storage for the CUSTOM mode.
- BINARY-ALLOCATION=RM1 is equivalent to BINARY-ALLOCATION=CUSTOM=1,2,4,8,16 when BINARY-ALLOCATION-SIGNED=YES. When BINARY-ALLOCATION-SIGNED=NO (the default), the only difference is that 19-digit unsigned binary is allocated with 16 bytes of storage for RM mode and 8 bytes of storage for the CUSTOM mode.
- BINARY-ALLOCATION=MF-RM is equivalent to BINARY-ALLOCATION=CUSTOM=1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16. Note, however, that since 30 digits only require 13 bytes of storage for signed or unsigned binary, no item will be allocated with 14, 15, or 16 bytes of storage.

Some examples of custom binary allocation are:

- If BINARY-ALLOCATION=CUSTOM=16, all binary items will be 16-bytes long.
- If BINARY-ALLOCATION=CUSTOM=1,4,16, then all one-digit and two-digit binary items will be 1-byte long, any binary items greater than two digits and less than ten digits will be 4-bytes long, and all remaining binary items will be 16-bytes long.

Use of the BINARY-ALLOCATION keyword will affect:

- File record structures that include binary data items.
- REDEFINES validity when the subject or object defines binary data items.
- Any LINAGE-COUNTER special registers.
- CALL statement arguments that are binary or are groups that contain binary data items. This includes CALL statement arguments for the supplied subprogram library, for example, arguments for C$CARG (on page 532) and C$SCRD (on page 559).
- Pop-up windows, because the Pop-Up Window Control Block (on page 208) is a group that contains binary data items that must be allocated a specific number of bytes.

Thus, this configuration capability should be used with care. The binary allocation override language feature of RM/COBOL (see the USAGE clause in the data description entry) is more appropriate in situations where the programmer wants to control the allocated sizes of certain binary data items on a case by case basis. The binary allocation configuration capability is intended mostly for easing conversion to RM/COBOL from other COBOL dialects that use a different allocation scheme. When this configuration keyword is used, it must be used consistently throughout a programming project, and, in some cases, the binary allocation override language feature may need to be used to resolve conflicts (for example, when defining argument data items for the Liant supplied subprogram library).

4. **BINARY-ALLOCATION-SIGNED.** This keyword causes unsigned binary data items to be allocated as if they were signed, so that signed and unsigned data items with the same number of digits will be allocated the same number of bytes of storage. If the value is set to YES, unsigned binary data items are allocated the same number of bytes of storage as if they were signed; only the allocation is affected, the data item is not treated as signed for any other purpose. If the value is set to NO, unsigned items are allocated the minimum number of bytes necessary to support the unsigned precision specified by the PICTURE character-string, which for several cases is one less byte of storage than for the corresponding signed precision. The default value for this keyword is NO.

5. **COBOL-74.** This keyword allows programs created for ANSI COBOL 1974 to be compiled. If the value is set to YES, ANSI COBOL 1974 semantics and I-O status values are assumed. If the value is set to NO, ANSI COBOL 1985 semantics and I-O status values are assumed. The default value for this keyword is NO.

   This keyword corresponds to the compiler 7 Option (see page 149).

6. **COMPUTATIONAL-AS-BINARY.** This keyword, when the value is set to YES, causes the compiler to treat data items described in the source program as usage COMP or COMPUTATIONAL as if they had been described as usage BINARY. If you set the value of this keyword to NO—and do not set a value for the COMPUTATIONAL-TYPE keyword—usage COMP and COMPUTATIONAL retain their usual format. For illustrations, see Unsigned Numeric COMPUTATIONAL-4 Data (on page 419) and Signed Numeric COMPUTATIONAL-4 Data (on page 422). The default value for this keyword is NO.

   **Note** Setting COMPUTATIONAL-AS-BINARY=YES creates compatibility between COMP data items and IBM OS/VS COMP data items. This can improve computational speed at runtime and reduce the amount of storage occupied by a COMP data item.

   When the value is set to YES, this keyword corresponds to the compiler U=B Option (see page 143).

   COMPUTATIONAL-AS-BINARY is an obsolete configuration capability retained for compatibility with existing configuration files. COMPUTATIONAL-AS-BINARY=YES is equivalent to COMPUTATIONAL-TYPE=BINARY.

   COMPUTATIONAL-TYPE and COMPUTATIONAL-AS-BINARY should not be specified together in the same configuration.

7. **COMPUTATIONAL-TYPE.** This keyword determines the data format used for data items described as COMPUTATIONAL or COMP in their data description entry.

   The COMPUTATIONAL-TYPE keyword may be assigned one of the following values: BINARY, DISPLAY, UNPACKED-DECIMAL, or PACKED-DECIMAL. If the value is set to BINARY, the format is the same as if BINARY had been specified in the USAGE clause in the data description entry. If the value is set to DISPLAY, the format is the same as if DISPLAY had been specified in the USAGE clause in the data description entry. If the value is set to PACKED-DECIMAL, the format is the same as if PACKED-DECIMAL had been specified in the USAGE clause in the data description entry. If the value is set to UNPACKED-DECIMAL, the format is the default unpacked decimal format for COMPUTATIONAL data items. For illustrations, see Unsigned Numeric COMPUTATIONAL (on page 414) and Signed Numeric COMPUTATIONAL (on page 415). The default data format for a COMPUTATIONAL or COMP data item is UNPACKED-DECIMAL.

   Setting COMPUTATIONAL-TYPE=BINARY corresponds to the compiler U=B Option. Setting COMPUTATIONAL-TYPE=DISPLAY corresponds to the compiler U=D Option. Setting COMPUTATIONAL-TYPE=PACKED-DECIMAL corresponds to the compiler U=P Option. See page 143.

   COMPUTATIONAL-TYPE and COMPUTATIONAL-AS-BINARY should not be specified together in the same configuration.

8. **COMPUTATIONAL-VERSION.** This keyword modifies the data format of data items described as signed COMPUTATIONAL (COMP) and signed COMPUTATIONAL-3 (COMP-3) in their data description entry. This configuration option affects the value used for positive sign representation.

   The COMPUTATIONAL-VERSION keyword may be assigned one of the following values: RMCOBOL85, RMCOBOL2 or RMCOS. The RMCOBOL85 value represents the default positive sign convention for RM/COBOL compilers and causes UNPACKED data items to use the hexadecimal value 0C to indicate positive values, and PACKED-DECIMAL data items to use the hexadecimal value C to indicate positive values. The RMCOBOL2 value selects the positive sign convention for previous RM/COBOL (74) version 2 compilers and causes UNPACKED data items to use the hexadecimal value 0B to indicate positive values, and PACKED-DECIMAL data items to use the hexadecimal value F to indicate positive values. The RMCOS value selects the positive sign convention for the RM/COBOL-74 compiler for the RM/COS operating system and causes UNPACKED data items to use the hexadecimal value 0B to indicate positive values, and PACKED-DECIMAL data items to use the hexadecimal value B to indicate positive values.

   The RMCOBOL2 and RMCOS options allow applications to access files containing COMPUTATIONAL and COMPUTATIONAL-3 data items that use previous sign representations. This keyword has no corresponding Compile Command line option. The default value for this keyword is RMCOBOL85.

**Note 1** The COMPUTATIONAL-VERSION keyword may be used in conjunction with the COMPUTATIONAL-TYPE keyword or the compiler U Option (see page 143). For example, by setting COMPUTATIONAL-TYPE=PACKED-DECIMAL (or the compiler U=P Option in the Compile Command) and COMPUTATIONAL-VERSION=RMCOBOL2, COMPUTATIONAL data items will be PACKED-DECIMAL with the RM/COBOL (74) version 2 sign representation.

**Note 2** When using the COMPUTATIONAL-VERSION keyword, you cannot specify an object version level less than 7.

9. **DEBUG.** This keyword determines whether source programs are to be compiled as if the WITH DEBUGGING MODE clause appeared in each program. If the value is set to YES, the debugging mode is selected. If the value is set to NO, debugging mode is not selected. The default value for this keyword is NO.

   This keyword corresponds to the compiler D Option (see page 149).

10. **DEBUG-TABLE-OUTPUT.** This keyword, when the value is set to YES, causes the compiler to include both the symbol table and the debug line table in the object program. Furthermore, when the value is set to ALL, the actual text of compiler-generated lines that do not appear in source or copy files is also included in the object file and is available during debugging.

    When the debug line table is included in the object program, CodeWatch can display the program's source at execution time. Setting YES is sufficient for most purposes. If ALL is set, the displayed source has the appearance of a printed listing. Note that this may lead to large object program files. After debugging is complete, this information may be removed by the STRIP option in the Combine Program (rmpgmcom) utility (on page 583).

    When the value of this keyword is set to NO, the line table is not included in the object file. The default value for this keyword is NO.

    Setting DEBUG-TABLE-OUTPUT=YES corresponds to the compiler Y=2 Option. Setting DEBUG-TABLE-OUTPUT=ALL corresponds to the compiler Y=3 Option. (See page 148 for a description of these options.) Both options imply SYMBOL-TABLE-OUTPUT=YES (see page 300).

11. **DERESERVE.** This keyword directs the compiler to remove words (and their associated language features) from the reserved words list. The value of the DERESERVE keyword is a comma-separated list of those words to be removed from the reserved words list. All words specified must be found in the compiler reserved words list. When a word is removed from the compiler reserved words list, the word is considered to be a user-defined word wherever it occurs in a source program. The context-sensitive words are contained in the compiler reserved words list and may be removed by use of the DERESERVE keyword. When a context-sensitive word is removed, it is considered a user-defined word even in the context in which it would normally be reserved. By default, no words are removed from the reserved words list.

    This keyword has no corresponding Compile Command line option.

12. **DISPLAY-UPDATE-MESSAGES.** This keyword controls which messages are displayed when the automatic update check determines that there is an update message available for the RM/COBOL compiler. The message is displayed at compiler termination. If the value of this keyword is set to ALL, then all update messages are displayed. If the value of this keyword is set to URGENT-ONLY, then only messages that Liant designates as urgent are displayed. The default value of this keyword is ALL.

13. **FLAGGING.** This keyword flags specified elements of the COBOL language in the listing file. Multiple values are separated by commas. One or more of the following values may be included in any order:

| | |
|---|---|
| COM1 | INTERMEDIATE |
| COM2 | OBSOLETE |
| EXTENSION | SEG1 |
| HIGH | SEG2 |

- COM1 flags COM1 and COM2 elements of the language.
- COM2 flags COM2 elements of the language.
- EXTENSION flags RM/COBOL extensions to ANSI COBOL 1985.
- HIGH flags HIGH elements of the language.
- INTERMEDIATE flags HIGH and INTERMEDIATE elements of the language.
- OBSOLETE flags obsolete elements of the language.
- SEG1 flags SEG1 and SEG2 elements of the language.
- SEG2 flags SEG2 elements of the language.

By default, none of the occurrences of the preceding items is flagged.

This keyword corresponds to the compiler F Option (see page 150).

14. **LINKAGE-ENTRY-SETTINGS.** This keyword allows configuration of the treatment of based linkage items, including formal arguments, upon entry to subprograms called during the run unit. The value assigned to the keyword establishes the behavior for the program or programs compiled with that setting of the keyword. The programs that are called in a run unit need not all be compiled with the same value for this keyword.

    This configuration keyword is intended to provide compatibility when porting existing programs from another COBOL implementation. New RM/COBOL programs should be designed to use the default behavior; that is, UNLINK-NONE, which is described in the value descriptions that begin on page 294.

    Much of the complexity of the following descriptions occurs only when a Format 5 or Format 6 SET statement explicitly modifies the base address of a formal argument (see the *RM/COBOL Language Reference Manual*). It is recommended that such use be restricted to those cases where it accomplishes a clear programming goal, such as setting a default value for an argument that will commonly be omitted.

    For a formal argument that corresponds to an existing actual argument, the actual argument address is used during execution of the subprogram when references are made to the formal argument, except that, if a Format 5 or 6 SET statement modifies the base address of the formal argument, then that base address overrides the actual argument address until the program exits. When a formal argument corresponds to an omitted actual argument, or to an actual argument that has a null base address, the last setting of the based linkage base address is used when references are made to the formal argument.

    **Note** A pointer data item, whose value is NULL, does not have a null base address.

The last setting of the based linkage base address may have resulted from any one of the following:

- The setting to NULL when the program was placed into its initial state.
- The setting established by an explicit Format 5 SET statement.
- The setting to NULL upon entry because of the UNLINK-ALL or UNLINK-FORMAL-ARGUMENTS values for this keyword.
- The setting to the last passed actual argument base address because of the LINK-FORMAL-ARGUMENTS value for this keyword.

In the value descriptions below, the word "link" is used to refer to setting the base address of a based linkage item. This setting is as if a Format 5 SET statement were executed where the sending item is the address of the item being linked and the receiving item is the address of the based linkage item. Because of the way base addresses are evaluated for based linkage items that are also formal arguments (as described above), this is equivalent to setting the base address of the formal argument to itself. For example:

```
SET ADDRESS OF formal-argument-1 TO ADDRESS OF formal-argument-1.
```

The word "unlink" is used to refer to execution of a similar Format 5 SET statement, except that the sending item is the figurative constant NULL. For example:

```
SET ADDRESS OF based-linkage-item-1 TO NULL.
```

The value for LINKAGE-ENTRY-SETTINGS may be specified in any of the following ways:

- UNLINK-ALL specifies that all based linkage items (including all formal arguments) be reset to a base address value of NULL upon each entry to the subprogram. The address for each existing actual argument corresponding to a formal argument will override this setting during the execution of the subprogram until the subprogram exits. Specifying this keyword causes behavior matching the Micro Focus COBOL behavior for the NOSTICKY-LINKAGE directive, which is the Micro Focus COBOL default behavior. The behavior is effectively as if, for the purposes of based linkage items only, the program was canceled each time it exited; that is, as if it had the PROGRAM IS INITIAL attribute specified.
- UNLINK-FORMAL-ARGUMENTS specifies that only based linkage items that are also formal arguments be reset to a base address of NULL upon each entry to the subprogram. The address for each existing actual argument corresponding to a formal argument will override this setting during the execution of the subprogram until the subprogram exits. Based linkage items that are not formal arguments will retain the base address last established by a Format 5 SET statement or modified by a Format 6 SET statement. Specifying this keyword causes behavior matching the Micro Focus COBOL behavior for the STICKY-LINKAGE"1" directive.
- UNLINK-NONE specifies that no based linkage items will be reset upon entry to the subprogram. All based linkage items will retain the base address last established by a Format 5 SET statement or modified by a Format 6 SET statement. The address for each existing actual argument corresponding to a formal argument will override any such setting during the execution of the subprogram until the subprogram exits. Where a

formal argument corresponds to an omitted actual argument, or to an actual argument that has a null base address, the last set base address will be used. Only when the program is placed into its initial state, either on the first CALL in the run unit or the first CALL after the subprogram has been canceled, will the based linkage base addresses be set to a NULL address value. For an initial program, that is, a program described with the PROGRAM IS INITIAL clause in the PROGRAM-ID paragraph, the based linkage base addresses will be set to a NULL address value for each entry because the program is effectively canceled after it exits for each time it is called. The behavior for this keyword is the default behavior for RM/COBOL and is more fully described in the description of the "Linkage Section" in the chapter entitled, *Data Division*, of the *RM/COBOL Language Reference Manual*. There is no corresponding Micro Focus COBOL behavior.

- LINK-FORMAL-ARGUMENTS specifies the base address of each existing actual argument corresponding to a formal argument is set as the base address of the formal argument. This is as if a Format 5 SET statement was executed that specified the ADDRESS OF the actual argument as the sending item and the ADDRESS OF the formal argument as the receiving item. The base address for a formal argument is not modified when the formal argument corresponds to an omitted actual argument, or to an actual argument that has a null base address. Other than these implicit settings for formal arguments, the behavior is the same as for UNLINK-NONE. With this specification, when an actual argument is omitted in a subsequent call, the last previously passed actual argument will be used by the subprogram except when the base address has been changed by an explicit Format 5 or Format 6 SET statement execution. Specifying this keyword causes behavior matching the Micro Focus COBOL behavior for the STICKY-LINKAGE"2" directive.

15. **LISTING-ATTRIBUTES.** This keyword determines which information is to be included in the program listing, and where the program listing will be directed. One or more of the following values may be included, in any order (multiple values must be separated by commas):

| | |
|---|---|
| ALLOCATION-MAP | PRINT-LISTING |
| CROSS-REFERENCE | SUPPRESS-COPY-FILES |
| ERROR-ONLY-LIST | SUPPRESS-REPLACED-LINES |
| LISTING-FILE | TERMINAL-LISTING |
| NO-TERMINAL-DISPLAY | |

- ALLOCATION-MAP generates an allocation map. When set to this value, the LISTING-ATTRIBUTES keyword corresponds to the compiler A Option (see page 144). By default, the allocation map is not generated.
- CROSS-REFERENCE generates a cross reference map. When set to this value, the LISTING-ATTRIBUTES keyword corresponds to the compiler X Option (see page 146). By default, the cross reference listing is not generated.
- ERROR-ONLY-LIST includes only erroneous source lines in the listing file. When set to this value, the LISTING-ATTRIBUTES keyword corresponds to the compiler E Option (see page 145). By default, the source program component of the listing is not suppressed.

- LISTING-FILE writes a copy of the listing file to disk. When set to this value, the LISTING-ATTRIBUTES keyword corresponds to the compiler L Option (see page 145). By default, a copy of the listing file is not written to disk.
- NO-TERMINAL-DISPLAY suppresses the display of informational messages on the screen. When set to this value, the LISTING-ATTRIBUTES keyword corresponds to the compiler K Option (see page 143). By default, the informational messages are displayed.
- PRINT-LISTING prints a copy of the listing file. When set to this value, the LISTING-ATTRIBUTES keyword corresponds to the compiler P Option (see page 146). By default, the listing file is not printed.
- SUPPRESS-COPY-FILES directs that text in the copy files not be placed in the listing file. When set to this value, the LISTING-ATTRIBUTES keyword corresponds to the compiler C or C=1 Option (see page 144). By default, copy files are placed into the listing file.
- SUPPRESS-REPLACED-LINES directs that the comment lines containing text that has been replaced as the result of the REPLACE statement or the REPLACING phrase of the COPY statement not be included in the listing file. This corresponds to the compiler C=2 Option (see page 144). By default, replaced lines are included as comments in the listing file.

  **Note** Specifying both of the values, SUPPRESS-COPY-FILES and SUPPRESS-REPLACED-LINES, in the LISTING-ATTRIBUTES keyword is equivalent to the compiler C=3 Option.
- TERMINAL-LISTING displays a copy of the listing file on the screen. When set to this value, the LISTING-ATTRIBUTES keyword corresponds to the compiler T Option (see page 146). By default, a copy of the listing is not written to the standard output device.

16. **LISTING-DATE-FORMAT.** This keyword directs the compiler to use a specific format for the compilation date in the header lines on listing pages. The value must be one of the following:

    - MMDDYY specifies month-of-year, day-of-month, year-of-century.
    - DDMMYY specifies day-of-month, month-of-year, year-of-century.
    - YYMMDD specifies year-of-century, month-of-year, day-of-month.
    - YYDDD specifies year-of-century, Julian day-of-year.
    - MMDDYYYY specifies month-of-year, day-of-month, year-of-millennium.
    - DDMMYYYY specifies day-of-month, month-of-year, year-of-millennium.
    - YYYYMMDD specifies year-of-millennium, month-of-year, day-of-month.
    - YYYYDDD specifies year-of-millennium, Julian day-of-year.

    The default value for this keyword is MMDDYYYY. This keyword has no corresponding Compile Command line option.

    **Note** This keyword affects the date inserted for the DATE-COMPILED paragraph. The same date format used in the listing header is inserted in the DATE-COMPILED paragraph.

17. **LISTING-DATE-SEPARATOR.** This keyword directs the compiler to use a specific separator character for the compilation date in the header lines on listing pages. The value of this keyword must be a single-character string or a number from 0 to 255. The default value for this keyword is "/" (47 or 0x2f).

    This keyword has no corresponding Compile Command line option.

    **Note** This keyword affects the date inserted for the DATE-COMPILED paragraph. The same date format used in the listing header is inserted in the DATE-COMPILED paragraph.

18. **LISTING-PATHNAME.** This keyword directs the compiler to write the program listing to the indicated directory. When using this keyword, it is not necessary to set LISTING-ATTRIBUTES=LISTING-FILE. The value must be a string that specifies the listing file pathname. The default value for this keyword is not to produces a listing file.

    This keyword corresponds to the compiler L Option (see page 145).

19. **LISTING-TIME-SEPARATOR.** This keyword directs the compiler to use a specific separator character for the compilation time in the header lines on listing pages. The value of this keyword must be a single-character string or a number from 0 to 255. The default value for this keyword is ":" (58 or 0x3a).

    This keyword has no corresponding Compile Command line option.

20. **NO-DIAGNOSTIC.** This keyword directs the compiler to suppress specified diagnostic messages. The value of this keyword must be a list (possibly with one entry) of numbers or certain named values. The list specifies the diagnostic messages to be suppressed. The default value for this keyword is an empty list, that is, no diagnostic messages are suppressed. A number in the value list causes suppression of the correspondingly numbered diagnostic message as shown in a compilation listing or as documented in Appendix B: *Compiler Messages* of the *Language Reference Manual*. The named values that may be specified in the value list are as follows:

    - The value INFO specifies that all informational messages should be suppressed. (Informational messages providing additional detail about a preceding summary diagnostic message are not suppressed unless the associated summary diagnostic message is also suppressed.)
    - The value WARNINGS specifies that all warnings and informational messages should be suppressed.
    - The value MINOR-ERRORS specifies that all minor errors, warnings, and informational messages should be suppressed. Minor errors are 94, 99, 319, 339, 361, 362, 363, 424, and 432. These errors represent violations of COBOL language rules that have been relaxed by other dialects of COBOL. RM/COBOL generally interprets these erroneous constructions in the same manner as those other dialects.
    - The value ERRORS specifies that all non-failure errors should be suppressed. Use of this named classification is not recommended; it was provided for completeness only. Use of this named classification can lead to confusing failure errors caused by suppressed non-failure errors. Also, the execution behavior of the program may not be as desired in some cases.

    Failure errors, that is, errors for which reasonable code cannot be generated or for which the compiler must skip scanning of some source in order to find valid COBOL syntax cannot be suppressed, either by number or by use of the ERRORS named value. Also, message number 5, "Scan resumed.", can only be

suppressed by listing its number after any named message classifications, because named message classifications always restore message number 5 for output.

This keyword has no corresponding Compile Command line option.

21. **OBJECT-PATHNAME.** This keyword directs the compiler to write the object file to the indicated directory. The value must be a string that specifies the object directory pathname. The default value for this keyword is to write the object file to the same directory as the source file.

    This keyword corresponds to the compiler O Option (see page 147).

22. **OBJECT-VERSION.** This keyword specifies the highest allowed object version level of code generated by the compiler. The value must be an integer in the range 7 through 12. The default value for this keyword is the current object version number, 12. For compilers licensed for evaluation or educational purposes or for a limited time (ValidThru date specified in the compiler license), the minimum object version value is 9.

    This keyword corresponds to the compiler Z Option (see page 149).

23. **RESEQUENCE-LINE-NUMBERS.** This keyword directs the compiler to generate a sequential line number in the first six columns of source records as they appear on the listing. The source file is not affected.

    If the value is set to YES, this keyword numbers records beginning with 1 for each source or copy input file. The number can be helpful when editing the source file. This line number cannot be used with the RM/COBOL Interactive Debugger.

    If the value is set to NO, the compiler will print the source record exactly as read, including any commentary information present in columns 1 through 6.

    The default value for this keyword is NO. This keyword corresponds to the compiler R Option (see page 146).

24. **RMCOBOL-2.** This keyword allows programs created for the RM/COBOL (74) version 2.*n* compiler to be compiled. If the value is set to YES, RM/COBOL (74) version 2.*n* programs are accepted. If the value is set to NO, they are not. The default value for this keyword is NO.

    This keyword corresponds to the compiler 2 Option (see page 150).

25. **SEPARATE-SIGN.** This keyword determines whether the compiler is to use a separate or a combined sign for a signed numeric data item with DISPLAY specified in the USAGE clause when a SIGN clause is not specified in the data description entry. If the value is set to YES, a separate sign is assumed. If the value is set to NO, a combined sign is assumed. The default value for this keyword is NO.

    This keyword corresponds to the compiler S Option (see page 143).

26. **SEQUENTIAL-FILE-TYPE.** This keyword determines the organization of sequential files not explicitly defined as binary sequential or line sequential in their SELECT entries.

    The SEQUENTIAL-FILE-TYPE keyword may be assigned one the following values: LINE or BINARY. If the value is set to LINE, all files not defined as binary sequential are line sequential. If the value is set to BINARY, all files not defined as line sequential are binary sequential. If this keyword is not configured and no selection is made at compile time, the decision on whether the file is BINARY or LINE is deferred to program execution. The choice is then

controlled by the configured DEFAULT-TYPE. The default value for this keyword is to defer making the decision until runtime, where the decision is based on the DEFAULT-TYPE keyword (see page 326) of the RUN-SEQ-FILES configuration record.

Setting SEQUENTIAL-FILE-TYPE=BINARY corresponds to the compiler B Option (see page 144). Setting SEQUENTIAL-FILE-TYPE=LINE corresponds to the compiler V Option (see page 144).

27. **STRICT-REFERENCE-MODIFICATION.** This keyword can be used to suppress the version 9 and later compiler default of relaxing the reference modification rules. Starting in version 9, the compiler does not require that the quantity offset plus length less one in reference modification be less than or equal to the length of the data item being reference-modified. If the value of this keyword is set to YES, then the strict, ISO 1989-1985-compliant rules prior to version 9 are enforced at compile and runtime. The default value for this keyword is NO, which results in the compiler allowing the relaxed rules for reference modification at compile and runtime. The relaxed rules do not allow offsets less than 1 to be used in reference modification, and lengths less than 1 may not be specified as literals. The relaxed rules treat a zero or negative length reference modifier at runtime as giving a zero-length source or destination.

28. **SUPPRESS-FILLER-IN-SYMBOL-TABLE.** This keyword can be used to suppress the version 7.5 and later compiler default of inserting all FILLER data items into the symbol table. Prior to version 7.5, the compiler reduced memory usage by not inserting all FILLER data items into the symbol table. Starting in version 7.5, the compiler inserts all FILLER data items into the symbol table to support the FILLER phrase of the INITIALIZE statement. If the value of this keyword is set to YES, then FILLER data items, other than group items, items that are described with the OCCURS clause or items that are conditional-variables for associated condition-names, are not inserted into the symbol table. When this keyword is set to YES, the compiler produces a warning if the FILLER phrase of the INITIALIZE statement is used, because the phrase is largely ineffective in this case. The default value for this keyword is NO, which results in the compiler inserting all FILLER data items into the symbol table.

29. **SUPPRESS-LITERAL-BY-CONTENT.** This keyword can be used to suppress the version 7.5 and later compiler default of passing literals specified in the USING phrase of CALL statements as if the BY CONTENT phrase applied. If the value is set to YES, then literals without an explicit BY CONTENT phrase will be passed by reference as they were prior to version 7.5. The default value for this keyword is NO, which causes literals to be passed by content, thus protecting the value of the literal in the calling program from changes made to the corresponding linkage section data item in the called program. The purpose of this keyword is only to provide strict backward compatibility. Setting the value to YES should be done only to determine if an application was depending on changing a literal value passed in the USING phrase of a CALL statement. Once this is determined, the program should be corrected to avoid such a dangerous dependence. For additional information, see Argument Considerations (on page 216).

30. **SUPPRESS-NUMERIC-OPTIMIZATION.** This keyword allows suppression of optimization of code for certain numeric operations. The optimized code that the compiler normally generates for numeric operations assumes that all nonbinary numeric data items contain only standard digits and signs, as described in Appendix C: *Internal Data Formats* (on page 403). Setting the value of the SUPPRESS-NUMERIC-OPTIMIZATION keyword to a value of YES directs the compiler to generate unoptimized code for all nonbinary

numeric operations. The unoptimized code is more likely to interpret correctly a nonbinary numeric field that contains nonstandard digits and signs. In particular, it will treat space characters and binary zero characters as if they were display zeros, and it will accept a wider range of representations of a positive sign. The unoptimized code takes longer to execute than the optimized code. The difference will be noticeable in programs that have a very high density of numeric operations. When the value of the SUPPRESS-NUMERIC-OPTIMIZATION keyword is set to NO, the compiler generates the normal, optimized code for numeric operations.

The default value for this keyword is NO. This keyword has no corresponding Compile Command line option.

31. **SYMBOL-TABLE-OUTPUT.** This keyword, when its value is set to YES, causes the compiler to include the symbol table in the object program. Note that this information may be removed by the STRIP option in the Combine Program (rmpgmcom) utility (on page 583). When the symbol table is included in the object program, source program data-names and index-names may be used in Debug commands at execution time. See Chapter 9: *Debugging* (on page 245).

    When the value of this keyword is set to NO, the symbol table is not included in the object program. The default value for this keyword is NO.

    Selecting SYMBOL-TABLE-OUTPUT=YES corresponds to the compiler Y Option (see page 148).

32. **WHEN-COMPILED-FORMAT.** This keyword specifies the format of the value for the WHEN-COMPILED special register.

    When the value of this keyword is set to OSVS, the WHEN-COMPILED special register has a 20-character string value with the format “%H.%M.%S%b %d, %Y”, where %H is replaced with the hour (00 - 23), %M is replaced with the minutes (00 - 59), %S is replaced with the seconds (00-61), %b is replaced with the month (Jan-Dec), %d is replaced with the day of month (01 - 31), and %Y is replaced with the four-digit year of the compilation date and time; for example, “15.21.39Apr 23, 2004”.

    When the value of this keyword is set to VSC2, the WHEN-COMPILED special register has a 16-character string value with the format “%m/%d/%y%H.%M.%S”, where %m is replaced with the month (01 - 12), %d is replaced with the day of month, %y is replaced with the two-digit year, %H is replaced with the hour (00 - 23), %M is replaced with the minutes (00-59), and %S is replaced with the seconds (00-61) of the compilation date and time; for example, “04/23/0415.21.39”.

    When the value of this keyword is a string other than OSVS or VSC2, it is interpreted as an strftime (from the standard C library) format string. In this case, the format string may generate a result that is up to 80 characters in length. If the format generates a string longer than 80 characters, the WHEN-COMPILED register will have the value “Cfg error: WHEN-COMPILED fmt > 80”. An strftime format string contains codes preceded by a “%” character. Characters not preceded by a “%” character are copied unchanged to the output.

    The default value for this keyword is OSVS. This keyword has no corresponding Compile Command line option.

    The supported codes for UNIX and Windows are described in Table 31. On Windows, the UNIX-only codes produce no characters in the result string. On some UNIX systems, the UNIX-only codes may produce no characters in the result string, may reproduce the code in the result string (for example, “%F” yields “%F”), or may produce another value than explained

(for example, "%G" may be equivalent to "%Y"). Thus, use of the UNIX-only codes makes the configuration file operating system dependent, but since the WHEN-COMPILED special register is evaluated at compile time, the object program is still portable to other systems.

**Table 31: Date and Time Format Codes**

| Code | Generated Result Description | Typical Result Values[1] | Operating System |
|---|---|---|---|
| **%%** | A (single) percent sign. | % | UNIX and Windows |
| **%a** | Abbreviated weekday name. | Mon – Fri | UNIX and Windows |
| **%A** | Full weekday name. | Monday – Friday | UNIX and Windows |
| **%b** | Abbreviated month name. | Jan – Dec | UNIX and Windows |
| **%B** | Full month name. | January – December | UNIX and Windows |
| **%c** | Date and time representation appropriate for locale. | "08/20/04 16:01:52" | UNIX and Windows |
| **%C** | Century as a decimal number; however, on some UNIX systems, this code is equivalent to %N, the default date and time. | 00 – 99 | UNIX |
| **%d** | Day of month as a decimal number. | 01 – 31 | UNIX and Windows |
| **%D** | Date as %m/%d/%y. | "06/24/04" | UNIX |
| **%e** | Day of month as a decimal number; leading zeroes are replaced by spaces. | 1 – 31 | UNIX |
| **%F** | Date as %Y-%m-%d; however, on HP-UX, this code is equivalent to %B, the full month name. | "2004-06-24" | UNIX |
| **%G** | Date in ISO 8601:1988 date format. | "2004" | UNIX |
| **%h** | Abbreviated month name; this code is equivalent to %b. | Jan – Dec | UNIX |
| **%H** | Hour of day as a decimal number for a 24-hour clock. | 00 – 23 | UNIX and Windows |
| **%I** | Hour of day as a decimal number for a 12-hour clock. | 01 – 12 | UNIX and Windows |
| **%j** | Day of year as a decimal number. | 001 – 366 | UNIX and Windows |
| **%k** | Hour as a decimal number for a 24-hour clock; leading zeroes are replaced by spaces. | 0 – 23 | UNIX |
| **%l** | Hour as a decimal number for a 12-hour clock; leading zeroes are replaced by spaces. | 1 – 12 | UNIX |
| **%m** | Month of year as a decimal number. | 01 – 12 | UNIX and Windows |

**Table 31: Date and Time Format Codes (*Cont.*)**

| Code | Generated Result Description | Typical Result Values[1] | Operating System |
|---|---|---|---|
| **%M** | Minutes past hour as a decimal number. | 00 – 59 | UNIX and Windows |
| **%n** | New-line character. | X'0A' | UNIX |
| **%N** | Default date and time representation; some UNIX systems equate %C and %N, producing either the default date and time or the century for both. The default date and time value is as produced by the system date command. | "Thu Jun 24 13:04:30 CDT 2004" | UNIX |
| **%p** | Current locale's A.M. or P.M. indicator for a 12-hour clock in upper case. | AM, PM | UNIX and Windows |
| **%P** | Current locale's a.m. or p.m. indicator for a 12-hour clock in lower case). | am, pm | LINUX |
| **%r** | Time as %I:%M:%S %p (12-hour clock time with A.M. and P.M. indicator). | "11:05:32 AM" | UNIX |
| **%R** | Time as %H:%M. (24-hour clock time with hours and minutes). | "16:43" | UNIX |
| **%S** | Seconds past minute as a decimal number; allows for leap seconds. | 00 – 61 | UNIX and Windows |
| **%t** | Tab character. | X'09' | UNIX |
| **%T** | Time as %H:%M:%S (24-hour clock time with hours, minutes and seconds). | "15:35:42" | UNIX |
| **%u** | Weekday as a decimal number; Monday is 1. | 1 – 7 | UNIX |
| **%U** | Week of year as a decimal number, with Sunday as first day of week 1. | 00 – 53 | UNIX and Windows |
| **%V** | Week of year as a decimal number per ISO 8601:1988, where if the week containing January 1st has four or more days in the new year, it is week 1; otherwise, it is week 53 of the preceding year. | 01 – 53 | UNIX |
| **%w** | Weekday as a decimal number; Sunday is 0. | 0 – 6 | UNIX and Windows |
| **%W** | Week of year as a decimal number, with Monday as first day of week 1. | 00 – 53 | UNIX and Windows |

**Table 31: Date and Time Format Codes (*Cont.*)**

| Code | Generated Result Description | Typical Result Values[1] | Operating System |
|---|---|---|---|
| **%x** | Date representation for current locale. | "08/20/04" | UNIX and Windows |
| **%X** | Time representation for current locale. | "14:34:24" | UNIX and Windows |
| **%y** | Year without century as a decimal number. | 00 – 99 | UNIX and Windows |
| **%Y** | Year with century as a decimal number. | 2004 – 2050 | UNIX and Windows |
| **%z** | Time zone as an hour and minute offset from GMT. Windows and some UNIX systems support %z as equivalent to %Z (time zone name or time zone abbreviation). Some UNIX systems, notably Linux, report +0000 despite reporting a time zone name other than GMT or UTC for %Z. | "-0600" (for CST) or "-0500" (for CDT)<br><br>"CST" or "CDT" | Linux<br><br>Windows and some UNIX |
| **%Z** | Either the time zone name or time zone abbreviation (on UNIX, it is determined by the TZ environment variable, TIMEZONE environment variable, or locale; on Windows, this depends on registry settings). | "CST", "CDT" | UNIX and Windows |

[1] *In the Typical Result Values column, quoted examples are a representative example, whereas unquoted examples indicate a range of possible example values.*

**Note 1** On Windows, the "#" flag may precede any formatting code, but this flag is ignored for the **a**, **A**, **b**, **B**, **p**, **X**, **z**, **Z** and **%** codes. For the **c** code, this flag causes the long date and time representation to be used; for example, "Friday, April 23, 2004 17:32:45". For the **x** code, this flag cause the long date representation to be used; for example "Friday, April 23, 2004". For the **d**, **H**, **I**, **j**, **m**, **M**, **S**, **U**, **w**, **W**, **y**, and **Y** codes, this flag causes leading zeroes, if any, to be suppressed. The "#" flag is generally ignored on some UNIX systems, but causes literal output of the code with or without the "%" on others and affects letter case on Linux systems for codes **a**, **A**, **b**, **B**, **h** and **Z**.

**Note 2** On UNIX, the flags "O" (letter "O") and "E" may be used preceding certain codes. The "O" flag causes use of the locale's alternate digit symbols (for example, roman numerals) with the format codes d, e, H, I, m, M, S, U, V, w, W, and y. The "E" flag causes the use of Era-specific values with the codes c, C, x, X, y, and Y. If the alternative format does not exist in the current locale, the "O" and "E" flags are ignored. On some UNIX systems, use of either flag causes the code to be unrecognized and to be output literally in the result.

33. **WORKSPACE-SIZE.** This keyword allows the specification of the amount of workspace area that the compiler will allocate for its internal tables. Control of this keyword will allow larger programs to be compiled, or memory to be conserved when compiling small programs on a system with limited memory. This keyword's value is a decimal number that reserves memory in increments of 1024 (1 KB) bytes; for example, a value of 100 would reserve 102400 bytes. The minimum value for this keyword is 32; the maximum value is 16384. The default value is 1024 kilobytes.

    This keyword may be overridden using the compiler W Option (see page 143). See the description of the W Option for information on appropriate values for WORKSPACE-SIZE. The compiler listing includes information on the amount of memory used for a compilation (see the line that starts "Maximum compilation memory …" in the summary listing, as described on page 163).

# DEFINE-DEVICE Configuration Record

The DEFINE-DEVICE configuration record is used to associate a physical device or process to a value for an RM/COBOL file access name. The primary use of DEFINE-DEVICE records is to define printer devices, but they are also used to define tape devices. For additional information about devices, see Device Support (on page 226). Since device support is highly system dependent, the DEFINE-DEVICE records for UNIX and Windows are described separately in the following sections.

When an RM/COBOL file access name, after it has been modified by any applicable environment variable replacement, matches either of the values specified for the DEVICE or PATH keywords, then that DEFINE-DEVICE record controls the input-output operations for the associated COBOL file. When it is the DEVICE value that is matched by the file access name, the file access name is effectively mapped to the PATH value. When it is the PATH value that is matched by the file access name, the file access name is unchanged, but the other options of that DEFINE-DEVICE record are applied.

The presence of a single DEFINE-DEVICE record overrides the automatic internal configuration for the PRINTER and TAPE support. For more information, see Default Configuration Files (on page 344), and the discussions of printer support (on page 227) and tape support (on page 227).

## For UNIX

The keywords DEVICE and PATH must be present in each DEFINE-DEVICE record and may be followed by one or more keywords. No keyword may be repeated in a single DEFINE-DEVICE record. If the keyword is allowed to have a value, it is followed by an equal sign (=) and the value.

The possible keywords supported under UNIX are as follows:

1. **DEVICE.** This keyword specifies the RM/COBOL file access name value that will be associated with the operating system device. If the value of the DEVICE keyword is the same as the value specified as the file access name, the value supplied in the PATH keyword will be used as the actual pathname. There is no default value; the keyword is required.

2. **PATH.** This keyword specifies the pathname to be used as the pathname or pipe for the device. This value may be enclosed in quotation marks to allow spaces to be included in a pipe (see the next item). If it is necessary to place a quotation mark in the pipe, use a pair of consecutive quotation marks for each quote within the string. There is no default value; the keyword is required.

3. **PIPE.** This keyword determines whether the value of the PATH keyword is a process to be spawned. If the value is set to YES, the runtime system will start another program to simulate a device sending, for an input file, or receiving, for an output file, the record by creating a pipe and forking a child shell process using the value of the PATH keyword as the command string passed to the shell. Thus, the value of the PATH keyword specifies the program or programs to start. If you use quotation marks to enclose the command, any shell command may be given along with any options. If the value is set to NO, the runtime system will use the PATH value as the actual pathname. If the PATH value contains a leading pipe character ('|'), the remainder of the path will be treated as a pipe destination regardless of the setting of the PIPE keyword. The information on page 27 regarding a file access name that contains an initial pipe character is also applicable to pipes created with the YES setting of the PIPE keyword. Setting the PIPE keyword to YES or supplying a PATH value with a leading pipe character is mutually exclusive with the TAPE=YES keyword setting. The default value for this keyword is NO.

   Two or more files open at the same time in the same run unit will share pipes created with a DEFINE-DEVICE configuration record when those files have a file access name that is resolved through the same DEFINE-DEVICE configuration record. In contrast, pipes created using the pipe character in the file access name and not resolved through a DEFINE-DEVICE configuration record will not be shared.

4. **TAPE.** This keyword determines whether the value of the PATH keyword specifies a tape device. If the value is set to YES, the path is assumed to represent a tape device. If the value is set to NO, a non-tape device is assumed. Tape devices are read or written with blocks of 512 characters unless the COBOL program specifies a different nonzero block size, in which case the specified block size is used. Setting the TAPE keyword to YES is mutually exclusive with the PIPE=YES keyword setting. The default value for this keyword is NO.

## For Windows

The keywords DEVICE and PATH must be present in each DEFINE-DEVICE record and may be followed by one or more keywords. No keyword may be repeated in a single DEFINE-DEVICE record. If the keyword is allowed to have a value, it is followed by an equal sign (=) and the value.

The possible keywords supported under Windows are as follows:

1. **DEVICE.** This keyword specifies the RM/COBOL file access name value that will be associated with the operating system device. If the value of the DEVICE keyword is the same as the value specified as the file access name, the value supplied in the PATH keyword will be used as the actual pathname. There is no default value; the keyword is required.

2. **ESCAPE-SEQUENCES.** This keyword determines whether the Windows printer described by the value of the PATH keyword allows embedded RM/COBOL-specific escape sequences (on page 525). If the value is set to YES, the RM/COBOL runtime system will recognize the sequences. If the

value is set to NO, the runtime system will not recognize those escape sequences. Setting the ESCAPE-SEQUENCES keyword to YES is mutually exclusive with any of the PIPE=YES, TAPE=YES, and RAW=YES keyword settings. The default value for this keyword is NO.

3. **PATH.** This keyword specifies the pathname to be used as the pathname for the device. This value may be enclosed in quotation marks to allow spaces to be included in the printer name, as described in Windows Printers (on page 306). If it is necessary to place a quotation mark in the printer name, use a pair of consecutive quotation marks for each quote within the string. There is no default value; the keyword is required.

4. **RAW.** This keyword determines whether the Windows printer described by the value of the PATH keyword is opened in RAW mode. If the value is set to YES, the runtime system will open the printer in RAW mode. This allows certain networked printers on Windows NT-class servers to respond to embedded escape sequences. See the P$SetRawMode (on page 491) subprogram for a more complete description of RAW mode. Most P$ subprograms are not available if RAW mode is used. If the value is set to NO, the runtime system will treat the printer as a normal Windows printer. Setting the RAW keyword to YES is mutually exclusive with any of the PIPE=YES, TAPE=YES, and ESCAPE-SEQUENCES=YES keyword settings. The default value for this keyword is NO.

5. **TAPE.** This keyword determines whether the value of the PATH keyword specifies a tape device. If the value is set to YES, the path is assumed to represent a tape device. If the value is set to NO, a non-tape device is assumed. Tape devices are read or written with blocks of 512 characters unless the COBOL program specifies a different nonzero block size, in which case the specified block size is used. Setting the TAPE keyword to YES is mutually exclusive with any of the RAW=YES, and ESCAPE-SEQUENCES=YES keyword settings. The default value for this keyword is NO.

## Windows Printers

The DEFINE-DEVICE record is used to associate a Windows printer device with an RM/COBOL file access name. It is not possible to bypass the Windows printer drivers by using the DEFINE-DEVICE record. If the Windows printer driver is used, it is not possible to send raw escape sequences to the printer. When using DEFINE-DEVICE to specify a Windows printer, both the DEVICE and PATH keywords are required.

The PATH keyword specifies the Windows printer device to use. The syntax is as follows:

```
PATH=[ [Device Name] [, [Port] [, [Font Name] [, [Size] ] ] ]
```

*Device Name* is the name of the printer.

The name “DEFAULT” can be used to indicate that the default Windows printer should be used. If a *Device Name* is specified, the *Port* is ignored.

*Port* specifies the port to which the printer is attached. *Port* must be defined on the Details tab for a printer shown in the Printers folder or COBOL I/O error 35 will be returned from the OPEN statement.

*Font Name* specifies the font to use.

*Size* specifies the size of the font to use. It is specified in points.

**Note** If neither a *Device Name* nor *Port* is specified, the default printer is used.

Examples of DEFINE-DEVICE records are as follows:

```
DEFINE-DEVICE DEVICE=PRINTER PATH=",LPT1"

DEFINE-DEVICE DEVICE=FOO PATH="HP LaserJet"
```

For compatibility with Windows, whenever a device name followed by the colon character is encountered in either the DEVICE or PATH keywords, it is treated as if the colon were not present. For example, the following configuration records:

```
DEFINE-DEVICE DEVICE=PRN: PATH=",LPT1"

DEFINE-DEVICE DEVICE=LPT2 PATH=",LPT2:"
```

would be treated as if they were:

```
DEFINE-DEVICE DEVICE=PRN PATH=",LPT1"

DEFINE-DEVICE DEVICE=LPT2 PATH=",LPT2"
```

To perform translation of "PRINTER*x*" names in the same way that the RM/COBOL for DOS runtime system does, the following default synonym table is used on Windows.

```
DEVICE=PRINTER   PATH="DEFAULT"
DEVICE=PRINTER?  PATH="DYNAMIC"
DEVICE=PRINTER1  PATH=",LPT1"
DEVICE=PRINTER2  PATH=",LPT2"
DEVICE=PRINTER3  PATH=",LPT3"
DEVICE=PRINTER4  PATH=",LPT4"
DEVICE=PRINTER5  PATH=",LPT5"
DEVICE=PRINTER6  PATH=",LPT6"
DEVICE=PRINTER7  PATH=",LPT7"
DEVICE=PRINTER8  PATH=",LPT8"
DEVICE=PRINTER9  PATH=",LPT9"
```

The "PRINTER?" synonym is provided to allow dynamic assignment of the Windows printer at printer open time. See Windows System Print Jobs (on page 65) for more information on the use of "PRINTER?".

# EXTENSION-NAMES Configuration Record

The EXTENSION-NAMES configuration record identifier is followed by one or more keywords indicating file usages. Each keyword is allowed to have a value, and is followed by an equal sign (=) and the value. More than one file usage keyword may be listed in one EXTENSION-NAMES record. The extension value must be one to three characters in length. The extension value may be specified as a single period to indicate that no extension is to be used for files of the file usage indicated by the keyword. In all other cases, the characters specified for the extension value must be in the set of characters that are valid for a filename extension under the current operating system.

The file usage keywords are the following:

1. **COPY.** This keyword specifies the extension to be used for files referenced by COPY statements in a COBOL source program. The default extension for copy files is **cbl**.
2. **LISTING.** This keyword specifies the extension to be used for COBOL listing files. The default extension for listing files is **lst**.
3. **OBJECT.** This keyword specifies the extension to be used for COBOL object program files. The default extension for object files is **cob**.
4. **SOURCE.** This keyword specifies the extension to be used for COBOL source program files. The default extension for source files is **cbl**.

# EXTERNAL-ACCESS-METHOD Configuration Record

The EXTERNAL-ACCESS-METHOD configuration record is used to identify external file access methods that should be applied. It can be repeated more than once to identify multiple access methods and the order in which they are to be accessed. UNIX supports the following external access methods: RMPLUSDB, RMINFOX, and USRMTACC. Windows supports RMBTRV32 and RMTCP32. Refer to Chapter 4: *System Considerations for Btrieve* (on page 111) for more information about RMBTRV32.

The EXTERNAL-ACCESS-METHOD record identifier is followed by one or more keywords. Each keyword is followed by an equal sign (=) and the value to be assigned to that keyword. The possible keywords are as follows:

1. **CREATE-FILES.** This keyword controls whether the external access method will be called to create new files. If the value is set to YES, the external access method will be allowed to create files. If the value is set to NO, and the file does not exist, the external access method will not be called to create it. The default value for this keyword is YES.
2. **NAME.** This keyword is used to identify the name of the external access method. It is required and has no default value. The external access method names currently identified are RMPLUSDB, RMINFOX, and USRMTACC for UNIX, and RMBTRV32 and RMTCP32 for Windows.
3. **OPTIONS.** This keyword is used to pass options to the external access method interface. The options must be enclosed in quotation marks. The possible

values depend on the external access method that is specified in the NAME keyword. If this keyword is not specified, no options will be passed to the external access method interface.

# INTERNATIONALIZATION Configuration Record

The INTERNATIONALIZATION configuration record is used to specify information regarding internationalization, including support for the euro symbol (€).

**Note 1** On Windows, the keywords for euro symbol support need to be specified only if the default behavior is not acceptable. The default behavior described for these keywords should provide acceptable euro symbol support on Windows for almost all users. Additional information is provided in Euro Support Considerations Under Windows (on page 310).

**Note 2** On UNIX, euro support is available through normal terminal configuration. RM/COBOL is ready to support the euro provided the UNIX operating system supports the euro. You may need to make changes to your UNIX operating system, such as installing a character set that supports the euro, before your system is euro-ready.

The INTERNATIONALIZATION record identifier is followed by one or more keywords. If the keyword is allowed to have a value, it is followed by an equal sign (=) and the value. The possible keywords are as follows:

1. **EURO-CODEPOINT-ANSI.** This keyword specifies the code point value to be used for the euro symbol in the Windows ANSI codepage (the codepage used to display and print ANSI encoded fonts). The value must be a number between decimal 0 and 255 (hexadecimal 0x00 and 0xff). When specified, the runtime system will use the given value when euro support is enabled and characters are being converted between OEM and ANSI. There is no default value, but if euro support is enabled and this keyword is not specified, then the runtime system will query Windows for the correct code point to use. If the ANSI codepage does not include a euro symbol, the runtime system will use code point 128 (0x80). This keyword is ignored if euro support is disabled by use of the keyword EURO-SUPPORT-ENABLE=NO.

   **Note** This keyword is supported only under Windows.

2. **EURO-CODEPOINT-OEM.** This keyword specifies the code point value to be used for the euro symbol in the Windows OEM codepage (the codepage used for data in memory). The value must be a number between decimal 0 and 255 (hexadecimal 0x00 and 0xff). When specified, the runtime system will use the given value when euro support is enabled and characters are being converted between OEM and ANSI. There is no default value, but if euro support is enabled and this keyword is not specified, then the runtime system will query Windows for the correct code point to use. If the OEM codepage does not include a euro symbol (as is true for most OEM codepages other than 858), the runtime system will use code point 213 (0xD5). This keyword is ignored if euro support is disabled by use of the keyword EURO-SUPPORT-ENABLE=NO.

   **Note** This keyword is supported only under Windows.

3. **EURO-SUPPORT-ENABLE.** This keyword determines whether the runtime system on Windows maps the euro symbol from OEM to ANSI when rendering characters for screen display or printing and maps the euro symbol from ANSI to OEM when accepting characters from the keyboard. If the value is set to YES, then the runtime system maps the euro symbol. Runtimes prior to version 7.5 did not map the euro symbol, so if this new behavior causes a problem, then the value can be set to NO, in which case the old behavior is restored. The default value for this keyword is YES.

   Some Windows systems, such as Windows 2000, provide native support for the euro symbol in OEM codepage 858. For systems with native euro support, specifying EURO-SUPPORT-ENABLE=NO will not disable euro support, provided that both the OEM and the ANSI codepages in use contain a euro symbol.

   **Note** This keyword is supported only under Windows.

## Euro Support Considerations Under Windows

In order to use the euro symbol, the font used must contain the euro character symbol (€). To determine whether the font contains the euro symbol, open the Character Map in the Windows System Tools utility. From the Character Map dialog box, you can display the maps of different fonts. The euro symbol will usually be located in position 128. Some fonts that contain the euro symbol are Courier New, Times New Roman, Arial, and Tahoma. Note that you may need to obtain new copies of these fonts from Microsoft at the following location:

- http://www.microsoft.com/typography/fontpack/default.htm

If the euro symbol displays correctly but does not print correctly, it is likely that the internal printer font was used instead of the display font. The printer font may not contain the euro symbol. Some printers offer a Font Substitutions Table from the Printer Properties dialog box that allows you to enable printing of the euro symbol by downloading the printer font as "Outline." Not all printers have this capability, however, and you should refer to your printer documentation for more details. For additional information on the euro symbol support in Windows, see the website at the following location:

- http://www.microsoft.com/windows/euro.asp

To be able to enter a euro symbol from the keyboard when the euro symbol is in the range 0 to 31 or 127 to 255, the DATA-CHARACTERS keyword (see page 328) of the TERM-ATTR configuration record must be specified to extend the range of text characters from the default range of 32 to 126.

When entering characters from the keyboard using the Windows ALT+<*number*> technique, the <*number*> should contain a leading '0' character (indicating an ANSI character) to enter a euro symbol that is in the range 0128 to 0255 of the ANSI codepage. When the <*number*> does not include a leading '0' character (indicating an OEM character) and the OEM codepage does not include a euro symbol, the Windows keyboard driver may convert the character to an ANSI character other than a euro symbol before the runtime system has a chance to map the character according to the configured euro support.

# PRINT-ATTR Configuration Record

The PRINT-ATTR configuration record is used to describe the characteristics of the printer to which printer files are assigned or on which printer files will eventually be printed. A printer file is a line sequential file that has any or all of the following RM/COBOL source program features:

- ASSIGN TO PRINT or ASSIGN TO PRINTER phrase in the file control entry for the file
- LINAGE phrase in the file description entry for the file
- ADVANCING phrase in a WRITE statement for the file

The compiler listing is a printer file.

The PRINT-ATTR record identifier is followed by one or more keywords. If the keyword is allowed to have a value, it is followed by an equal sign (=) and the value. The possible keywords are as follows:

1. **AUTO-LINE-FEED.** This keyword determines whether a line feed is needed after a carriage return to cause a single line advance of the carriage. If the value is set to YES, a single line advance is automatic so no line feed character will be written. If the value is set to NO, a single line advance is not automatic, thus requiring a line feed character. The default value for this keyword is NO.
2. **COLUMNS.** This keyword determines the number of columns across a line, represented as a decimal number in the range 1 through 65280. All records written to a printer file will be truncated to this value. The default value for this keyword is not to truncate.
3. **FORM-FEED-AVAILABLE.** This keyword indicates whether the printer supports the form feed character, FF, to slew to the top of a new page. If the value is set to NO, top-of-page will be reached by issuing an appropriate number of line feeds; the LINES keyword should be used to describe the page size. If the value is set to YES, top-of-page will be reached by writing a form feed character. The default value for this keyword is YES.

   **Note** This option only determines the method used by the runtime I-O system to position the file to a new physical page. The runtime I-O system would normally print a form feed character to accomplish a physical page break. If the user's printer does not support advancing to the next physical page when a form feed character is printed, the value of this option can be set to NO to tell the runtime I-O system not to use a form feed character for this purpose. Regardless of the setting of this option, files described with the LINAGE clause do not normally use physical page breaks because the LINAGE clause describes logical rather than physical pages. Files described with the LINAGE clause will be affected by this option only if either of the PRINT-ATTR configuration record keywords, LINES or LINAGE-PAGES-PER-PHYSICAL-PAGE, is set to a nonzero value.
4. **LINAGE-INITIAL-FORM-POSITION.** This keyword determines the assumed initial position of the form in the printer for a file described with the LINAGE clause (sequential file description entry) on page 224. If set to TOP-OF-FORM, the form is assumed to be positioned at the top of the page (that is, on the first line of the top margin). In this case, the runtime writes the first line after advancing over the top margin of the first page so as to reach the first line of the logical page body. The value TOP-OF-FORM would normally be used for page printers that do not use continuous forms. If set to PAGE-BODY-

LINE-1, the form is assumed to be positioned at line one of the page body and the runtime ignores the top margin specified for the first logical page. The value PAGE-BODY-LINE-1 would normally be used with line printers that use continuous forms that have been positioned by the operator to print the first line of the logical page body. The default value for this keyword is PAGE-BODY-LINE-1.

5. **LINAGE-PAGES-PER-PHYSICAL-PAGE.** This keyword determines whether physical page breaks are generated for files described with the LINAGE clause (see page 224). When this keyword is set to the value 0, the set of logical pages is printed contiguously with no additional spacing provided between logical pages, except as explained in the note regarding the PRINT-ATTR configuration record keyword LINES. Form feed characters, in particular, are not normally used between pages. When this keyword is set to a value from 1 to 255, the value indicates the number of logical pages that fit on a physical page. Each time that many pages have been printed, a physical page break will be generated. For example, a value of 1 causes a physical page break (see the note below for additional details on physical page breaks) between each logical page. As another example, if there are three logical pages (for example, checks) per physical page, a value of 3 could be used to cause a physical page break between each set of three logical pages. A nonzero value is typically useful for page printers when the logical page, or a set of logical pages, does not fill a physical page. A zero value is typically useful for line printers using continuous forms. The default value for this keyword is 0.

   **Note** This option instructs the runtime I-O system to insert a physical page break between certain logical pages. The physical page break is normally a form feed character. However, the PRINT-ATTR configuration record keyword settings FORM-FEED-AVAILABLE=NO and LINES=*n* may be used together in those cases where a form feed character is either not available or not desirable. In this case, a physical page break is accomplished by printing the number of line feed characters necessary to ensure *n* lines per physical page.

6. **LINES.** This keyword determines the number of lines on a page, represented as a decimal number in the range 1 through 65535. Use this keyword when FORM-FEED-AVAILABLE=NO (see page 311). This keyword may be used with FORM-FEED-AVAILABLE=YES to cause form feed characters to be placed in the file after the specified number of lines have been written. The default value for this keyword is not to have page size processing.

   This keyword also determines the number of lines on a page of a compilation listing. If not specified, the RM/COBOL compiler assumes 66 lines per page.

   **Note** This option, when set to a nonzero value, will cause files described with the LINAGE clause to advance to a new physical page whenever that number of lines have been printed. This may result in unintended additional spacing between or within logical pages, depending on the relationship between the value specified for LINES and the size(s) of logical pages. For page printers, setting LINES to the size of the logical page may have the desired effect of ejecting pages from the printer when a logical page is complete. However, the PRINT-ATTR configuration record keyword LINAGE-PAGES-PER-PHYSICAL-PAGE, which only affects files described with the LINAGE clause, is better suited to this purpose.

7. **TOP-OF-FORM-AT-CLOSE.** This keyword determines whether the printer file is positioned to top-of-form when closed. If the value is set to YES, the printer file will be positioned to top-of-form with form feed or line feed characters. If the value is set to NO, no additional control characters will be written when the printer file is closed. The default value for this keyword is NO.

8. **WRAP-COLUMN.** This keyword determines the column number after which automatic line wrap-around occurs, represented as a decimal number in the range 1 through 65535. The default value for this keyword is to assume that the printer does not automatically wrap long lines.

9. **WRAP-MODE.** This keyword specifies whether automatic line wrapping occurs when the WRAP-COLUMN character is written. If the value is set to EXACT, wrapping occurs when the WRAP-COLUMN character is written. If the value is set to NEXT, wrapping occurs when the character following the WRAP-COLUMN character is written. The default value for this keyword is NEXT.

# RUN-ATTR Configuration Record

The RUN-ATTR configuration record identifier is followed by one or more keywords. If the keyword is allowed to have a value, it is followed by an equal sign (=) and the value. The following descriptions include the default values that are used if the keyword is not modified by a RUN-ATTR record. The possible keywords are as follows:

1. **ACCEPT-FIELD-FROM-SCREEN.** This keyword controls the behavior of ACCEPT statements that do not specify either the PROMPT or the UPDATE phrase. If the value is set to YES, the field is initialized with the current contents of the field on the display. If the value is set to NO, the field is initialized to all blanks. The default value for this keyword is NO. In either case, the contents of the field as displayed are unchanged.

   Only fields that are output by the RM/COBOL runtime system can be reliably retrieved. The contents of a field that appeared on the display prior to the invocation of the runtime are considered undefined.

2. **ACCEPT-INTENSITY.** This keyword determines the default intensity level used within ACCEPT statements. If the value is set to HIGH, high intensity is used. If the value is set to LOW, low intensity is used. The default value for this keyword is HIGH.

3. **ACCEPT-PROMPT-CHAR.** This keyword enables the ACCEPT statements default prompt character to be overridden with the character corresponding to the value of the keyword. Changing the default prompt character does not affect ACCEPT statements that do not use the PROMPT phrase. The value of this keyword must be a single-character string or a number from 0 to 255. The default value for this keyword is "_" (95 or 0x5F).

4. **BEEP.** This keyword determines whether the runtime system should override the beeps (BEEP) that are coded in ACCEPT and DISPLAY statements. If the value is set to YES, the beeps that are coded in the statements (including the default beeps on ACCEPT statements) cause the terminal to beep. If the value is set to FORCED-ACCEPT, all Format 3 ACCEPT statements cause the terminal to beep; all other ACCEPT and DISPLAY statements behave as if the value were set to YES. If the value is set to NO, the beeps that are coded in the statements are ignored. The default value for this keyword is YES.

5. **BLINK.** This keyword determines whether the runtime system should override blinking (BLINK) coded in the ACCEPT and DISPLAY statements. If the value is set to YES, blinking is used as directed by the statements. If the value is set to NO, blinking is not used. The default value for this keyword is YES.

   **Note** The blinking attribute is not available under Windows.

6. **DISPLAY-INTENSITY.** This keyword determines the default intensity level used within DISPLAY statements. If the value is set to HIGH, high intensity is used. If the value is set to LOW, low intensity is used. The default value for this keyword is HIGH.

7. **EDIT-COMMA.** This keyword enables the comma edit character (thousands separator) to be overridden with the character corresponding to the value of the keyword. This configuration option is not affected by the presence of the DECIMAL-POINT IS COMMA clause in the source program, except for the default value. The value of this keyword must be a single-character string or a number from 0 to 255. The default value for this keyword is "," (44 or 0x2c), or, if DECIMAL-POINT IS COMMA clause is specified in the source program, the default is "." (46 or 0x2e).

8. **EDIT-CURRENCY-SYMBOL.** This keyword enables the currency symbol edit character (cs) to be overridden with the character corresponding to the value of the keyword. This keyword has effect only when a CURRENCY SIGN IS clause is present in the Configuration Section of the program. This configuration option does not affect the currency sign ($). See the EDIT-DOLLAR keyword for overriding the currency sign value. The value must be a single-character string or a number from 0 to 255. The default value for this keyword is "$" (36 or 0x24).

9. **EDIT-DECIMAL.** This keyword enables the decimal point edit character to be overridden with the character corresponding to the value of the keyword. This configuration option is not affected by the presence of the DECIMAL-POINT IS COMMA clause in the source program, except for the default value. The value of this keyword must be a single-character string or a number from 0 to 255. The default value for this keyword is "." (46 or 0x2e), or, if the DECIMAL-POINT IS COMMA clause is specified in the source program, the default is "," (44 or 0x2c).

10. **EDIT-DOLLAR.** This keyword enables the currency sign ($) to be overridden with the character corresponding to the value of the keyword. This configuration option is not affected by the presence of the CURRENCY SIGN clause in the source program, unless the program specifies CURRENCY SIGN IS "$". In that case, the "$" in that program is the currency symbol and the EDIT-DOLLAR keyword has no effect. (See the EDIT-CURRENCY-SYMBOL keyword for overriding the currency symbol value.) The value must be a single-character string or a number from 0 to 255. The default value for this keyword is "$" (36 or 0x24).

11. **ERROR-MESSAGE-DESTINATION.** This keyword determines the destination to which the runtime system should direct error messages, Interactive Debugger input and output, STOP literal messages and STOP RUN messages. ACCEPT . . . FROM CONSOLE and DISPLAY . . . UPON CONSOLE are not affected by this keyword. The two possible values are STANDARD-ERROR and STANDARD-INPUT-OUTPUT. If the value is set to STANDARD-ERROR, these messages are directed to the standard error device. If the value is set to STANDARD-INPUT-OUTPUT, these messages are written to standard output and, in the cases of Debug input and STOP *literal* message operator responses, the responses are read from standard input. The value STANDARD-

ERROR does not allow redirection of the messages; STANDARD-INPUT-OUTPUT does allow redirection. The default value for this keyword is STANDARD-ERROR. For information on the standard input, output, and error devices, see Redirection of Input and Output (on page 44).

12. **REVERSE.** This keyword determines whether the runtime system should override reverse video (REVERSE) coded in the ACCEPT and DISPLAY statements. If the value is set to YES, reverse video is used as directed by the statements. If the value is set to NO, reverse video is not used. The default value for this keyword is YES.

13. **SCROLL-SCREEN-AT-TERMINATION.** This keyword determines whether the runtime system should scroll the screen by one line before returning to the shell. If the value is set to NO, the screen is not scrolled. If the value is set to YES, the screen is scrolled only if the COBOL program performed any screen I/O, such as ACCEPT or DISPLAY statements. The default value for this keyword is YES.

14. **STRIP-LIKE-PATTERN-TRAILING-SPACES.** This keyword determines whether the runtime system should strip trailing spaces from a pattern specified as an alphanumeric variable in a LIKE condition. If the value is set to NO, trailing spaces are not stripped; that is, they are considered significant in the pattern. If the value is set to YES, all trailing spaces are stripped from the pattern value. The default value for this keyword is YES.

    **Note** To match runtime behavior prior to version 9, the value NO must be configured.)

    When trailing spaces are stripped (STRIP-LIKE-PATTERN-TRAILING-SPACES=YES) and the pattern needs to specify significant trailing spaces, it can do so by specifying a space followed by a quantifier. For example, “Hello {1}” requires exactly one trailing space for a match and “Hello +” requires one or more trailing spaces for a match.

    This configuration keyword does not affect literal pattern values, where trailing spaces are always considered significant, and it also does not affect a pattern specified as the first argument to the C$CompilePattern library subprogram, where trailing space stripping is controlled by the second argument.

15. **TAB.** This keyword controls the default behavior of ACCEPT statements that do not have a TAB phrase. If the value of the keyword is set to YES, ACCEPT statements are executed as if the TAB phrase were present. If value is set to NO, ACCEPT statements are executed as if the NO TAB keyword were present in the CONTROL phrase. The default value for this keyword is NO.

16. **UNDERLINE.** This keyword determines whether the runtime system should override underlining (UNDERLINE) coded in the Screen Section or the CONTROL phrase of ACCEPT and DISPLAY statements. If the value is set to YES, underlining is used as directed by the statements. If the value is set to NO, underlining is not used. The default value for this keyword is YES.

For complete descriptions of the ACCEPT and DISPLAY statements, see Chapter 6: *Procedure Division Statements* of the *RM/COBOL Language Reference Manual*.

# RUN-FILES-ATTR Configuration Record

The RUN-FILES-ATTR configuration record identifier is followed by one or more keywords. If the keyword is allowed to have a value, it is followed by an equal sign (=) and the value. Some of the following keywords are allowed on both the RUN-FILES-ATTR record and on the RUN-INDEX-FILES, RUN-REL-FILES, and RUN-SEQ-FILES records. For these keywords, specifying a value on a RUN-FILES-ATTR record is equivalent to specifying the same value on RUN-INDEX-FILES, RUN-REL-FILES, and RUN-SEQ-FILES records. If a keyword exists on both a RUN-FILES-ATTR and on a RUN-*xxx*-FILES record, then the last one in the configuration file will be used. The possible keywords are as follows:

1. **ALLOW-EXTENDED-CHARS-IN-FILENAMES**. This keyword controls whether extended characters in the range 128 through 255 (0x80 through 0xFF) are allowed in filenames. If this keyword is set to YES, these characters are allowed in filenames. If this keyword is set NO, these characters are excluded from filenames. If this keyword is set to ANSI, extended characters are allowed in filenames and the Windows function, SetFileApisToANSI, is called to ensure that Windows interprets filenames in all Windows filename functions as containing characters represented with code points from the ANSI codepage. If this keyword is set to OEM, extended characters are allowed in filenames and the Windows function, SetFileApisToOEM, is called to ensure that Windows interprets filenames in all Windows filename functions as containing characters represented with code points from the OEM codepage. The default value for this keyword is NO.

   **Note** This keyword is supported only under Windows. See the topic Character Set Considerations for Windows (on page 97) for additional information on the ANSI and OEM codepages on Windows.

2. **BLOCK-SIZE.** This keyword determines the default disk block size for all file organizations except program files, represented as a decimal number. It can be overridden by a configuration record for a particular organization. A program can override this with the BLOCK CONTAINS clause in the file description entry. The maximum value for this keyword is 65489; the minimum value is 256. Under UNIX, the default value is BUFSIZ, taken from the C include file <**stdio.h**>. Under Windows, the default value is the disk sector size. Program files always use 512-byte blocks.

3. **BUFFER-POOL-SIZE.** This keyword determines the amount of memory that should be allocated in the disk buffer pool, represented as a decimal number. The maximum value of this keyword is 10000000 (ten million); the minimum value is 1. A value of 1 will cause the minimum disk buffer pool to be allocated. The default value is 20480 for the compiler and 256000 for the runtime system.

4. **DEFAULT-USE-PROCEDURE.** This keyword can be used to set the runtime action to be taken when an I/O error occurs on a file for which there is no applicable USE procedure. If the value is set to TERMINATE, the runtime terminates with an appropriate error message. If the value is set to CONTINUE, the runtime continues the program execution at the next executable statement following the I/O statement that caused the error as if an empty USE procedure had been executed, that is, a USE procedure that did nothing but exit. The default value for this keyword is TERMINATE. For purposes of determining whether to wait on a record locked condition or return the record locked I-O status value, specifying the value CONTINUE implies that there is an applicable USE procedure.

**WARNING** Setting DEFAULT-USE-PROCEDURE=CONTINUE causes program continuation even after serious errors. This can cause the program to behave in difficult to understand ways when unexpected errors occur. When using this configuration setting, it is the responsibility of the program to check for errors after each I/O operation for which an applicable USE procedure is not provided.

5. **DISABLE-LOCAL-ACCESS-METHOD.** This keyword can be used to prevent the use of the local access method within the file manager based on one or more specified conditions:

   - If the value is set to UNQUALIFIED-NAMES, then the local access method will not be called to create a file with an unqualified name. An unqualified name is one with no path part, that is, just a simple file name, like FILE.EXT. Without this value, files with unqualified names will be created in the current working directory. Use this value when you want to create new files in the first directory in RUNPATH rather than in the current working directory. This value has no effect on finding existing files, only on creating new files.

   - If the value is set to UNC-NAMES, then the local access method will not be called for any UNC-style name. UNC-style names begin with either \\ (double backslash) or // (double forward slash). UNC-style names then contain a server name, a directory path, and a file name (for example, //IXSERVER/DATA/FILE.EXT). This value may be useful when all files accessed via UNC-style names reside on an RM/InfoExpress server. Typically, the RM/InfoExpress external access method is configured for the runtime and directory names such as //IXSERVER/DATA are placed in the RUNPATH environment variable.

   The default value for this keyword is not to disable creation of files by the local access method for any of the possible conditions.

6. **ENABLE-OLD-DOS-FILENAME-HANDLING.** This keyword controls whether filenames are processed in the way they were handled in earlier DOS runtimes. If this keyword is set to YES, filenames are converted to uppercase, all spaces are eliminated, and each node name (characters between separators) is truncated to an 8.3 format. For example, the filename “c:\long directory.name\long filename.extension” would become “C:\LONGDIRE.NAM\LONGFILE.EXT”. If this keyword is set to NO, filenames remain in mixed case, spaces are allowed, and long node names are allowed (that is, no truncation). The default value for this keyword is NO.

   **Note** This keyword is supported only under Windows.

7. **EXPANDED-PATH-SEARCH.** This keyword controls when the directory search sequence is used. If a filename is specified with no directory path, the directory search sequence will always be used. If the filename begins with a forward slash (/), a backward slash (\), or a tilde (~), the directory search sequence will not be used. If the filename contains a directory path that does not begin with a slash or tilde, the directory search sequence will be used only if this keyword is set to YES. In this case, the entire name, including the directory path, will be appended to each entry in the directory search sequence. The default value for this keyword is NO.

8. **FATAL-RECORD-LOCK-TIMEOUT.** This keyword affects record locking. For file descriptors whose file control entry has no FILE STATUS clause and file descriptors that have no USE declarative procedure defined, this keyword determines how many seconds to wait when attempting to lock a record that is locked by another run unit before returning a fatal error to the calling program.

A value of 0, the default value for this keyword, indicates an infinite wait. The minimum value is 0; the maximum value is 65,535 seconds.

If the record is locked using a different file descriptor in the same run unit, the error will always be returned immediately to prevent a deadlock situation.

9. **FILE-LOCK-LIMIT.** This keyword determines the limit for the location to apply locks to a file. This number can vary depending on the system on which the file resides. The lock limit applies to all file organizations. For record and file locks to perform correctly, all run units opening a file must use the same file lock limit. The lock limit also limits the actual amount of data that can be stored in a file. This data limit is dependent on the file organization. Sequential and relative files can store slightly less than half this number. For indexed files, the data size limit varies with the block size: a block size of 1024 will allow eighty percent of this number for data storage; a block size of 4096 will allow over ninety percent. The maximum value for this keyword is 07FFFFFFEh (approximately 2 gigabytes); the minimum value is 1. The default value is 07FFFFFFEh.

   For information on superseding the limit for the location to apply locks to a file that will be accessed as a large file, see the description of the LARGE-FILE-LOCK-LIMIT keyword later in this section.

10. **FILE-PROCESS-COUNT.** This keyword determines the maximum number of run units that can have a file open at the same time. It applies to all file organizations. For record and file locks to perform correctly, all run units opening a file must use the same file process count. The maximum value for this keyword is 4096; the minimum value is 8. The default value is 1024.

11. **FORCE-USER-MODE.** This keyword determines whether the runtime system should assume a single-user or a shared environment. If the value is set to SINGLE, local files are locked and treated as if they were unshared. If the value is set to MULTI, local files are treated as if they were shared, and locks are applied as specified in OPEN statements or the applicable LOCK MODE clause. This keyword does not affect remote files. Remote files are always assumed to be in a shared environment. The default value for this keyword is MULTI.

    **Note** This keyword is supported only under Windows.

12. **KEEP-FLOPPY-OPEN.** This keyword determines whether the runtime system will open and close program files that reside on a floppy drive every time it accesses them. If the value is set to NO, the RM/COBOL file management system closes these files as often as possible in order to prevent floppy corruption problems that can occur when swapping diskettes during a program execution. If the value is set to YES, the RM/COBOL file management system will not attempt to close floppy-based program files after every access. The default value for this keyword is NO.

    **Note** This keyword is supported only under Windows.

13. **LARGE-FILE-LOCK-LIMIT.** This keyword determines the limit for the location to apply locks to a file that will be accessed as a large file, superseding the limit specified by the FILE-LOCK-LIMIT keyword. For record and file locks to perform correctly, all run units opening a particular file must use the same file lock limit. The lock limit also limits the actual amount of data that can be stored in a file. (See the description of the FILE-LOCK-LIMIT keyword, described earlier in this section, for details on this relationship.) The value assigned to this keyword is specified in gigabytes (GB). The maximum value for this keyword is 1048576, which equates to 1 petabyte ($2^{50}$). The minimum value is 1. The default value is 64.

For an explanation of how to indicate that a relative file or a sequential file will be accessed as a large file, see the description of the USE-LARGE-FILE-LOCK-LIMIT keyword of the RUN-REL-FILES configuration record and RUN-SEQ-FILES configuration record on pages 325 and 326, respectively. See also the descriptions of File Version Level 3 (on page 244) and the DEFAULT-FILE-VERSION-NUMBER keyword (on page 320) of the RUN-INDEX-FILES configuration record for information on using the LARGE-FILE-LOCK-LIMIT with indexed files.

14. **RESOLVE-LEADING-NAME.** This keyword controls when the first directory name specified in a file access name is resolved from the environment. The first directory name is defined as a name that is not preceded by a slash character. Under Windows, the slashes that may appear in a volume name are ignored. If the name is not found in the environment, no substitution will occur, and the name will remain as specified (after the possible removal of the leading character). There are several possible values for this keyword, including ALWAYS, NEVER, or one of these seven leading characters: !, @, #, $, %, ^, or &.

    If RESOLVE-LEADING-NAME is set to ALWAYS, and the directory name exists in the environment, the value of the environment variable will replace the name. If RESOLVE-LEADING-NAME is set to one of the seven leading characters, the directory name begins with that character, and the directory name without that character exists in the environment, then the value of that environment variable will replace the name. If the value is set to NEVER, then the leading directory name will never be replaced. The default value for this keyword is NEVER.

    In the special case that the file access name does not contain any directory specifiers, substitution will always be attempted.

    For example, if the environment contains a variable name DIR with the value MYDIR, and does not contain the variable D1, the following substitutions would occur.

| **Name Specified** | **RESOLVE-LEADING-NAME Value** | | |
|---|---|---|---|
| | **ALWAYS** | **NEVER** | **@** |
| DIR/FILE | MYDIR/FILE | DIR/FILE | DIR/FILE |
| @DIR/FILE | @DIR/FILE | @DIR/FILE | MYDIR/FILE |
| @D1/FILE | @D1/FILE | @D1/FILE | D1/FILE |
| DIR | MYDIR | MYDIR | MYDIR |
| @DIR | @DIR | @DIR | MYDIR |

15. **RESOLVE-SUBSEQUENT-NAMES.** This keyword controls when directory names or the filename specified in a file access name are resolved from the environment. It does not apply to the leading name (see RESOLVE-LEADING-NAME keyword, described previously). If the name is not found in the environment, no substitution will occur, and the name will remain as specified (after the possible removal of the leading character). There are several possible values for this keyword, including ALWAYS, NEVER, or one of these seven leading characters: !, @, #, $, %, ^, or &.

    If RESOLVE-SUBSEQUENT-NAMES is set to ALWAYS, and the directory or filename exists in the environment, the value of the environment variable will replace the name. If RESOLVE-SUBSEQUENT-NAMES is set to one of the seven leading characters, the directory or filename begins with that character, and the directory or filename without that character exists in the environment, then the value of that environment variable will replace the name. If the value is

set to NEVER, the directory or filename will never be replaced. The default value for this keyword is NEVER.

For example, if the environment contains a variable name DIR with the value MYDIR, a variable name FILE with the value MYFILE, and does not contain the variable D1, the following substitutions would occur.

| Name Specified | RESOLVE-SUBSEQUENT-NAMES Value | | |
|---|---|---|---|
| | ALWAYS | NEVER | @ |
| DIR/FILE | DIR/MYFILE | DIR/FILE | DIR/FILE |
| /DIR/@FILE | /MYDIR/@FILE | /DIR/@FILE | /DIR/MYFILE |
| /@D1/FILE | /@D1/MYFILE | /@D1/FILE | /D1/FILE |

16. **USE-PROCEDURE-RECORD-LOCK-TIMEOUT.** This keyword affects record locking. For file descriptors whose file control entry has a FILE STATUS clause and for which a USE declarative procedure is defined, it determines how many seconds to wait when attempting to lock a record that is locked by another run unit before returning an error to the calling program. A value of 0, the default value for this keyword, indicates that the error should be returned immediately. The minimum value is 0; the maximum value is 65,535 seconds.

    If the record is locked using a different file descriptor in the same run unit, the error is always returned immediately to prevent a deadlock situation.

# RUN-INDEX-FILES Configuration Record

The RUN-INDEX-FILES configuration record identifier is followed by one or more keywords. If the keyword is allowed to have a value, it is followed by an equal sign (=) and the value. The possible keywords are as follows:

1. **ALLOCATION-INCREMENT.** This keyword determines the allocation increment of indexed files created by the runtime system. The value is the decimal number of blocks to be added to the file. The maximum value for this keyword is 9999; the minimum value is 1. The default value is 8.

2. **BLOCK-SIZE.** This keyword determines the default disk block size for indexed files, represented as a decimal number. A program can override this with the BLOCK CONTAINS clause in the file description entry (see the discussion of this clause on page 233 of this manual and in Chapter 4: *Data Division* of the *RM/COBOL Language Reference Manual*). The maximum value for this keyword is 65489; the minimum value is 256. Under UNIX, the default value is BUFSIZ, taken from the C include file <**stdio.h**>. Under Windows, the default value is the disk sector size. See also the description of the MINIMUM-BLOCK-SIZE and ROUND-TO-NICE-BLOCK-SIZE keywords (on page 321) of the RUN-INDEX-FILES configuration record for additional information on computing the actual block size for indexed files.

3. **DATA-COMPRESSION.** This keyword determines whether the indexed files created by the runtime system use data compression. If the value is set to YES, data compression is used. If the value is set to NO, data is stored in uncompressed form. The default value for this keyword is YES.

4. **DEFAULT-FILE-VERSION-NUMBER.** This keyword determines the default file version number for new files when an OPEN OUTPUT is performed. The version number for existing files or files predefined with the Define Indexed File (rmdefinx) utility (on page 594) will not be changed when an OPEN

OUTPUT is performed. Allowable values are 0, 2, 3, and 4. The default value is 4. For more information, see Indexed File Version Levels (on page 243).

5. **ENABLE-ATOMIC-IO.** This keyword determines whether the indexed files created by the runtime system use atomic I/O. If the value is set to YES, atomic I/O is enabled and if the default version number is less than 4, the version number is set to 4. If the value is set to NO, atomic I/O is disabled. The default value for this keyword is NO.

6. **FORCE-CLOSED.** This keyword determines whether an indexed file created by the runtime system is marked closed on disk between file modification operations. If the value is set to YES, the file header is marked open at the beginning of a DELETE, REWRITE, or WRITE operation and marked closed at the end of the operation. This minimizes the risk that a power interruption would necessitate a recovery of the indexed file. If the value is set to NO and the file is opened I-O, OUTPUT, or EXTEND, any power interruption will require a recovery of the indexed file. The default value for this keyword is NO.

7. **FORCE-DATA.** This keyword determines—for an indexed file created by the runtime system and opened WITH LOCK—whether the runtime system forces data blocks to be given to the operating system when modified. If the value is set to YES, data block write requests are issued to the operating system whenever information is changed. If the value is set to NO, data block write requests are made as dictated by buffer space. The default value for this keyword is NO.

8. **FORCE-DISK.** This keyword determines—for an indexed file created by the runtime system—whether the runtime system forces disk blocks to be written to disk when modified. If the value is set to YES, blocks written to the operating system are forced to be written to disk as well. If the value is set to NO, blocks written to the operating system remain in the operating system buffer pool. The default value for this keyword is NO.

9. **FORCE-INDEX.** This keyword determines—for an indexed file created by the runtime system and opened WITH LOCK—whether the runtime system forces index blocks to be given to the operating system when modified. If the value is set to YES, index block write requests are issued to the operating system whenever the index block is changed. If the value is set to NO, index block write requests are made as buffer space dictates. The default value for this keyword is NO.

10. **KEY-COMPRESSION.** This keyword determines whether the indexed files created by the runtime system use key compression. If the value is set to YES, key compression is used. If the value is set to NO, keys are stored in uncompressed form. The default value for this keyword is YES.

11. **MINIMUM-BLOCK-SIZE.** This keyword determines the minimum disk block size for the indexed files created by the runtime system, represented as a decimal number. (See the discussion of the BLOCK CONTAINS clause on page 233). The maximum value for this keyword is 4096; the minimum value is 256. The default value is 1024.

12. **ROUND-TO-NICE-BLOCK-SIZE.** This keyword determines whether the block size computed for the indexed files created by the runtime system is forced to be a multiple of 512 (under Windows) or the value of BUFSIZ, taken from the C include file <**stdio.h**> (under UNIX). The default value for this keyword is YES.

13. **USE-LARGE-FILE-LOCK-LIMIT.** This keyword determines which value to use for the lock limit when creating a version 4 indexed file. If the value of this keyword is set to NO, the value of the FILE-LOCK-LIMIT keyword of the RUN-FILES-ATTR configuration record is used. If the value is set to YES, the value of the LARGE-FILE-LOCK-LIMIT keyword is used. (For a description of these keywords in the RUN-FILES-ATTR record, see page 318.) The default value for this keyword is NO.

    **Note** This keyword affects only the lock limit placed into the Key Information Block (KIB) for version 4 indexed files created by the runtime system; existing version 4 indexed files always use the lock limit stored when the file was created.

# RUN-OPTION Configuration Record

The RUN-OPTION configuration record identifier is followed by one or more keywords. If the keyword is allowed to have a value, it is followed by an equal sign (=) and the value. The possible keywords are as follows:

1. **B.** This keyword controls the default ACCEPT and DISPLAY buffer size and is represented as a decimal number. The maximum value for this keyword is 65280; the minimum value is 1. The default value is 264.

   A default value specified with this keyword may be overridden by the runtime B Option (see page 181).

2. **DISPLAY-UPDATE-MESSAGES.** This keyword controls which messages are displayed when the automatic update check determines that there is an update message available for the RM/COBOL runtime. The message is displayed at runtime termination. If the value of this keyword is set to ALL, then all update messages are displayed. If the value of this keyword is set to URGENT-ONLY, then only messages that Liant designates as urgent are displayed. For a runtime licensed as part of an RM/COBOL development system, the default value is ALL; for a runtime licensed as part of an RM/COBOL runtime system, the default value is URGENT-ONLY.

3. **ENABLE-LOGGING.** This keyword controls the generation of various error and information log files. The LOG-PATH keyword (described on page 324) must be included to specify the location of the directory for the log file. By default, no logging is performed. Logging normally should be enabled only when required to meet a particular need. If large amounts of data are logged, there will be noticeable performance degradation in the runtime. Multiple keyword values are separated by commas. One or more of the following values may be included:

   | | |
   |---|---|
   | 98-ERRORS | OTHER-CLOSE |
   | ALL | OTHER-OPEN |
   | ATOMIC-IO | TERMINAL-INFO |
   | FILE-CLOSE | TERMINATION |
   | FILE-OPEN | PERM-OS-ERRORS |

   - ENABLE-LOGGING=98-ERRORS turns on logging of 98,*nn* file structure errors. Most of these errors indicate problems with indexed files. This logging option is normally turned on only at the request of a Liant support representative. The log file generated is named **RM98ERR.LOG**.

- ENABLE-LOGGING=ALL turns on all logging.
- ENABLE-LOGGING=ATOMIC-IO turns on logging of possible problems with indexed files which were created with atomic I/O enabled. This logging option is normally turned on only at the request of a Liant support representative. The log file generated is named **RMATOMIO.LOG**.
- ENABLE-LOGGING=FILE-CLOSE turns on logging of CLOSE statements of sequential, relative, and indexed files. The log file generated is named **RMOPNCLS.LOG**. This logging option may be used to aid in understanding some complex applications.
- ENABLE-LOGGING=FILE-OPEN turns on logging of successful OPEN statements of sequential, relative, and indexed files. The log file generated is named **RMOPNCLS.LOG**. This logging option may be used to aid in understanding some complex applications and to help diagnose certain types of problems involving large numbers of file opens.
- ENABLE-LOGGING=OTHER-OPEN turns on logging of successful opens of program files. The log file generated is named **RMOPNCLS.LOG**. This logging option may be used to aid in understanding some complex applications and to help diagnose certain types of problems.
- ENABLE-LOGGING=OTHER-CLOSE turns on logging of closes of program files. The log file generated is named **RMOPNCLS.LOG**. This logging option may be used to aid in understanding some complex applications.
- ENABLE-LOGGING=TERMINAL-INFO turns on informational logging of terminfo and termcap processing. The log file generated is named **RMINSEQ.LOG**. This option is available only for UNIX. This logging option is intended to help solve problems encountered with the terminfo or termcap configuration.
- ENABLE-LOGGING=TERMINATION turns on logging of termination errors, including all traceback information. The log file generated is named **RMTERM.LOG**. Termination logging is available or Windows and UNIX, however, UNIX users also can redirect standard error (STDERR).
- ENABLE-LOGGING=PERM-OS-ERRORS turns on logging of detailed information about errors encountered by the runtime when making operating system (OS) calls. On UNIX, this option also enables logging of information to help diagnose the cause of Procedure Errors 251 through 256 related to the terminfo or termcap configuration. This logging option is normally turned on only at the request of a Liant support representative. The log file generated is named **RMOSERR.LOG**.

4. **FILL-CHARACTER.** This keyword specifies the fill character to be used to initialize read-write memory allocated for the run unit. The default value for this keyword is " " (space = 0x20 = 20h). Working-Storage data items that do not specify a VALUE clause in their data description entry will be filled with this character value at program load time.

   This option corresponds to the runtime F Option (see page 183).

   For consistency with the command line treatment of the F option value, a single-digit character, 0 – 9, represents the corresponding single ASCII digit character, that is, "0" – "9", 0x30 – 0x31 or decimal 48 – 57, whether or not it is quoted. Multi-digit decimal values without quotes can be 00 – 255 and represent the corresponding numeric code point. Quoted values must contain only one

character and are never considered to be a numeric value, that is, they always represent a single ASCII character value to be used as the fill character.

5. **K.** This keyword controls the suppression of the banner notice and the STOP RUN message. If the value is set to SUPPRESS, the banner notice and STOP RUN message are suppressed. If the value is set to DISPLAY, the suppression of the banner notice and the STOP RUN message is controlled by the runtime K Option (see page 180). The default value for this keyword is DISPLAY. If the K keyword specifies the value SUPPRESS, a later specification of K=DISPLAY in the configuration is ignored.

6. **L.** This keyword specifies RM/COBOL object or non-COBOL subprogram libraries to be loaded during run unit initialization. The value is a string specifying the pathname of the library file to be loaded. This keyword may be specified multiple times to load multiple libraries. Libraries are loaded in the left to right order of L keywords in the configuration file. The default value for this keyword is not to add any libraries to the list of libraries to be loaded.

   This option corresponds to the runtime L Option (see page 183). The same rules regarding locating libraries for the L Option apply. Libraries specified with this keyword are loaded after libraries specified with the L Option. Liant recommends that multiple RM/COBOL programs be combined into libraries using the Combine Program (rmpgmcom) utility (on page 583) because doing so improves application startup time.

7. **LIBRARY-PATH.** This keyword specifies the location (directory) of a set of RM/COBOL object files, all of which are to be loaded during run unit initialization. The value is a string specifying the directory pathname. The specified directory must be locally accessible, although it may be accessed through a network drive or UNC name; in particular, RM/InfoExpress cannot be used to access the directory. This keyword may be specified multiple times to load from multiple directories. The default value for this keyword is not to add any directories to the list of directories from which to load COBOL programs.

   Directories are processed in the left to right order of LIBRARY-PATH keywords in the configuration file, but the order of loading from any one directory is operating system-dependent. Libraries loaded because of this keyword are loaded after libraries loaded with either the runtime L Option or the L keyword of the RUN-OPTION configuration record, but before non-COBOL libraries loaded automatically from the **rmcobolso** (on UNIX) or **RmAutoLd** (on Windows) subdirectories of the execution directory. All RM/COBOL object files in the specified directory are loaded. A file is determined to be an RM/COBOL object file solely by its having an extension matching the RM/COBOL object extension as specified by the OBJECT keyword of the EXTENSION-NAMES configuration record (by default, **.cob**). On UNIX, the test for this extension is case sensitive. Liant recommends that multiple RM/COBOL programs be combined into libraries using the Combine Program (rmpgmcom) utility (on page 583) because doing so improves application startup time.

8. **LOG-PATH.** This keyword specifies the location (directory) where the log file (as specified in the ENABLE-LOGGING keyword described earlier in this section) will be written. The directory must already exist before the runtime is started, and the user must have create and write permission for the directory. The runtime will create the log file, if necessary, and the file will be opened in append mode. The default for this keyword is to suppress all logging regardless of the value specified for the ENABLE-LOGGING keyword (described on page 322).

9. **M.** This keyword controls which level of ANSI semantics is used with ACCEPT and DISPLAY statements. If the value is set to 2, ANSI level 2 semantics are used. If the value is set to 1, the level of ANSI semantics is controlled by the runtime M Option (see page 181). The default value for this keyword is 1. If the M keyword specifies the value 2, a later specification of M=1 in the configurations is ignored.

10. **MAIN-PROGRAM.** This keyword specifies the name of the main program to be executed. The value is a string that specifies the program name or file name that is to be executed. The name overrides the name specified on the command line. The first occurrence of this keyword encountered during configuration processing is effective and later occurrences are ignored. The default value for this keyword is to use the filename specified on the command line.

    **Note** If a main program name is configured, the filename parameter may be omitted in the Runtime Command. In this case, options can be specified in the Runtime Command only by using a hyphen before each option.

11. **V.** This keyword controls the display of the list of the support modules (shared objects on UNIX and dynamic load libraries on Windows) loaded by the RM/COBOL runtime system. If the value is set to DISPLAY, the list will be displayed. If the value is set to SUPPRESS, the runtime V Option or the setting of the RM_DYNAMIC_LIBRARY_TRACE environment variable control the list display (see page 180). The default value for this keyword is SUPPRESS. If the V keyword specifies the value DISPLAY, a later specification of V=SUPPRESS in the configuration is ignored.

# RUN-REL-FILES Configuration Record

The RUN-REL-FILES configuration record identifier is followed by one keyword. The keyword is followed by an equal sign (=) and a value. The possible keyword is as follows:

1. **BLOCK-SIZE.** This keyword determines the default block size, represented as a decimal number, for relative files opened WITH LOCK. The maximum value for this keyword is 65489. The minimum value is 0, which specifies that the default value is no blocking. The presence of a BLOCK CONTAINS phrase in the file description entry of an unshared relative file forces that file to be blocked, even if no blocking is specified by this keyword. The default value for this keyword is BUFSIZ, taken from the C include file <**stdio.h**>, if the file is opened for RANDOM access. Under UNIX, if the file is opened for DYNAMIC or SEQUENTIAL access, the default value is BUFSIZ, taken from the C include file <**stdio.h**>, or 4096, whichever is larger. Under Windows, the default value is the disk sector size or 4096, whichever is larger.

2. **USE-LARGE-FILE-LOCK-LIMIT.** This keyword determines which value to use for the limit when applying locks to a relative file. If the value of this keyword is set to NO, the value of the FILE-LOCK-LIMIT keyword of the RUN-FILES-ATTR configuration record is used. If the value is set to YES, the value of the LARGE-FILE-LOCK-LIMIT keyword is used. For a description of these keywords, see the RUN-FILES-ATTR configuration record (on page 316). For record and file locks to perform correctly, all run units opening a file must use the same file lock limit. The default value for this keyword is NO.

# RUN-SEQ-FILES Configuration Record

The RUN-SEQ-FILES configuration record identifier is followed by one or more keywords. If the keyword is allowed to have a value, it is followed by an equal sign (=) and the value. The possible keywords are as follows:

1. **BLOCK-SIZE.** This keyword determines the default block size represented as a decimal number, for sequential disk files opened WITH LOCK. The maximum value for this keyword is 65489. The minimum value is 0, which specifies that the default value is no blocking. The presence of a BLOCK CONTAINS phrase in the file description entry of an unshared sequential file forces that file to be blocked, even if no blocking is specified by this keyword. Under UNIX, the default value for this keyword is BUFSIZ, taken from the C include file <**stdio.h**>, or 4096, whichever is larger. Under Windows, the default value is the disk sector size or 4096, whichever is larger.

2. **DEFAULT-TYPE.** This keyword determines whether unspecified sequential files are line sequential or binary sequential. If the value is set to BINARY, unspecified sequential files are binary sequential. If the value is set to LINE, unspecified sequential files are line sequential. The default value for this keyword is BINARY.

3. **DEVICE-SLEWING-RESERVE.** This keyword determines the number of character positions to be reserved for form feed and line feed characters. The value for this keyword is a decimal number. The maximum value for this keyword is 999; the minimum value is 10. The default value is 255.

   This keyword applies to line sequential files written directly to a nontape device. It also applies to line sequential files on disk that are not opened WITH LOCK or are not blocked. The value required for this keyword is generally the maximum value specified by any BEFORE/AFTER ADVANCING phrase in a WRITE statement in the program, plus 2. This value is configurable for performance considerations only, since it enables the record—along with the appropriate number of control characters—to be written in one write request. If the value specified for this keyword is less than the maximum specified by the BEFORE/AFTER ADVANCING phrase, the operation will be performed correctly, but will require more than one write request to complete the operation.

4. **TAB-STOPS.** This keyword determines an ascending sequence of tab stop columns represented as decimal numbers separated by commas. Up to 18 tab stop columns are allowed. TAB-STOPS=0 specifies that no tab stop columns exist. The default TAB-STOPS sequence is 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60, 64, 68, and 72.

5. **USE-LARGE-FILE-LOCK-LIMIT.** This keyword determines which value to use for the limit when applying locks to a sequential file. If the value of this keyword is set to NO, the value of the FILE-LOCK-LIMIT keyword of the RUN-FILES-ATTR configuration record is used. If the value is set to YES, the value of the LARGE-FILE-LOCK-LIMIT keyword is used. For a description of these keywords, see the RUN-FILES-ATTR configuration record (on page 316). For record and file locks to perform correctly, all run units opening a file must use the same file lock limit. The default value for this keyword is NO.

# RUN-SORT Configuration Record

The RUN-SORT configuration record identifier is followed by one or more keywords. If the keyword is allowed to have a value, it is followed by an equal sign (=) and the value. The possible keywords are as follows:

1. **INTERMEDIATE-FILES.** This keyword determines the number of intermediate sort files represented as a decimal number. The maximum value for this keyword is 9; the minimum value is 3. The default value is 5.
2. **MEMORY-SIZE.** This keyword determines the default sort memory size represented as a decimal number. The maximum value for this keyword is available memory; the minimum value is 0. If unspecified, the default sort memory size is 256000 bytes. The maximum allowed value is 2147483647 bytes. If the MERGE or SORT statements are used in the run unit, sort memory size must be non-zero.

# TERM-ATTR Configuration Record

The TERM-ATTR configuration record provides information about terminal attributes. The TERM-ATTR record identifier is followed by one or more keywords. If the keyword is allowed to have a value, it is followed by an equal sign (=) and the value. The possible keywords are as follows:

1. **ALWAYS-USE-CURSOR-POSITIONING.** This keyword determines whether the absolute cursor positioning sequence is always used to position the cursor on the screen. If the value is set to YES, the absolute cursor positioning sequence is always used. If the value is set to NO, the runtime system attempts to optimize cursor positioning. The default value for this keyword is NO. This keyword may be used in those circumstances in which the optimized cursor position is sometimes incorrect.

   **Note** This keyword is supported only under UNIX.

2. **BCOLOR.** This keyword indicates the initial background color value to use for ACCEPT and DISPLAY statements. The terminal will be restored to this color when the run unit ends. The default color value for this keyword is BLACK.

   **Note** Under Windows, the default background color is determined by the system settings. In particular, the default background color is the default “window background” color.

3. **CHARACTER-TIMEOUT.** This keyword sets the time-out period related to the inter-character timer used by the RM/COBOL runtime system when processing input sequences. The timer is used to determine whether a character is a final character in a sequence. The default wait time before deciding that no more characters are pending is .5 seconds. The inter-character timer may need to be increased on certain systems or in some cases when the user is logged in from a network (rlogin). The value for the CHARACTER-TIMEOUT keyword is specified in tenths of seconds.

   **Note 1** This keyword is supported only under UNIX.

   **Note 2** The value of the inter-character timer also affects the BEFORE TIME phrase of the ACCEPT statement. For more information, see Time Phrase (on page 203).

4. **COLUMNS.** This keyword determines the number of columns the runtime system will display on the terminal screen, represented as a decimal number between 1 and 255. The default value for this keyword is 80.

   **Note** This keyword is supported only under Windows.

5. **DATA-CHARACTERS.** This keyword has as its value a comma-separated pair of decimal integers. The first number is the lower bound of characters to be interpreted as text characters. The second number is the upper bound of characters to be interpreted as text characters. On input, characters not in this range and not part of a TERM-INPUT sequence (see page 332) are illegal. The default range is 32 (20h) to 126 (7Eh). See Term-Input Configuration Record (on page 332).

   The first occurrence of a DATA-CHARACTERS keyword specification clears the default range and any previously specified DBCS-CHARACTERS (see page 328) or PASS-THRU-ESCAPE (see page 330) characters. Multiple disjoint character ranges can be specified to be data characters by multiple specifications of the DATA-CHARACTERS keyword. For UNIX, the decimal integers must specify code points in the current locale's set of text characters. For Windows, the decimal integers must specify the character code points in the OEM codepage when the native character set uses the OEM codepage, which is usually 437 in the U.S. and 850 in Western Europe, and must specify the character code points in the ANSI codepage when the native character set uses the ANSI codepage, which is usually 1252 in the U.S. and Western Europe. The following is an example for specifying a disjoint range of text code points:

   ```
   /* Set normal default range
   TERM-ATTR DATA-CHARACTERS=32,126
   /* Add euro symbol (as OEM code point 213)
   TERM-ATTR DATA-CHARACTERS=213,213
   ```

   Multiple DATA-CHARACTERS keyword specifications are also useful when converting the period on the numeric keypad into a comma for numeric input in countries that use the comma as the fractional separator rather than the period. This allows the user to enter numbers using only the numeric keypad. An example of how this might be done is as follows:

   ```
   /* Specify the data characters, omitting the period (46)
   TERM-ATTR DATA-CHARACTERS=32,45
   TERM-ATTR DATA-CHARACTERS=47,126
   /* Map the numeric keypad period to comma
   TERM-INPUT   DATA=44   NUL 110
   /* Leave the keyboard (regular) period as period
   TERM-INPUT   DATA=46   NUL 190
   ```

6. **DBCS-CHARACTERS.** This keyword enables support for double-byte character set (DBCS) characters and specifies the lead-byte character ranges.

   In DBCS, a character is represented by either one or two bytes. Double-byte character set characters have the following components:

   - **Lead byte.** If the value of this byte falls into the lead-byte range, a DBCS character is indicated.
   - **Trail byte.** This byte further refines the selected character. The trail byte can have any value except zero (0), but is typically restricted to a subset of all possible characters.

Double-byte character set characters occupy two, physically adjacent positions on the screen and are usually displayed as a double-width character.

Double-byte character set characters, however, are not the same as "wide" characters. In a wide-character set, such as Unicode, all characters are represented by two bytes, which provides up to 65,536 characters. In DBCS, the value of the first byte determines whether the second byte is considered part of the current character.

Values for this keyword are specified as a pair of numbers separated by a comma. The syntax of this keyword is as follows:

```
TERM-ATTR DBCS-CHARACTERS=n1,n2
```

These numbers specify a range of characters to be considered lead-byte characters. Multiple ranges of characters can be specified by repeating the DBCS-CHARACTERS keyword, but ranges must not overlap. These numbers must also be wholly contained within the DATA-CHARACTERS range, which by default, is 32 through 126. The default value for this keyword is that no characters are considered to be lead-byte characters.

The DBCS-CHARACTERS keyword must be specified after the first occurrence of the DATA-CHARACTERS keyword if both keywords are specified. The first occurrence of the DATA-CHARACTERS keyword causes values specified in the preceding DBCS-CHARACTERS keyword and the PASS-THRU-ESCAPE keyword (see page 330) to be discarded.

**Note** This keyword is only supported under UNIX.

### Restrictions

The following restrictions apply to double-byte character set characters in RM/COBOL:

- Because a DBCS character cannot be entered in a field where there is only one remaining character position, DBCS characters cannot be any of the following:
  - Entered into one-character ACCEPT fields.
  - Typed into the last position of a field.
  - Inserted into a field where only one position remains available.
  - Typed over single-byte characters in a field where no positions remain available.
- A DBCS character cannot be displayed on a position that would physically or logically separate the trail byte from the lead byte. DBCS characters, therefore, cannot be displayed in the last column of a screen or window. In addition, when a new window is opened, any lead bytes or trail bytes that would be orphaned by the window are blanked and restored when the window is removed.
- If an ACCEPT field spans a window or screen row, RM/COBOL will permit the entry of a DBCS character when the cursor is in the last column of the window or screen. The cursor advances two character positions and the DBCS character is saved but not displayed (two spaces are displayed instead). If characters are inserted or deleted from the field such that the character no longer wraps across rows, it becomes visible. If another DBCS

character wraps as a result, it becomes invisible. All entered characters will be returned to the RM/COBOL program when the ACCEPT is terminated.

- Such an invisible character cannot be retrieved with the ACCEPT-FIELD-FROM-SCREEN keyword in the RUN-ATTR record or the C$SCRD subprogram. Instead, spaces are returned.
- DBCS characters must not be used as a character specified either in the PROMPT phrase of an ACCEPT statement or in the GRAPHICS keyword of the CONTROL phrase of an ACCEPT or DISPLAY statement.
- DBCS characters must not be used in filenames or pathnames.

7. **FCOLOR.** This keyword indicates the initial foreground color value to use for ACCEPT and DISPLAY statements. The terminal will be restored to this color when the run unit ends. The default color value for this keyword is WHITE.

   **Note** Under Windows, the default foreground color is determined by the system settings. In particular, the default foreground color will be the default "window text" color. The default high-intensity foreground color is determined by the default "selected text" color.

8. **PASS-THRU-ESCAPE.** This keyword allows certain DISPLAY statements to write sequences directly to standard output, bypassing screen optimization and buffering. If used, any DISPLAY statement that begins with one of the specified characters will exhibit the PASS-THRU behavior. No cursor positioning is performed. All attributes specified in the DISPLAY statement are ignored. The characters are represented in ordinal form (for example, the ASCII ESC character is 27) within a comma separated list as both individual values and ranges (for example, 0-31,255). The default value for this keyword is to have no PASS-THRU-ESCAPE characters.

   The PASS-THRU-ESCAPE keyword must be specified after the first occurrence of the DATA-CHARACTERS keyword (see page 328) if both keywords are specified. The first occurrence of the DATA-CHARACTERS keyword causes values specified in preceding DBCS-CHARACTERS (see page 328) and PASS-THRU-ESCAPE keywords to be discarded.

   **Note** This keyword is supported only under UNIX.

9. **REDRAW-ON-CALL-SYSTEM.** This keyword sets a variable that is checked by the C language subprogram, SYSTEM, before restoring the screen. The values for this keyword are YES and NO. The default value for this keyword is YES. A value of YES causes the runtime system to redraw the screen in order to maintain screen integrity. An additional argument, (PIC X) to CALL "SYSTEM", gives control of this feature on an individual call basis. For more information, see UNIX Considerations (on page 578) and the "C Subprograms Performing Terminal I/O" topic in Appendix H: *Non-COBOL Subprogram Internals for UNIX* of the *CodeBridge* manual.

   **Note** This keyword is supported only under UNIX.

10. **ROWS.** This keyword determines the number of rows the runtime system will display on the terminal screen, represented as a decimal number between 1 and 255. The default value for this keyword is 25.

    **Note** This keyword is supported only under Windows.

11. **SCREEN-CONTENT-OPTIMIZE.** This keyword determines whether the runtime system optimizes DISPLAY output based on current screen contents. If the value is set to YES, the runtime system avoids displaying an unnecessary character if the screen image already contains that character in the desired

location, thereby increasing terminal I/O efficiency. If the value is set to NO, the runtime system forces every character to be written to the screen even if this results in duplicating the existing screen contents. Because turning off screen content optimization clears the screen image, repainting the screen will result in a blank display. The default value for this keyword is YES.

**Note** This keyword is supported only under UNIX.

12. **SUPPRESS-NULLS.** This keyword controls whether the RM/COBOL runtime system sends NULL or LOW-VALUE characters to the screen. Setting the value of this keyword to YES causes NULL characters to be stripped from all displayed fields. The NULL characters will, however, remain in the internal data storage. The default value for this keyword is NO.

    **Note** This keyword is supported only under UNIX.

13. **USE-COLOR.** Setting the value of this keyword to YES forces a COBOL program to perform all DISPLAY and ACCEPT statements using either the default color values or the color values specified in a CONTROL phrase. If the termcap or terminfo database contains color support and the set_foreground and set_background strings are present, these strings are used to process the request. Otherwise, the RM/COBOL runtime system will generate the seven-bit SGR (Set Graphics Rendition) sequences assigned by the International Standards Organization for color. These sequences are defined as follows:

| | Control Sequence Introducer | Parameter Selection | Terminator |
|---|---|---|---|
| Foreground | ESC [ | 3 [0-7] | m |
| Background | ESC [ | 4 [0-7] | m |

The color values associated with the parameter values zero through seven are as follows:

| Color | Parameter Second Digit |
|---|---|
| BLACK | 0 |
| RED | 1 |
| GREEN | 2 |
| YELLOW | 3 |
| BLUE | 4 |
| MAGENTA | 5 |
| CYAN | 6 |
| WHITE | 7 |

If the value of the USE-COLOR keyword is set to NO, color sequences will not be used until the first ACCEPT or DISPLAY that requests to use color and only if there is support for color in the termcap or terminfo database. The default value for this keyword is NO.

If none of the color keywords is specified, color processing will be disabled until a CONTROL phrase is encountered that specifies the use of a valid color. When a CONTROL phrase is encountered, it is honored provided there is color support in the termcap or terminfo database. When the run unit ends, the terminal will be restored to the values specified in the FCOLOR and BCOLOR keywords in the TERM-ATTR configuration record.

**Note** This keyword is supported only under UNIX.

# TERM-INPUT Configuration Record

The TERM-INPUT configuration record associates field editing semantics, exception codes, or a single data character with incoming character sequences. For more information on defining keys under your operating system, see Defined Keys (on page 188). The TERM-INPUT record also defines how Toolbar icon strings are interpreted, which is described in Setting Toolbar Properties (on page 88).

The TERM-INPUT record identifier is followed by one or more optional keywords and finally, by the character sequence specification. If the keyword is allowed to have a value, it is followed by an equal sign (=) and the value. The keywords may appear in any order, but must precede the character sequence specification (on page 333).

The possible keywords are as follows:

1. **ACTION.** This keyword specifies the field editing action or screen action to be performed. The allowed values, which are described in detail in Field Editing Actions (on page 338), are as follows:

   | | |
   |---|---|
   | BACKSPACE | REPAINT-SCREEN |
   | CONTROL-BREAK | RESET-ANSI-INSERTION |
   | DELETE-CHARACTER | RIGHT ARROW |
   | ERASE-ENTIRE | SCREEN-ESCAPE |
   | ERASE-REMAINDER | SCREEN-HOME |
   | ESCAPE-TO-COMMAND | SCREEN-PREVIOUS-FIELD |
   | ESCAPE-TO-OS | SCREEN-TERMINATE |
   | FIELD-HOME | SET-ANSI-INSERTION |
   | INSERT-CHARACTER | SET-RM-INSERTION |
   | LEFT-ARROW | TOGGLE-ANSI-INSERTION |

   The default value for this keyword is not to associate a field editing action with the given character sequence specification.

2. **CODE.** This keyword specifies the numeric exception code value to be returned to the program. The exception value must be in the range 0 to 255. The values 98 and 99 are reserved to indicate input data conversion rule violation and input timed out, respectively. The definition of an exception code identifies a key sequence that terminates input. Key sequences without exception codes do not terminate field input unless one of the screen actions is defined for that key sequence (as described in the ACTION keyword). The default value for this keyword is not to associate an exception with the given character sequence specification.

3. **DATA.** This keyword specifies the data character or character equivalent to be returned to the program when the incoming character sequence is received. For more information, see Character Sequence Specification (on page 333). If a DATA keyword is provided, neither the ACTION nor CODE keywords may be specified. The default value for this keyword is not to associate a data character with the given character sequence specification.

4. **EXCEPTION.** This keyword specifies whether the ON EXCEPTION clause of the ACCEPT statement should be executed, as explained in Input Sequence Specification (on page 338). The EXCEPTION keyword is legal only if the CODE keyword, described above, is also provided. If a value of YES is specified, the ON EXCEPTION clause is executed; if a value of NO is specified, the ON EXCEPTION clause is not executed. The default value for this keyword is NO unless the CODE keyword is specified, in which case the default value is

YES. The EXCEPTION keyword is ignored during an ACCEPT *screen-name* statement.

5. **PRECEDENCE.** This keyword assigns a precedence to the TERM-INPUT record. The value specified is a decimal number from 0 to 14, with 0 considered to have the highest precedence. The PRECEDENCE keyword is used only when a terminal generates the same input character sequence for two different keys. The Left Arrow and Backspace keys form a typical example. The PRECEDENCE keyword can specify which interpretation of the input sequence is to be used. The TERM-INPUT record with the highest precedence replaces that with a lower one. If two TERM-INPUT records have the same precedence, a procedure error occurs when the unit is first accessed. The default value for this keyword is 0.

   **Note** This keyword is supported only under UNIX.

## Character Sequence Specification

Terminal configuration allows the specification of incoming character sequences. The syntax of the sequences is one or more characters or character equivalents, separated by spaces. Any single character delimited by spaces is interpreted as the represented ASCII character. More than one character delimited by spaces is interpreted as a character equivalent. If the characters are digits, they are interpreted as the decimal value of the ASCII code. The recognized character equivalents may be one of those listed in Table 32 and Table 33 (on page 336).

Table 32 lists the ASCII equivalents.

**Table 32: ASCII Equivalents**

| ASCII Code | Character Equivalent | ASCII Code | Character Equivalent |
|---|---|---|---|
| 000 | NUL | 017 | DC1 |
| 001 | SOH | 018 | DC2 |
| 002 | STX | 019 | DC3 |
| 003 | ETX | 020 | DC4 |
| 004 | EOT | 021 | NAK |
| 005 | ENQ | 022 | SYN |
| 006 | ACK | 023 | ETB |
| 007 | BEL | 024 | CAN |
| 008 | BS | 025 | EM |
| 009 | HT | 026 | SUB |
| 010 | LF | 027 | ESC |
| 011 | VT | 028 | FS |
| 012 | FF | 029 | GS |
| 013 | CR | 030 | RS |
| 014 | SO | 031 | US |
| 015 | SI | 032 | SP |
| 016 | DLE | 127 | DEL |

## Translation of TERM-INPUT Sequences on Windows

When the sequence specification does not begin with a NUL on Windows, the translation of TERM-INPUT sequences on Windows is as follows:

1. Character values 1 through 26 (SOH through SUB in Table 32) are translated to Ctrl+“a” through Ctrl+“z”, respectively. For example, the sequence “BS” is the same as “NUL WCNT H”, that is, Ctrl+“h”.
2. Character value 27 (ESC) is not translated and corresponds to the Windows virtual-key code for the Esc key.
3. Character value 28 (FS) is translated to Ctrl+“\” for U.S. keyboards. The translation uses VK_OEM_5 (0xDC), which may correspond to a different key on non-U.S. keyboards.
4. Character value 29 (GS) is translated to Ctrl+“]” for U.S. keyboards. The translation uses VK_OEM_6 (0xDD), which may correspond to a different key on non-U.S. keyboards.
5. Character value 30 (RS) is translated to Ctrl+“6”.
6. Character value 31 (US) is translated to Ctrl+“-”. The translation uses VK_OEM_MINUS (0xBD), which should be the minus key for any country.
7. Character values 32 (SP) through 255, with twenty-two exceptions, are not translated and correspond directly to the Windows virtual-key code values. The exceptions are as follows:
   - 034/039 (""") are translated to VK_OEM_7=0xDE
   - 043/061 (“+=”) are translated to VK_OEM_PLUS=0xBB
   - 044/060 (“,<”) are translated to VK_OEM_COMMA=0xBC
   - 045/095 (“-_”) are translated to VK_OEM_MINUS=0xBD
   - 046/062 (“.>”) are translated to VK_OEM_PERIOD=0xBE
   - 047/063 (“/?”) are translated to VK_OEM_2=0xBF
   - 058/059 (“;:”) are translated to VK_OEM_1=0xBA
   - 091/123 (“[{”) are translated to VK_OEM_4=0xDB
   - 092/124 (“\|”) are translated to VK_OEM_5=0xDC
   - 093/125 (“]}”) are translated to VK_OEM_6=0xDD
   - 096/126 (“`~”) are translated to VK_OEM_3=0xC0

These exceptions allow a character sequence to specify a nonalphanumeric character to obtain the Windows virtual-key code for that key on a U.S. keyboard. For non-U.S. keyboards, the translation is often incorrect. Thus, outside the U.S., this method of specifying a sequence should be avoided by specifying a leading NUL in the sequence.

When the sequence specification begins with a NUL on Windows, the translation is as follows:

1. Two 0 (NUL) characters in sequence (NUL NUL) represents a Ctrl+Break key press. (RM/COBOL internally converts the 0x03 virtual-key code returned by Ctrl+Break to zero, an unused virtual-key code value, for historical reasons having to do with RM/COBOL on MS/DOS).

2. The value 127 (DEL) indicates that the next character, if there is one, is an ASCII OEM character code. If there is no next character, 127 is interpreted the same as WF16 (VK_F16 = 0x7F = 127).
3. Any other value is treated as a Windows virtual-key code value. The value may be specified as one of the following:
   - a single OEM ASCII character (example: A);
   - a quoted single OEM ASCII character (example: "A");
   - one of the ASCII equivalents from Table 32 on page 333 (example: ETX);
   - one of the Code Names from Table 33 (example: WF2);
   - a decimal number (example: 113 for F2); or
   - a hexadecimal number (example: 0x71 for F2).

However, even though OEM ASCII values can be specified in a TERM-INPUT character sequence, for Windows they represent virtual-key code values, except as described in the translation used when the sequence does not begin with a NUL. The description of a value specification in a configuration record on page 282 describes how to specify a decimal or hexadecimal numeric value and when quotes are required around an ASCII character. The Windows virtual-key codes for letters are the uppercase version of the letter; the lowercase letters represent other keys on the keyboard (for example, the letter "a", with the value 0x61, is the virtual-key code for the numeric keypad 1 key). Documentation on Windows virtual-key codes is available from Microsoft on their MSDN Library website at the following location:

- http://msdn.microsoft.com/library/

Additional character equivalents, listed in Table 33, have been defined for the character sequence specifications in Windows. If a character equivalent, which actually specifies a Windows virtual-key code value, is used to specify a character sequence, the sequence specification should begin with a NUL. This is necessary because character values are translated in the absence of a leading NUL, and there is overlap between character values and virtual-key code values.

Another special incoming character sequence has been added. Specify NUL DEL <ascii-char-code> on the TERM-INPUT record to match on the ASCII character code rather than the Windows virtual key code. <ascii-char-code> is the decimal value of the ASCII code in the range 0 through 255. In order for this record to be effective, the <ascii-char-code> must not be included in the TERM-ATTR record DATA-CHARACTERS range. As an example, an Umlaut-Uppercase-U can be input by:

```
TERM-INPUT   DATA=154   NUL DEL 154
```

**Note** Alt-key sequences are not available under RM/COBOL for Windows because the underlying Windows-based environment traps the Alt-key sequences. Alt-key sequences are entered as Ctrl-Shift-key combination sequences. For example, use Ctrl-Shift-I instead of Alt-I.

**Table 33: Additional Character Equivalents Under RM/COBOL for Windows**

| Code Name | Description (Virtual key code) |
|---|---|
| APPS | Applications key (0x5D) |
| ATTN | ATTN key (0xF6) |
| CAPITAL | Caps Lock key (0x14) |
| CRSEL | CRSEL key (0xF7) |
| EREOF | Erase EOF key (0xF9) |
| EXSEL | EXSEL key (0xF8) |
| KB'" | Regular KeyBoard apostrophe/quotation mark (0xDE) |
| KB,< | Regular KeyBoard comma/less than (0xBC) |
| KB.> | Regular KeyBoard period/greater than (0xBE) |
| KB/? | Regular KeyBoard slash/question mark (0xBF) |
| KB;: | Regular KeyBoard semicolon/colon (0xBA) |
| KB[{ | Regular KeyBoard left bracket/left brace (0xDB) |
| KB\| | Regular KeyBoard backslash/vertical bar (0xDC) |
| KB]} | Regular KeyBoard right bracket/right brace (0xDD) |
| KB-_ | Regular KeyBoard minus sign/underscore (0xBD) |
| KB`~ | Regular KeyBoard grave accent/tilde (0xC0) |
| KB=+ | Regular KeyBoard equal sign/plus sign (0xBB) |
| KB0) | Regular KeyBoard zero/right parenthesis (0x30) |
| KB1! | Regular KeyBoard one/exclamation point (0x31) |
| KB2@ | Regular KeyBoard two/at sign (0x32) |
| KB3# | Regular KeyBoard three/number sign (0x33) |
| KB4$ | Regular KeyBoard four/dollar sign (0x34) |
| KB5% | Regular KeyBoard five/percent sign (0x35) |
| KB6^ | Regular KeyBoard six/caret (0x36) |
| KB7& | Regular KeyBoard seven/ampersand (0x37) |
| KB8* | Regular KeyBoard eight/asterisk (0x38) |
| KB9( | Regular KeyBoard nine/left parenthesis (0x39) |
| LWIN | Left Windows logo key (0x5B) |
| NKP- | Numeric KeyPad Subtract (minus sign) (0x6D) |
| NKP* | Numeric KeyPad Multiply (asterisk) (0x6A) |
| NKP. | Numeric KeyPad Decimal (period) (0x6E) |
| NKP/ | Numeric KeyPad Divide (slash) (0x6F) |
| NKP+ | Numeric KeyPad Add (plus sign) (0x6B) |
| NKP0 … NKP9 | Numeric KeyPad 0 … 9 (zero … nine) (0x60 … 0x69) |
| NKPS | Numeric KeyPad Separator (not on most keyboards) (0x6C) |
| NUMLOCK | Num Lock key (0x90) |
| OEM_1 | “;:” for US (0xBA) |
| OEM_2 | “/?” for US (0xBF) |

**Table 33: Additional Character Equivalents Under RM/COBOL for Windows (*Cont.*)**

| Code Name | Description (Virtual key code) |
|---|---|
| OEM_3 | "`~" for US (0xC0) |
| OEM_4 | "[{" for US (0xDB) |
| OEM_5 | "\|" for US (0xDC) |
| OEM_6 | "]}" for US (0xDD) |
| OEM_7 | """" for US (0xDE) |
| OEM_8 | (0xDF) |
| OEM_COMMA | "," for any country (0xBC) |
| OEM_MINUS | "-" for any country (0xBD) |
| OEM_PERIOD | "." for any country (0xBE) |
| OEM_PLUS | "+" for any country (0xBB) |
| PA1 | PA1 key (0xFD) |
| PAUSE | Pause key (0x13) |
| PLAY | Play key (0xFA) |
| RWIN | Right Windows logo key (0x5C) |
| SCROLL | Scroll Lock key (0x91) |
| WAGR | AltGr key (Ctrl+Alt under Windows) |
| WCNT | Control key (0x11) |
| WCNT | Control key (0x11) |
| WDEL | Delete key (0x2E) |
| WDWN | Down Arrow key (0x28) |
| WEND | End key (0x23) |
| WEND | End key (0x23) |
| WF1 . . . WF23 | Function 1 … Function 23 (0x70 … 0x86) |
| WHOM | Home key (0x24) |
| WINS | Insert key (0x2D) |
| WLFT | Left Arrow key (0x25) |
| WPGD | PgDn key (0x22) |
| WPGU | PgUp key (0x21) |
| WPRT | Print key (0x2C) |
| WRGT | Right Arrow key (0x27) |
| WSFT | Shift key (0x10) |
| WUP | Up Arrow key (0x26) |
| ZOOM | Zoom key (0xFB) |

### Example

The NKP*x* and KB*xx* names are useful if you want the numeric keypad to return a different character than the same key on the regular portion of the keyboard. Suppose you want the period key on the regular keyboard to continue to return a period (ASCII decimal 46), while the period key on the numeric keypad returns a comma (ASCII decimal 44). To remove the period (46) from the range, replace the normal record:

```
TERM-ATTR DATA-CHARACTERS=32,126
```

with the following two new records:

```
TERM-ATTR DATA-CHARACTERS=32,45

TERM-ATTR DATA-CHARACTERS=47,126
```

Then, to obtain the desired behavior, add the following two new records:

```
TERM-INPUT DATA=46  NUL KB.>

TERM-INPUT DATA=44  NUL NKP.
```

## Input Sequence Specification

Incoming character sequences are specified by the corresponding attribute name. For example, assume you have a keyboard with a Next page key (**knp** in the terminfo database, and **kN** in the termcap database) and you want that to cause the ON EXCEPTION branch of your ACCEPT statement to be taken with the decimal value 80 stored in the EXCEPTION variable. This could be described in the following ways.

For terminfo:

```
TERM-INPUT       CODE=80      EXCEPTION=YES  knp
```

For termcap:

```
TERM-INPUT       CODE=80      EXCEPTION=YES  kN
```

For Windows:

```
TERM-INPUT       CODE=80      EXCEPTION=YES  NUL WPGU
```

## Field Editing Actions

RM/COBOL ACCEPT statements define fields on the terminal in which the operator may enter data. Depending on the phrases specified in the ACCEPT statement and the setting of the ACCEPT-FIELD-FROM-SCREEN keyword of the RUN-ATTR configuration record (on page 313), the initial contents of the screen field may be empty (all spaces), filled with characters from the screen, filled with prompt characters or filled with the current value of the associated ACCEPT operand. The operator may then modify the displayed contents of the screen field; all positions of

that screen field may continue to be modified until a field termination key is entered. This modification of the displayed data is called field editing.

Data entry is processed in either insertion mode or replacement mode. In insertion mode, incoming text characters are inserted at the current cursor position, and following characters are moved to the right in the field. In replacement mode, incoming text characters replace the characters at the current cursor position. The RM/COBOL terminal model defines three types of insertion mode:

1. Single-character insertion mode reverts to replacement mode after one text character is inserted.
2. RM insertion mode reverts to replacement mode when data entry in the field is terminated or when a field editing key is entered.
3. ANSI insertion mode remains active across ACCEPT operations until reset by the operator to RM insertion mode or replacement mode.

The RM/COBOL terminal interface provides the following field editing facilities that may be made available to the operator. These facilities can be specified with the ACTION keyword of the TERM-INPUT configuration record (on page 332).

1. **BACKSPACE.** The BACKSPACE value accomplishes a destructive backspace by moving the cursor left one position in the field, deleting the character at that position, moving all following characters left one position, and inserting a prompt character at the right end of the screen field. If entered in the leftmost position of the screen field, the key sequence is treated as a field termination key or as an illegal keystroke depending on whether an exception code has been assigned to the key sequence.
2. **CONTROL-BREAK.** The CONTROL-BREAK value causes the run unit to be terminated immediately, as if a STOP RUN statement were executed.
3. **COPY-TO-CLIPBOARD.** The COPY-TO-CLIPBOARD value causes the currently selected text to be copied to the clipboard. This action is effective only on Windows.
4. **DELETE-CHARACTER.** The DELETE-CHARACTER value deletes the character at the current cursor position, moves all following characters left one position, and inserts a prompt character at the right end of the screen field.
5. **ERASE-ENTIRE.** The ERASE-ENTIRE value replaces all character positions in the screen field with prompt characters, and moves the cursor to the leftmost position of the field.
6. **ERASE-REMAINDER.** The ERASE-REMAINDER value replaces characters from the current position to the rightmost position of the screen field with prompt characters.
7. **ESCAPE-TO-COMMAND.** The ESCAPE-TO-COMMAND value causes the runtime system to run a user-specified command whenever the configured key is pressed. The screen is cleared and the terminal is returned to a shell state before the shell command is run. After the shell or command terminates, the COBOL screen is restored. The SHELL environment variable determines the shell that is used. If none is set, **/bin/sh** is used. The RM_ESCAPE_TO_COMMAND environment variable determines the command that is used. If none is set, **/bin/sh** is used.

   **Note** This value is supported only under UNIX.

8. **ESCAPE-TO-OS.** The ESCAPE-TO-OS value causes the runtime system to bring up a command shell whenever the configured key is pressed. This allows the user to press a configured key to bring up a shell, execute OS commands, and then return to the COBOL program without having to change the COBOL source code.

   **Note** This value is supported only under UNIX.

9. **FIELD-HOME.** The FIELD-HOME value moves the cursor to the leftmost position of the screen field.

10. **INSERT-CHARACTER.** The INSERT-CHARACTER value inserts a single space character at the current character position. When entered in replacement mode, this causes the subsequent text character to appear to be inserted.

11. **LEFT-ARROW.** The LEFT-ARROW value moves the cursor left one position in the screen field. If entered in the leftmost position of the screen field, this value is treated as a field termination key or as an illegal keystroke, depending on whether an exception code has been assigned to the key sequence.

12. **REPAINT-SCREEN.** The REPAINT-SCREEN value causes the runtime system to reinitialize the terminal and redraw the screen from an in-memory image. This allows the user to restore a terminal's display after events such as a loss of power to the terminal.

    **Note** This value is supported only under UNIX.

13. **RESET-ANSI-INSERTION.** The RESET-ANSI-INSERTION value changes the handling of subsequent text characters to replacement mode.

14. **RIGHT-ARROW.** The RIGHT-ARROW value moves the cursor right one position in the screen field. The key sequence is illegal if the character at the current character position is a prompt character. If entered in the rightmost screen field position, the key sequence is treated as a field termination key or as an illegal keystroke, depending on whether an exception code has been assigned to the key sequence.

15. **SCREEN-ESCAPE.** The SCREEN-ESCAPE value terminates an ACCEPT *screen-name* statement. The current field is neither checked for errors nor moved to the intermediate area for screen data items. If the current ACCEPT is not an ACCEPT *screen-name* statement, no action is taken (unless the keyword CODE is defined for the same key sequence).

16. **SCREEN-HOME.** The SCREEN-HOME value moves the cursor to the first input field defined in the Screen Section for the current screen. The current field is checked for errors. No semantic action is taken for ACCEPT *identifier* statements.

17. **SCREEN-PREVIOUS-FIELD.** The SCREEN-PREVIOUS-FIELD value moves the cursor to the beginning of the previous field (as defined in the Screen Section) after the current field is validated. No semantic action is taken for ACCEPT *identifier* statements.

18. **SCREEN-TERMINATE.** The SCREEN-TERMINATE value terminates an ACCEPT *screen-name* statement, but, unlike the SCREEN-ESCAPE value, the current field is validated and moved to the intermediate area. No semantic action is taken for ACCEPT *identifier* statements.

19. **SET-ANSI-INSERTION.** The SET-ANSI-INSERTION value changes the handling of following text characters. Upon entry to the run unit, terminal handling is in replacement mode. The SET-ANSI-INSERTION value causes later incoming text characters to be handled in insertion mode. Insertion mode continues until a TOGGLE-ANSI-INSERTION, RESET-ANSI-INSERTION or SET-RM-INSERTION value is entered.
20. **SET-RM-INSERTION.** The SET-RM-INSERTION value changes the handling of following text characters. Upon entry to the run unit, terminal handling is in replacement mode. The SET-RM-INSERTION value causes later incoming text characters to be handled in insertion mode.

    RM insertion mode is reset when any field editing key sequence is entered or when the field input is terminated.
21. **TOGGLE-ANSI-INSERTION.** The TOGGLE-ANSI-INSERTION value changes the handling of following text characters. If terminal handling is currently in replacement mode, it is changed to ANSI insertion mode. If terminal handling is currently in RM or ANSI insertion mode, it is changed to replacement mode.

If a key sequence assigned to an exception code has also been assigned to any of the field editing facilities (excluding BACKSPACE, LEFT-ARROW, and RIGHT-ARROW), the editing action is performed and field editing will terminate.

We recommend that all terminal configurations include values to generate the RM/COBOL generic exception keys, which are shown in Table 34.

**Table 34: RM/COBOL Generic Exception Keys**

| Code | Generic Name | Code | Generic Name |
|---|---|---|---|
| 13 | Enter | 18 | Function 18 |
| 01 | Function 1 | 19 | Function 19 |
| 02 | Function 2 | 20 | Function 20 |
| 03 | Function 3 | 40 | Command |
| 04 | Function 4 | 41 | Attention |
| 05 | Function 5 | 52 | Up Arrow |
| 06 | Function 6 | 53 | Down Arrow |
| 07 | Function 7 | 54 | Home |
| 08 | Function 8 | 55 | New Line |
| 09 | Function 9 | 56 | Tab Left |
| 10 | Function 10 | 57 | Erase Right |
| 11 | Function 11 | 58 | Tab Right |
| 12 | Function 12 | 59 | Insert Line |
| 13 | Function 13 | 61 | Delete Line |
| 14 | Function 14 | 64 | Send |
| 15 | Function 15 | 83 | Help |
| 16 | Function 16 | 84 | Redo |
| 17 | Function 17 | | |

# TERM-INTERFACE Configuration Record

The terminal configuration records have different formats, depending on the type of terminal interface the runtime system uses: termcap, terminfo, or graphical user interface (GUI). The TERM-INTERFACE configuration record describes the format of records in this configuration file. The runtime system will use this information to correctly process the records and to ensure that it is the correct version of the runtime system (termcap, terminfo, or GUI) to be using the information.

The TERM-INTERFACE record must precede all other terminal interface configuration records except TERM-UNIT.

The TERM-INTERFACE record identifier is followed by one keyword that describes the format. The possible keywords are as follows:

1. **GUI.** The value of this keyword must be specified for use with RM/COBOL for Windows. It indicates that the terminal interface is a graphical user interface (GUI).
2. **TERMCAP.** This keyword specifies that the runtime system is expected to process the termcap database and handle all input and output to the terminal directly. TERM-INPUT configuration records use termcap input sequence.
3. **TERMINFO.** This keyword specifies that the runtime system is expected to use the terminfo database and handle all input and output to the terminal directly. TERM-INPUT configuration records use terminfo input sequence.
4. **WINDOWS.** This value is identical to the value of GUI.

**Note** The default terminal interface is GUI (or WINDOWS, which is equivalent) for Windows and is selected at installation for UNIX.

# TERM-UNIT Configuration Record

The TERM-UNIT configuration record is used to associate RM/COBOL units to stations (devices). The TERM-UNIT record identifier is followed by one or more keywords. If the keyword is allowed to have a value, it is followed by an equal sign (=) and the value.

**Note** The TERM-UNIT configuration record is supported only under UNIX.

The TERM-UNIT record must contain the PATH keyword if the UNIT keyword does not specify the default unit. The possible keywords are as follows:

1. **BPS.** This keyword specifies the Bits Per Second to which the communication port should be initialized when the unit is first accessed by the RM/COBOL program. The default for this keyword is the value to which the operating system has set the port. The valid values for this keyword are 50, 75, 110, 134.5, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, EXT-A and EXT-B (for external A and external B).
2. **CHARACTER-WIDTH.** This keyword specifies the character size, in bits, to which the communication port should be initialized when the unit is first accessed by the RM/COBOL program. The default value for this keyword is the value to which the operating system has set the port. The valid values for this keyword are 5, 6, 7 and 8.

3. **DEFINE-CONTROL-CHARACTERS.** This keyword specifies whether the terminal control characters, Ctrl+A through Ctrl+Z, are to be predefined in the unit's input sequences to terminate input and generate exception codes of 1 through 26. On the termcap and terminfo runtime systems, Ctrl+H, Ctrl+I, Ctrl+J, and Ctrl+M are always predefined to be Backspace, Tab, New Line, and Carriage Return, respectively.

   If the DEFINE-CONTROL-CHARACTERS keyword has the value YES, the characters are defined. If the DEFINE-CONTROL-CHARACTERS keyword has the value NO, they are not defined. The default value for this keyword is YES.

4. **MOVE-ATTR.** This keyword specifies the handling of terminals with positional attributes. On such terminals, attributes are implemented by placing special characters on the terminal screen. These attribute characters take up screen positions and overwrite any data there.

   If the MOVE-ATTR keyword has the value NO, it specifies that the attribute characters are placed at the line and position requested by the RM/COBOL program and the actual start of the field is moved to the right, accordingly.

   If the MOVE-ATTR keyword has the value YES, it specifies that the field should be placed in the line and position requested and the attributes moved to the left. The only time this will not be done is when the termcap capability database indicates that the attributes set by the attribute characters will not cross a line boundary or if the field starts in line 1, position 1 and the database does not indicate that the attribute will wrap from the bottom to the top of the screen.

   The default value for this keyword is NO.

5. **PARITY.** This keyword specifies the parity to which the communication port should be initialized when the unit is first accessed by the RM/COBOL program. The default value for this keyword is the value to which the operating system has set the port. The valid values of this keyword are: EVEN, ODD and NONE.

6. **PATH.** This keyword specifies the device pathname that will be used for this unit. This must be a tty port that currently has no other processes attached to it and has read and write privileges for the user who starts the run unit. This is a required field if the unit being described is not the default unit. If the unit being described is the default unit, this keyword must not be present.

7. **STOP-BITS.** This keyword specifies the number of stop bits that the communication port should send following transmitted characters. The default value for this keyword is the value to which the operating system has set the port. The valid values of this keyword are 1 and 2.

8. **TYPE.** This keyword specifies the terminal type connected to the port. This is the name used when searching the termcap or terminfo database. The default value for this keyword is the value of the TERM environment variable.

9. **UNIT.** This keyword specifies the RM/COBOL unit number being described and is either a decimal number between 0 and 255 or the value DEFAULT-UNIT. If the value is DEFAULT-UNIT, this TERM-UNIT record describes the unit that is used when no UNIT clause appears on the ACCEPT or DISPLAY statement. If it is a number, this TERM-UNIT record describes a particular UNIT number that will be used in an RM/COBOL ACCEPT or DISPLAY statement. The default value for this keyword is DEFAULT-UNIT.

# Default Configuration Files

The following topics illustrate samples of default configuration files for terminfo, termcap, and Windows systems:



## Termcap Example

The following records define the default configuration as provided by Liant on systems that use termcap.

```
RUN-ATTR accept-intensity=high beep=yes blink=yes
&display-intensity=high reverse=yes
RUN-ATTR error-message-destination=standard-error
RUN-OPTION b=264 k=display m=1
RUN-INDEX-FILES allocation-increment=8
RUN-INDEX-FILES data-compression=yes force-closed=no force-data=no
&force-disk=no
RUN-INDEX-FILES force-index=no key-compression=yes
RUN-SEQ-FILES default-type=binary
RUN-SEQ-FILES device-slewing-reserve=255
RUN-SEQ-FILES tab-stops=8,12,16,20,24,28,32,36,40,44,48,52,
&56,60,64,68,72
RUN-SORT intermediate-files=5 memory-size=16000
TERM-UNIT move-attr=no define-control-characters=yes
TERM-INTERFACE TERMCAP
TERM-INPUT action=left-arrow                    kl
TERM-INPUT action=right-arrow                   kr
TERM-INPUT action=backspace precedence=1        kb
TERM-INPUT action=set-rm-insertion              kI
TERM-INPUT action=delete-character              kD
TERM-INPUT action=erase-entire                  kC
TERM-INPUT action=screen-terminate code=01      k1
TERM-INPUT action=screen-terminate code=02      k2
TERM-INPUT action=screen-terminate code=03      k3
TERM-INPUT action=screen-terminate code=04      k4
TERM-INPUT action=screen-terminate code=05      k5
TERM-INPUT action=screen-terminate code=06      k6
TERM-INPUT action=screen-terminate code=07      k7
TERM-INPUT action=screen-terminate code=08      k8
TERM-INPUT action=screen-terminate code=09      k9
TERM-INPUT action=screen-terminate code=10      k;
TERM-INPUT code=11                              F1
TERM-INPUT code=12                              F2
TERM-INPUT code=13                              F3
TERM-INPUT code=14                              F4
TERM-INPUT code=15                              F5
TERM-INPUT code=16                              F6
TERM-INPUT code=17                              F7
TERM-INPUT code=18                              F8
TERM-INPUT code=19                              F9
TERM-INPUT code=20                              FA
TERM-INPUT code=21                              FB
```

```
TERM-INPUT code=22                              FC
TERM-INPUT code=23                              FD
TERM-INPUT code=24                              FE
TERM-INPUT code=25                              FF
TERM-INPUT code=26                              FG
TERM-INPUT code=27                              FH
TERM-INPUT code=28                              FI
TERM-INPUT code=29                              FJ
TERM-INPUT code=30                              FK
TERM-INPUT code=31                              FL
TERM-INPUT code=32                              FM
TERM-INPUT code=33                              FN
TERM-INPUT code=34                              FO
TERM-INPUT code=35                              FP
TERM-INPUT code=36                              FQ
TERM-INPUT code=37                              FR
TERM-INPUT code=38                              FS
TERM-INPUT code=39                              FT
TERM-INPUT code=40                              FU
TERM-INPUT code=41                              FV
TERM-INPUT code=42                              FW
TERM-INPUT code=43                              FX
TERM-INPUT code=44                              FY
TERM-INPUT code=45                              FZ
TERM-INPUT code=46                              Fa
TERM-INPUT code=47                              Fb
TERM-INPUT code=48                              Fc
TERM-INPUT code=49                              Fd
TERM-INPUT code=50                              Fe
TERM-INPUT code=51                              Ff
TERM-INPUT code=52                              Fg
TERM-INPUT code=53                              Fh
TERM-INPUT code=54                              Fi
TERM-INPUT code=55                              Fj
TERM-INPUT code=56                              Fk
TERM-INPUT code=57                              Fl
TERM-INPUT code=58                              Fm
TERM-INPUT code=59                              Fn
TERM-INPUT code=60                              Fo
TERM-INPUT code=61                              Fp
TERM-INPUT code=62                              Fq
TERM-INPUT code=63                              Fr
TERM-INPUT code=13 exception=no                 cr
TERM-INPUT code=40                              k0
TERM-INPUT code=41                              K3
TERM-INPUT code=49                              l0
TERM-INPUT action=screen-previous-field code=52 ku
TERM-INPUT code=53                              kd
TERM-INPUT action=screen-home code=54           kh
TERM-INPUT code=55                              nw
TERM-INPUT code=56                              K4
TERM-INPUT action=erase-remainder code=57       kE
TERM-INPUT code=58                              K5
TERM-INPUT code=59                              kA
TERM-INPUT code=61                              kL
TERM-INPUT code=64                              K2
TERM-INPUT code=67                              kP
TERM-INPUT code=68                              kN
TERM-INPUT code=82                              K1
TERM-INPUT code=83 precedence=1                 %1
TERM-INPUT code=84                              %0
```

```
PRINT-ATTR auto-line-feed=no
PRINT-ATTR form-feed-available=yes top-of-form-at-close=no
DEFINE-DEVICE device=PRINTER path="lp -s" pipe=yes
DEFINE-DEVICE device=PRINTER1 path="lp -s" pipe=yes
DEFINE-DEVICE device=TAPE path=/dev/rtp tape=yes
```

**Note** The actual path values for the DEFINE-DEVICE records are system-dependent and may differ from the values shown in the example.

## Terminfo Example

The following records define the default configuration as provided by Liant on systems that use terminfo.

```
RUN-ATTR accept-intensity=high beep=yes blink=yes
&display-intensity=high reverse=yes
RUN-ATTR error-message-destination=standard-error
RUN-OPTION b=264 k=display m=1
RUN-INDEX-FILES allocation-increment=8
RUN-INDEX-FILES data-compression=yes force-closed=no force-data=no
&force-disk=no
RUN-INDEX-FILES force-index=no key-compression=yes
RUN-SEQ-FILES default-type=binary
RUN-SEQ-FILES device-slewing-reserve=255
RUN-SEQ-FILES tab-stops=8,12,16,20,24,28,32,36,40,44,48,52,
&56,60,64,68,72
RUN-SORT intermediate-files=5 memory-size=16000
TERM-UNIT move-attr=no define-control-characters=yes
TERM-INTERFACE TERMINFO
TERM-INPUT action=left-arrow                    kcub1
TERM-INPUT action=right-arrow                   kcuf1
TERM-INPUT action=backspace precedence=1        kbs
TERM-INPUT action=set-rm-insertion              kich1
TERM-INPUT action=delete-character              kdch1
TERM-INPUT action=erase-entire                  kclr
TERM-INPUT action=screen-terminate code=01      kf1
TERM-INPUT action=screen-terminate code=02      kf2
TERM-INPUT action=screen-terminate code=03      kf3
TERM-INPUT action=screen-terminate code=04      kf4
TERM-INPUT action=screen-terminate code=05      kf5
TERM-INPUT action=screen-terminate code=06      kf6
TERM-INPUT action=screen-terminate code=07      kf7
TERM-INPUT action=screen-terminate code=08      kf8
TERM-INPUT action=screen-terminate code=09      kf9
TERM-INPUT action=screen-terminate code=10      kf10
TERM-INPUT code=11                              kf11
TERM-INPUT code=12                              kf12
TERM-INPUT code=13                              kf13
TERM-INPUT code=14                              kf14
TERM-INPUT code=15                              kf15
TERM-INPUT code=16                              kf16
TERM-INPUT code=17                              kf17
TERM-INPUT code=18                              kf18
TERM-INPUT code=19                              kf19
TERM-INPUT code=20                              kf20
TERM-INPUT code=21                              kf21
TERM-INPUT code=22                              kf22
TERM-INPUT code=23                              kf23
```

```
TERM-INPUT code=24                                kf24
TERM-INPUT code=25                                kf25
TERM-INPUT code=26                                kf26
TERM-INPUT code=27                                kf27
TERM-INPUT code=28                                kf28
TERM-INPUT code=29                                kf29
TERM-INPUT code=30                                kf30
TERM-INPUT code=31                                kf31
TERM-INPUT code=32                                kf32
TERM-INPUT code=33                                kf33
TERM-INPUT code=34                                kf34
TERM-INPUT code=35                                kf35
TERM-INPUT code=36                                kf36
TERM-INPUT code=37                                kf37
TERM-INPUT code=38                                kf38
TERM-INPUT code=39                                kf39
TERM-INPUT code=40                                kf40
TERM-INPUT code=41                                kf41
TERM-INPUT code=42                                kf42
TERM-INPUT code=43                                kf43
TERM-INPUT code=44                                kf44
TERM-INPUT code=45                                kf45
TERM-INPUT code=46                                kf46
TERM-INPUT code=47                                kf47
TERM-INPUT code=48                                kf48
TERM-INPUT code=49                                kf49
TERM-INPUT code=50                                kf50
TERM-INPUT code=51                                kf51
TERM-INPUT code=52                                kf52
TERM-INPUT code=53                                kf53
TERM-INPUT code=54                                kf54
TERM-INPUT code=55                                kf55
TERM-INPUT code=56                                kf56
TERM-INPUT code=57                                kf57
TERM-INPUT code=58                                kf58
TERM-INPUT code=59                                kf59
TERM-INPUT code=60                                kf60
TERM-INPUT code=61                                kf61
TERM-INPUT code=62                                kf62
TERM-INPUT code=63                                kf63
TERM-INPUT code=13 exception=no                   cr
TERM-INPUT code=40                                kf0
TERM-INPUT code=41                                ka3
TERM-INPUT code=49                                lf0
TERM-INPUT action=screen-previous-field code=52 kcuu1
TERM-INPUT code=53                                kcud1
TERM-INPUT action=screen-home code=54             khome
TERM-INPUT code=55                                nel
TERM-INPUT code=56                                kc1
TERM-INPUT action=erase-remainder code=57         kel
TERM-INPUT code=58                                kc3
TERM-INPUT code=59                                kil1
TERM-INPUT code=61                                kdl1
TERM-INPUT code=64                                kb2
TERM-INPUT code=67                                kpp
TERM-INPUT code=68                                knp
TERM-INPUT code=82                                ka1
TERM-INPUT code=83 precedence=1                   khlp
TERM-INPUT code=84                                krdo
PRINT-ATTR auto-line-feed=no
```

```
PRINT-ATTR form-feed-available=yes top-of-form-at-close=no
DEFINE-DEVICE device=PRINTER path="lp -s" pipe=yes
DEFINE-DEVICE device=PRINTER1 path="lp -s" pipe=yes
DEFINE-DEVICE device=TAPE path=/dev/rtp tape=yes
```

**Note** The actual path values for the DEFINE-DEVICE records are system-dependent and may differ from the values shown in the example.

## Windows Example

The following records define the default configuration provided by Liant for the Windows operating system.

```
RUN-ATTR accept-intensity=high beep=yes blink=yes
&display-intensity=high reverse=yes
RUN-ATTR error-message-destination=standard-error
RUN-OPTION b=264 k=display m=1
RUN-INDEX-FILES allocation-increment=8
RUN-INDEX-FILES data-compression=yes force-closed=no force-data=no
&force-disk=no
RUN-INDEX-FILES force-index=no key-compression=yes
RUN-SEQ-FILES default-type=binary
RUN-SEQ-FILES device-slewing-reserve=255
RUN-SEQ-FILES tab-stops=8,12,16,20,24,28,32,36,40,44,48,52,
&56,60,64,68,72
RUN-SORT intermediate-files=5 memory-size=16000
PRINT-ATTR auto-line-feed=no
PRINT-ATTR form-feed-available=yes top-of-form-at-close=no
TERM-INTERFACE GUI
TERM-ATTR Data-Characters=32,126
TERM-INPUT Code=1                                SOH
TERM-INPUT Code=2                                STX
TERM-INPUT Action=Control-Break                  ETX
TERM-INPUT Code=4                                EOT
TERM-INPUT Code=5                                ENQ
TERM-INPUT Code=6                                ACK
TERM-INPUT Code=7                                BEL
TERM-INPUT Action=Backspace                      BS
TERM-INPUT Code=9                                HT
TERM-INPUT Code=10                               LF
TERM-INPUT Code=11                               VT
TERM-INPUT Code=12                               FF
TERM-INPUT Code=13 Exception=No                  CR
TERM-INPUT Code=14                               SO
TERM-INPUT Code=15                               SI
TERM-INPUT Code=16                               DLE
TERM-INPUT Code=17                               DC1
TERM-INPUT Code=18                               DC2
TERM-INPUT Code=19                               DC3
TERM-INPUT Code=20                               DC4
TERM-INPUT Code=21                               NAK
TERM-INPUT Code=22                               SYN
TERM-INPUT Code=23                               ETB
TERM-INPUT Code=24                               CAN
TERM-INPUT Code=25                               EM
TERM-INPUT Code=26                               SUB
TERM-INPUT Action=Screen-Escape Code=27          ESC
```

```
TERM-INPUT Code=28                                FS
TERM-INPUT Code=29                                GS
TERM-INPUT Code=30                                RS
TERM-INPUT Code=31                                US
TERM-INPUT Code=27                                WCNT [
TERM-INPUT Action=Control-Break                   NUL NUL
TERM-INPUT Action=Backspace                       NUL BS
TERM-INPUT Code=58                                NUL HT
TERM-INPUT Code=56                                NUL WSFT HT
TERM-INPUT Code=13 Exception=No                   NUL CR
TERM-INPUT Action=Erase-Remainder Code=57         WSFT WCNT E
TERM-INPUT Code=58                                WSFT WCNT R
TERM-INPUT Code=59                                WSFT WCNT I
TERM-INPUT Code=49                                WSFT WCNT P
TERM-INPUT Code=41                                WSFT WCNT A
TERM-INPUT Code=64                                WSFT WCNT S
TERM-INPUT Code=61                                WSFT WCNT D
TERM-INPUT Action=Erase-Remainder Code=13 Exception=No WSFT WCNT K
TERM-INPUT Code=40                                WSFT WCNT C
TERM-INPUT Code=55                                WSFT WCNT N
TERM-INPUT Action=Screen-Home Code=54             NUL WHOM
TERM-INPUT Action=Screen-Previous-Field Code=52 NUL WUP
TERM-INPUT Code=67                                NUL WPGU
TERM-INPUT Action=Left-Arrow                      NUL WLFT
TERM-INPUT Action=Right-Arrow                     NUL WRGT
TERM-INPUT Code=82                                NUL WEND
TERM-INPUT Code=53                                NUL WDWN
TERM-INPUT Code=68                                NUL WPGD
TERM-INPUT Action=Set-RM-Insertion                NUL WINS
TERM-INPUT Action=Delete-Character                NUL WDEL
TERM-INPUT Action=Screen-Terminate Code=1         NUL WF1
TERM-INPUT Action=Screen-Terminate Code=2         NUL WF2
TERM-INPUT Action=Screen-Terminate Code=3         NUL WF3
TERM-INPUT Action=Screen-Terminate Code=4         NUL WF4
TERM-INPUT Action=Screen-Terminate Code=5         NUL WF5
TERM-INPUT Action=Screen-Terminate Code=6         NUL WF6
TERM-INPUT Action=Screen-Terminate Code=7         NUL WF7
TERM-INPUT Action=Screen-Terminate Code=8         NUL WF8
TERM-INPUT Action=Screen-Terminate Code=9         NUL WF9
TERM-INPUT Action=Screen-Terminate Code=10        NUL WF10
TERM-INPUT Code=11                                NUL WSFT WF1
TERM-INPUT Code=12                                NUL WSFT WF2
TERM-INPUT Code=13                                NUL WSFT WF3
TERM-INPUT Code=14                                NUL WSFT WF4
TERM-INPUT Code=15                                NUL WSFT WF5
TERM-INPUT Code=16                                NUL WSFT WF6
TERM-INPUT Code=17                                NUL WSFT WF7
TERM-INPUT Code=18                                NUL WSFT WF8
TERM-INPUT Code=19                                NUL WSFT WF9
TERM-INPUT Code=20                                NUL WSFT WF10
TERM-INPUT Code=21                                NUL WCNT WF1
TERM-INPUT Code=22                                NUL WCNT WF2
TERM-INPUT Code=23                                NUL WCNT WF3
TERM-INPUT Code=24                                NUL WCNT WF4
TERM-INPUT Code=25                                NUL WCNT WF5
TERM-INPUT Code=26                                NUL WCNT WF6
TERM-INPUT Code=27                                NUL WCNT WF7
TERM-INPUT Code=28                                NUL WCNT WF8
```

```
TERM-INPUT Code=29                                  NUL WCNT WF9
TERM-INPUT Code=30                                  NUL WCNT WF10
TERM-INPUT Code=31                                  NUL WSFT WCNT WF1
TERM-INPUT Code=32                                  NUL WSFT WCNT WF2
TERM-INPUT Code=33                                  NUL WSFT WCNT WF3
TERM-INPUT Code=34                                  NUL WSFT WCNT WF4
TERM-INPUT Code=35                                  NUL WSFT WCNT WF5
TERM-INPUT Code=36                                  NUL WSFT WCNT WF6
TERM-INPUT Code=37                                  NUL WSFT WCNT WF7
TERM-INPUT Code=38                                  NUL WSFT WCNT WF8
TERM-INPUT Code=39                                  NUL WSFT WCNT WF9
TERM-INPUT Code=40                                  NUL WSFT WCNT WF10
TERM-INPUT Code=65                                  NUL WCNT WLFT
TERM-INPUT Code=66                                  NUL WCNT WRGT
TERM-INPUT Code=83                                  NUL WCNT WEND
TERM-INPUT Code=70                                  NUL WCNT WPGD
TERM-INPUT Code=81                                  NUL WCNT WHOM
TERM-INPUT Code=71                                  WSFT WCNT 49
TERM-INPUT Code=72                                  WSFT WCNT 50
TERM-INPUT Code=73                                  WSFT WCNT 51
TERM-INPUT Code=74                                  WSFT WCNT 52
TERM-INPUT Code=75                                  WSFT WCNT 53
TERM-INPUT Code=76                                  WSFT WCNT 54
TERM-INPUT Code=77                                  WSFT WCNT 55
TERM-INPUT Code=78                                  WSFT WCNT 56
TERM-INPUT Code=79                                  WSFT WCNT 57
TERM-INPUT Code=80                                  WSFT WCNT 48
TERM-INPUT Code=85                                  WSFT WCNT -
TERM-INPUT Code=87                                  WSFT WCNT =
TERM-INPUT Code=69                                  NUL WCNT WPGU
TERM-INPUT Code=11                                  NUL WF11
TERM-INPUT Code=12                                  NUL WF12
DEFINE-DEVICE  DEVICE=PRINTER   PATH=DEFAULT
DEFINE-DEVICE  DEVICE=PRINTER1  PATH=",LPT1"
DEFINE-DEVICE  DEVICE=PRINTER2  PATH=",LPT2"
DEFINE-DEVICE  DEVICE=PRINTER3  PATH=",LPT3"
DEFINE-DEVICE  DEVICE=PRINTER4  PATH=",LPT4"
DEFINE-DEVICE  DEVICE=PRINTER5  PATH=",LPT5"
DEFINE-DEVICE  DEVICE=PRINTER6  PATH=",LPT6"
DEFINE-DEVICE  DEVICE=PRINTER7  PATH=",LPT7"
DEFINE-DEVICE  DEVICE=PRINTER8  PATH=",LPT8"
DEFINE-DEVICE  DEVICE=PRINTER9  PATH=",LPT9"
DEFINE-DEVICE  DEVICE=PRINTER?  PATH=DYNAMIC
```

# Chapter 11: Instrumentation

RM/COBOL provides a method for examining the performance of RM/COBOL programs at the statement level. This facility—called Instrumentation—involves a two-step process. First, program data must be gathered. Instrumentation provides the tools for data gathering. Next, the gathered data must be analyzed in a manner consistent with your specific requirements. The delivered RM/COBOL Instrumentation contains one example of a data analysis program, called **analysis**. This program also can be used as a starting point for creating your own data analysis program.

The data gathered by Instrumentation and reported by **analysis** can be used during program development to optimize program flow, identify bugs caused by run-away loop control, and improve program integrity by pinpointing unexecuted program code.

## Invoking Instrumentation

Instrumentation is invoked when you enter the I Runtime Command Option (see page 181). If you intend to use **analysis** as well, all programs in the run unit should be compiled with the L Compile Command Option (see page 145).

Keep in mind that the runtime system needs additional memory when the I Runtime Command Option is used. Furthermore, each program in the run unit requires additional memory as it is loaded by the Runtime Command or by a CALL statement (see the “CALL Statement” section in Chapter 6: *Procedure Division Statements* of the *RM/COBOL Language Reference Manual*). The exact amount of required memory depends on the number of lines in the PROCEDURE DIVISION, as described in the next section.

# Data Collection

For each program in a run unit, a file is generated that contains the following information:

- PROGRAM-ID value
- Line number of the PROCEDURE DIVISION header
- Total number of source lines in the program
- Total number of statements executed in the program
- Number of times each statement was executed
- Number of times each paragraph and section were executed

This information is gathered in a data structure that could be described by the RM/COBOL record description, shown in Figure 39.

**Figure 39: Sample Data Structures Description**

```
01  PROGRAM-IDENTIFICATION.
    02  PROGRAM-ID-VALUE                    PIC X(30).
    02  PROCEDURE-DIVISION-LINE             PIC 9(8) BINARY.
    02  SOURCE-LINE-COUNT                   PIC 9(8) BINARY.
    02  TOTAL-STATEMENTS-EXECUTED           PIC 9(8) BINARY.
    02  TOTAL-STATEMENTS-PITCHED            PIC 9(8) BINARY.
    02                                      PIC X(14).
    02  OCCURS 1 TO 65000
        DEPENDING ON number of procedure lines.
        03  OCCURS 2.
            04  STATEMENT-TYPE              PIC X(1).
                88  NO-STATEMENT-OR-NOT-EXECUTED
                                            VALUE LOW-VALUE.
                88  SECTION-COUNT          VALUE 'X'.
                88  PARAGRAPH-COUNT        VALUE 'P'.
                88  STATEMENT-COUNT        VALUE 'S'.
            04  COUNT-VALUE                PIC 9(9) BINARY.
```

The size of the required structure may be calculated as follows:

```
n = source-line-count - procedure-division-line + 1
size (in bytes) = 60 + (n * 10)
```

The data structure for a program is allocated when the first statement in that program is executed. The data structure remains allocated even when the associated program is canceled.

As with the RM/COBOL Interactive Debugger, Instrumentation treats section names, paragraph names and procedural statements as statements for the purpose of data gathering. For the definitions of statements and line and intraline numbers, see Statements (on page 246) and Line and Intraline Numbers (on page 247). Likewise, programs within the run unit that were compiled with the Q Compile Command Option (see page 148) are invisible to Instrumentation. No information will be gathered nor file created for such programs.

As each statement is executed, Instrumentation adds 1 to the total number of statements executed for that program (TOTAL-STATEMENTS-EXECUTED in the record description above). If the executing statement is the first or second statement on a line, 1 is added to the count of executions (COUNT-VALUE (line, intraline + 1)

in the record descriptor above) and the appropriate statement type (STATEMENT-TYPE (line, intraline + 1) in the record descriptor above) is set to true for that statement. If the executing statement is the third or subsequent statement on a line, 1 is added to a count of statements not counted individually (TOTAL-STATEMENTS-PITCHED in the record description above).

When program execution completes—normally or abnormally—each data structure is written to disk. The name of the disk file is constructed from the first eight characters of the PROGRAM-ID value, concatenated to the file extension **.CNT**. If the program-name is less than eight characters, the entire name is concatenated to the file extension. For example, for a program named **example**, the data collection file would be named **example.CNT**. If the name of the program was **generates**, the data collection file would be named **generate.CNT**.

The runtime directory search sequence is used when writing these files. For more details, see Directory Search Sequences on UNIX (on page 24) and Directory Search Sequences on Windows (on page 61). An existing file within the directories named in the RUNPATH specification will be replaced by a data collection file with the same name. If no file with the same name exists, a new data collection file will be created.

Instrumentation either replaces or creates new data collection files for each invocation of a run unit. Historical information is not maintained from run unit to run unit. This is a function of the **analysis** program, which processes the data collection files after each run unit.

Instrumentation writes messages to the screen as each data collection file is written. The form of the message when no errors occur is the following:

```
name.CNT Opening Writing Closing
```

where, *name* is derived from the first eight characters of the PROGRAM-ID, as described above. Should an error occur during the opening, writing or closing of the file, an error message, as described in Appendix A: *Runtime Messages* (on page 357), appears after the name of the operation encountering the error. If Opening receives an error, Writing and Closing are not attempted and will not be included in the message.

# Data Analysis

As was mentioned at the start of this chapter, a data analysis program—named **analysis**—is provided with RM/COBOL Instrumentation. Use this program as delivered, or modify it to fit your precise requirements. As delivered, **analysis** provides the following:

- Statement count data from the **.CNT** file is added to the statement count data gathered from earlier invocations of Instrumentation. This keeps a total count of the statements executed.
- Statement count data from the total count file is merged with the listing file, creating a new listing file showing execution counts for each statement.
- A summary of execution counts for each paragraph and section is appended to the new listing file.

To use **analysis** as delivered, first compile the program with the RM/COBOL Compile Command:

```
rmcobol analysis A L X
```

This creates a source listing, allocation map, and cross reference on a listing file named **analysis.lst** in the directory containing **analysis.cbl**. An RM/COBOL object file, **analysis.cob**, is also created in that directory. See Chapter 6: *Compiling* (on page 135) for more information about the compilation options and parameters.

Next, execute the **analysis** program by entering the RM/COBOL Runtime Command:

```
runcobol analysis [A='path']
```

The optional A='*path*' may be used to supply a pathname of the directory where the merged listing files created by **analysis** are to reside. If this parameter is omitted, the RUNPATH specification is used. See Chapter 7: *Running* (on page 177) for more information about the Runtime Command options and parameters.

The **analysis** program is listing-file driven, that is, the filenames of one or more listing files are supplied by the user. These listing files are then processed by **analysis**, one at a time, as follows:

1. The merged listing file is opened using the same filename as the listing file, with the extension **.HST**.
2. The source listing section is scanned to find the PROGRAM-ID paragraph.
3. The program-name found in the PROGRAM-ID paragraph is used to construct the filename of the **.CNT** file.
4. The **.CNT** file is opened.
5. The program-name used in the PROGRAM-ID paragraph is used to construct the filename of the total count (**.TOT**) file.
6. The **.TOT** file is opened.
7. The **.TOT** file is updated (or created if it does not exist) by adding the statement counts from the **.CNT** file to the corresponding statement counts in the **.TOT** file.
8. The source listing section is reformatted merging the statement counts from the **.CNT** or **.TOT** file with the source listing.
9. Summary information about the program is written to the merged file.
10. All listing file information between the end of one program and the beginning of another in the file or end of file is copied to the merged file.
11. If another program exists in the listing file, this process is repeated starting at Step 2.
12. If the name of another listing file is supplied by the user, this process is repeated starting at Step 1.
13. **analysis** terminates.

When prompted to supply a listing filename, enter only the filename portion of the name. The **.lst** extension is supplied automatically. To indicate there are no more listing files, press Enter without typing a name. Use redirected input on the Runtime Command to automate the entry of filenames where repeated runs are desired.

Figure 40 illustrates an excerpt from a merged listing, as it pertains to **analysis**.

**Figure 40: Excerpt of a Merged Listing**

```
Count1 Count2  LINE  PG/LN  A..B..2..3..4..5..6..7..ID...8   FPERF
                     1        IDENTIFICATION DIVISION.
                     2        PROGRAM-ID.                   FPERF.
                                      .
                                      .
                                      .
205                 77        PROCEDURE DIVISION.
  1                 78        MAIN SECTION.
                                      .
                                      .
                                      .
  1                 91        A.
  1                 92          MOVE ZEROS TO COUNT-ER, ERROR-COUNT.
  1                 93          MOVE SPACE TO PRINT-RECORD.
  1    1            94          MOVE "PATH" TO PATH. MOVE "A" TO P-R.
  1                 95          PERFORM B.
  1                 96          IF COUNT-ER = 1
  1                 97             WRITE PRINT-RECORD FROM PASS-LINE
                    98           ELSE
                    99             WRITE PRINT-RECORD FROM FAIL-LINE
                   100               ADD 1 TO ERROR-COUNT.
  1                101          MOVE "B" TO P-R.
                   102
  8    8           103        B.WRITE PRINT RECORD FROM PASS-LINE.
  8                104          ADD 1 TO COUNT-ER.
Summary Statistics for FPERF
The program contains 242 lines, of which 166 are procedure division lines.
The procedure division has 92 lines which had COBOL verbs executed at least
once, and 26 lines which had procedure-names executed at least once.
There were 205 verb executions counted, of which 151 were COBOL statements
and 54 were procedures.
                                      .
                                      .
                                      .
Count      %     LineParagraph/Section
    8    14.81    103   B
    6    11.11    197   K
                                      .
                                      .
                                      .
    1    1.85     200   A
```

Count1, associated with the PROCEDURE DIVISION line, is the total number of statements executed in this program. Count2, associated with the PROCEDURE DIVISION line, is the total number of statements pitched (unattributed) in this program. For all other lines in the Procedure Division, Count1 is the number of times the section, paragraph or statement was executed. Count2 is the number of times the next statement on the line was executed. If there are more than two statements on the line, the execution counts of the remaining statements are not displayed.

If the line was not referenced or executed, Count1 and Count2 will be blank. This indicates that the line did not contain a verb, or that it is code that was never executed.

Some parts of the merged listing file may be suppressed with the S Runtime Command Option. In this case, the Runtime Command would be:

```
runcobol analysis S=xpnuc [A='path']
```

The values of *x*, *p*, *n*, *u*, and *c* have the following results:

- *x* = 1 excludes all lines in the original listing file from the merged listing file; *x* = 0 includes these lines. *x* = 0 is the default.
- *p* = 1 excludes the paragraph and section execution counts from the summary information; *p* = 0 includes these counts. *p* = 0 is the default.
- *n* = 1 excludes all lines from the merged listing file that are not contained in the Procedure Division of the source listing portion; *n* = 0 includes such lines. *n*= 0 is the default. *n* has no additional effect if *x* = 1.
- *u* = 1 suppresses the update of the total execution counts derived from the total count file; *u* = 0 allows the update. *u* = 0 is the default.
- *c* = 1 produces the merged listing file from the **.CNT** file. *c* = 0 produces the merged listing file from the **.TOT** file. *c* = 0 is the default. *c* has no effect if *x* = 1.

For example:

```
runcobol analysis S=10101
```

produces a merge listing file with only paragraph and section execution counts and some summary information from the **.CNT** file but updates historical data for all statements.

# Appendix A: Runtime Messages

This appendix presents the types of messages generated during program execution, including those generated following normal termination as well as those generated when an error occurs.

## Error Message Types

**Data Reference**, **Procedure**, **Input/Output**, **Sort-Merge**, **Message Control**, **Configuration**, and **Initialization** errors have error numbers along with the error messages to help pinpoint the error being diagnosed. See the discussion of Error Message Format (on page 358).

A **Traceback** message traces back through one or more calling programs when an error occurs within a called subprogram. The traceback traces the path from the statement causing the error through all programs currently active in the run unit.

An **Internal** error message indicates that an inconsistency not normally caused by a flaw in the source program has been detected. The numbers within the error message are needed by Liant technical support services should an internal error occur.

An **Operator-Requested Termination** error occurs when an operator ends execution by pressing the Ctrl and Break keys (Interrupt key under UNIX).

A **COBOL Normal Termination** message is displayed when program execution terminates successfully.

# Error Message Format

The different types of messages use the same general format:

```
COBOL type error code at line number in
      program prog-id compiled date time
```

*type* is one of the following types of messages:

- Data reference
- Procedure
- Input/output
- Internal
- Traceback
- Operator-requested termination
- Sort-merge
- Message control
- Configuration
- Initialization

*code* is as defined in the appropriate sections of this appendix. Following the *code* in a procedure error, a parenthesized brief text description of the error is provided. This text description is shown in the descriptions of the procedure errors later in this appendix.

*number* identifies a particular line in the Procedure Division of the source program. It is the line in which the statement being referred to starts, and it can be looked up in the leftmost column (labeled "Line") of the source listing produced by the compiler. If a question mark is shown in this position, the following *prog-id* field refers to a machine language subprogram, or indicates that a valid line number has not been established following an Interactive Debug R (Resume) Command (on page 272).

If the program has been compiled with the Q Compile Command Option, line numbers are not available. Instead, the statement address, which is shown under the "Debug" heading in the listing, will be displayed. The statement address consists of a segment number and a segment offset and can be distinguished from a line number since the segment offset is always displayed as a six-digit number (with leading zeros, if necessary). The segment number is not displayed if it is zero. If a segment number is present, it precedes the segment offset and the two are separated by a slash (/) character.

**Note** The statement address in the error message may not match exactly any of the statement addresses in the program listing. If the statement address in the error message does match a statement address in the program listing, the error condition may have been caused by the statement whose address is just prior to the error address.

*prog-id* identifies the program interrupted in order to produce this message. It has the following format:

```
program-name (pathname.ext)
```

*program-name* is taken from the PROGRAM-ID paragraph of the source program.

*pathname.ext* is the fully qualified pathname of the object library in which the object program resides.

*date* and *time* are the date and time the program was compiled. They correspond exactly to the date and time printed on the program listing.

**Note 1** Traceback and operator-requested termination messages do not include the "error *code*" portion of this message.

**Note 2** The format of configuration and initialization errors does not precisely conform to the format shown above. See the discussion of configuration errors (on page 284).

# Data Reference Errors

Data reference errors include invalid data types, improper data definitions, improper data values and illegal subscripting.

| Number | Description |
|---|---|
| 101 | For one of the following reasons, no operand exists corresponding to the referenced Linkage Section item:<br>1. There are more data items specified in the Procedure Division header than are specified in the USING phrase of the CALL statement in the calling program.<br>2. The Procedure Division header in the first (or main) program in the run unit specifies more than one data item (for more details, see the discussion of the A Runtime Command Option that begins on page 182). This is just a special case of reason 1 since the main program is called with only one argument.<br>3. The CALL statement in the calling program specified OMITTED for the argument corresponding to the Procedure Division header argument for the Linkage Section data item in the called program.<br>**Note** This error does not occur if the Linkage Section data item is referenced in the ADDRESS OF special register or in an ADDRESS OF phrase in a SET statement. Thus, this error can be prevented by first testing if ADDRESS OF *identifier-1* NOT = NULL before attempting to reference *identifier-1* directly. |

| Number | Description |
|---|---|
| 102 | A reference to a variable length group is illegal because the value in the DEPENDING data item (*data-name-1*) is less than the minimum value (*integer-1*) or greater than the maximum value (*integer-2*) in the OCCURS clause. |
| 103 | An identifier or literal referenced in an INSPECT CONVERTING statement is illegal for one of the following reasons:<br>1. The source translation template (*identifier-6* or *literal-4*) contains multiple occurrences of the same value.<br>2. The source translation template (*identifier-6* or *literal-4*) does not have the same length as the destination translation template (*identifier-7* or *literal-5*).<br>3. The destination translation template (*literal-5*) is figurative and its length is not one. |
| 104 | A reference to a data item is illegal for one of the following reasons:<br>1. The computed composite subscript value for a subscripted reference has a value that is negative, zero or exceeds the maximum value for the referenced item.<br>2. There is a reference to a Linkage Section data item that is a formal argument whose description specifies more characters than are present in the corresponding operand in the USING or GIVING phrases of the CALL statement that called the current called program.<br>3. There is a reference to a Linkage Section data item in the first (or main) program in the run unit whose description specifies more characters than are supplied by the A Runtime Command Option.<br>4. There is a reference to a Linkage Section data item that is a based linkage record whose description specifies more characters than are present in the area of memory covered by the pointer value that was used to set the base address of the record.<br>5. There is a reference to a Linkage Section data item that is a based linkage record and the offset value for the base address has been set outside the area of memory covered by the address and length fields of the base address. That is, a Format 6 SET statement has set the pointer offset value outside the area of memory covered by the pointer data item. In this case, the error occurs not when the Format 6 SET statement is executed, but when the resultant pointer value is used as the base address of a based linkage record. |
| 105 | A subscript calculation overflowed or underflowed. |

| Number | Description |
|---|---|
| 106 | An index-name value indicates more than 65535 occurrences. |
| 107 | A reference modification is illegal for one of the following reasons:<br>1. A reference modification offset value is less than or equal to zero, or is greater than the length of the data item being reference modified.<br>2. A reference modification length value is less than or equal to zero or is greater than the remaining length of the data item being reference modified after application of the offset value. |
| 108 | The referenced Linkage Section data item (which is other than one associated with an argument listed in the USING or GIVING phrases of the Procedure Division header), has a null base address because of one of the following reasons:<br>1. The base address has never been set during this run unit.<br>2. The base address has been explicitly set to NULL or to a pointer data item with a null value during this run unit.<br>**Note** This error does not occur if the Linkage Section data item is referenced in the ADDRESS OF special register or in an ADDRESS OF phrase in a SET statement. Thus, this error can be prevented by first testing if ADDRESS OF *identifier-1* NOT = NULL before attempting to reference *identifier-1* directly. |
| 110 | A reference to a data item is illegal because the base address for the data item has been set to a pointer value, other than NULL, that does not point to memory that the program may access. This error occurs when the based linkage item is referenced after, but not at, the time the bad base address is established in a Format 6 SET statement. |

# Procedure Errors

Procedure errors include improper program structure or invalid calls.

| Number | Description |
|---|---|
| 201 | (canceling active program) A CANCEL statement has attempted to cancel a program that is still active. That is, a program that has called, directly or indirectly, the program attempting the cancel. |
| 202 | (program-name equal spaces) The program-name on a CALL statement has a value that is equal to spaces. |
| 203 | (calling library by file-name) The program-name on the Runtime Command or CALL statement does not match any of the PROGRAM-ID names in any library but does match a valid RM/COBOL library object filename. The call-by-filename technique is valid only for single-program object files. |
| 204 | (program not found) The program-name on the Runtime Command or CALL statement does not match any of the PROGRAM-ID names in any library and does not match a valid RM/COBOL object filename or non-COBOL executable file. Note that an object program with a higher object version number than that supported by the runtime system is not considered a valid program; in this case, error message 233 will also be displayed. For more information, see Appendix H: *Object Versions* (on page 617). When error 204 terminates execution, error messages for any load errors on files considered a candidate for loading because of the RM/COBOL extension search are displayed along with the full pathname of the candidate file.<br><br>**Note** The RM/COBOL extension search continues after a load error and, if a successful load occurs for a given extension, no errors are displayed nor is the ON EXCEPTION path taken for a CALL statement. In contrast, the RM/COBOL search of the RUNPATH for any given extension stops at the first file found, if any. If the desired file could be found later in the RUNPATH, the problem must be fixed by deleting the file that will not load, replacing that file with a file that will load successfully, or changing the RUNPATH value so that the desired file is found earlier in the RUNPATH path search sequence.<br><br>Under Windows, if the CALL statement specified SYSTEM, this error can occur when the external routine SYSTEM was successfully found and loaded, but the command processor required by SYSTEM could not be found. This can occur when the COMSPEC environment variable is not defined or its value contains an invalid drive, path, or filename. This error can also indicate that the length of the parameter passed to SYSTEM exceeds the limits specified in the documentation of SYSTEM (on page 578). |

| Number | Description |
|---|---|
| 204 (*Cont.*) | Under Windows, if the CALL statement specified a DLL file that does not export either of the special entry points **RM_EntryPoints** or **RM_EnumEntryPoints** and does not contain a nonresident ordinal one entry point, this error occurs. See "Preparing C Subprograms" in Appendix G: *Non-COBOL Subprogram Internals for Windows* of the *CodeBridge* manual for information about calling a non-COBOL support module by file name, as opposed to loading it as a library of program names.<br><br>Under UNIX and Windows, if a non-COBOL support module specifies a name in the **EntryPointName** entry of the subprogram name table that is not an exported symbol for the support module, this error occurs. See C Program Name Table Structure" in Appendixes G (for Windows) and H (for UNIX) of the *CodeBridge* manual for additional information about the program name table. When this is the cause of the error, a message is displayed indicating the unknown symbol. Contact the supplier of the support module for a corrected version of the module (all names are checked on any load of the module, other than a "call-by-filename" load on Windows, so this should not occur except during module development).<br><br>For UNIX and Windows, this error can indicate problems finding or searching a directory specified in the LIBRARY-PATH keyword of the RUN-OPTION configuration record. In this case, the pathname displayed contains a trailing directory separator character, which is "\" (on Windows) or "/" (on UNIX). The trailing directory separator character indicates that the directory pathname caused the problem. If this error is caused because of an attempt to load a library found in the directory, the full pathname of the library file itself is displayed, without a trailing directory separator character.<br><br>If the CALL statement specified the ON EXCEPTION or ON OVERFLOW phrase, this procedure error is suppressed and execution continues with the imperative statement in the ON EXCEPTION or ON OVERFLOW phrase. |
| 205 | (calling active program) A CALL statement has attempted to call a program that is still active. An active program is one that has called, directly or indirectly, the program attempting the call in error. |
| 206 | (object file not valid) The called filename is not a valid RM/COBOL object file. The file may be corrupt or contain information that makes it invalid for this run unit. A corrupt file could be caused by a system failure or abnormal termination of the RM/COBOL compiler. The file also could be invalid for this run unit if the registration information is not correct or if the object was compiled with features that make it incompatible with the calling program (for example, the computational versions may not match). |

| Number | Description |
|---|---|
| 207 | (insufficient memory for loading) There is not enough memory to load the program from the Runtime Command or the CALL statement, or to build the in-memory library structures indicated in the Runtime Command, or to reserve memory for the ACCEPT and DISPLAY buffers. This may be caused by memory fragmentation resulting from the dynamics of CALL and CANCEL operations and file I/O, or it may mean the requested program is too large for the available memory. More memory can be made available during a SORT statement by using the T Runtime Command Option to reduce the memory requested by sort. Additional memory can be made available by reducing the amount of buffer pool memory through the use of the BUFFER-POOL-SIZE keyword (see page 316) on the RUN-FILES-ATTR configuration record. For details on memory size requirements for object programs, see page 162.<br><br>Under Windows, if the CALL statement specified SYSTEM, this error can occur when there is insufficient memory to load the command processor required by SYSTEM.<br><br>If the CALL statement specified the ON EXCEPTION or ON OVERFLOW phrase, this procedure error is suppressed and execution continues with the imperative-statement in the ON EXCEPTION or ON OVERFLOW phrase. |
| 208 | (compilation error in ALTER statement) The ALTER statement cannot be executed because of an error in the source program. The compilation listing provides the specific reason for the error; for example, an undefined procedure-name, an ambiguous procedure-name reference, an attempt to ALTER a procedure-name that is not alterable, a conflict with segmentation rules, and so forth. |
| 209 | (unaltered GO TO statement) The GO TO statement cannot be executed because it does not specify a default procedure-name and it was not altered before attempting execution. The source program may have a compilation error if no ALTER statement specifies the paragraph containing the GO TO statement. However, the source program may compile without error if at least one ALTER statement exists that specifies the paragraph containing the GO TO statement. In the latter case, no such ALTER statement is executed in the logical sequence of statements leading to the execution of the GO TO statement. |
| 210 | (compilation error in GO TO or PERFORM statement) The GO TO or PERFORM statement cannot be executed because of an error in the source program. The compilation listing provides the specific reason for the error; for example, an undefined procedure-name, an ambiguous procedure-name reference, a conflict with segmentation rules, and so forth. |
| 211 | (general compilation error in source program) An “E” level compilation error has been encountered. |

| Number | Description |
|---|---|
| 212 | (SORT/MERGE USE procedure error) The USE procedure cannot exit because it was invoked by the execution of a SORT or MERGE statement, and the sort-merge operation is either no longer active or the exit location has been lost. |
| 213 | (library not found) The RM/COBOL object library file specified in the Runtime Command cannot be found. |
| 214 | (library not valid) The RM/COBOL object library file specified in the Runtime Command does not contain a valid object program. |
| 215 | (segmentation error for PERFORM statement) A PERFORM statement in an independent segment has performed a section or paragraph in a fixed segment that performed a section or paragraph in a different independent segment. |
| 216 | (mismatched EXTERNAL data item) An external item with the same name and type (data record, file connector or index name) as an existing external in the run unit has a different description than the existing external.<br><br>For an external data record, the length of the record is different.<br><br>For an index-name, the span of the table item associated with the index name is different, or the index-name is associated with a different external record.<br><br>For a file connector, any of the file control clauses, file description clauses or record description lengths are different. For a relative organization external file connector, this error is caused if the new external does not reference the same external data item for the relative key as is referenced by the existing external file connector.<br><br>For additional details on the matching rules required for external objects with the same name described in more than one program of a run unit, see the discussion of External Objects (on page 217). |
| 217 | (mismatched EXTERNAL file) An external file connector is invalid since it indicates a SAME AREA or MULTIPLE FILE TAPE association. Typically, the compiler prevents this error from occurring by diagnosing the problem at compile time. |
| 218 | (insufficient memory for EXTERNAL data item or file) There is not enough memory to allocate the data structures necessary to support an external item declared in the program currently being loaded. |
| 219 | (insufficient memory for USE GLOBAL procedure) There is not enough memory to allocate the data structures necessary to support entry into a USE GLOBAL procedure following the occurrence of an I/O error for which the USE GLOBAL procedure is applicable. The program is terminated as if no applicable USE procedure were found. |

| Number | Description |
|---|---|
| 222 | ("CALL SYSTEM" load failure) Under Windows, the SYSTEM routine was called but the command processor required by SYSTEM could not be loaded for some unexpected reason, such as, bad environment, access denied, too many open files, or bad format for the command processor. If the command processor could not be found, error 204 would occur instead of this error. If there were insufficient memory, procedure error 207 would occur instead of this error. If the operating system fails to load the command processor for any other reasons, then this error occurs.<br><br>If the CALL statement specified the ON EXCEPTION or ON OVERFLOW phrase, this procedure error is suppressed and execution continues with the imperative-statement in the ON EXCEPTION or ON OVERFLOW phrase. |
| 223 | (non-COBOL library load failure) Under Windows, an error occurred while loading a DLL file. If the DLL file could not be found, error 204 would occur instead of this error. If there were insufficient memory, procedure error 207 would occur instead of this error. If the operating system fails to load the DLL for any other reasons, then this error occurs. This error generally indicates that the DLL was found, but has an invalid format for the operating system being used. Some "system out of memory" conditions may cause an error 223, since Windows returns an ambiguous error status in some low memory situations.<br><br>Under UNIX and Windows, an optional support module was unable to complete initialization successfully. Contact the provider of the failing support module if the information provided is not sufficient to resolve the problem. See special entry point **RM_AddOnInit** in Appendices G (for Windows) and H (for UNIX) of the *CodeBridge* manual for additional information regarding optional support module initialization.<br><br>If the CALL statement specified the ON EXCEPTION or ON OVERFLOW phrase, this procedure error is suppressed and execution continues with the imperative-statement in the ON EXCEPTION or ON OVERFLOW phrase. |

| Number | Description |
|---|---|
| 225 | (RM/COBOL object header not valid) The object header record for an RM/COBOL object program could not be successfully read or the contents of the header record are not valid. This error can occur for the header records of nested programs as well as separately compiled programs, including second or later separately compiled programs in a library of object programs. This error indicates that the file is not a valid RM/COBOL object file. The file may be a valid non-COBOL file (DLL or shared object), in which case this error will be ignored and the non-COBOL file will be loaded. Other possibilities are that the object file was corrupted or the load was attempted on a file that was never an object file, such as a text file having a name that matches a filename that RM/COBOL uses in its normal load search sequence. If the search sequence completes without finding a valid loadable file, this error will precede the error indicating that the search was unsuccessful, such as a procedure error 204 or 214. |
| 226 | (incorrect program descriptor size) The object header record for an RM/COBOL object program specifies a program descriptor size that is not valid for the object version specified in the header record. This error is a special case of error 225 in that it indicates the header record is not valid, but provides the specific reason to aid in determining the cause of the problem. |
| 227 | (expired object) The RM/COBOL object program was produced by a compiler that has expired. The source program needs to be re-compiled with a non-expired compiler. (Currently, objects expire only when they are produced by compilers licensed for evaluation or educational purposes. If the evaluation or educational license is updated to a normal license, re-compiling objects previously produced by the evaluation compiler is necessary.) |
| 228 | (runtime license ID mismatch) The RM/COBOL object program was produced by a compiler licensed for evaluation or educational purposes and is being run with a runtime also licensed for evaluation or educational purposes, but with a different license identifier. Evaluation or educational runtimes can run evaluation or educational objects only when the same development system (matching license identifier) is used.<br><br>**Note 1** Evaluation runtimes can run any non-evaluation object (such as the utility program objects shipped in an evaluation licensed system). Also, un-expired evaluation objects can be run by any non-evaluation runtimes.<br><br>**Note 2** Educational runtimes can run any non-educational object (such as the utility program objects shipped in an educational licensed system). However, un-expired educational objects can be run only by the matching license ID educational runtime. |

| Number | Description |
|---|---|
| 229 | (library defines no valid program-names) A non-COBOL library does not specify any valid program-names that can be called from COBOL. Thus, the library is essentially empty and should not be specified as part of a run unit. |
| 230 | (program size is zero) The RM/COBOL object program has zero size and thus cannot be a valid RM/COBOL object program. This error is a special case of error 225 in that it indicates the object header record is not valid, but provides the specific reason to aid in determining the cause of the problem. |
| 231 | (library TOC not valid) The table of contents for an RM/COBOL object library could not be successfully read or is logically inconsistent. The file may be corrupted or may not be an RM/COBOL object library (see error 225 for further information). |
| 232 | (unknown load entry type) The RM/COBOL object program has a load entry type that is not supported by the runtime being used to run the program. This error is a special case of error 225 in that it indicates the object header record is not valid, but provides the specific reason to aid in determining the cause of the problem. |
| 233 | (unsupported object version) An RM/COBOL object program being loaded has an object version or object flags that are not supported by the runtime being used to run the program. This normally means that the runtime is not a recent enough version to run the specific program file indicated in the pathname displayed with this message. However, it could also indicate that the object file has been corrupted, although this is unlikely since the file has already passed several other tests that would have produced a different error, such as procedure error 225.<br><br>**Note** Non-educational runtimes produce error 233 if an attempt is made to run an educational object program. |
| 234 | (no object file found) The main program or a called subprogram name could not be found in any loaded library. The RM/COBOL runtime system, therefore, searched for an object file to satisfy the request. This search for an object file did not find any candidate files for loading. For further information on the search for an object file, see Subprogram Loading (on page 214). |
| 251 | (termcap entry syntax) Under UNIX, a syntax error was detected while scanning the termcap entry for a terminal type. |
| 252 | (terminal type name unknown) Under UNIX, the terminal type name specified by the TERM environment variable or by the termcap entry tc cannot be located. |
| 253 | (terminal entry table overflow) Under UNIX, an internal table overflowed while processing a termcap or terminfo entry. The entry is too complex and its size should be reduced. |

| Number | Description |
|---|---|
| 254 | (duplicate terminal input sequences) Under UNIX, two termcap or terminfo entries have identical input sequences for this terminal. To indicate that one entry takes precedence over another, use the PRECEDENCE= keyword (see page 333). |
| 255 | (terminal does not support positioning) Under UNIX, the terminal described in the termcap or terminfo entry has no cursor positioning sequence, or the rows or columns for the terminal are zero. |
| 256 | (terminal unit undefined) Under UNIX, the unit number specified in an ACCEPT or DISPLAY statement has not been defined using a TERM-UNIT configuration record. |
| 257 | (non-COBOL dynamic load not supported by OS) Under UNIX, an attempted dynamic load of a machine language subprogram or library failed because dynamic load is not supported by your operating system. |
| 258 | (insufficient memory for pattern matching) There is not enough memory to match the pattern regular expression in a LIKE condition to the subject string. This occurs because there are too many possible match states to consider. For example, the pattern "((a{1,10}){1,10}){1,10}" when matching a string of 30 consecutive "a" characters will cause this error. |
| 299 | (instrumentation) An attempt to use Instrumentation on a run unit failed because a program in the run unit contains more than 65535 source lines or there was insufficient memory to allocate the data collection structure for a program in the run unit. See Data Collection (on page 352) for additional information on the memory requirements of Instrumentation. |

# Input/Output Errors

Input/output errors include all errors that can occur during file access. The format is as follows:

```
COBOL I/O error number on COBOL-filename
     file file-access-name
```

The numerically ordered list presented below shows the values that can be displayed as *number* in the I/O error messages, and a description of each error. The list is presented in numerical order. The I/O error number has the form:

```
mm, nn
```

*mm* is a two-digit decimal number indicating the general class of error that occurred. It is also the value stored into the file status data item if such an item has been specified for the associated file. Thus, this value is available to the program.

*nn* is a two-digit code that provides more specific information on the nature of the error. This value is available to the program only if you call the subprogram C$RERR (on page 557).

When the I/O error is 30, the I/O error number has the form:

```
30, OS error nnnnn
```

*OS* is the operating system that generated the error and indicates how the *nnnnn* code should be interpreted.

*nnnnn* is the operating system error code that was returned when the error occurred. This value is available to the program only if you call the subprogram C$RERR (on page 557).

The phrase “1985 mode” indicates that the error message description applies only to ANSI COBOL 1985. The phrase “1974 mode” indicates that the error message description applies only to ANSI COBOL 1974. Messages not marked with either phrase indicate that the description applies to both ANSI COBOL 1985 and 1974.

| Number | Description |
|---|---|
| 00 | The operation was successful. |
| 02 | The operation was successful but a duplicate key was detected. For a READ statement, the key value for the current key of reference is equal to the value of that same key in the next logical record within the current key of reference. For a REWRITE or WRITE statement, the record just written created a duplicate key value for at least one alternate record key for which duplicates are allowed. |
| 04, 05 | The record read from the file is shorter than the minimum record length. (1985 mode) |
| 04, 06 | The record read from the file is longer than the record area. (1985 mode) |
| 05 | The operation was successful but the file was not present at the time the statement began. For a DELETE FILE statement, the file was not found. For an OPEN statement, the optional file was not found. If the open mode is I-O or EXTEND, the file has been created. (1985 mode) |
| 07 | The operation was successful. If the operation was a CLOSE statement with a NO REWIND, REEL, UNIT or FOR REMOVAL clause, or if the operation was an OPEN statement with the NO REWIND clause, the file is not on a unit or reel medium. (1985 mode) |
| 10 | A sequential READ statement was attempted and no next (or previous) logical record exists in the file because the end (or beginning) of file was reached, or a sequential READ statement was attempted for the first time on an optional input file that is not present. |
| 14 | A sequential READ statement was attempted for a relative file and the number of significant digits in the relative record number is larger than the size of the relative key data item. (1985 mode) |
| 21 | A sequence error exists for a sequentially or dynamically accessed indexed file using duplicate prime keys. For a REWRITE statement, the prime record key was changed by the program between the execution of the preceding READ statement for the file and the execution of the REWRITE statement. Or, for a WRITE statement, the ascending sequence requirements for successive record key values were violated. |
| 22 | For an indexed file, the new record value attempts to duplicate an indexed file key that prohibits duplicates. For a relative file, the relative key data item duplicates a relative record number that already exists. |

| Number | Description |
|---|---|
| 23 | An attempt was made to randomly access a record that does not exist in the file, or a START or random READ statement was attempted on an optional input file that is not present. For a relative file, this means the relative key data item contains a value that is less than one, refers to a deleted record, or is greater than the highest relative record number existing in the file. For an indexed file, this means the specified value of the record or alternate record key does not refer to a record existing in the file. |
| 24 | There is insufficient disk space for the operation on a relative or indexed file. |
| 24, 01 | A sequential WRITE statement was attempted for a relative file and the number of significant digits in the relative record number is larger than the size of the relative key data item, or for a random WRITE statement the relative key data item contains a value that is less than one. (1985 mode) |
| 24, 02 | There is insufficient room left in the file for the operation. See the descriptions of the FILE-LOCK-LIMIT and FILE-PROCESS-COUNT keywords (on page 318) of the RUN-FILES-ATTR configuration record for more details.<br><br>This error may occur when attempting to delete a record from an indexed file if enough free blocks cannot be made available to split an index block. Running the Indexed File Recovery (recover1) utility (on page 598) on the file could consolidate enough index blocks to allow continued use of the file.<br><br>This error may occur on UNIX systems if the file's size exceeds the user's ulimit. |
| 24, 03 | An attempt was made to add the 4294967296$^{th}$ record to an indexed file. The maximum number or records allowed in an indexed file in 4294967295. |
| 24, 04 | An attempt was made to add a record to a shared relative file that is so far beyond the current EOF that the entire intervening region exceeds the maximum size of a file region lock. |

| Number | Description |
|---|---|
| 30, *OS* error *code* | RM/COBOL returned a permanent error in this 30, *OS* error *code* format. *OS* indicates the operating system that is the source of the error *code*. If *OS* is Btrieve, *code* is defined in Table 35 on page 388. If *OS* is UNIX, *code* is defined in the **errno.h** include file. If *OS* is Windows, refer to the file **WINERROR.TXT**, which is distributed with the media. If *OS* is C Library, *code* is defined in Table 36 on page 390. If *OS* is File Manager Detected, *code* is defined in Table 37 on page 390. If *OS* is any other value, then *code* and its meaning are system-dependent. Please refer to the documentation provided with your operating system for more information.<br><br>Using the eleven-character data item on the C$RERR subprogram will allow permanent errors to include the *OS* error *code*. For example, 30,06,00058 is Btrieve error 58. See C$RERR (on page 557).<br><br>A Procedure Division statement that explicitly or implicitly causes an overlay segment to be loaded may receive this error if the RM/COBOL object file cannot be read when the statement is executed. Appendix K: *Troubleshooting RM/COBOL* (on page 655) describes more information on 30 errors. |
| 30, 25 | This permanent error may be returned for Btrieve files. It indicates that an error occurred during the creation of the file. It may also mean that the file was already opened during the creation. If the file did not exist at the time the RM/COBOL-to-Btrieve Adapter program checked for the existence of the file but did exist at the time of the attempt to create it, then the Adapter program will misinterpret the error as an I/O error and return the permanent error code. |
| 30, 58 | The Btrieve compression size is too small. Use the appropriate Btrieve Setup utility to increase the size for the Largest Compressed Record Size setting (see the appropriate Btrieve installation and operation manual for more details). The Btrieve MicroKernel Database Engine (MKDE) must be restarted before the increase will take effect. |
| 30, 64 | End of file occurred when ACCEPTing from a redirected input file. |
| 30, 97 | The Btrieve requester's data message length is too small. Reconfigure the requester and specify a higher value. Be certain also to ensure that the maximum record length (Communication Buffer Size), configured on server-based Btrieve, is at least as large as the requester's value (see the appropriate Btrieve installation and operation manual for more details). |
| 34 | There is insufficient disk space for the operation on a sequential file. |
| 34, 02 | There is insufficient room left in the file for the operation. For more details, see the descriptions of FILE-LOCK-LIMIT and FILE-PROCESS-COUNT (on page 318) of the RUN-FILES-ATTR configuration record. |

| Number | Description |
|---|---|
| 35 | The file is not available because the file identified by the resultant file access name could not be found. The pathname or filename may be misspelled or may not be valid for the operating system. Specifying a pathname or filename that is not a valid name or that is longer than allowed also results in this error. The directory search sequence specified by RUNPATH may be incorrect. For information on the resultant file access name, see Locating RM/COBOL Files on UNIX (on page 24) or Locating RM/COBOL Files on Windows (on page 61). (1985 mode) |
| 37, 01 | The file must be mass storage. The device-name specified for the file was DISC, DISK or RANDOM, but the resultant file access name identifies a file that does not reside on a disk. (1985 mode) |
| 37, 07 | The requested operation conflicts with the permissions allowed to the run unit for the file. This error can occur under any of the following conditions: a DELETE FILE statement failed because the run unit did not have write permission for the directory containing the file; an OPEN statement with the OUTPUT or EXTEND phrase failed because the run unit does not have write permission for the file; an OPEN statement with the INPUT phrase failed because the run unit does not have read permission for the file; an OPEN statement with the I-O phrase failed because the run unit does not have read and write permissions for the file; or, for an indexed file, an OPEN I-O failed due to future file version attributes that allow the file to be read but not written. (1985 mode) |
| 38 | An OPEN or DELETE FILE operation failed because the filename was previously closed WITH LOCK. (1985 mode) |
| 39, 01 | The file organization specified for the filename does not match the actual file organization of the physical file. (1985 mode)<br><br>This message may not occur if the file is actually a Btrieve file. The Btrieve MicroKernel Database Engine (MKDE) always opens its files with lock, and the OPEN WITH LOCK error condition is encountered by the RM/COBOL file management system, preventing it from determining the organization of the file. |
| 39, 02 | The minimum record length specified in the RECORD CONTAINS clause or implied by the record descriptions of the filename does not match the actual minimum record length of the physical file. (1985 mode)<br><br>For Btrieve files, if the minimum record length of the file is less than four bytes, the file will be implemented using four-byte records. In this situation, the RM/COBOL-to-Btrieve Adapter program cannot detect the initial minimum record length and will fail to diagnose the mismatched minimum record length error condition. |

| Number | Description |
|---|---|
| 39, 03 | The maximum record length specified in the RECORD CONTAINS clause or implied by the record descriptions of the filename does not match the actual maximum record length of the physical file. (1985 mode)<br><br>This error can occur when reopening a print file closed WITH NO REWIND if the OPEN statement specifies a different record length than was used on the previous OPEN statement.<br><br>This error does not occur with variable-length record Btrieve files. Btrieve files do not support a mechanism to record this information and, thus, it cannot be verified. |
| 39, 04 | The minimum block length specified in the BLOCK CONTAINS clause for the filename does not match the actual minimum block size of the physical file. (1985 mode) |
| 39, 05 | The maximum block length specified in the BLOCK CONTAINS clause for the filename does not match the actual maximum block size of the physical file. (1985 mode)<br><br>This error does not occur with variable-length record Btrieve files. Btrieve files do not support a mechanism to record this information and, thus, it cannot be verified. |
| 39, 06 | The record delimiting technique, LINE-SEQUENTIAL or BINARY-SEQUENTIAL, specified for the filename does not match the actual record delimiting technique of the physical file. (1985 mode) |
| 39, 07 | The CODE-SET specified for the filename does not match the actual character code of the physical file. (1985 mode) |
| 39, 08 | The COLLATING SEQUENCE specified for the indexed file does not match the actual collating sequence of the physical file. (1985 mode) |
| 39, 09 | The record type attribute, fixed or variable, specified in the RECORD CONTAINS clause or implied by the record descriptions of the filename does not match the record type attribute of the physical file. (1985 mode) |
| 39, 0A | The character specified in the PADDING CHARACTER clause for the filename does not match the actual padding character of the file on the external medium. (1985 mode) |

| Number | Description |
|---|---|
| 39, 30 through 39, 3E | The key duplicates allowed flag specified for keys 0 through 14 does not match the corresponding key duplicates allowed flag of the physical file. The prime record key is 0. Alternate record keys are numbered in ascending order of key offset, starting with 1. (1985 mode)<br><br>For Btrieve files, this error also indicates that in the physical file, a key attribute other than duplicate, modifiable, binary, null, alternate collating sequence, or extended type was specified for the Btrieve key that corresponds with the COBOL key defined for the key offset. (For example, descending, supplemental, and manual key attributes may not be used for RM/COBOL keys.) In addition, the primary key may not have a null attribute.<br><br>Btrieve allows multiple keys to be defined at the same key offset, but only the first is considered by the RM/COBOL-to-Btrieve Adapter program. |
| 39, 3F | The key duplicates allowed flag specified for an alternate record key 15 through 254 does not match the corresponding key duplicates allowed flag of the physical file. (1985 mode)<br><br>For Btrieve files, this error also indicates that in the physical file, a key attribute other than duplicate, modifiable, binary, null, alternate collating sequence or extended type was specified for the Btrieve key that corresponds with the COBOL key defined for the key offset. (For example, descending, supplemental and manual key attributes may not be used for RM/COBOL keys.)<br><br>Btrieve allows multiple keys to be defined at the same key offset, but only the first is considered by the RM/COBOL-to-Btrieve Adapter program. |
| 39, 40 through 39, 4E | The offset from the start of the record area to the start of the key area for keys 0 through 14 does not match the corresponding key offset of the physical file. The prime record key is 0. Alternate record keys are numbered in ascending order of key offset, starting with 1. (1985 mode) |
| 39, 4F | The offset from the start of the record area to the start of the key area for an alternate key 15 through 254 does not match the corresponding key offset of the physical file. (1985 mode) |
| 39, 50 through 39, 5E | The length of the key area for keys 0 through 14 does not match the corresponding key length of the physical file. The prime record key is key 0. Alternate record keys are numbered in ascending order of key offset, starting with 1. (1985 mode)<br><br>This error also occurs if the number of keys specified does not match the actual number of keys in the physical file. The key number 1 through E, if greater than the number of keys in the file description, indicates that the file contains more keys than the program describes. The key number 1 through E, if less than or equal to the number of keys in the file description, provides a value one greater than the number of keys contained in the file. (1985 mode) |

| Number | Description |
|---|---|
| 39, 5F | The length of the key area for an alternate key 15 through 254 does not match the corresponding key offset of the physical file. (1985 mode)<br><br>This error also occurs if the number of keys specified does not match the actual number of keys in the physical file. (1985 mode) |
| 39, 60 through 39, 6E | The number of segments for keys 0 through 14 does not match the corresponding key number of segments of the physical file. The prime record key is 0. Alternate record keys are numbered in ascending order of key offset, starting with 1. (1985 mode) |
| 39, 6F | The number of segments for an alternate key 15 through 254 does not match the corresponding key number of segments of the physical file. (1985 mode) |
| 41, 01 | A duplicate open was rejected by a system that does not allow the physical file to be opened twice. (1985 mode) |
| 41, 02 | A duplicate open was rejected by a system that does not allow the COBOL filename to be opened twice. (1985 mode) |
| 41, 03 | A DELETE FILE was rejected because the file was open. (1985 mode) |
| 42 | A CLOSE or UNLOCK operation was attempted on an unopened file. (1985 mode) |
| 43 | A DELETE or REWRITE operation was attempted on a file declared to be ACCESS MODE SEQUENTIAL or on an indexed file declared to be ACCESS MODE DYNAMIC that specifies a prime key that allows duplicate values, and the last operation on the file was not a successful READ operation. (1985 mode) |
| 44, 03 | The length of the record area specified in the WRITE, REWRITE or RELEASE statement is less than the minimum record length of the file. (1985 mode) |
| 44, 04 | The length of the record area specified in the WRITE, REWRITE, or RELEASE statement is greater than the maximum record length of the file. (1985 mode) |
| 44, 07 | A REWRITE statement attempted to change the length of a record in a sequential organization file. (1985 mode) |
| 46 | No file position is currently defined. A sequential READ operation was attempted, but the last READ or START operation was unsuccessful or returned an at end condition. (1985 mode) |
| 46, 02 | The position was lost. (1985 mode)<br><br>For Btrieve files, A READ NEXT operation was attempted and could not be completed because the Btrieve MicroKernel Database Engine (MKDE) position was lost due to the current record (and surrounding records) being deleted by programs at other computers. See Current Record Position Limitations (on page 126). |

| Number | Description |
|---|---|
| 47, 01 | The requested operation conflicts with the open mode of the file. A START or READ operation was attempted on a file that is not open in the INPUT or I-O mode. (1985 mode) |
| 47, 02 | A READ or START operation was attempted on an unopened file. (1985 mode) |
| 48, 01 | The requested operation conflicts with the open mode of the file. This error can occur under the following conditions: a WRITE operation was attempted on a file that is not open in the EXTEND, I-O, or OUTPUT mode; or, a WRITE operation was attempted on a file in the sequential access mode that is open in the I-O mode. (1985 mode) |
| 48, 02 | A WRITE operation was attempted on an unopened file. (1985 mode) |
| 49, 01 | The requested operation conflicts with the open mode of the file. A DELETE or REWRITE operation was attempted on a file that is not open in the I-O mode. (1985 mode) |
| 49, 02 | A DELETE or REWRITE operation was attempted on an unopened file. (1985 mode) |
| 90 | An unrecognizable message has been received by the RM/COBOL-to-Btrieve Adapter program from the RM/COBOL file management system.<br><br>This error also indicates that an invalid request has been made to the RM/COBOL file management system or some other external access method. This may be caused by an internal error or when communicating to an earlier version of RM/InfoExpress. |
| 90, 01 | The requested operation conflicts with the open mode of the file. This error can occur under the following conditions: a READ or START operation was attempted on a file that is not open in the INPUT or I-O mode; a WRITE operation was attempted on a file that is not open in the EXTEND, I-O, or OUTPUT mode or a WRITE operation was attempted on a file in the sequential access mode that is open in the I-O mode; or, a DELETE or REWRITE operation was attempted on a file that is not open in the I-O mode. (1974 mode) |
| 90, 02 | A DELETE or REWRITE operation was attempted on a file declared to be ACCESS MODE SEQUENTIAL, and the last operation on the file was not a successful READ operation. (1974 mode) |
| 90, 03 | The requested operation conflicts with the media type. This error can occur under the following conditions: a READ or OPEN INPUT operation was attempted on a file with a device-name of OUTPUT, PRINT or PRINTER; a WRITE, OPEN OUTPUT or EXTEND operation was attempted on a file with a device-name of CARD-READER or INPUT; or a DELETE, REWRITE, START or OPEN I-O operation was attempted on a file with a device-name other than DISC, DISK or RANDOM. |

| Number | Description |
|---|---|
| 90, 04 | The requested operation conflicts with the defined organization. A DELETE or START operation was attempted on an ORGANIZATION SEQUENTIAL file. |
| 90, 05 | A file truncate operation conflicts with other users. An OPEN OUTPUT operation was attempted on a physical file that is currently in an open mode for another file connector of this run unit or a file connector of another run unit that shares the file.<br><br>On Btrieve files, Error 90, 05 also indicates that an I/O error occurred on the creation of the file. (If the file already existed and the Btrieve MicroKernel Database Engine (MKDE) returned a Create I/O Error, 25, then the RM/COBOL-to-Btrieve Adapter program will misinterpret the error as truncation conflict because the Btrieve MKDE uses this error for both conditions.) |
| 90, 06 | The file access name specified in the OPEN statement indicates that the file is accessed through an external access method and the external access method refused to accept the request by the RM/COBOL file management system to establish a session.<br><br>This error also indicates that the RM/COBOL file management system refused one of the configuration parameters passed to it. The runtime system validates all the configuration parameters of the RM/COBOL file management system; however, if there is insufficient memory to create a buffer pool of the requested size, this will not be detected until later. Reducing the size of the buffer pool may resolve the problem. |
| 90, 07 | The requested operation conflicts with the permissions allowed to the run unit for the file. This error can occur under the following conditions: a DELETE FILE statement failed because the run unit did not have write permission for the directory containing the file; an OPEN statement with the OUTPUT or EXTEND phrase failed because the run unit does not have write permission for the file; or, an OPEN statement with the INPUT or I-O phrase failed because the run unit does not have read permission for the file. (1974 mode)<br><br>For Btrieve files, this error code may also indicate that a DELETE, WRITE, or REWRITE operation was performed on a file that has been opened for read-only access using an RM/COBOL-to-Btrieve Adapter program mode option of M=R (read-only). For more information about this option, see M (Mode) Option (on page 121). (1974 and 1985 mode) |
| 90, 08 | The requested operation is not supported by the external access method. A COBOL I/O statement was attempted to a non-RM/COBOL file and the access method for the file does not support the statement. |
| 91 | A CLOSE or UNLOCK operation was attempted on an unopened file. (1974 mode) |
| 91, 02 | A READ, START, WRITE, DELETE or REWRITE operation was attempted on an unopened file. (1974 mode) |

| Number | Description |
|---|---|
| 92, 01 | A duplicate open was rejected by a system that does not allow the physical file to be opened twice. (1974 mode) |
| 92, 02 | A duplicate open was rejected by a system that does not allow the COBOL filename to be opened twice. (1974 mode) |
| 92, 03 | A DELETE FILE was rejected because the file was in an open mode. (1974 mode) |
| 93, 02 | An operation was rejected because file lock conflicts with another user.<br><br>An OPEN WITH LOCK was attempted on a file that is already open, or an OPEN without lock was attempted and the file is already open WITH LOCK.<br><br>A DELETE FILE was attempted on a file that is currently open.<br><br>This message may occur in cases with Btrieve files when it would not occur with RM/COBOL indexed files, because the Btrieve MicroKernel Database Engine (MKDE) always opens its files WITH LOCK. For more information, see error messages 39, 01 and 94, 01.<br><br>For Btrieve files, this error code may also indicate either of the following conditions:<br><br>• Another computer has a transaction in progress on this file.<br><br>• Or, an attempt was made to open a file that another computer had opened already with a conflicting RM/COBOL-to-Btrieve Adapter program mode option. For more information about the mode option, see M (Mode) Option (on page 121). For example, if the first computer opens a file with a value of A (accelerated) for the mode option, then the accelerated mode option must be specified by all other computers that subsequently open the file. Conversely, if the first computer opens a file and does not specify the accelerated mode option, then no other computers that subsequently open the file can specify the accelerated mode option either. These restrictions remain in effect until all computers have closed the file. |
| 93, 03 | An OPEN or DELETE FILE operation failed because the filename was previously closed WITH LOCK. (1974 mode) |
| 93, 04 | The file could not be opened because another file in the same SAME AREA clause is currently open. |
| 93, 05 | The file could not be opened because another file in the same MULTIPLE FILE TAPE clause is already open. |
| 93, 06 | The file could not be created because a file with the same name already exists. |
| 93, 07 | The file could not be opened because a lock table for the requested open mode was full. See the FILE-PROCESS-COUNT keyword (on page 318) of the RUN-FILES-ATTR configuration record for more information. |

| Number | Description |
|---|---|
| 94, 01 | The file organization specified for the filename does not match the actual file organization of the physical file. (1974 mode)<br><br>This message may not occur if the file is actually a Btrieve file. The Btrieve MicroKernel Database Engine always opens its files WITH LOCK, and the OPEN WITH LOCK error condition will be encountered by the RM/COBOL file management system, preventing it from determining the organization of the file. |
| 94, 02 | The minimum record length specified in the RECORD CONTAINS clause or implied by the record descriptions of the filename does not match the actual minimum record length of the physical file. (1974 mode)<br><br>For Btrieve files, if the minimum record length of the file is less than four bytes, the file will be implemented using four-byte records. In this situation, the RM/COBOL-to-Btrieve Adapter program cannot detect the initial minimum record length and will fail to diagnose the mismatched minimum record length error condition. |
| 94, 03 | The maximum record length specified in the RECORD CONTAINS clause or implied by the record descriptions of the filename does not match the actual maximum record length of the physical file. (1974 mode)<br><br>This error can occur when reopening a print file closed WITH NO REWIND if the OPEN statement specifies a different record length than was used on the previous OPEN statement.<br><br>This error will not occur with variable-length record Btrieve files. Btrieve files do not support a mechanism to record this information and, thus, it cannot be verified. |
| 94, 04 | The minimum block length specified in the BLOCK CONTAINS clause for the filename does not match the actual minimum block size of the physical file. (1974 mode) |
| 94, 05 | The maximum block length specified in the BLOCK CONTAINS clause for the filename does not match the actual maximum block size of the physical file. (1974 mode)<br><br>This error will not occur with variable-length record Btrieve files. Btrieve files do not support a mechanism to record this information and, thus, it cannot be verified. |
| 94, 06 | The record delimiting technique, LINE-SEQUENTIAL or BINARY-SEQUENTIAL, specified for the filename does not match the actual record delimiting technique of the physical file. (1974 mode) |
| 94, 07 | The CODE-SET specified for the filename does not match the actual character code of the physical file. (1974 mode) |
| 94, 08 | The COLLATING SEQUENCE specified for the indexed file does not match the actual collating sequence of the physical file. (1974 mode) |

| Number | Description |
|---|---|
| 94, 09 | The record type attribute, fixed or variable, specified in the RECORD CONTAINS clause or implied by the record descriptions of the filename does not match the record type attribute of the physical file. (1974 mode) |
| 94, 20 | The file is not available because the file identified by the resultant file access name could not be found. The pathname or filename may be misspelled or may not be valid for the operating system. Specifying a pathname or filename that is not a valid name or that is longer than allowed also results in this error. The directory search sequence specified by RUNPATH may be incorrect. For information on the resultant file access name, see Locating RM/COBOL Files on UNIX (on page 24) or Locating RM/COBOL Files on Windows (on page 61). (1974 mode) |
| 94, 21 | The file organization specified is invalid or unsupported, or the requested open operation is illegal on the specified organization. |
| 94, 22 | The minimum record length is invalid. The minimum record length specified in the RECORD CONTAINS clause for the filename exceeds the maximum record length. |
| 94, 23 | The maximum record length is invalid. The maximum record length specified in the RECORD CONTAINS clause of the filename exceeds 65280, or the indexed records are not compressed and the maximum record length exceeds the block size. |
| 94, 24 | The minimum block size is invalid. The minimum block size specified in the BLOCK CONTAINS clause of the filename exceeds the maximum block size. |
| 94, 25 | The maximum block size is invalid. The maximum block size specified in the BLOCK CONTAINS clause of the filename is too large. The method of computing the block size and the limitations on the block size for each organization are described in sequential files (on page 222), relative files (on page 228), and indexed files (on page 111).<br><br>For indexed organization files, the computed block size is also a function of the maximum record size. In general, if the BLOCK CONTAINS clause is omitted, the runtime system defaults to the valid block size that is a multiple of the disk sector size. For files with a very large record size, specifying BLOCK CONTAINS 1 RECORDS yields the minimum possible block size. |
| 94, 26 | The record delimiter is invalid. A record delimiting technique other than LINE-SEQUENTIAL or BINARY-SEQUENTIAL was specified. |
| 94, 27 | The CODE-SET specified is invalid or unsupported. |
| 94, 28 | The COLLATING SEQUENCE specified for an indexed file is invalid or unsupported. |

| Number | Description |
|---|---|
| 94, 29 | The record type attribute, fixed or variable, specified for the filename is unsupported. |
| 94, 30 through 94, 3E | The key duplicates allowed flag specified for keys 0 through 14 is invalid (1974 and 1985 modes) or does not match the corresponding key duplicates allowed flag of the physical file (1974 mode). The prime record key is 0. Alternate record keys are numbered in ascending order of key offset, starting with 1.<br><br>For Btrieve files, this error also indicates a key attribute other than duplicate, modifiable, binary, null, alternate collating sequence, or extended type was specified for the Btrieve key that corresponds with the COBOL key defined for the key offset. (For example, descending, supplemental, and manual key attributes may not be used for RM/COBOL keys.) In addition, the primary key may not have a null attribute.<br><br>Btrieve allows multiple keys to be defined at the same key offset, but only the first key is considered by the RM/COBOL-to-Btrieve Adapter program. |
| 94, 3F | The key duplicates allowed flag specified for an alternate record key 15 through 254 is invalid (1974 and 1985 modes) or does not match the corresponding key duplicates allowed flag of the physical file. (1974 mode)<br><br>For Btrieve files, this error also indicates that in the physical file, a key attribute other than duplicate, modifiable, binary, null, alternate collating sequence or extended type was specified for the Btrieve key that corresponds with the COBOL key defined for the key offset. (For example, descending, supplemental and manual key attributes may not be used for RM/COBOL keys.)<br><br>Btrieve allows multiple keys to be defined at the same key offset, but only the first key is considered by the RM/COBOL-to-Btrieve Adapter program. |
| 94, 40 through 94, 4E | The offset from the start of the record area to the start of the key area for keys 0 through 14 is invalid (1974 and 1985 modes) or does not match the corresponding key offset of the physical file (1974 mode). The prime record key is 0. Alternate record keys are numbered in ascending order of key offset, starting with 1.<br><br>Error 94, 40 also occurs if more than 254 alternate record keys are specified.<br><br>For Btrieve files, this error also indicates that the key extends into the variable portion of a Btrieve file record. |
| 94, 4F | The offset from the start of the record area to the start of the key area for an alternate key 15 through 254 is invalid (1974 and 1985 modes) or does not match the corresponding key offset of the physical file. (1974 mode)<br><br>For Btrieve files, this error also indicates that the key extends into the variable portion of a Btrieve file record. |

| Number | Description |
|---|---|
| 94, 50 through 94, 5E | The length of the key area for keys 0 through 14 is invalid (1974 and 1985 modes) or does not match the corresponding key length of the physical file (1974 mode). The prime record key is key 0. Alternate record keys are numbered in ascending order of key offset, starting with 1.<br><br>This error also occurs if the number of specified keys does not match the actual number of keys in the physical file. The key number 1 through E, if greater than the number of keys in the file description, indicates that the file contains more keys than the program describes. The key number 1 through E, if less than or equal to the number of keys in the file description, provides a value one greater than the number of keys contained in the file. (1974 mode)<br><br>A Btrieve file is allowed to have more keys defined in it than in the COBOL description of the file, but not less. |
| 94, 5F | The length of the key area for an alternate key 15 through 254 is invalid (1974 and 1985 modes) or does not match the corresponding key offset of the physical file. (1974 mode)<br><br>This error also occurs if the number of keys specified does not match the actual number of keys in the physical file (see error messages 94, 50 through 94, 5E). (1974 mode) |
| 94, 60 | There is insufficient memory to open a file. The amount of memory required to open a file can be reduced by specifying a smaller maximum block size in the BLOCK CONTAINS clause. Additional memory can be made available by decreasing the amount of buffer pool memory by using the BUFFER-POOL-SIZE keyword (see page 316) on the RUN-FILES-ATTR configuration record. |
| 94, 61 | There is insufficient disk space to create a file. |
| 94, 62 | The LINAGE parameters are invalid for an OPEN statement. One or more LINAGE parameters are negative or greater than 32767, LINAGE equals zero, FOOTING equals zero, or FOOTING is greater than LINAGE. |
| 94, 63 | An OPEN WITH LOCK was attempted on a system that does not support WITH LOCK. |
| 94, 64 | The filename specified is invalid. This error can occur if the filename is set to spaces. |
| 94, 65 | An OPEN was attempted on the controlling console device. |
| 94, 66 | There are no more file handles available. An OPEN was rejected because an operating system limit on the number of files was reached. |

| Number | Description |
|---|---|
| 94, 67 | The file is too large. An attempt was made to open a file that is too large for this system. The file was probably created on another system using the LARGE-FILE-LOCK-LIMIT configuration keyword or is a version 3 indexed file, or an attempt was made to use a LARGE-FILE-LOCK-LIMIT value on a system that does not support files larger than 2 GB. For more information, see the description of the LARGE-FILE-LOCK-LIMIT keyword (on page 318) in the RUN-FILES-ATTR configuration record or the description of File Version Level 3 (on page 244) files. |
| 94, 68 | An attempt was made to open an indexed file that has a future file version number. The indexed file may have been created by a later version of RM/COBOL or the indexed file may be corrupt. |
| 94, 69 | The specified large file lock limit is too large for either the operating system on which the runtime is running, or the file system on which the file would reside. For example, files on a Windows 9x-class operating system do not support a large file lock limit greater than four gigabytes (4 GB). |
| 95, 01 | The file must be mass storage. The device-name specified for the file was DISC, DISK or RANDOM, but the resultant file access name identifies a file that does not reside on disk. (1974 mode) |
| 96 | No file position is currently defined. A sequential READ operation was attempted, but the last READ or START operation was unsuccessful or returned an at end condition. (1974 mode) |
| 96, 02 | The position was lost. (1974 mode)<br><br>For Btrieve files, A READ NEXT operation was attempted and could not be completed because the Btrieve MicroKernel Database Engine (MKDE) position was lost due to the current record (and surrounding records) being deleted by programs at other computers. See Current Record Position Limitations (on page 126). |
| 97, 01 | One or more characters in the record are illegal in a line sequential file. |
| 97, 02 | One or more characters could not be translated from the native character set to the external code-set. |
| 97, 03 | The length of the record area specified in the WRITE, REWRITE or RELEASE statement is less than the minimum record length of the file. (1974 mode) |
| 97, 04 | The length of the record area specified in the WRITE, REWRITE, or RELEASE statement is greater than the maximum record length of the file. (1974 mode) |
| 97, 05 | The record read from the file is shorter than the minimum record length. |
| 97, 06 | The record read from the file is longer than the record area. |

| Number | Description |
|---|---|
| 97, 07 | A REWRITE statement attempted to change the length of a record in a sequential organization file. (1974 mode) |
| 97, 08 | The LINAGE parameters are invalid for a WRITE statement. One or more LINAGE parameters are negative or greater than 32767, LINAGE equals zero, FOOTING equals zero, or FOOTING is greater than LINAGE. |
| 97, 09 | The TO LINE value specified in a WRITE statement for a file described with the LINAGE clause is not an allowed value. The value is either zero, greater than the number of lines in the current logical page body, or, when the NEXT PAGE option has not been specified, greater than the current line number within the logical page body. |
| 98, 01 | The indexed file structure includes a count of the number of times the file is currently open for modification. The count should be zero whenever a file in a single-user environment is opened or a file in a shared environment is opened WITH LOCK. If the count is non-zero when the file is opened WITH LOCK, a 98, 01 error is returned. The conditions that determine whether the runtime system assumes a single-user or shared environment are described in File Types and Structure (on page 222) and in the explanation of the FORCE-USER-MODE keyword (see page 318) of the RUN-FILES-ATTR configuration record.<br><br>The count is incremented when a program opens the file I-O, OUTPUT, or EXTEND and decremented when the program closes the file. If the count is non-zero when the file is opened in a single-user environment or opened WITH LOCK in a shared environment, then the system must have terminated without closing the file. This error can also occur when a file with a non-zero count is moved from a shared environment to a single-user environment. The indexed file will be inconsistent if all the modifications to it were not written to disk. Use the Indexed File Recovery (recover1) utility (on page 598) to rebuild or recover disk information.<br><br>You can reduce the likelihood of encountering this error by changing the Indexed File Recovery utility strategy to “Force File Closed”, as described in Data Recoverability (on page 231). This causes the count to be changed around every write operation instead of during open and close. Use the Define Indexed File (rmdefinx) utility (on page 594) to change the recovery strategy of an existing indexed file. |
| 98, 02 | A fatal error occurred during a DELETE, REWRITE, or WRITE statement when the file was last open. The index structure is inconsistent and must be rebuilt. Use the Indexed File Recovery (recover1) utility (on page 598) to rebuild or recover disk information. |

| Number | Description |
|---|---|
| 98, *nn* | Invalid file structure. The *nn* subcode may be useful in determining which runtime system procedure detected the error.<br><br>For an indexed organization file, an inconsistency in the file structure was detected. If the error occurs when the file is being read, it may be a disk read error, which may go away when the operation is retried. If the error occurs during a DELETE, REWRITE, or WRITE statement, a later OPEN statement will probably receive a 98, 02 error. It may be possible to correct the file structure inconsistency by rebuilding the index structure. Use the Indexed File Recovery (recover1) utility (on page 598) to rebuild or recover disk information.<br><br>For a sequential or relative organization file, this error usually indicates the file description does not match the organization of the file, record type, record delimiting technique, or record length. It may also indicate that the file data was not written or read correctly.<br><br>Appendix K: *Troubleshooting RM/COBOL* (on page 655) describes more information on 98 errors. |
| 99 | A DELETE, READ, or REWRITE statement failed because the record is locked by another user. |

**Table 35: Btrieve Status Codes and Messages[1]**

| Code | Message | Code | Message |
|---|---|---|---|
| 1 | Invalid operation | 36 | Transaction error |
| 2 | I/O error | 37 | Transaction is active |
| 3 | File not open | 38 | Transaction control I/O error |
| 4 | Key value not found | 39 | End/abort transaction error |
| 5 | Duplicate key value | 40 | Transaction max files |
| 6 | Invalid key number | 41 | Operation not allowed |
| 7 | Different key number | 42 | Incomplete accelerated access |
| 8 | Invalid positioning | 43 | Invalid record address |
| 9 | End-of-file | 44 | Null key path |
| 10 | Modifiable key value error | 45 | Inconsistent key flags |
| 11 | Invalid filename | 46 | Access to file denied |
| 12 | File not found | 47 | Maximum open files |
| 13 | Extended file error | 48 | Invalid alternate sequence definition |
| 14 | Pre-image open error | 49 | Key type error |
| 15 | Pre-image I/O error | 50 | Owner already set |
| 16 | Expansion error | 51 | Invalid owner |
| 17 | Close error | 52 | Error writing cache |
| 18 | Disk full | 53 | Invalid interface |
| 19 | Unrecoverable error | 54 | Variable page error |
| 20 | Record manager inactive | 55 | Auto-increment error |
| 21 | Key buffer too short | 56 | Incomplete index |
| 22 | Data buffer length | 57 | Expanded memory error |
| 23 | Position block length | 58 | Compression buffer too short (see page 373 for resolution suggestion) |
| 24 | Page size error | 59 | File already exists |
| 25 | Create I/O error | 60 | Reject count reached |
| 26 | Number of keys | 61 | Work space too small |
| 27 | Invalid key position | 62 | Incorrect descriptor |
| 28 | Invalid record length | 63 | Invalid extended insert buffer |
| 29 | Invalid key length | 64 | Filter limit reached |
| 30 | Not a Btrieve file | 65 | Incorrect field offset |
| 31 | File already extended | 74 | Transaction rolled back |
| 32 | Extend I/O error | 75 | Server routing list too small |
| 34 | Invalid extension name | 76 | File server list too small |
| 35 | Directory error | 77 | VAP wait error |

[1] *This list of Btrieve status codes and messages is provided for convenience only, and is not intended to supply complete information. For a full list of codes and messages with descriptions, refer to the appropriate Btrieve installation and operation manual.*

**Table 35: Btrieve Status Codes and Messages[1] *(Cont.)***

| Code | Message | Code | Message |
|---|---|---|---|
| 78 | Deadlock detected | 130 | MKDE out of system locks |
| 79 | Programming error | 132 | File full |
| 80 | Conflict | 133 | More than 5 concurrent engines accessing same file |
| 81 | Lock error | 1001 | Lock parameter is invalid |
| 82 | Lost position | 1002 | Memory allocation error |
| 83 | Read outside transaction | 1003 | Invalid memory size parameter |
| 84 | Record in use | 1004 | Page size error |
| 85 | File in use | 1005 | Preimage drive parameter invalid |
| 86 | File table full | 1006 | Preimage buffer parameter invalid |
| 87 | Handle table full | 1007 | Files parameter invalid |
| 88 | Incompatible mode error | 1008 | Initialization parameter invalid |
| 90 | Redirected device table full | 1009 | Transaction filename parameter invalid |
| 91 | Server error | 1010 | Transaction control file error |
| 92 | Transaction table full | 1011 | Compression buffer parameter invalid |
| 93 | Incompatible lock type | 1012 | Invalid /n option (pre-v6.0) |
| 94 | Permission error | 1013 | Task list full |
| 95 | Session no longer valid | 1014 | Stop warning. Files still active |
| 96 | Communications environment error | 1015 | Pointer parameter invalid |
| 97 | Data message too small (see page 373 for resolution suggestion) | 1016 | MKDE already initialized |
| 98 | Internal transaction error | 1017 | Requester cannot locate wbtrvres.dll |
| 99 | Requester cannot access NetWare Runtime server | 1018 | MKDE called from callback |
| 100 | No cache buffers | 1019 | Operation canceled by callback |
| 101 | Insufficient OS memory | 1020 | Requester communications error |
| 102 | Insufficient stack space | 2001 | Memory allocation insufficient |
| 103 | Chunk offset too big | 2002 | Requester option invalid |
| 104 | Unknown locale | 2003 | Requester cannot access local file |
| 105 | Cannot create VAT file | 2004 | SPX not installed |
| 106 | Cannot get next chunk | 2005 | Incorrect SPX version |
| 107 | Chunk operation on pre-v6.0 file | 2006 | No available SPX connection |
| 109 | Unknown error creating or accessing semaphore | 2007 | Pointer parameter invalid |

**Table 36: C Library Error Codes[1]**

| Code | Message | Code | Message |
|---|---|---|---|
| 1 | No such file or directory | 21 | I/O error |
| 2 | Arg list too big | 22 | Is a directory |
| 3 | Exec format error | 23 | Not a directory |
| 4 | Bad file number | 24 | Too many links |
| 5 | Not enough memory | 25 | Block device required |
| 6 | Permission denied | 26 | Not a character device |
| 7 | File exists | 27 | No such device or address |
| 8 | Cross-device link | 28 | Not owner |
| 9 | Invalid argument | 29 | Broken pipe |
| 10 | File table overflow | 30 | Read-only file system |
| 11 | Too many open files | 31 | Illegal seek |
| 12 | No space left on device | 32 | No such process |
| 13 | Argument too large | 33 | Text file busy |
| 14 | Result too large | 34 | Bad address |
| 15 | Resource deadlock would occur | 35 | Name too long |
| 16 | Interrupt | 36 | No such device |
| 17 | Child does not exist | 37 | No locks available in system |
| 18 | Resource unavailable, try again | 38 | Unknown system call |
| 19 | Device or resource busy | 39 | Directory not empty |
| 20 | File too large | | |

[1] *This list of C library error codes and meanings is provided for convenience only, and is not intended to supply complete information.*

**Table 37: File Manager Detected Error Codes**

| Code | Message |
|---|---|
| 1 | Memory management failure |
| 2 | Operator requested termination |
| 3 | Locks lost |

# Internal Errors

In general, internal errors are caused when the object file has been corrupted. If the corruption was caused by an undetected data error reading the object file from disk or over a network, the failure should disappear or change when the program is run again. If the object file on disk is corrupted, compiling the program to generate correct object should fix the problem.

# Sort-Merge Errors

Sort-merge errors include errors processing a SORT or MERGE statement.

| Number | Description |
|---|---|
| 301 | There was insufficient memory available to initiate a sort or merge process. The default or specified sort memory size was insufficient to hold ten records of the record length to be sorted, or the specified sort memory size is not available. Use the T Runtime Command Option to increase the memory requested by the SORT statement. |
| 302 | Fewer than three intermediate files were available to begin a SORT statement. The sort procedure cannot begin unless it is able to create at least three intermediate files. |
| 303 | A record read from a MERGE file or SORT USING file was not long enough to include all the keys. |
| 304 | Too many out of sequence records were passed to the sort process. Use the T Runtime Command Option to increase the memory available to sort. Or, divide the records to be sorted into several files, sort the several files, and merge the resulting files. |
| 305 | A SORT or MERGE statement was attempted while a sort or merge process was already active. |
| 306 | A RELEASE or RETURN statement was attempted and no sort or merge was active. |
| 307 | A RELEASE or RETURN statement was attempted for a sort or merge description other than the one currently being sorted or merged. |
| 308 | A RELEASE statement was attempted in an OUTPUT PROCEDURE, or a RETURN statement was attempted in an INPUT PROCEDURE. |
| 309 | A RETURN statement was attempted in an OUTPUT PROCEDURE after the at end condition was returned on the sort or merge file. |
| 310 | An application I/O statement was attempted on a file currently opened as a sort or merge USING or GIVING file. |

# Message Control Errors

Message control errors include errors that occur when using the Message Control System (MCS).

| Number | Description |
|---|---|
| 351 | An ENABLE, DISABLE, SEND, RECEIVE or ACCEPT . . . FROM MESSAGE COUNT statement was encountered and no MCS was present. |

# Configuration Errors

Configuration errors include all errors that occur because of an error in the configuration. The formats are as follows:

```
COBOL configuration error code at record
      number in location.
```

```
COBOL configuration error code.  Error
      processing configuration.
```

*code* is the error number listed below.

*number* identifies the logical record in the configuration file (location) at which the error was found. Each logical record is identified with a configuration record type. In other words, when you are using the record number provided in the message to determine the erroneous record, count the lines combined with their corresponding continuation lines as one line, and do not count the comment lines and blank lines.

*location* identifies the configuration file containing the error. The possible values are as follows:

- Attached configuration or automatic configuration file
- Overriding configuration file
- Supplemental configuration file

Attached configuration or automatic configuration file refers to configuration files either attached to the executable on Windows or located automatically by the automatic configuration support module. For more details, see Automatic and Attached Configuration Files (on page 282). Overriding configuration file refers to a configuration file specified by the C Runtime Command Option. Supplemental configuration file refers to a configuration file specified by the X Runtime Command Option. For an explanation of the Runtime Command Options, see Chapter 7: *Running* (on page 177).

The first format is used if an error is detected during the processing of a configuration record. The error message will be followed by a line containing the portion of the record being processed when the error occurred and another line placing a currency symbol underneath the item being processed when the error occurred.

The second format is used if an error is detected after all configuration records have been processed or if an error is detected with which a record is not associated.

| Number | Description |
|---|---|
| 401 | A character has been defined as a data character in a TERM-ATTR record as well as the beginning character of an input sequence in a TERM-INPUT record. |
| 402 | An invalid delimiter was found. |
| 403 | The input sequence in the current TERM-INPUT configuration record has been defined in a previous TERM-INPUT configuration record. |
| 404 | A keyword has not been provided where expected or the keyword is invalid. |
| 405 | The resulting terminal input configuration table (used by the runtime system during ACCEPT statements) is too large or out of memory. |
| 406 | An attempt has been made to redefine the terminal interface. |
| 407 | Syntax error. |
| 408 | An attempt has been made to describe a terminal characteristic and either a terminal interface has not been defined via a TERMINAL-INTERFACE record, or the terminal characteristic is not valid with the defined terminal interface. |
| 409 | A value has not been provided where expected or the value is invalid. |
| 410 | The configuration file requested in the Runtime Command was not found. |
| 411 | A logical configuration record exceeds the maximum length. |

# Runcobol Initialization Messages

During its initialization, the runtime system may issue several messages, some of which are error diagnostics and some of which are informational.

## Initialization Errors

If the runtime system receives an error from the operating system during initialization, the following message is displayed:

```
RM/COBOL: Operating System Initialization Error. OS Error Code: code
```

*code* depends on the operating system on which the file resides. See the description of the 30, OS error *code* on page 373 for more information.

## Support Module Initialization Errors

During initialization, the runtime system attempts to initialize each optional support module that has defined an initialization entry point (**RM_AddOnInit**). If the support module determines that successful initialization is not possible, the runtime system produces the following message:

RM/COBOL: Add-on software failed to initialize - *module-name*.

Other messages should indicate the reason why the support module could not complete initialization. Contact the provider of the failing support module if the information provided is not sufficient to resolve the problem.

## Support Module Version Errors

During initialization, the runtime system locates and loads various support modules, including either the terminfo or the termcap Terminal Interface support module. Also, at initialization, the runtime system verifies that each support module is the correct version for the runtime system. If a support module is not the correct version, the following message is displayed:

RM/COBOL: *module-name* version mismatch, expected 9.0*n.nn*, found *n.nn.nn*.

The runtime system may issue one of the following messages if a support module indicates either that it does not support any of the interface versions supported by the runtime system or that it cannot run with this version of the runtime system:

RM/COBOL: *module-name* bad interface version: *nn*; must be *nn* to *nn*.

RM/COBOL: *module-name* version check failed.

When any of the previous messages are displayed, the runtime system terminates with the following message:

Error invoking mismatched runtime and support module.

## Option Processing Errors

If the runtime system detects an unknown option letter, a known option letter in error or an option not terminated with a space or comma, the following message is displayed:

```
letter option not valid.
```

*letter* is the erroneous option letter.

Further processing is done for libraries (as specified by the L Runtime Command Option) during which control structures are built for use later in the run unit. If errors are encountered during this processing, the following message is displayed:

```
COBOL procedure error code. Error processing library
   library-name.
```

*code* is as defined in Procedure Errors (on page 362).

*library-name* is the name of the library being loaded.

## Main Program Loading Errors

If errors are encountered during the load and initialization of the main program, the following message is displayed:

```
COBOL procedure error code (description).  Error loading main
   program prog-name.
```

*code* is as defined in Procedure Errors (on page 362).

*prog-name* is the name of the program being loaded (or the pathname of the file being loaded, as described in the note below).

**Note** In some cases, this will be a procedure error 204 indicating that the application main program could not be found. If one or more program object files were found, but could not be successfully loaded, the 204 error will be preceded by one or more messages in the preceding format indicating the specific reason that the load failed along with the pathname of the file that could not be loaded in place of *prog-name*. Multiple files may be attempted because of the extension search the runtime performs. When the search completes without successfully loading a program, the 204 error is displayed after any of the load errors as follows:

```
COBOL procedure error 204 (program not found).  Error starting
   application prog-name.
```

204 is the procedure error code as defined in Procedure Errors (on page 362).

*prog-name* is the name of the program as specified in the runtime command line.

If the extension search does not find any matching file names, then only the error below is displayed:

```
COBOL procedure error 204 (program not found).  Error loading
   main program prog-name.
```

204 is the procedure error code as defined in Procedure Errors (on page 362).

*prog-name* is the name of the program as specified in the runtime command line.

## Runcobol Banner Message

The banner message is displayed when you first invoke RM/COBOL. This message may be suppressed with the K Runtime Command Option:

```
RM/COBOL Runtime - Version 9.0n.nn for operating system.
Copyright(c) 1985-2005 by Liant Software Corp. All rights reserved.
Registration Number:  xx-nnnn-nnnnn-nnnn
```

A verbose banner has additional information about the product and environment in which it is running. The verbose banner is obtained for the runtime by using the V Runtime Command Option or by setting the environment variable RM_VERBOSE_BANNER to a value that begins with “Y” or “y”. The verbose banner adds the following lines to the banner:

```
RM/COBOL: User user-name running on machine machine-name (system-name)
RM/COBOL: Native character set: ncs (Codepage: cp-number)
```

A verbose banner also includes the support module list as explained in the description of the V Runtime Command Option. The lines in the verbose banner are not suppressed by the K Runtime Command Option.

## Runcobol Usage Message

The following message is displayed in response to entering a Runtime Command without parameters:

```
Usage:   RUNCOBOL name [options]

Options: [A=arguments] [B=buffersize] [C=cfgfile1] [D]
         [F=fillchar] [I] [K] [L=libname] [M] [Q=mcssym]
         [S=switches] [T=sortsize] [V] [X=cfgfile2]
```

## Registration Error Messages

Attempting to execute more runtime systems than are authorized causes the following message to be displayed:

```
   Error invoking unauthorized copy of runtime.
```

# COBOL Normal Termination Messages

When the runtime system encounters a STOP RUN statement or a GOBACK statement in the main program of a run unit, the following message is displayed:

```
COBOL STOP RUN at line number in program prog-id
```

This message may be suppressed with the K Runtime Command Option.

When the runtime system encounters a STOP literal or STOP data item statement, the following message is displayed:

```
"literal or data item" at line number in program prog-id
   Continue (Y/N)?
```

If you enter Y or y, execution continues with the next executable statement. If you enter N or n, execution ends as if a STOP RUN statement was encountered.

# Appendix B: Limits and Ranges

This appendix describes RM/COBOL limits and ranges and file locking limitations.

**Note** See also Chapter 4: *System Considerations for Btrieve* (on page 111) for a description of the limitations of the Btrieve MicroKernel Database Engine (MKDE), and the way in which these limitations affect RM/COBOL indexed files. Although these two systems perform the same functions, they do not operate in the same manner.

## RM/COBOL Limits and Ranges

No more than 98133 unique user-defined words (alphabet-names, cd-names, class-names, condition-names, data-names, file-names, index-names, key-names, mnemonic-names, paragraph-names, section-names, and symbolic-characters), where each name is 30 characters in length, may be defined in a single source program. When names average less than 30 characters in length, more unique user-defined words are allowed. For example, names averaging 12 characters in length double the limit to 196,266 unique user-defined words. The COBOL user-defined word categories level-number, library-name, program-name, segment-number and text-name do not count towards this limit.

No more than 65534 identifiers may be defined in a single source program, including any of its nested source programs. Only 65533 identifiers may be defined in the Data Division (see below for information about compiler generated identifiers that occur in the Procedure Division). Note that the compiler implicitly defines identifier entries, reducing the number that can be explicitly defined, as follows:

- one for each LINAGE-COUNTER associated with any files described with the LINAGE clause;
- one for each variable length record file that has multiple record descriptions and no record description is a group having the maximum record length;
- one for each indexed file with a CODE-SET or COLLATING-SEQUENCE that requires that an external collating sequence be provided, that is, when the specified or implicit code-set for the file requires a mapping other than identity to obtain the specified or implicit collating sequence used for record keys of the file;

- one for each RERUN clause;
- one for a RETURN-CODE special register item if the program contains an explicit reference to RETURN-CODE or contains a STOP RUN identifier/integer statement;
- one for each temporary index needed for any INITIALIZE, INSPECT, MOVE, STRING or UNSTRING statements that reference one or more subscripted data items (the compiler reuses these for each statement, thus defining only the maximum needed for any one such statement); and
- five for control information if one or more simple (COBOL nucleus level 1) INSPECT statements are contained in the source program.

The maximum length of an elementary data item is 65280 characters.

The maximum length of an element of a table is 65280 characters (that is, data items subordinate to an OCCURS clause cannot exceed 65280 characters in length).

The maximum number of occurrences of a table element is 65535. That is, in an OCCURS clause, the value of *integer-2* must be less than or equal to 65535.

Data items greater than 65280 characters in length may be referenced in a MOVE statement or the USING phrase of a CALL statement, but may not be referred to in any other context.

Data items greater than 65280 characters in length may not be reference modified.

The maximum guaranteed precision for mathematical calculations is 36 decimal digits. This limit applies to all mathematical operations except exponentiation where the maximum guaranteed precision is 18 decimal digits.

A record key of an indexed file may not exceed 254 characters in length.

A maximum of 255 keys may be declared for an indexed file (one prime key and up to 254 alternate keys).

A record of a file may not exceed 65280 characters in length.

A single paragraph cannot exceed 32512 bytes of generated object code.

The maximum number of procedures (paragraphs or sections) defined in all the fixed permanent and fixed overlayable segments of a separately compiled program, including any of the programs contained in the separately compiled program, is 8191. The maximum number of procedures in any one independent overlayable segment is also 8191.

The maximum number of USING items in a CALL statement or a Procedure Division header is 255.

The maximum length of literals generated for any one segment type within a program may not exceed 65535 bytes. Segment type refers to fixed permanent, fixed overlayable and independent. Segmentation can greatly increase the number of literals a program may have, since literals are overlaid for each segment.

The maximum number of external items in a single, separately compiled program—including any of its contained programs—is limited to 2046.

**Note** For additional compilation limits, refer to Table 14: Abnormal Termination Messages in Compiler Status Messages (on page 168).

# File Locking

When this user's guide was produced, RM/COBOL for Windows and UNIX had the following file locking limitations:

- RM/COBOL implements file and record locks through region locks. The algorithm for computing the region to be locked is unique to RM/COBOL. This means that the OPEN and READ statements only lock out other RM/COBOL applications; an application not using the RM/COBOL file system can still access data in locked files. This may cause inconsistent data when a file is shared between RM/COBOL applications and other applications.
- RM/COBOL implements the WITH LOCK phrase of the OPEN statement and the WITH NO LOCK phrase of the READ statement by applying region locks to segments of the file. To ensure consistent results and to improve performance when the WITH LOCK phrase is specified, the runtime system must recognize when two distinct filenames identify the same file. This is accomplished under Windows by resolving the user filename into a fully specified filename, including the remote machine name when the file is remote. If a remote file resides on a UNIX server accessed by Network File System (NFS), however, two very different filenames can be associated with a single file through the **ln** command. This can cause a Windows client program to read invalid data from the file or to diagnose file integrity errors.

# Appendix C: Internal Data Formats

This appendix describes and illustrates the internal representations of the data types.

## Internal Data Formats

The particular format in which the RM/COBOL compiler stores data is determined by the specification of any of the following clauses in the data description entry for the data item:

1. **PICTURE clause.** This clause classifies a data item as signed or unsigned numeric or as nonnumeric.
2. **USAGE clause.** This clause modifies the format of a numeric data item. All nonnumeric data items are always considered to be DISPLAY usage.
3. **SIGN clause.** This clause determines the format and the position (separate leading, combined leading, separate trailing or combined trailing) of the operational sign for signed numeric DISPLAY data items. The SIGN clause cannot be used for any other type of data item.

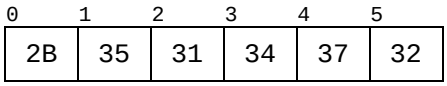
The following formats are described:

- **Nonnumeric data**
  - Alphanumeric (ANS)
  - Alphanumeric Edited (ANSE)
  - Alphabetic (ABS)
  - Alphabetic Edited (ABSE)
  - Numeric Edited (NSE)

- **Numeric data**
  - Numeric unsigned DISPLAY (NSU)
  - Numeric signed DISPLAY, TRAILING SEPARATE sign (NTS)
  - Numeric signed DISPLAY, LEADING SEPARATE sign (NLS)
  - Numeric signed DISPLAY, TRAILING sign (NTC)
  - Numeric signed DISPLAY, LEADING sign (NLC)
  - Numeric unsigned COMPUTATIONAL (NCU)
  - Numeric signed COMPUTATIONAL (NCS)
  - Numeric signed COMPUTATIONAL-1 (NBS)
  - Numeric unsigned positive COMPUTATIONAL-3 or PACKED-DECIMAL (NPP)
  - Numeric signed COMPUTATIONAL-3 or PACKED-DECIMAL (NPS)
  - Numeric unsigned COMPUTATIONAL-4 or BINARY (NBU)
  - Numeric signed COMPUTATIONAL-4 or BINARY (NBS)
  - Numeric unsigned COMPUTATIONAL-5 (NBUN)
  - Numeric signed COMPUTATIONAL-5 (NBSN)
  - Numeric unsigned COMPUTATIONAL-6 (NPU)

Illustrations are presented as follows:

```
01 EXAMPLE       PIC S9(3)V9(2)  VALUE +514.72
                 DISPLAY SIGN LEADING SEPARATE.
```




The line of source code shows the RM/COBOL data description entry. The values inside the box show the hexadecimal data values in bytes, as stored in memory. The numbers above the box show the hexadecimal relative byte address.

The ASCII code-set is used to represent all data items whose usage is DISPLAY.

# Nonnumeric Data

Nonnumeric data items are formatted one character per byte. The leftmost character starts at the lowest address. Edited nonnumeric data items have an associated editing PICTURE character-string that is used only when the data item is a receiving operand. Nonnumeric data items always have DISPLAY usage.

The SIGN clause does not apply to nonnumeric data items. The BLANK WHEN ZERO clause—when associated with a numeric PICTURE character-string—causes the data item to be treated as numeric edited.

Table 38 lists the types of nonnumeric data, the compiler designation for each type, and the valid picture symbols for each type.

**Table 38: Nonnumeric Data**

| Data Type | Compiler Designation | Picture Symbols |
|---|---|---|
| Alphanumeric | ANS | A X 9 |
| Alphanumeric Edited | ANSE | A B X 0 9 / |
| Alphabetic | ABS | A |
| Alphabetic Edited | ABSE | A B |
| Numeric Edited | NSE | B P V Z 0 9 / , . $ + – * CR DB |

When the CURRENCY SIGN clause is specified in the SPECIAL-NAMES paragraph, the character specified in that clause is also a permissible picture symbol for numeric edited data items.

The full ASCII code-set is stored as follows:

```
 01 ANS1          PIC X(95)  VALUE
                                " !""#$%&'()*+,-./0123456789:;<=>?
-                                "@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
-                                "`abcdefghijklmnopqrstuvwxyz{|}~"
                  DISPLAY.
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 00: | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |
| 08: | 28 | 29 | 2A | 2B | 2C | 2D | 2E | 2F |
| 10: | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 |
| 18: | 38 | 39 | 3A | 3B | 3C | 3D | 3E | 3F |
| 20: | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| 28: | 48 | 49 | 4A | 4B | 4C | 4D | 4E | 4F |
| 30: | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 |
| 38: | 58 | 59 | 5A | 5B | 5C | 5D | 5E | 5F |
| 40: | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 |
| 48: | 68 | 69 | 6A | 6B | 6C | 6D | 6E | 6F |
| 50: | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 |
| 58: | 78 | 79 | 7A | 7B | 7C | 7D | 7E | |

## Alphanumeric Edited

```
01 ANSE1          PIC XX/9900/AABB  VALUE "**/1200/QR  " DISPLAY.
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2A | 2A | 2F | 31 | 32 | 30 | 30 | 2F | 51 | 52 | 20 | 20 |

## Alphabetic

```
01 ABS1           PIC A(12)  VALUE "STOCK NUMBER" DISPLAY.
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 53 | 54 | 4F | 43 | 4B | 20 | 4E | 55 | 4D | 42 | 45 | 52 |

## Alphabetic Edited

```
01 ABSE1          PIC AABAABAA  VALUE "XX YY ZZ" DISPLAY.
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 58 | 58 | 20 | 59 | 59 | 20 | 5A | 5A |

## Numeric Edited

```
01 NSE1           PIC $*(5).**CR  VALUE "$***24.13  " DISPLAY.
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a |
|---|---|---|---|---|---|---|---|---|---|---|
| 24 | 2A | 2A | 2A | 32 | 34 | 2E | 31 | 33 | 20 | 20 |

```
01 NSE2           PIC $(6).99+  VALUE 4913.78 DISPLAY.
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 20 | 24 | 34 | 39 | 31 | 33 | 2E | 37 | 38 | 2B |

# Numeric Data

Any numeric data item, regardless of usage, must contain at least one digit position and may contain as many as thirty digit positions.

A numeric data item may vary in size from 1 to 31 bytes. Storage requirements for a numeric item depend on the usage, the number of digits, and the presence of an operational sign.

Numeric DISPLAY data items are formatted one digit character per byte. The number is formatted as an integer aligned with the most significant digit at the lowest address and the least significant digit at the highest address. The position of the implied decimal point is maintained in a separate data descriptor. The format descriptions of the numeric DISPLAY data items begin in the following topic.

Numeric computational data items are formatted in a variety of ways depending on the specific USAGE applied to the data item and whether the data item is unsigned or signed. The various numeric computational data item formats are described beginning with Unsigned Numeric Computational (on page 414).

## Unsigned Numeric DISPLAY (NSU)

No storage is allocated for an operational sign.

Valid picture symbols are 9, V and P. Usage is always DISPLAY.

### Format Illustrations

```
01 NSU1          PIC 9(3)V9(2)  VALUE 731.24  DISPLAY.
```

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 37 | 33 | 31 | 32 | 34 |

```
01 NSU2          PIC 9(6)V9(6)  VALUE 3456.7  DISPLAY.
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 30 | 30 | 33 | 34 | 35 | 36 | 37 | 30 | 30 | 30 | 30 | 30 |

```
01 NSU3          PIC 9(5)PP  VALUE 5712300 DISPLAY.
```

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 35 | 37 | 31 | 32 | 33 |

```
01 NSU4          PIC P9(7)  VALUE 0.09431726 DISPLAY.
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 39 | 34 | 33 | 31 | 37 | 32 | 36 |

# Signed Numeric DISPLAY (TRAILING SEPARATE)

Signed numeric DISPLAY data items described with the SIGN IS TRAILING SEPARATE clause are formatted with an additional byte for the operational sign. This sign occupies a separate byte following the least significant digit byte. A positive sign is represented by a value of 2Bh. A negative sign is represented by a value of 2Dh.

If the S (separate sign) Compile Command Option is specified, the compiler assumes the presence of the SIGN IS TRAILING SEPARATE clause for all signed numeric DISPLAY data items for which the SIGN clause is not explicitly stated.

Signed numeric DISPLAY data (TRAILING SEPARATE) is designated as NTS. The valid picture symbols are S, 9, V and P. Usage is always DISPLAY. The sign is always TRAILING SEPARATE.

### Format Illustrations

```
01 NTS1          PIC S9(3)V9(2)  VALUE -731.24
                 SIGN TRAILING SEPARATE DISPLAY.
```

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 37 | 33 | 31 | 32 | 34 | 2D |

```
01 NTS2          PIC S9(4)V9(7)  VALUE -34.567
                 SIGN TRAILING SEPARATE DISPLAY.
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 30 | 30 | 33 | 34 | 35 | 36 | 37 | 30 | 30 | 30 | 30 | 2D |

```
01 NTS3          PIC S9(5)PP  VALUE +5712300
                 SIGN TRAILING SEPARATE DISPLAY.
```

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 35 | 37 | 31 | 32 | 33 | 2B |

```
01 NTS4          PIC SP9(7)  VALUE -.09431726
                 SIGN TRAILING SEPARATE DISPLAY.
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 39 | 34 | 33 | 31 | 37 | 32 | 36 | 2D |

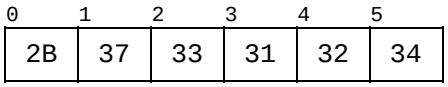
# Signed Numeric DISPLAY (LEADING SEPARATE)

Signed numeric DISPLAY data items described with the SIGN IS LEADING SEPARATE clause are formatted with an additional byte for the operational sign. The operational sign occupies a separate byte preceding the most significant digit byte. A positive sign is represented by a value of 2Bh. A negative sign is represented by a value of 2Dh.

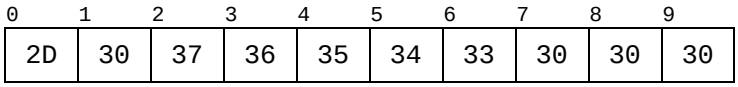
Signed numeric DISPLAY data (LEADING SEPARATE) is designated as NLS. The valid picture symbols are S, 9, V and P. Usage is always DISPLAY. The sign is always LEADING SEPARATE.

### Format Illustrations

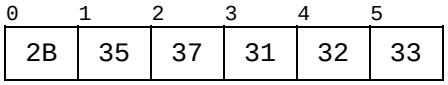
```
01 NLS1          PIC S9(3)V9(2)  VALUE +731.24
                 SIGN LEADING SEPARATE DISPLAY.
```



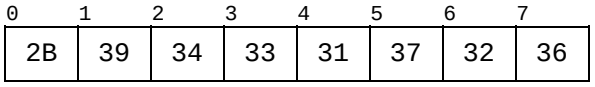

```
01 NLS2          PIC S9(2)V9(7)  VALUE -7.6543
                 SIGN LEADING SEPARATE DISPLAY.
```




```
01 NLS3          PIC S9(5)PP  VALUE +5712300
                 SIGN LEADING SEPARATE DISPLAY.
```




```
01 NLS4          PIC SP9(7)  VALUE +.09431726
                 SIGN LEADING SEPARATE DISPLAY.
```



## Signed Numeric DISPLAY (TRAILING)

Signed numeric DISPLAY data items described with the SIGN IS TRAILING clause are formatted with the highest addressed byte containing the operational sign combined with the low-order digit.

Because of the combined digit and sign, this format is sometimes called trailing combined sign. It is also sometimes called trailing zoned sign because the sign is represented in a manner consistent with a sign zone punch and digit punch on Hollerith punch cards.

The hexadecimal values that result from this combination are shown in Table 39.

**Table 39: Combined Digit and Sign**

| Digit | Positive | Negative |
|---|---|---|
| 0 | 7B | 7D |
| 1 | 41 | 4A |
| 2 | 42 | 4B |
| 3 | 43 | 4C |
| 4 | 44 | 4D |
| 5 | 45 | 4E |
| 6 | 46 | 4F |
| 7 | 47 | 50 |
| 8 | 48 | 51 |
| 9 | 49 | 52 |

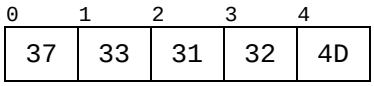
This format is the default for all signed numeric DISPLAY data items when the SIGN clause is not specified, unless the S (separate sign) Compile Command Option is specified.

Signed numeric DISPLAY data (TRAILING) is designated as NTC. The valid picture symbols are S, 9, V and P. Usage is always DISPLAY. The sign is always TRAILING.

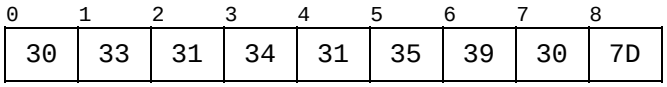
Note that if an NTC data item is sent directly to a printer or display device, the position that contains the combined sign and digit appears as a special character or letter. A positive zero value appears as {. A negative zero value appears as }. Positive 1 through 9 values appear as A through I. Negative 1 through 9 values appear as J through R.

### Format Illustrations

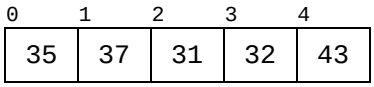
```
01 NTC1          PIC S9(3)V9(2)  VALUE -731.24
                 SIGN TRAILING DISPLAY.
```



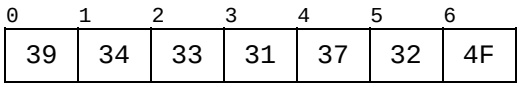

```
01 NTC2          PIC S9(2)V9(7)  VALUE -3.14159
                 SIGN TRAILING DISPLAY.
```




```
01 NTC3          PIC S9(5)PP  VALUE +5712300
                 SIGN TRAILING DISPLAY.
```




```
01 NTC4          PIC SP9(7)  VALUE -.09431726
                 SIGN TRAILING DISPLAY.
```



# Signed Numeric DISPLAY (LEADING)

Signed numeric DISPLAY data items described with the SIGN IS LEADING clause are formatted with the lowest addressed byte containing the operational sign combined with the high-order digit. The hexadecimal values that result from this combination are listed in Table 39 on page 411.

Because of the combined digit and sign, this format is sometimes called leading combined sign. It is also sometimes called leading zoned sign because the sign is represented in a manner consistent with a sign zone punch and digit punch on Hollerith punch cards.

Signed numeric DISPLAY data (LEADING) is designated as NLC. The valid picture symbols are S, 9, V and P. Usage is always DISPLAY. The sign is always LEADING.

Note that if an NLC data item is sent directly to a printer or display device, the position that contains the combined sign and digit appears as a special character or letter. A positive zero value appears as {. A negative zero value appears as }. Positive 1 through 9 values appear as A through I. Negative 1 through 9 values appear as J through R.

### Format Illustrations

```
01 NLC1          PIC S9(3)V9(2)  VALUE -731.24
                 SIGN LEADING DISPLAY.
```

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 50 | 33 | 31 | 32 | 34 |

```
01 NLC2          PIC S9V9(7)  VALUE -2.9876
                 SIGN LEADING DISPLAY.
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 4B | 39 | 38 | 37 | 36 | 30 | 30 | 30 |

```
01 NLC3          PIC S9(5)PP  VALUE +5712300
                 SIGN LEADING DISPLAY.
```

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 45 | 37 | 31 | 32 | 33 |

```
01 NLC4          PIC SP9(7)  VALUE +.09431726
                 SIGN LEADING DISPLAY.
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 49 | 34 | 33 | 31 | 37 | 32 | 36 |

## Unsigned Numeric COMPUTATIONAL

For the default configuration, COMPUTATIONAL (or COMP) data items are formatted as one binary coded decimal digit per byte.

**Note** The compiler can be configured to treat data items described as COMPUTATIONAL (or COMP) as having been described as BINARY, PACKED-DECIMAL, or DISPLAY using the COMPUTATIONAL-TYPE keyword of the COMPILER-OPTIONS configuration record. In those cases, the default format for COMPUTATIONAL data items described here does not apply; please refer to the corresponding section of this appendix if you have configured one of the other formats for COMPUTATIONAL data.

The number is formatted as an integer aligned with the most significant digit at the lowest address and the least significant digit at the highest address. The position of the implied decimal point is maintained in a separate data descriptor.

No storage is allocated for an operational sign.

Unsigned numeric COMPUTATIONAL data is designated as NCU. The valid picture symbols are 9, V and P. Usage is always COMPUTATIONAL or COMP.

### Format Illustrations

```
01 NCU1          PIC 9(3)V9(2)  VALUE 731.24
                 COMPUTATIONAL.
```

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 07 | 03 | 01 | 02 | 04 |

```
01 NCU2          PIC 9(3)V9(4)  VALUE 23.45
                 COMPUTATIONAL.
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 00 | 02 | 03 | 04 | 05 | 00 | 00 |

```
01 NCU3          PIC 9(5)PP  VALUE 5712300
                 COMPUTATIONAL.
```

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 05 | 07 | 01 | 02 | 03 |

```
01 NCU4          PIC P9(7)  VALUE .09431726
                 COMPUTATIONAL.
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 09 | 04 | 03 | 01 | 07 | 02 | 06 |

# Signed Numeric COMPUTATIONAL

For the default configuration, signed numeric COMPUTATIONAL (or COMP) data items are formatted as one binary coded decimal digit per byte.

**Note** The compiler can be configured to treat data items described as COMPUTATIONAL (or COMP) as having been described as BINARY, PACKED-DECIMAL, or DISPLAY using the COMPUTATIONAL-TYPE keyword of the COMPILER-OPTIONS configuration record. In those cases, the default format for signed numeric COMPUTATIONAL data items described here does not apply; please refer to the corresponding section of this appendix if you have configured one of the other formats for COMPUTATIONAL data.

The number is formatted as an integer aligned with the most significant digit at the lowest address and the least significant digit at the highest address. The position of the implied decimal point is maintained in a separate data descriptor.

The operational sign occupies a separate byte following the least significant digit. A negative sign is represented by the value 0Dh. A positive sign is normally represented by a value of 0Ch. The COMPUTATIONAL-VERSION keyword of the COMPILER-OPTIONS configuration record may alter the positive sign representation. Selecting a value of RMCOBOL2 or RMCOS for COMPUTATIONAL-VERSION will cause the positive sign to be represented by a value of 0Bh.

Signed COMPUTATIONAL data is designated as NCS, unpacked signed. The valid picture symbols are S, 9, V and P. Usage is always COMPUTATIONAL or COMP.

### Format Illustrations

```
01 NCS1          PIC S9(3)V9(2)  VALUE -731.24
                 COMPUTATIONAL.
```

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 07 | 03 | 01 | 02 | 04 | 0D |

```
01 NCS2          PIC S9(3)V9(4)  VALUE 123.4567
                 COMPUTATIONAL.
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 01 | 02 | 03 | 04 | 05 | 06 | 07 | 0C |

```
01 NCS3          PIC S9(5)PP  VALUE +5712300
                 COMPUTATIONAL.
```

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 05 | 07 | 01 | 02 | 03 | 0C |

```
01 NCS4          PIC SP9(7)  VALUE -.09431726
                 COMPUTATIONAL.
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 09 | 04 | 03 | 01 | 07 | 02 | 06 | 0D |

## Signed Numeric COMPUTATIONAL-1 Data

Signed numeric COMPUTATIONAL-1 (or COMP-1) data items are formatted as 2's complement binary words. The number of 9's in the PICTURE character-string does not affect the allocation size for COMPUTATIONAL-1 data items. Such data items are always considered signed without regard to the presence or absence of an S in the associated PICTURE character-string. COMP-1 usage is restricted to integer data items without "P" scaling.

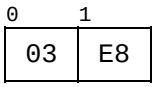
The number is formatted as a 2's complement binary word with the most significant byte at the lowest address and the least significant byte at the highest address.

The operational sign is indicated by the 2's complement format. If the most significant bit is zero, the number is positive. If this bit is one, the number is negative.

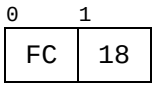
Signed numeric COMPUTATIONAL-1 data is designated as NBS, Binary Signed. The valid picture symbols are S, 9 and V. Usage is always COMPUTATIONAL-1 or COMP-1.

### Format Illustrations

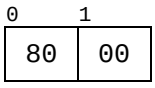
```
01 NBS1-COMP-1   PIC S9(4)  VALUE +1000
                 COMPUTATIONAL-1.
```



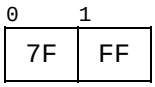

```
01 NBS2-COMP-1   PIC S9(4)  VALUE -1000
                 COMPUTATIONAL-1.
```




```
01 NBS3-COMP-1   PIC S9(5)  VALUE -32768
                 COMPUTATIONAL-1.
```




```
01 NBS4-COMP-1   PIC S9(5)  VALUE +32767
                 COMPUTATIONAL-1.
```



# Unsigned Numeric COMPUTATIONAL-3 Data

Unsigned numeric COMPUTATIONAL-3 (or COMP-3) and PACKED DECIMAL data items are formatted as two binary coded decimal digits per byte, except for the highest addressed byte which contains one digit followed by an operational sign. If the associated PICTURE character-string contains an even number of 9's, an additional high-order digit is included in the storage allocated for the data item so as to complete an integral number of bytes.

The number is formatted as an integer aligned with the most significant digit at the lowest address and the least significant digit at the highest address. The position of the implied decimal point is maintained in a separate data descriptor.

The operational sign is concatenated with the least significant digit in the highest addressed byte. The high-order four bits of the byte is the least significant digit and the low-order four bits of the byte is the operational sign. The sign is always Fh, denoting a positive value.

Unsigned numeric COMPUTATIONAL-3 data is designated as NPP, packed unsigned. The valid picture symbols are 9, V and P. Usage is COMPUTATIONAL-3, COMP-3, or PACKED-DECIMAL.

### Format Illustrations

```
01 NPP1          PIC 9(3)V9(3)  VALUE 731.246
                 PACKED-DECIMAL.
```

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 07 | 31 | 24 | 6F |

```
01 NPP2          PIC 9(8)V9(7)  VALUE 123456.789
                 PACKED-DECIMAL.
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 00 | 12 | 34 | 56 | 78 | 90 | 00 | 0F |

```
01 NPP3          PIC 9(5)PP  VALUE 5712300
                 PACKED-DECIMAL.
```

| 0 | 1 | 2 |
|---|---|---|
| 57 | 12 | 3F |

```
01 NPP4          PIC P9(7)  VALUE .09431726
                 PACKED-DECIMAL.
```

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 94 | 31 | 72 | 6F |

## Signed Numeric COMPUTATIONAL-3 Data

Signed numeric COMPUTATIONAL-3 (or COMP-3) and PACKED DECIMAL data items are formatted as two binary coded decimal digits per byte, except for the highest addressed byte, which contains one digit followed by an operational sign. If the associated PICTURE character-string contains an even number of 9's, an additional high-order digit is included in the storage allocated for the data item so as to complete an integral number of bytes.

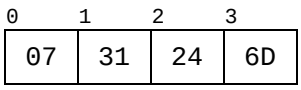
The number is formatted as an integer aligned with the most significant digit at the lowest address and the least significant digit at the highest address. The position of the implied decimal point is maintained in a separate data descriptor.

The operational sign is concatenated with the least significant digit in the highest addressed byte. The high-order four bits of the byte is the least significant digit and the low-order four bits of the byte is the operational sign. A negative sign is represented by the value Dh. A positive sign is normally represented by a value of Ch. The COMPUTATIONAL-VERSION keyword of the COMPILER-OPTIONS configuration record may alter the positive sign representation. Selecting a value of RMCOBOL2 for COMPUTATIONAL-VERSION causes the positive sign to be represented by a value of Fh. Selecting a value of RMCOS for COMPUTATIONAL-VERSION causes the positive sign to be represented by a value of Bh.

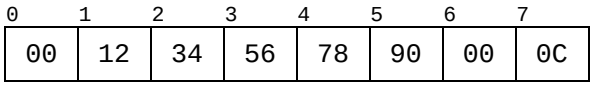
Signed numeric COMPUTATIONAL-3 data is designated as NPS, packed signed. The valid picture symbols are S, 9, V and P. Usage is COMPUTATIONAL-3, COMP-3 or PACKED-DECIMAL.

### Format Illustrations

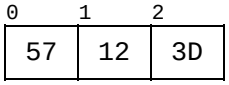
```
01 NPS1          PIC S9(3)V9(3)  VALUE -731.246
                 PACKED-DECIMAL.
```



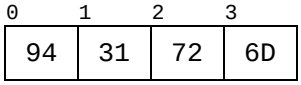

```
01 NPS2          PIC S9(8)V9(7)  VALUE 123456.789
                 PACKED-DECIMAL.
```




```
01 NPS3          PIC S9(5)PP  VALUE -5712300
                 PACKED-DECIMAL.
```




```
01 NPS4          PIC SP9(7)  VALUE -.09431726
                 PACKED-DECIMAL.
```



## Unsigned Numeric COMPUTATIONAL-4 Data

Unsigned numeric COMPUTATIONAL-4 (or COMP-4) and BINARY data items are formatted as binary bytes. If the USAGE clause specified a binary allocation override, the number of bytes allocated is the number specified in the binary allocation override. Otherwise, the number of bytes depends on the number of 9's in the associated PICTURE character-string and the setting of the BINARY-ALLOCATION and BINARY-ALLOCATION-SIGNED keywords of the COMPILER-OPTIONS configuration record according to Table 40 (on page 420).

**Note** When BINARY-ALLOCATION=CUSTOM=*integer-list* is configured for the compiler, then the compiler allocates the smallest number of bytes that support the decimal precision indicated by the PICTURE character-string and that is an allowed size in *integer-list*. See Table 40 under BINARY-ALLOCATION=MF-RM for the smallest number of bytes needed for an unsigned binary numeric data item of a given decimal precision. However, if BINARY-ALLOCATION-SIGNED=YES is also configured for the compiler, the number of byes allocated will be according to Table 41 (on page 423) instead of Table 40. If *integer-list* does not include a size large enough to support the decimal precision, the compiler produces a warning and uses the default allocation sizes shown for BINARY-ALLOCATION=RM.

The number is formatted as a binary integer with the most significant byte at the lowest address and the least significant byte at the highest address. The position of the decimal point is maintained in a separate data descriptor.

For unsigned binary items, there is no operational sign. In particular, the most significant bit is not indicative of the sign. The value is always interpreted as a positive number.

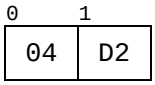
Unsigned numeric COMPUTATIONAL-4 data is designated as NBU, binary unsigned. The valid picture symbols are 9, V and P. Usage is COMPUTATIONAL-4, COMP-4 or BINARY.

**Table 40: Bytes Allocated for an Unsigned Binary Numeric Data Item**

| **BINARY-ALLOCATION=RM (Default)** | |
|---|---|
| **Number of 9's in PICTURE character-string** | **Bytes Allocated** |
| 1-4 | 2 |
| 5-9 | 4 |
| 10-18 | 8 |
| 19-30 | 16 |
| **BINARY-ALLOCATION=RM1** | |
| **Number of 9's in PICTURE character-string** | **Bytes Allocated** |
| 1-2 | 1 |
| 3-4 | 2 |
| 5-9 | 4 |
| 10-18 | 8 |
| 19-30 | 16 |
| **BINARY-ALLOCATION=MF-RM** | |
| **Number of 9's in PICTURE character-string** | **Bytes Allocated** |
| 1-2 | 1 |
| 3-4 | 2 |
| 5-7 | 3 |
| 8-9 | 4 |
| 10-12 | 5 |
| 13-14 | 6 |
| 15-16 | 7 |
| 17-19 | 8 |
| 20-21 | 9 |
| 22-24 | 10 |
| 25-26 | 11 |
| 27-28 | 12 |
| 29-30 | 13 |

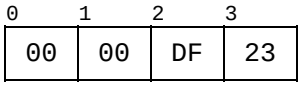
### Format Illustrations

```
01 NBU1          PIC 9(3)V9  VALUE 123.4
                 BINARY.
```



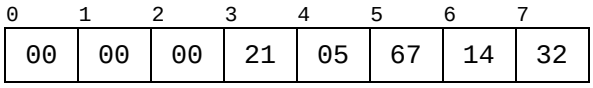
| 0 | 1 |
|---|---|
| 04 | D2 |

```
01 NBU2          PIC 9(5)PP  VALUE 5712300
                 BINARY.
```



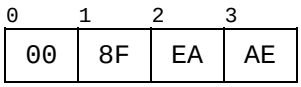
| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 00 | 00 | DF | 23 |

```
01 NBU3          PIC 9(18)  VALUE 141824562226
                 BINARY.
```



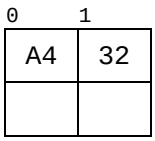
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 00 | 00 | 00 | 21 | 05 | 67 | 14 | 32 |

```
01 NBU4          PIC P9(7)  VALUE .09431726
                 BINARY.
```



| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 00 | 8F | EA | AE |

```
01 NBU5          PIC 9(5)  VALUE 42034
                 BINARY(2).
```



| 0 | 1 |
|---|---|
| A4 | 32 |
| | |

## Signed Numeric COMPUTATIONAL-4 Data

Signed numeric COMPUTATIONAL-4 (or COMP-4) and BINARY data items are formatted as 2's complement binary bytes. If the USAGE clause specified a binary allocation override, the number of bytes allocated is the number specified in the binary allocation override. Otherwise, the number of bytes depends on the number of 9's in the associated PICTURE character-string and the setting of the BINARY-ALLOCATION keyword of the COMPILER-OPTIONS configuration record according to Table 41 (on page 423).

**Note** When BINARY-ALLOCATION=CUSTOM=*integer-list* is configured for the compiler, then the compiler allocates the smallest number of bytes that support the decimal precision indicated by the PICTURE character-string and that is an allowed size in *integer-list*. See Table 41 under BINARY-ALLOCATION=MF-RM for the smallest number of bytes needed for a signed binary numeric data item of a given decimal precision. If *integer-list* does not include a size large enough to support the decimal precision, the compiler produces a warning and uses the default allocation sizes shown for BINARY-ALLOCATION=RM.

The number is formatted as a binary integer with the most significant byte at the lowest address and the least significant byte at the highest address. The position of the decimal point is maintained in a separate data descriptor.

The operational sign is indicated by the 2's complement format. If the most significant bit is zero, the number is positive. If this bit is one, the number is negative.

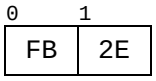
Signed numeric COMPUTATIONAL-4 data is designated as NBS, binary signed. The valid picture symbols are S, 9, V and P. Usage is COMPUTATIONAL-4, COMP-4 or BINARY.

**Table 41: Bytes Allocated for a Signed Binary Numeric Data Item**

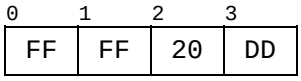
| **BINARY-ALLOCATION=RM (Default)** | |
|---|---|
| **Number of 9's in PICTURE character-string** | **Bytes Allocated** |
| 1-4 | 2 |
| 5-9 | 4 |
| 10-18 | 8 |
| 19-30 | 16 |
| **BINARY-ALLOCATION=RM1** | |
| **Number of 9's in PICTURE character-string** | **Bytes Allocated** |
| 1-2 | 1 |
| 3-4 | 2 |
| 5-9 | 4 |
| 10-18 | 8 |
| 19-30 | 16 |
| **BINARY-ALLOCATION=MF-RM** | |
| **Number of 9's in PICTURE character-string** | **Bytes Allocated** |
| 1-2 | 1 |
| 3-4 | 2 |
| 5-6 | 3 |
| 7-9 | 4 |
| 10-11 | 5 |
| 12-14 | 6 |
| 15-16 | 7 |
| 17-18 | 8 |
| 19-21 | 9 |
| 22-23 | 10 |
| 24-26 | 11 |
| 27-28 | 12 |
| 29-30 | 13 |

### Format Illustrations

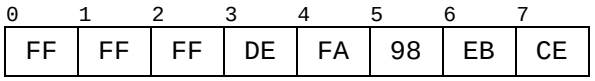
```
01 NBS1          PIC S9(3)V9  VALUE -123.4
                 BINARY.
```



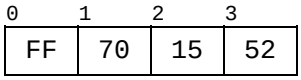

```
01 NBS2          PIC S9(5)PP  VALUE -5712300
                 BINARY.
```




```
01 NBS3          PIC S9(18)  VALUE -141824562226
                 BINARY.
```




```
01 NBS4          PIC SP9(7)  VALUE -.09431726
                 BINARY.
```




```
01 NBS5          PIC S9(3)  VALUE 95
                 BINARY(1).
```

## Unsigned Numeric COMPUTATIONAL-5 Data

Unsigned numeric COMPUTATIONAL-5 (or COMP-5) data items are formatted as binary bytes. If the USAGE clause specified a binary allocation override, the number of bytes allocated is the number specified in the binary allocation override. Otherwise, the number of bytes depends on the number of 9's in the associated PICTURE character-string and the setting of the BINARY-ALLOCATION and BINARY-ALLOCATION-SIGNED keywords of the COMPILER-OPTIONS configuration record according to Table 40 (on page 420).

**Note** When BINARY-ALLOCATION=CUSTOM=*integer-list* is configured for the compiler, then the compiler allocates the smallest number of bytes that support the decimal precision indicated by the PICTURE character-string and that is an allowed size in *integer-list*. See Table 40 under BINARY-ALLOCATION=MF-RM for the smallest number of bytes needed for an unsigned binary numeric data item of a given decimal precision. However, if BINARY-ALLOCATION-SIGNED=YES is also configured for the compiler, the number of byes allocated will be according to Table 41 (on page 423) instead of Table 40. If *integer-list* does not include a size large enough to support the decimal precision, the compiler produces a warning and uses the default allocation sizes shown for BINARY-ALLOCATION=RM.

The number is formatted as a binary integer with machine native byte ordering as follows:

- On "little-endian" machines, the format is the least significant byte at the lowest address and most significant byte at the highest address, that is, reversed byte ordering from COMPUTATIONAL-4 (COMP-4, BINARY). The position of the decimal point is maintained in a separate data descriptor.
- On "big endian" machines, the format is the same as COMPUTATIONAL-4 (COMP-4, BINARY), with the most significant byte at the lowest address and the least significant byte at the highest address.

For unsigned binary items, there is no operational sign. In particular, the most significant bit is not indicative of the sign. The value is always interpreted as a positive number.

Unsigned numeric COMPUTATIONAL-5 data is designated as NBUN, binary unsigned native. The valid picture symbols are 9, V and P. Usage is COMPUTATIONAL-5 or COMP-5.

**Note** COMPUTATIONAL-5 data is intended only for interfacing with non-COBOL programs. The data items are not portable between differing machine memory architectures and thus should not be used in data descriptions entries subordinate to a file description entry; that is, they should only be specified in the Working-Storage and Linkage Sections of a COBOL program.

### Format Illustrations

Because the format is machine dependent, format illustrations are not provided for COMPUTATIONAL-5 (COMP-5) data items. On "big-endian" machines, the illustrations given for unsigned COMPUTATIONAL-4 (COMP-4, BINARY) apply, since the format is the same. On "little-endian" machines, the illustrations given for unsigned COMPUTATIONAL-4 (COMP-4, BINARY) apply except that the bytes would appear in the reverse order; that is, the format illustration addressing should be numbered starting with zero on the right and ascending to the left.

## Signed Numeric COMPUTATIONAL-5 Data

Signed numeric COMPUTATIONAL-5 (or COMP-5) data items are formatted as 2's complement binary bytes. If the USAGE clause specified a binary allocation override, the number of bytes allocated is the number specified in the binary allocation override. Otherwise, the number of bytes depends on the number of 9's in the associated PICTURE character-string and the setting of the BINARY-ALLOCATION keyword of the COMPILER-OPTIONS configuration record according to Table 41 (on page 423).

**Note** When BINARY-ALLOCATION=CUSTOM=*integer-list* is configured for the compiler, then the compiler allocates the smallest number of bytes that support the decimal precision indicated by the PICTURE character-string and that is an allowed size in *integer-list*. See Table 41 under BINARY-ALLOCATION=MF-RM for the smallest number of bytes needed for a signed binary numeric data item of a given decimal precision. If *integer-list* does not include a size large enough to support the decimal precision, the compiler produces a warning and uses the default allocation sizes shown for BINARY-ALLOCATION=RM.

The number is formatted as a binary integer with machine native byte ordering as follows:

- On "little-endian" machines, the format is the least significant byte at the lowest address and most significant byte at the highest address, that is, reversed byte ordering from COMPUTATIONAL-4 (COMP-4, BINARY). The position of the decimal point is maintained in a separate data descriptor.
- On "big-endian" machines, the format is the same as COMPUTATIONAL-4 (COMP-4, BINARY), with the most significant byte at the lowest address and the least significant byte at the highest address.

The operational sign is indicated by the 2's complement format. If the most significant bit is zero, the number is positive. If this bit is one, the number is negative.

Signed numeric COMPUTATIONAL-5 data is designated as NBSN, binary signed native. The valid picture symbols are S, 9, V and P. Usage is COMPUTATIONAL-5 or COMP-5.

**Note** COMPUTATIONAL-5 data is intended only for interfacing with non-COBOL programs. The data items are not portable between differing machine memory architectures and thus should not be used in data descriptions entries subordinate to a file description entry; that is, they should only be specified in the Working-Storage and Linkage Sections of a COBOL program.

### Format Illustrations

Because the format is machine dependent, format illustrations are not provided for COMPUTATIONAL-5 (COMP-5) data items. On "big-endian" machines, the illustrations given for signed COMPUTATIONAL-4 (COMP-4, BINARY) apply, since the format is the same. On "little-endian" machines, the illustrations given for signed COMPUTATIONAL-4 (COMP-4, BINARY) apply except that the bytes would appear in the reverse order; that is, the format illustration addressing should be numbered starting with zero on the right and ascending to the left.

# Unsigned Numeric COMPUTATIONAL-6 Data

Unsigned numeric COMPUTATIONAL-6 (COMP-6) data items are formatted as two binary coded decimal digits per byte. If the picture contains an odd number of 9's, an additional high-order digit is included in the storage allocated for the item.

The number is formatted as an integer aligned with the most significant digit at the lowest address and the least significant digit at the highest address. The position of the implied decimal point is maintained in a separate data descriptor.

No storage is allocated for an operational sign.

Unsigned numeric COMPUTATIONAL-6 data is designated as NPU, packed unsigned. The valid picture symbols are 9, V and P. Usage is COMPUTATIONAL-6 or COMP-6.

### Format Illustrations

```
01 NPU1          PIC 9(3)V9(2)  VALUE 731.24
                 COMPUTATIONAL-6.
```

| 0 | 1 | 2 |
|---|---|---|
| 07 | 31 | 24 |

```
01 NPU2          PIC 9(8)V9(8)  VALUE 123456.789
                 COMPUTATIONAL-6.
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 00 | 12 | 34 | 56 | 78 | 90 | 00 | 00 |

```
01 NPU3          PIC 9(5)PP  VALUE 5712300
                 COMPUTATIONAL-6.
```

| 0 | 1 | 2 |
|---|---|---|
| 05 | 71 | 23 |

```
01 NPU4          PIC P9(6)  VALUE .0943172
                 COMPUTATIONAL-6.
```

| 0 | 1 | 2 |
|---|---|---|
| 94 | 31 | 72 |

# Pointer Data

Pointer data items have USAGE POINTER specified in their data description entry and are not described with a PICTURE clause. The format of pointer data items may change in future releases of the RM/COBOL product, so programs that depend on the information provided here may be tied to a particular version of RM/COBOL. This information is provided for clarification only. Programs should depend only on the characteristics of pointers described in the *RM/COBOL Language Reference Manual*.

An RM/COBOL pointer data item is an array of three 64-bit quantities as follows:

1. The base address of the memory area to which the pointer refers.
2. The current offset from the base address as set by SET *pointer* UP/DOWN statements.
3. The length of the memory area to which the base address points.

All three quantities are stored from most significant byte in the lowest address to least significant byte at the highest address, regardless of the machine memory architecture.

Programs that modify any of the three values of a pointer data item directly, for example, by redefinition as numeric quantities and use of arithmetic statements, may result in program failures that are difficult to explain. In COBOL, pointer data items should be manipulated only with the Formats 5 and 6 of the SET statement, as described in the *RM/COBOL Language Reference Manual*, or by use of the C$MemoryAllocate (on page 552), C$MemoryDeallocate (on page 553), and C$CompilePattern (on page 535) subprograms, as described in Appendix F: *Subprogram Library*. In non-COBOL subprograms, CodeBridge Library routines should be used to access or modify pointer data items passed as arguments to the non-COBOL subprogram (see the *CodeBridge* manual).

# Appendix D: Support Modules (Non-COBOL Add-Ons)

This appendix introduces you to the optional support modules that are included with RM/COBOL and other support modules from several add-on packages available from Liant Software. This appendix also describes which support modules are used with the runtime system, the compiler, and the Indexed File Recovery (**recover1**) utility components. Also provided are details about how RM/COBOL locates support modules and information on how to build a COBOL-callable, non-COBOL subprogram library support module or a Message Control System (MCS) support module of your own.

## Introduction

Beginning with version 7.1 of RM/COBOL for UNIX, the **customiz**/relinking method for adding features to RM/COBOL was replaced with a simpler, more flexible system using optional support modules. Consequently, it is no longer necessary or possible to relink the runtime system, the compiler, or the recovery utility. The **customiz** Bourne Shell script that was available in previous versions of RM/COBOL is no longer available.

This new method of adding features to RM/COBOL does not require the developer or the end-user to relink any of the RM/COBOL products, and, unless you are adding new C routines, does not require a C compiler or linker. Furthermore, multiple different versions of the RM/COBOL runtime system and the recovery utility are no longer required to support the two different terminal interfaces, terminfo and termcap. RM/COBOL for UNIX now uses separate support modules to support the two terminal interfaces so that only a single runtime and recovery utility are present on the distribution media. The terminal interface is still selected at installation.

While it is no longer possible to relink the runtime, it is still possible to build custom add-on, non-COBOL library optional support modules. Liant Software encourages developers to use the CodeBridge cross-language call system to produce specialized support modules that will be useful to other RM/COBOL customers.

RM/COBOL for Windows has long supported add-on, non-COBOL library optional support modules as Windows dynamic-link libraries (DLLs). Starting with version 7.5 of RM/COBOL for Windows, this support for optional support modules as dynamic-link libraries has been more closely aligned with the support for optional support modules as shared objects on UNIX.

# Overview of Optional Support Modules

The optional support modules are implemented as shared objects on UNIX and as dynamic-link libraries on Windows. Shared objects (with an extension of **.so**) and dynamic-link libraries (with an extension of **.dll**) allow an executable program to load additional code at execution time rather than at link time. Shared objects are sometimes called dynamic libraries because they are dynamically loaded. RM/COBOL uses several support modules to add additional features and allows developers to write their own support modules to add COBOL-callable, non-COBOL subprograms to the runtime system.

Installation for the support modules that are distributed as part of the RM/COBOL system occurs as part of the normal product installation, as described in System Installation for UNIX (on page 18) and System Installation for Windows (on page 51). Other support modules are installed according to the directions associated with particular add-on packages. Normally, the installation is as easy as copying the support module into either the execution directory (typically, **/usr/bin** on UNIX or **C:\Program Files\Liant\RMCOBOL** on Windows, but selectable during product installation) or the **rmcobolso** subdirectory of the UNIX execution directory or the **RmAutoLd** subdirectory of the Windows execution directory. A user-written support module (on page 439) should be installed in the **rmcobolso** subdirectory of the UNIX execution directory or the **RmAutoLd** subdirectory of the Windows execution directory. For information on how the RM/COBOL runtime, the RM/COBOL compiler, and the Indexed File Recovery utility locate optional support modules, see Locating Optional Support Modules (on page 431).

**Note** Versions of the RM/COBOL runtime system prior to 7.1 allowed the Pop-Up Window Manager to be included or excluded when relinking the runtime. In this version of RM/COBOL, the Pop-Up Window Manager is always present and cannot be removed.

Not all support modules can be used with all RM/COBOL components. While all of the support modules are designed to be used with the runtime system, only a subset of the support modules may be used with either the compiler and/or the Indexed File Recovery utility. Table 42 (for UNIX) and Table 43 (for Windows) list the support modules (and the dynamic library filenames) that can be used with each of the RM/COBOL components.

**Table 42: Optional Support Modules Used by RM/COBOL Components on UNIX**

| Support Module | Filename | Runtime | Compiler | Recover1 |
|---|---|---|---|---|
| Terminal Interface | librmterm.so | ✓ | | ✓ |
| Automatic Configuration File | librmconfig.so | ✓ | ✓ | ✓ |
| RM/InfoExpress Client | librminfox.so | ✓ | ✓ | |
| FlexGen | libflexgen.so | ✓ | | |
| Softis Network Access [1] | librmsoftis.so | ✓ | ✓ | |
| VanGui Interface Server | librmvgib.so | ✓ | | |
| RPC (Remote Procedure Calls) Server | libetsrpc.so | ✓ | | |
| Cobol-CGIX Server | libetscgix.so | ✓ | | |
| User-Written | *.so | ✓ | | |
| Message Control System | librmmcs.so | ✓ | | |

[1] *Used by Enterprise CodeBench Server and VanGui Interface Server.*

**Table 43: Optional Support Modules Used by RM/COBOL Components on Windows**

| Support Module | Filename | Runtime | Compiler | Recover1 |
|---|---|---|---|---|
| Automatic Configuration File | librmcfg.dll | ✓ | ✓ | ✓ |
| User-Written | *.dll | ✓ | | |
| Message Control System | librmmcs.dll | ✓ | | |

The support modules available for an RM/COBOL system are described in more detail beginning on page 435.

# Locating Optional Support Modules

In general, support modules are located automatically by the RM/COBOL runtime, the RM/COBOL compiler, and the Indexed File Recovery utility. Some of the support modules are located in the execution directory (that is, the directory from which the RM/COBOL component is executed) and some of the support modules are located in a subdirectory of the execution directory; this subdirectory is named **rmcobolso** on UNIX and **RmAutoLd** on Windows. Because support modules are designed to be loaded automatically rather than by using the L (Library) Option on the Runtime Command, it is not necessary to know about the other possible locations for support modules for normal "production" mode (see the following topic). However, for developers creating and testing their own support modules, there is considerable flexibility in locating support modules while in "test" mode. See In Test Mode (on page 433). Normally, this flexibility is not needed when running your application in production mode. You may elect to use any of these techniques whenever appropriate.

Filenames of optional support modules must be unique even if the modules are located in different directories. The runtime system assumes that support modules with the same name are the same and, therefore, ignores all subsequent support modules with the same name as one already loaded. You may use the V Option on

the Runtime Command, set V=DISPLAY in the RUN-OPTION configuration record, or define the RM_DYNAMIC_LIBRARY_TRACE environment variable to determine the exact load location for each support module.

Each of the support modules distributed as part of RM/COBOL or available from an add-on package is discussed in detail in Support Modules Available for RM/COBOL (on page 435). For information on building user-written support modules using CodeBridge or an alternative method, see Building Your Own Support Module (on page 439). Details on building a Message Control System support module are provided in Building a Message Control System (MCS) on page 441.

## In Production Mode

Only certain, special support modules are loaded from the execution directory. These support modules include the following:

- Terminal Interface (**librmterm.so**)
- Automatic Configuration File (**librmconfig.so** or **librmcfg.dll**)
- RM/InfoExpress Client (**librminfox.so**)
- Softis Network Access, used by both the Enterprise CodeBench Server and the VanGui Interface Server (**librmsoftis.so**)
- VanGui Interface Server (**librmvgib.so**)
- RPC Server support module (**libetsrpc.so**)
- Application-specific Message Control System (**librmmcs.so** or **librmmcs.dll**)

For UNIX, only the Terminal Interface support module is required. For UNIX and Windows, other support modules may or may not be present depending upon the RM/COBOL installation options chosen or the presence of the various add-on packages.

**Note** Users should not add user-written support modules to the execution directory unless they plan to specify the L (Library) Option on the Runtime Command to load the support module.

All other support modules are automatically loaded from the **rmcobolso** subdirectory of the UNIX execution directory or the **RmAutoLd** subdirectory of the Windows execution directory. These support modules include the following:

- FlexGen (**libflexgen.so**)
- Cobol-CGIX Server (**libetscgix.so**)
- User-written support modules (***.so** on UNIX; ***.dll** on Windows)

None of these support modules is required by the RM/COBOL runtime system. Only the FlexGen support module is included on the RM/COBOL release, and it is included only for UNIX platforms. All other support modules are included in various add-on packages or must be written by the RM/COBOL developer.

**Note** Only support modules with the **.so** extension will be loaded from the **rmcobolso** subdirectory on UNIX. Only support modules with the **.dll** extension will be loaded from the **RmAutoLd** subdirectory on Windows.

## In Test Mode

When you are developing a new support module, there are some additional options available for ensuring that your new support module is not available to others until the completion of the testing process.

### *Using a Different Execution Directory*

You may create a new execution directory by copying the RM/COBOL runtime and other necessary files, including support modules, to a directory other than the "production" execution directory (normally **/usr/bin on** UNIX and **C:\Program Files\Liant\RMCOBOL** on Windows). During testing, you will need to ensure that the new directory is included in your load path before the normal execution directory and, on Windows, you must register the new location of the runtime, as described in Runtime Registration (on page 56). When you execute the **runcobol** command from this new directory, you will establish it as the execution directory. Support modules will be loaded from this execution directory or from the **rmcobolso** (on UNIX) or **RmAutoLd** (on Windows) subdirectory of this directory. Any support modules you are testing should be placed in the **rmcobolso** or **RmAutoLd** subdirectory, for UNIX or Windows, respectively. Other support modules required for your application should also be placed in this subdirectory of the execution directory. These support modules will only be available to programs using the **runcobol** command in this new execution directory.

### *Using a Different Subdirectory*

It is also possible to use a subdirectory other than the default subdirectory (**rmcobolso** on UNIX and **RmAutoLd** on Windows) of the execution directory by defining the RM_LIBRARY_SUBDIR environment variable with the value of the test subdirectory. For example, specifying a value of **rmlibtest** for the environment variable will allow you to put your test support module in subdirectory **rmlibtest** of the normal execution directory. The environment variable, RM_LIBRARY_SUBDIR, does not specify a complete pathname, but rather a "synonym" (replacement) for the name of the subdirectory and the named subdirectory must exist within the execution directory.

Other support modules required for your application should also be placed in the specified subdirectory of the execution directory. It is not possible to use the environment variable to cause support modules to be loaded from a directory other than a directory subordinate to the execution directory, but the directory need not be directly subordinate (for example, the environment variable value may be **rmcobolso/test** or **RmAutoLd\test**). Your test support module will only be available to users with the environment variable defined with the correct value. When RM_LIBRARY_SUBDIR specifies a value, only the indicated subdirectory is automatically searched; the default subdirectory is not searched in this case.

## *Using the L Option*

The methods previously described for isolating support modules during testing both involved altering the automatic loading mechanism for support modules. It is also possible to use the L (Library) Option on the Runtime Command to specify a test support module. The L Option may specify a relative filename (starting with either "**.**"or "**..**" to indicate the current working directory or its parent directory) or a complete filename (one or more directory separators present), or a "simple" filename (one which is neither relative nor complete). The following sections explain these procedures on both UNIX and Windows.

**On UNIX:**

- If a relative or complete filename is specified on the L Runtime Command Option (for example, **L=./libusr.so** or **L= /home/user/libusr.so**), the runtime system looks for the support module only in the location specified.
- If a simple filename is specified on the L Option (for example, **L=libusr.so**), the runtime system looks for the support module first in the execution directory, then in the current working directory, then the runtime allows the UNIX dynamic-load open library call (dlopen on many UNIX systems) to load the support module, and, finally, the runtime system looks for the support module in any of the directories specified by the RUNPATH environment variable.
- The behavior of the dynamic-load open library call may—on some UNIX systems—be modified by defining a UNIX-specific environment variable to indicate the load location that the dynamic-load open library call should use. On some platforms, this environment variable may be named LD_LIBRARY_PATH. For additional information about this environment variable, you should consult the man pages for dlopen (or other dynamic-load open library call). Given the wide range of choices for locating your support module while testing and in production mode, you should not need to use this capability.
- It is not necessary to specify the **.so** extension for the support module when using the L Option. For example, if the actual support module filename is **libusr.so**, the L Option may specify the filename without the **.so** extension and the runtime system will still be able to load the support module. However, when the **.so** extension is omitted, care must be taken that there is not a COBOL program library file of the same name (ignoring the **.cob** or other configured object extension from the EXTENSION-NAMES configuration record) in the search sequence since COBOL program library files take precedence in this search. For further information about the search used by the L Option, see Subprogram Loading (on page 214).

**On Windows:**

- If a relative or complete filename is specified on the L Runtime Command Option (for example **L=.\libusr.dll** or **L=C:\rmtest\libusr.dll**), the runtime system looks for the support module only in the location specified.
- If a simple filename is specified on the L Option (for example, **L=libusr.dll**), the runtime system looks for the support module first in the execution directory, then in the directory of the runtime library module, **rmlibrun.dll**, then the runtime uses the Windows LoadLibrary function to search for the support module, and, finally, the runtime system looks for the support module in any of the directories specified by the RUNPATH environment variable.

**Note** The Windows LoadLibrary function searches in the following order: the executable directory, the current directory, the 32-bit Windows system directory (typically, **C:\Windows\System32**—this step occurs only on Windows NT-class operating systems), the Windows system directory (typically, **C:\Windows\System**), the Windows directory (typically, **C:\Windows**), and the directories specified by the PATH environment variable.

- It is not necessary to specify the **.dll** extension for the support module when using the L Option. For example, if the actual support module filename is **libusr.dll**, the L Option may specify the filename without the **.dll** extension and the runtime system will still be able to load the support module. However, in this case, care must be taken that there is not a COBOL program library file of the same name (ignoring the **.cob** or other configured object extension from the EXTENSION-NAMES configuration record) in the search sequence since COBOL program library files take precedence in this search. For further information about the search used for the L Option, see Subprogram Loading (on page 214).

# Support Modules Available for RM/COBOL

The following support modules are available with the RM/COBOL system you purchased or are available from add-on packages obtained from Liant Software.

## Terminal Interface Support Modules on UNIX

The RM/COBOL system can support either the terminfo or the termcap terminal interface. The terminal interface for your system is selected at installation time. If you wish to change your terminal interface, you need only run the installation command again and respond appropriately to the prompts. Additional information about the terminfo and termcap terminal interfaces may be found in Terminal Interfaces (on page 33).

Both the terminfo terminal support module, **librmti.so**, and the termcap terminal support module, **librmtc.so**, are present on the RM/COBOL distribution media and in the installation directory (normally **/usr/rmcobol**). At installation time, one of these is chosen to be the Terminal Interface support module and is first copied to **librmterm.so** in the installation directory, and then copied to the execution directory specified by the user. The runtime system and the recovery utility look only for **librmterm.so**. It is not possible to execute the runtime system from the installation directory without first installing RM/COBOL. The **librmti.so** and **librmtc.so** files should not be placed in the execution directory since the runtime and recovery utility will not use them.

While support modules are generally optional, the **librmterm.so** file must be present for the RM/COBOL runtime system or recovery utility to run. The only optional aspect of the Terminal Interface support modules is the choice of the terminfo interface or the termcap interface.

**Note** On Windows, the RM/COBOL runtime system supports only the Windows graphical user interface (GUI) as a terminal interface. This is supported by **rmguife.dll**, but this file is considered part of the RM/COBOL system for Windows rather than a support module.

## Automatic Configuration File Support Module

The RM/COBOL runtime, compiler, and recovery utility all allow a configuration file to be automatically located (that is, without the need to specify the configuration file on the command line). UNIX versions of RM/COBOL prior to 7.1 allowed a configuration file to be linked in the runtime, compiler, or recovery utility by modifying source file **oscnfg.c** and using a Makefile generated by the **customiz** script. Additional information about how to use automatic configuration files may be found in Automatic Configuration File (on page 282).

In order to be able to use automatic configuration files for the runtime, the compiler, or the recovery utility, you need to install the Automatic Configuration File support module during the RM/COBOL installation procedure. When automatic configuration support is installed on UNIX, the **librmconfig.so** file will be copied to the execution directory. When automatic configuration support is installed on Windows, the **librmcfg.dll** file will be copied to the execution directory. The automatic configuration support module does not contain the configuration file; it only enables automatic configuration file support by adding the code that searches for and, if found, reads the appropriately named configuration file during component initialization. If the support module is not present, the RM/COBOL component will not look for the automatic configuration file. If the support module is present, but does not find an appropriately named configuration file during component initialization, no automatic configuration occurs.

If you later decide that you do not want the Automatic Configuration File support module, you may remove the automatic configuration support module from the execution directory. On UNIX, the RM/COBOL **rmuninstall** command, described in System Removal for UNIX (on page 23), can be used in "selective (prompted) mode" to remove the automatic configuration support module. On Windows, the automatic configuration support module, **librmcfg.dll**, may be deleted or moved out of the execution directory.

## RM/InfoExpress Client Support Module on UNIX

The RM/COBOL system supports optimized access to remote RM/COBOL files on various local area networks (LANs) and wide area networks (WANs) via the RM/InfoExpress Server product. RM/InfoExpress allows significantly faster access to sequential, relative, and indexed files than conventional network access methods allow.

For the RM/COBOL compiler and runtime system to have access to remote files using the RM/InfoExpress Server, you need to install the RM/InfoExpress Client support module during the RM/COBOL installation procedure. The **librminfox.so** file will be copied to the execution directory.

If you later decide that you do not want the RM/InfoExpress Client support module, the **rmuninstall** command, as described in System Removal for UNIX (on page 23), can be used in "selective (prompted) mode" to remove the support module.

## FlexGen Support Module on UNIX

The FlexGen support module may be installed to add C language subprograms needed for Transoft Inc.'s FlexGen product. Prior to version 7.1 of the RM/COBOL runtime, support for the FlexGen C subprograms required a special version of the runtime. With the release of RM/COBOL version 7.1, the FlexGen support module is always present on the RM/COBOL distribution media and in the installation directory (normally **/usr/rmcobol**). It is no longer necessary to order a special FlexGen version of the runtime system.

For the RM/COBOL runtime system to have access to the FlexGen C subprograms, you need to install the FlexGen support module during the RM/COBOL installation procedure. The **libflexgen.so** file will be copied to the **rmcobolso** subdirectory of the execution directory.

If you later decide that you do not want the FlexGen support module, the **rmuninstall** command, as described in System Removal for UNIX (on page 23), can be used in "selective (prompted) mode" to remove the support module.

## Enterprise CodeBench Server Support Module on UNIX

Enterprise CodeBench, an add-on product for RM/COBOL, allows you to create, edit, compile, debug, and run RM/COBOL applications on UNIX within the Windows environment.

The Softis Network Access support module (distributed with the Enterprise CodeBench Server) may be added to the RM/COBOL compiler and runtime system to allow Enterprise CodeBench running on Windows to interact with RM/COBOL applications on UNIX. The Softis Network Access support module is not included with the RM/COBOL system. The Enterprise CodeBench Server must be purchased separately.

**Note** It may be possible to use the **makefile** (provided with the RM/COBOL release), two wrapper modules, **lnrmudll.o** and **vgibdll.o** (also provided with the release), and the objects and libraries provided with a prior version of the Enterprise CodeBench Server to link your own **librmsoftis.so**, provided you have a C compiler and linker. Executable files **rmlogin**, **rmfileio**, and **rmtermio** from a prior version of Enterprise CodeBench may be used without change.

In order to enable Enterprise CodeBench access to RM/COBOL on UNIX, you need to install the Softis Network Access support module during the Enterprise CodeBench Server installation procedure. The **librmsoftis.so** file will be copied to the execution directory. Other files needed by the Enterprise CodeBench Server, **rmlogin**, **rmfileio**, and **rmtermio**, are also copied to the execution directory.

If you later decide that you do not want the Softis Network Access support module, the RM/COBOL **rmuninstall** command, as described in System Removal for UNIX (on page 23), can be used in "selective (prompted) mode" to remove the support module.

**Note** If both the Enterprise CodeBench Server and the VanGui Interface Server are installed, the **librmsoftis.so** file is used by both. In this case, it is not possible to remove **librmsoftis.so** without removing both the Enterprise CodeBench Server and the VanGui Interface Server.

## VanGui Interface Server Support Modules on UNIX

The add-on product, VanGui Interface Builder (referred to as VanGui), is designed to allow you to use your choice of industry-standard GUI tools, such as Delphi or Visual Basic, to update the client user interface of both Windows- and UNIX-based RM/COBOL applications.

The VanGui Interface Server support modules may be added to the RM/COBOL runtime to allow the VanGui Interface Builder and/or the VanGui Interface Client running on Windows to run RM/COBOL applications on UNIX. The VanGui Interface Server support modules are not included with the RM/COBOL system. VanGui must be purchased separately.

**Note** It may be possible to use the **makefile** (provided with the RM/COBOL release), three wrapper modules, **lnrmudll.o**, **vgibdll.o**, and **vgib.o** (also provided with the release), and the objects and libraries provided with a prior version of the VanGui Interface Server to link your own **librmsoftis.so** and **librmvgib.so**, provided you have a C compiler and linker.

In order to enable VanGui access to RM/COBOL on UNIX, you need to install the VanGui Interface Server support modules during the VanGui installation procedure. The **librmsoftis.so** file and the **librmvgib.so** file will be copied to the execution directory.

If you later decide that you do not want the VanGui Server support modules, the RM/COBOL **rmuninstall** command, as described in System Removal for UNIX (on page 23), can be used in "selective (prompted) mode" to remove the support modules.

**Note** If both the Enterprise CodeBench Server and the VanGui Interface Server are installed, the **librmsoftis.so** file is used by both. In this case, it is not possible to remove **librmsoftis.so** without removing both the Enterprise CodeBench Server and the VanGui Interface Server.

## RPC Server Support Module on UNIX

The add-on product, RPC (Remote Procedure Calls), is designed to allow you to distribute your RM/COBOL application across multiple systems, with different operating systems, as easily as it can execute alone. RPC allows you to access data, devices, or centralized business logic on remote server systems.

The RPC Server support module may be added to the RM/COBOL runtime to allow a RPC Client to run RM/COBOL programs on UNIX. The RPC Server support module is not included with the RM/COBOL system. RPC must be purchased separately.

**Note** Versions of RPC (COBOL-RPC) distributed for use with versions of the RM/COBOL runtime prior to version 7.1 required that the runtime be relinked to add support for the RPC Server. This relinking is no longer necessary.

In order to enable RPC access to RM/COBOL on UNIX, you need to install the RPC Server support module. The **libetsrpc.so** file will be copied to the execution directory. The RPC Server support module must have this name and extension, and must be placed in the execution directory in order for the RM/COBOL runtime to find it.

If you later decide that you do not want the RPC Server support module, remove the **libetsrpc.so** file from the execution directory.

## Cobol-CGIX Server Support Module on UNIX

The add-on product, Cobol-CGIX, extends the World Wide Web's CGI interface and turns RM/COBOL into a web programming language. Cobol-CGIX allows you to develop web applications using an HTML editor and a few simple calls. It makes HTML work like a user interface tool designed for RM/COBOL and allows use of standard HTML design tools for creating and maintaining the presentation.

The Cobol-CGIX Server support module may be added to the RM/COBOL runtime to enable the special Cobol-CGIX functions. The Cobol-CGIX Server support module is not included with the RM/COBOL system. Cobol-CGIX must be purchased separately.

**Note** Versions of Cobol-CGIX distributed for use with versions of the RM/COBOL runtime prior to version 7.1 required that the runtime be relinked to add support for the Cobol-CGIX Server. This relinking is no longer necessary.

In order to enable Cobol-CGIX access to RM/COBOL on UNIX, you need to install the Cobol-CGIX Server support module. The **libetscgix.so** file will be copied to the **rmcobolso** subdirectory of the execution directory. The Cobol-CGIX Server support module should have this name and extension, and must be placed in the **rmcobolso** subdirectory of the execution directory in order for the RM/COBOL runtime to find it.

If you later decide that you do not want the Cobol-CGIX Server support module, remove the **libetscgix.so** file from the **rmcobolso** subdirectory of the execution directory.

# Building Your Own Support Module

A user-written optional support module may be built using CodeBridge, Liant Software Corporation's cross-language call system, or it may be built using the method described in Appendix G: *Non-COBOL Subprogram Internals for Windows* and Appendix H: *Non-COBOL Subprogram Internals for UNIX* in the *CodeBridge* manual. These appendices document the interface specification for RM/COBOL runtime calls to non-COBOL subprograms and provide information useful for developing a support module without using CodeBridge (described as the "old way"). Appendix H also provides information for UNIX developers who previously modified **sub.c** and relinked the runtime using a Makefile generated by the **customiz** script.

## User-Written Support Module

The RM/COBOL runtime allows for calls from RM/COBOL to non-COBOL subprogram libraries in the form of support modules. The CodeBridge Builder can produce a support module and is the preferred means of doing so. The alternate method, described in Appendices G and H of the *CodeBridge* manual, however, may be more convenient for developers who provided a non-COBOL subprogram library for use with previous versions of RM/COBOL. Either method, however, may be used.

It is not necessary to put all of your non-COBOL subprograms into a single support module. You may build as many (or as few) separate support modules as you think necessary. Since user-written support modules do not need to be listed on the

runtime command line (although they may be listed for testing or other reasons), you need not worry about changing shell or batch scripts. Each support module defines only those COBOL-callable functions defined in that support module using either the **RM_EntryPoints** symbol declaration or the **RM_EnumEntryPoints** entry point. Each user-written support module must include one of these two mechanisms to allow the runtime to determine which COBOL-callable functions are included in the support module (or, on Windows, include an .EDATA section). Information about all of the special entry points for support modules may be found in the "Special Entry Points for Support Modules" sections in Appendices G and H of the *CodeBridge* manual.

UNIX versions of RM/COBOL prior to 7.1 allowed non-COBOL subprogram libraries to be linked into the runtime by modifying the source file **sub.c** and using a Makefile generated by the **customiz** script. The source file **sub.c** is no longer present on the distribution media for UNIX. However, the source file **usrsub.c** and the various header files that it needs (all present on the RM/COBOL for UNIX distribution media and in the installation directory) provide a starting point for an application designer to build an application-specific support module. All of the special entry points for support modules are illustrated in the source file **usrsub.c**. As shipped with the RM/COBOL release, the source module **usrsub.c** merely produces trace messages when the COBOL-callable subprograms are called by the RM/COBOL runtime.

**Note** The special entry points, SYSTEM, DELETE, and RENAME, which were included in the C source **sub.c** on previous releases of RM/COBOL for UNIX, are not present in **usrsub.c**. These COBOL-callable functions are now part of the runtime system and are fully documented in Appendix F: *Subprogram Library* (on page 527).

Windows versions of RM/COBOL now include an enhanced **msgbox.c** example, which illustrates the special entry points for support modules.

On UNIX, the **makefile** (provided with the RM/COBOL release) provides a target to build the user-written support module, **libusr.so**. Although it is unnecessary to name your support module **libusr.so**, the name chosen must have an extension of **.so**.

In order to enable the completed support module for use, you should place the file in the **rmcobolso** subdirectory of the execution directory on UNIX or the **RmAutoLd** subdirectory of the execution directory on Windows. If you later decide that you do not want your support module, you should remove the file from the **rmcobolso** or **RmAutoLd** subdirectory of the execution directory. For additional information on how support modules are located, see Locating Optional Support Modules (on page 431).

## User-Written Support Module from Old sub.c or sub.o

The section "Creating a Support Module from a C Object (No Source)" in Appendix H of the *CodeBridge* manual describes how one might use **usrsub.c** as a "wrapper" module to surround an existing **sub.o** object from an older release of RM/COBOL in order to build a support module with the same non-COBOL subprograms as were previously available. This technique is most useful when the **sub.c** source module cannot be located.

**Note** It is not necessary to include all of the previously available non-COBOL subprograms in the **RM_EntryPoints** table. In particular, you should not include SYSTEM, DELETE, or RENAME unless you intend to provide your own

subprograms for these functions. Even though these functions are now provided in the runtime, as described in Appendix F: *Subprogram Library* (on page 527), including them in a user-written support module that is loaded at runtime will cause the user-written routines to be executed instead.

The same "wrapper" technique may be used to wrap an existing **sub.c** source so that no change need be made to the existing source in order to build a support module with the same non-COBOL subprograms as were previously available.

# Building a Message Control System (MCS)

The RM/COBOL runtime provides an open Message Control System (MCS) interface with which an application designer can design and include an MCS. A Message Control System is required for correct operation of the Communications Module statements: ACCEPT MESSAGE COUNT, DISABLE, ENABLE, PURGE, RECEIVE, and SEND. Beginning with RM/COBOL for UNIX version 7.1 and RM/COBOL for Windows version 7.5, in order to use the Communications Module statements, the application designer must produce a Message Control System support module. This section will explain how to design, build, and use a Message Control System.

## Message Control System (MCS) Support Module

UNIX versions of RM/COBOL prior to 7.1 allowed an MCS to be linked into the runtime by modifying or replacing source file **osmcs.c** and using a Makefile generated by the **customiz** script; however, starting with version 7.1, a user-written MCS support module replaces this method of supporting a Message Control System. Windows versions of RM/COBOL prior to 7.5 did not support a Message Control System; however, starting with version 7.5, a user-written MCS support module may be provided as a Windows DLL.

No Message Control System is provided with the RM/COBOL system. However, source file **usrmcs.c**, header file **rtccd.h**, and other necessary header files (all present on the RM/COBOL distribution media and in the installation directory) provide a starting point for an application designer to build an application-specific MCS. The **makefile** (provided with the RM/COBOL for UNIX release) provides a target to build the MCS support module, **librmmcs.so**. As shipped with the RM/COBOL release, source module **usrmcs.c** merely produces trace messages when the MCS entry points are called by the RM/COBOL runtime.

**Note** The Message Control System support module differs from most other support modules developed by users in that it does not necessarily contain any COBOL-callable subprograms. The only required entry points in the MCS support module are OSMCSINITIALIZE, used by the runtime to initialize the MCS, and OSMCS, used by the runtime to perform all other MCS operations. Other special support module entry points, described in Appendices G and H of the *CodeBridge* manual, may be added as desired. COBOL callable entry points may be added by defining the **RM_EnumEntryPoints** special entry point or the **RM_EntryPoints** table, although this is not typical. The MCS is typically entered from COBOL only because of execution of the communication verbs, which are ACCEPT … FROM MESSAGE COUNT, DISABLE, ENABLE, PURGE, RECEIVE, and SEND. Nevertheless, in some cases it may be desirable to extend the MCS with additional capabilities through COBOL callable functions. For example, setting MCS

configuration options might be handled in this way since the COBOL communication verbs do not provide this capability.

To enable the Message Control System support module, you should place the **librmmcs.so** file in the execution directory on UNIX or the **librmmcs.dll** file in the execution directory on Windows. The MCS support module must have the indicated name and extension, and must be placed in the execution directory in order for the RM/COBOL runtime to find it.

If you later decide that you do not want your Message Control System support module, remove the **librmmcs.so** (on UNIX) or **librmmcs.dll** (on Windows) file from the execution directory.

## Initializing the MCS

The routine OSMCSINITIALIZE is called at runtime initialization to initialize the MCS interface. If the interface is present and initialization completes properly, OSMCSINITIALIZE should return TRUE (non-zero). Otherwise, it should return FALSE (zero).

## Message Control System Data Structures

The MCS uses two data structures to communicate with the COBOL program. The first structure is the McsPointerArea, described below. The second structure is the communications descriptor map (CCD), which is described in the following section and illustrated in Figure 41 on page 444. Both data structures are defined in the **rtccd.h** header file, which is provided with RM/COBOL systems.

The McsPointerArea is described with the following C data structure:

```
typedef struct CCDPStruct
{
          CCD  *HostBlockAddress;
          BYTE *CdAddress;
          BYTE *MessageAreaAddress;
          BYTE *McsPointer[10];
} CCDPB;
```

This data structure is defined in C and contains only pointers (addresses). The size of the structure (in bytes) depends on the size of pointers for a given system. The structure is aligned so that pointers may be referenced directly.

The HostBlockAddress contains the address of the CCD.

The CdAddress contains a pointer to the record area implicitly associated with the CD.

The MessageAreaAddress contains a pointer to the message area for a SEND or RECEIVE operation, or the key (password) value for an ENABLE or DISABLE operation.

The McsPointer array is reserved for use by the MCS function.

The MCS functions have prototypes defined as follows:

```
ERRCODE OSMCSINITIALIZE(void);
```

and

```
ERRCODE OSMCS(long McsFunction, CCDPB *McsPointerArea);
```

The valid McsFunction codes are as follows:

```
1  ACCEPT MESSAGE COUNT
2  DISABLE
3  ENABLE
4  RECEIVE
5  SEND
6  PURGE
7  Terminate CCD (see below)
8  Terminate MCS (see below)
```

The parameter McsPointerArea contains a pointer to the C data structure described previously.

Upon exit, the MCS must return a completion code as defined in Table 44.

**Table 44: MCS Completion Codes**

| Code | Description |
|---|---|
| 0 | Function processed. No error. |
| 1 | RECEIVE . . . NO DATA processed. No data. |
| 2 – 1499 | Error. Terminate run unit with MCS error. The displayed error code will be the value plus 350. |

The MCS is notified when a program containing one or more CCDs is terminated due to the execution of a CANCEL statement or to run unit termination. In this case, upon entry to the MCS, McsFunction will be 7 and the McsPointerArea will contain a pointer to the CCD to be terminated. The MCS must then determine the action to be taken for the CCD solely from the CD Type (byte offset 2) field and the area reserved for the MCS. The MCS will be called once for each communications description defined in the program being terminated.

The MCS is explicitly notified of run unit termination after all CCDs have been terminated. When this happens, upon entry to the MCS, McsFunction will be 8 and the McsPointerArea pointer value will be NULL.

No error codes are anticipated or processed when the MCS is called with McsFunction 7 or 8.

## RM/COBOL Communications Descriptor (CCD)

The RM/COBOL communications descriptor (CCD) contains values derived from the CD description in the Data Division as well as from the Procedure Division statement that caused the request. (The C layout of a CCD is provided in the header file, **rtccd.h**.) Figure 41 illustrates a map of the CCD.

**Figure 41: Communications Descriptor Map (CCD)**

| BYTE OFFSET | |
|---|---|
| 0<br>1 | Reserved - do not modify |
| 2 | CD Type |
| 3 | Options |
| 4<br>5 | Destination Table Occurrences |
| 6<br>.<br>.<br>.<br>11 | Reserved - do not modify |
| 12<br>.<br>.<br>.<br>19 | Reserved for use by MCS |
| 20<br>21 | Message Area Length |
| 22 | Reserved - do not modify |
| 23 | Message Indicator |
| 24 | Advancing Flags |
| 25 | Reserved - do not modify |
| 26<br>27 | Advancing Count |
| 28<br>.<br>.<br>.<br>79 | Reserved for use by MCS |

**CD Type** has:

Bit 0 (0x01) is set if CD . . .FOR INPUT.

Bit 1 (0x02) is set if CD . . .FOR OUTPUT.

Both bits are set if CD . . .FOR I-O.

**Options** has:

Bit 0 (0x01) is set if RECEIVE . . .MESSAGE.

Bit 1 (0x02) is set if RECEIVE . . .SEGMENT.

Bit 2 (0x04) is set if RECEIVE . . .NO DATA.

Bit 3 (0x08) is set if ENABLE INPUT TERMINAL or DISABLE INPUT TERMINAL.

**Destination Table Occurrences** contains the value specified in the DESTINATION TABLE OCCURS clause, most-significant-byte first. Zero indicates that there is no DESTINATION TABLE OCCURS clause.

**Message Area Length** contains the length, as a short integer of the message area for a SEND or RECEIVE operation, and of the key (password) for an ENABLE or DISABLE operation. Zero indicates no message area (no FROM phrase in a SEND statement), or no key value (no KEY phrase in a DISABLE or ENABLE statement).

**Message Indicator** contains the binary value of the message indicator specified by the WITH phrase of the SEND statement. 0=no indicator, 1=ESI, 2=EMI, and 3=EGI. The value is incremented by 16 if the REPLACING LINE phrase was specified in the SEND statement.

**Advancing Flags** contains information about the ADVANCING phrase of the SEND statement. Bit 7 (0x80) is set if the BEFORE phrase was specified, and is cleared if the AFTER phrase was specified. Bit 6 (0x40) is set if *mnemonic-name* was specified. Bits 0-3 (0x0F) contain the channel number associated with *mnemonic-name* in this case. If bit 6 is clear, bit 3 (0x08) is set to indicate the PAGE phrase was specified, or bit 0 (0x01) is set to indicate the LINE(S) phrase was specified.

**Advancing Count** contains the binary value, stored as a short integer, of the literal or identifier specified in the LINE(S) clause of the ADVANCING LINE(S) phrase of the SEND statement.

The remaining area is set to zero by the RM/COBOL compiler, and may be used by the MCS, as required.

# Appendix E: Windows Printing

The RM/COBOL for Windows runtime system supplies a P$ subprogram library that allows access to Windows printing features. This appendix describes the required RM/COBOL calling sequence and the USING list parameters for each P$ subprogram. Note that failure to comply with the USING list requirements will halt the run unit with a STOP RUN indication at the line containing the incorrect CALL statement.

To facilitate Windows printing program development, COBOL copy files are supplied with the RM/COBOL development system. These are described in Copy Files (on page 492). Common uses for the P$ subprograms are illustrated in Example Code Fragments (on page 514).

In addition to the Windows printing subprogram library, this appendix also describes an alternative means of printing under Windows using a set of RM/COBOL-specific escape sequences (on page 525).

## P$ Subprogram Library

**Note** P$ subprogram names are case-insensitive. For readability, mixed case is used in this document.

The P$ subprograms can be logically grouped into the following categories that control:

- The standard Windows Print dialog box
- Drawing activities
- Text manipulation
- Activities common to both drawing and text manipulation
- Printer control activities

Table 45 lists the subprograms alphabetically within each of these categories.

**Table 45: RM/COBOL Windows Printing Subprogram Library**

| Windows Print Dialog Box Subprograms | Function |
|---|---|
| P$ClearDialog (on page 459) | Clears the standard Windows Print dialog box values back to their default (unset) state. |
| P$DisableDialog (on page 460) | Causes the Windows Print dialog box not to display the next time "PRINTER?" is opened. |
| P$DisplayDialog (on page 460) | Invokes the standard Windows Print dialog box. |
| P$EnableDialog (on page 461) | Causes the standard Windows Print dialog box to display automatically the next time the predefined printer device, "PRINTER?", is opened. |
| P$GetDialog (on page 461) | Retrieves values from the standard Windows Print dialog box. |
| P$SetDialog (on page 462) | Sets values for the standard Windows Print dialog box. |
| **Drawing Subprograms** | **Function** |
| P$DrawBitmap (on page 463) | Prints a bitmap file (.bmp). |
| P$DrawBox (on page 464) | Draws a box. |
| P$DrawLine (on page 464) | Draws a line. |
| P$GetPosition (on page 465) | Retrieves the ending position of the last print operation. |
| P$LineTo (on page 465) | Draws a line starting at the current position. |
| P$MoveTo (on page 466) | Repositions the line-draw pen without drawing a line. |
| P$SetBoxShade (on page 466) | Sets shading color and percentage for a box. |
| P$SetPen (on page 467) | Sets the style, width, and color of the pen tool for a box or a line. |
| P$SetPosition (on page 467) | Sets a new position for the next print operation. |

**Table 45: RM/COBOL Windows Printing Subprogram Library (*Cont.*)**

| Text Manipulation Subprograms | Function |
|---|---|
| P$ClearFont (on page 468) | Clears font description values back to their default (unset) state. |
| P$GetFont (on page 468) | Retrieves the characteristics of the current font to match the values used by the P$SetFont subprogram. |
| P$GetTextExtent (on page 470) | Retrieves the bounding rectangle size for the text passed to the function, calculated using the current font size. The returned values can be used to draw boxes around text. |
| P$GetTextMetrics (on page 470) | Retrieves the characteristics of the current font. |
| P$GetTextPosition (on page 473) | Retrieves the ending position of the last print operation adjusted to the top or bottom of the current font. |
| P$SetDefaultAlignment (on page 473) | Sets default alignment used in text positioning. |
| P$SetFont (on page 473) | Changes fonts for subsequent text print operations. |
| P$SetLineExtendMode (on page 476) | Concatenates output from two COBOL WRITE statements on the same line. |
| P$SetLineSpacing (on page 476) | Sets the number of lines per inch. |
| P$SetPitch (on page 477) | Sets normal, compressed, or expanded font pitch. |
| P$SetTabStops (on page 477) | Sets the tab stop increment. |
| P$SetTextColor (on page 478) | Sets the color for text output. |
| P$SetTextPosition (on page 478) | Sets a new position for the next print operation adjusted from the top or bottom of the current font. |
| P$TextOut (on page 479) | Allows the program to control the position of the text. This provides an alternative to using the COBOL WRITE to print text. |
| **Common Drawing and Text Manipulation Subprograms** | **Function** |
| P$SetDefaultMode (on page 480) | Sets default mode used in positioning and sizing parameters. |
| P$SetDefaultUnits (on page 480) | Sets default unit of measurement in positioning and sizing parameters. |
| P$SetLeftMargin (on page 481) | Sets left margin for subsequent printer output. |
| P$SetTopMargin (on page 481) | Sets the top margin for subsequent printer output. |

**Table 45: RM/COBOL Windows Printing Subprogram Library (*Cont.*)**

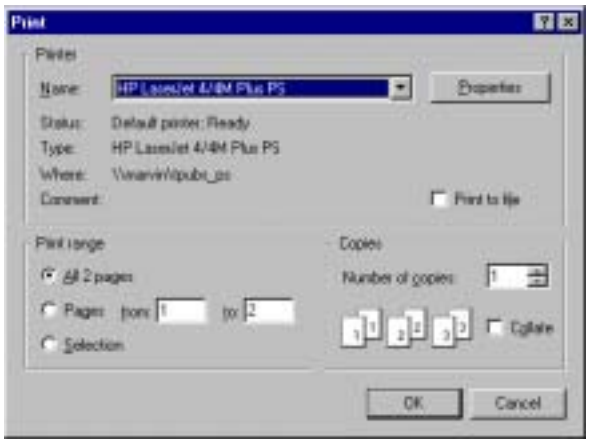
| Printer Control Subprograms | Function |
|---|---|
| P$ChangeDeviceModes (on page 482) | Changes device mode (DEVMODE) values for the standard Windows Print dialog box. Values take effect beginning with the next page. |
| P$EnableEscapeSequences (on page 483) | Enables RM/COBOL-specific escape sequences. |
| P$EnumPrinterInfo (on page 483) | Retrieves detailed information about all of the printers on a system. |
| P$GetDefineDeviceInfo (on page 484) | Retrieves the define device information as specified in the DEFINE-DEVICE configuration record of the runtime configuration file for the current P$ printer. |
| P$GetDeviceCapabilities (on page 485) | Retrieves the device capabilities of a P$ printer. |
| P$GetHandle (on page 487) | Retrieves the handle of the current P$ printer. Optionally, can be used to retrieve the true Windows printer handle. |
| P$GetPrinterInfo (on page 488) | Retrieves detailed information about a P$ printer. |
| P$NewPage (on page 490) | Forces the next printer output to a new page. |
| P$ResetPrinter (on page 490) | Resets the P$ printer. |
| P$SetDocumentName (on page 490) | Sets the name of the document as it appears in the Windows printer status window. |
| P$SetHandle (on page 491) | Changes the current P$ printer. |
| P$SetRawMode (on page 491) | Bypasses Windows printer drivers, enabling printing with escape sequences to a remote printer on a Windows NT-server. |

# Overview

Prior to version 7.0 of RM/COBOL for Windows, a COBOL Windows application had limited control of printing, as printer and font selection could only be specified using a configuration record. RM/COBOL version 7.0 for Windows runtime provides enhanced capabilities and flexibility when printing under Windows.

A new predefined printer device, “PRINTER?”, has been added to the RM/COBOL runtime, as discussed in Windows Printers (on page 306). When this device is opened, a standard Windows Print dialog box, as shown in Figure 42, is presented to the user to allow dynamic selection of the Windows printer.

**Figure 42: Standard Windows Print Dialog Box**




A broad range of COBOL-callable subprograms (P$) has been added to the runtime to allow printer control, font control, drawing of bitmaps, lines, and boxes, color control, positioning of printed objects, and other print-related functions.

These new functions can be applied to a single printer, or, if the application uses multiple printers, a printer "handle" is available to allow selection of an open printer on which subsequent P$ subprograms will operate. The printer handle can be ignored by the application if it opens only one printer at a time. The true Windows handle of the printer is available so that non-COBOL subprograms can further enhance the information on the page. This allows the use of special graphics and bar codes. This Windows handle can be ignored by the application if there is no need for non-COBOL programs to write to the printer.

C$SetDevelopmentMode (on page 567) may be used to enable expanded error information reporting (known as "development mode") for the P$ subprograms. This may be useful during program development. Development mode may also be used to bracket particular P$ calls by using both C$SetDevelopmentMode and C$ClearDevelopmentMode (on page 534).

**Note** Some of the more advanced printer functions require knowledge of Windows printing structures. Only limited documentation is given here because it is assumed that the developer who requires advanced functions has access to the appropriate Windows documentation.

**WARNING** Due to differences among printers and printer drivers, output produced using P$ subprograms can vary from printer to printer. To avoid surprises after application deployment, test your application on printers that you plan to support.

## Using Windows Printing Functions

The basic steps for using the Windows printing functions are as follows:

1. Open a printer that selects the “PRINTER?” device using the standard COBOL OPEN statement.

   This allows the end-user to choose the desired P$ printer.

2. Optionally retrieve the printer handle by calling P$GetHandle (on page 487).

   Disregard this step if the application does not open more than one printer at a time.

3. Call various P$ subprograms to control the font, orientation, color, position, and so on, of printed text or drawing objects.

   If more than one printer is opened at one time, call P$SetHandle (on page 491) to switch between them.

4. Intermix P$ subprogram calls with standard COBOL WRITE statements to the printer to produce the desired output.

5. Close the COBOL printer file.

## Returning to a "Normal" Font

In order to implement the concept of returning to a “normal” font (after using the RM/COBOL-specific escape sequences, Shift In or Shift Out, to specify expanded or compressed fonts), the RM/COBOL runtime keeps a copy of the normal font for each printer. That normal font is updated whenever P$SetFont (on page 473) or the Print Pitch or Font Height escape sequences, as described in the example Setting Text Position (on page 524), are used.

## Common P$ Subprogram Arguments

Many of the P$ subprograms use similar arguments to control positioning, size, color, and other common values. The arguments are described using the following terms: *Position*, *Size*, *Point*, *Amount*, and *Increment*, *Mode*, *Units*, *Yes/No*, and *Color*. The descriptions of these types of arguments are given below, but are omitted from the individual subprogram argument descriptions.

- ***Position.*** *XPosition, YPosition, XPoint,* and *YPoint* arguments can be any COBOL numeric data type. The default value for these arguments is the current position. An *XPosition* or *XPoint* argument must have a corresponding *YPosition* or *YPoint* argument, respectively.

  **Note 1** For any of the P$Get subprograms receiving a *Position* argument, the format of the receiving field’s PICTURE clause varies, depending on the type of *Units* (see page 453) being used. If *Units* is “Device Units”, the format of the PICTURE clause must be PIC S9(10). If *Units* is “Characters”, the format of the PICTURE clause must be PIC S9(3). If *Units* is “Inches” or “Metric”, the format of the PICTURE clause must be PIC S99v99.

  **Note 2** Position 0,0 is the upper-left corner of the printable area of the page.

- ***Alignment.*** *Alignment* arguments can be any alphabetic or alphanumeric COBOL data type. *Alignment* arguments allow the application to specify "Top" alignment or "Bottom" alignment of text positioning. The default mode is "Top" unless changed by the P$SetDefaultAlignment (on page 473) call. Only the first letter of the value is relevant, and it is case-insensitive. Possible values are contained in the copy file WINDEFS.CPY (on page 512).

- ***Size.*** *Size* arguments can be any COBOL numeric data type. *Size* arguments are used to specify the width (*SizeWidth*) and height (*SizeHeight*) of objects. Except when used with P$DrawBitmap (on page 463), *Size* arguments have default values of 1 for width and 1 for height.

  **Note** For any of the P$Get subprograms receiving a *Size* argument, the format of the receiving field's PICTURE clause varies, depending on the type of *Units* (see below) being used. If *Units* is "Device Units", the format of the PICTURE clause must be PIC 9(10). If *Units* is "Characters", the format of the PICTURE clause must be PIC 9(3). If *Units* is "Inches" or "Metric", the format of the PICTURE clause must be PIC 99v99.

- ***Amount and Increment.*** *Amount* and *Increment* arguments can be any COBOL numeric data type. These arguments specify the value of an argument used in subprogram calls.

- ***Mode.*** *Mode* arguments can be any alphabetic or alphanumeric COBOL data type. *Mode* arguments allow the application to specify "Absolute" positioning (with 0,0 being the upper-left corner of the page) or "Relative" positioning (relative to the last printed object or text). The default mode is "Absolute" unless changed by the call to P$SetDefaultMode (on page 480). Only the first letter of the value is relevant, and it is case-insensitive. Possible values are contained in the copy file WINDEFS.CPY (on page 512).

- ***Units.*** *Units* arguments can be any alphabetic or alphanumeric COBOL data type. *Units* arguments allow the application to select one of four units of measurement: "Inches", "Metric", "Characters", and "Device Units". "Inches" are expressed in inches with 2.5 meaning 2½ inches and have precision to 1/1000th of an inch. "Metric" is expressed in centimeters and has precision to 1/1000th of a centimeter. "Characters" are expressed in character cell row/column position (computed using the current font). "Device Units" are expressed in the low-level Windows device unit measurement. The default value for *Units* is "Inches" unless changed by the call to P$SetDefaultUnits (on page 480). Only the first letter of the value is relevant, and it is case-insensitive. Possible values are contained in the copy file WINDEFS.CPY (on page 512).

- ***Yes/No.*** *Yes/No* arguments can be any alphabetic or alphanumeric COBOL data type. Only the first letter of the value is relevant, and it is case-insensitive. The default value is N (No). For possible values to make the COBOL statement more readable, see the copy file WINDEFS.CPY (on page 512).

- ***Color.*** *Color* arguments may be specified by one of following methods:

  - Color-name/Percentage method. Any COBOL alphabetic or alphanumeric variable specifying the color-name. For possible values, see Table 46 and the copy file WINDEFS.CPY. A second optional argument indicates the percentage (intensity) to apply to the color value. For example, the following code fragment sets the box shading color to 30 percent red:

    ```
    CALL "P$SetBoxShade" USING ColorRed 30.
    ```

- RGB (Red, Green, Blue) triplet method. These values may be any COBOL numeric data items. Possible values for each data item are 0 through 255. See Table 46 for a list of default colors to use with RM/COBOL. Note that if you use the RGB triplet method, you must specify a value for all three colors.

**Table 46: Default Colors Used with RM/COBOL**

| String | 78-Level Value | Red, Green, Blue Values |
|---|---|---|
| ColorBlack | "Black" | 000,000,000 |
| ColorDarkBlue | "Dark Blue" | 000,000,127 |
| ColorDarkGreen | "Dark Green" | 000,127,000 |
| ColorDarkCyan | "Dark Cyan" | 000,127,127 |
| ColorDarkRed | "Dark Red" | 127,000,000 |
| ColorDarkMagenta | "Dark Magenta" | 127,000,127 |
| ColorBrown | "Brown" | 127,127,000 |
| ColorDarkGray | "Dark Gray" | 085,085,085 |
| ColorLightGray | "Light Gray" | 192,192,192 |
| ColorBlue | "Blue" | 000,000,255 |
| ColorGreen | "Green" | 000,255,000 |
| ColorCyan | "Cyan" | 000,255,255 |
| ColorRed | "Red" | 255,000,000 |
| ColorMagenta | "Magenta" | 255,000,255 |
| ColorYellow | "Yellow" | 255,255,000 |
| ColorWhite | "White" (will not print) | 255,255,255 |

## Omitting P$ Subprogram Arguments

The COBOL CALL statement in RM/COBOL now supports the OMITTED keyword to explicitly omit an argument in the USING list. For more information, see the "CALL Statement" topic in the *RM/COBOL Language Reference Manual*.

The calling COBOL program may omit arguments by passing fewer arguments than expected or passing the reserved word, OMITTED, for one or more arguments. The following example illustrates how the OMITTED keyword may be used with P$ subprograms.

```
CALL "P$SetPosition" USING 5, 3, OMITTED, "Inches"
```

This example causes the next drawn object to appear at the specified coordinates. The *Mode* argument (see page 453) has been omitted, which causes the RM/COBOL runtime to use the default value for *Mode*.

# Windows Print Dialog Box Subprograms

The following subprograms control the configuration of the standard Windows Print dialog box (see Figure 42 on page 451):

- P$ClearDialog (on page 459)
- P$DisplayDialog (on page 460)
- P$EnableDialog (on page 461)
- P$GetDialog (on page 461)
- P$SetDialog (on page 462)

**Note 1** P$GetDialog and P$SetDialog use the printer dialog/device mode parameters listed in Table 47. In this table, the fields listed in the "Parameter Name" column represent the string that must be passed to the P$ subprograms that use the *ParameterName/Value* pairs calling sequence, which allows parameters to be set or changed individually. Possible values are provided in the 78-level entries in the copy file PRINTDLG.CPY (on page 498).

The description in the "PICTURE Clause" column indicates whether the field is numeric or alphanumeric and its required number of digits or characters. When setting values using P$SetDialog with the *ParameterName/Value* pairs calling sequence, there is no minimum requirement on the number of digits or characters. When retrieving values using P$GetDialog with the *ParameterName/Value* pairs calling sequence, the PICTURE must specify at least the number of digits or characters shown.

**Note 2** Some user selections in the Windows Print dialog box must be acted on by the COBOL application, while the device driver will handle others automatically. Still other user selections will vary from driver to driver. For more information, see Printing Multiple Copies (on page 458) and Printing Partial Reports (on page 459).

**Table 47: Printer Dialog/Device Mode Parameters**

| Parameter Name | PICTURE Clause | Description |
|---|---|---|
| Return | PIC X | Standard Windows Print dialog box status: Y = OK and N = An error occurred (see Extended Error below). |
| Extended Error | PIC 9(5) | Extended error code. See PD-ExtendedErrorValue in PRINTDLG.CPY (on page 498) for details. |
| All Pages Flag | PIC X | "All" option button is selected. |
| Selection Flag | PIC X | "Selection" option button is selected. See Printing Partial Reports (on page 459). |
| Page Numbers Flag | PIC X | "Pages" option button is selected. |
| No Selection Flag | PIC X | Disables the "Selection" option button. |
| No Page Numbers Flag | PIC X | Disables the "Pages" option button. |
| Collate Flag | PIC X | "Collate" check box is selected. |
| Print Setup Flag | PIC X | Displays the Print Setup dialog box rather than the Print dialog box. The Print dialog box has Print Range and Copies features that are replaced by Paper and Orientation features in the Print Setup dialog box. |
| Print to File Flag | PIC X | "Print to File" check box is selected. |
| No Warning Flag | PIC X | Prevents the warning message from being displayed when there is no default printer. |
| Use Device Mode Copies Flag | PIC X | Indicates whether your application supports multiple copies and collation. See Printing Multiple Copies (on page 458). |
| Disable Print to File Flag | PIC X | Disables the "Print to File" check box. |
| Hide Print to File Flag | PIC X | Hides the "Print to File" check box. |
| No Network Button Flag | PIC X | Hides and disables the "Network" button. |
| From Page | PIC 9(5) | First page to print. |
| To Page | PIC 9(5) | Last page to print. |
| Min Page | PIC 9(5) | Minimum value for From Page and To Page. If Min Page equals Max Page, the "Pages" option button and the starting and ending page edit controls are disabled. See Printing Partial Reports (on page 459). |
| Max Page | PIC 9(5) | Maximum value for From Page and To Page. See Printing Partial Reports (on page 459). |
| Print Dialog Copies | PIC 9(5) | Initial number of copies for the "Copies" edit control. If a value is specified for Device Name (or any of the parameters following Device Name in this table), the value specified for Device Mode Copies overrides the value specified for Print Dialog Copies. See Printing Multiple Copies (on page 458). |

**Table 47: Printer Dialog/Device Mode Parameters *(Cont.)***

| Parameter Name | PICTURE Clause | Description |
|---|---|---|
| Device Name | PIC X(80) | Name of the printer selected by the user. |
| Fields | Group | **Note** This parameter is not available when using the *ParameterName/Value* pairs calling sequence because the pairs calling sequence automatically sets the appropriate Fields bits when setting Device Mode fields. The Fields parameter is available when using the *PrinterDialogDescription* group data item calling sequence. The developer is responsible for setting the appropriate Fields bits. |
| Orientation | PIC 9(5) | Portrait versus landscape. |
| Paper Size | PIC 9(5) | The size of the paper. |
| Paper Length | PIC 9(5) | Length of the paper (overrides Paper Size). |
| Paper Width | PIC 9(5) | Width of the paper (overrides Paper Size). |
| Scale | PIC 9(5) | Scale factor applied while printing, expressed as a percentage. For example, 50 would print text and graphics at 50% of their specified height and width. |
| Device Mode Copies | PIC 9(5) | The number of copies to print (overrides Print Dialog Copies). See Printing Multiple Copies (on page 458). |
| Default Source | PIC 9(5) | The default paper bin. |
| Print Quality | PIC S9(5) | High, medium, low, or draft. |
| Color | PIC 9(5) | Color versus monochrome. |
| Duplex | PIC 9(5) | One-sided versus two-sided printing. |
| Y Resolution | PIC 9(5) | Y-resolution of the printer specified in dots-per-inch. If this parameter is set, Print Quality specifies the X-resolution of the printer in dots-per-inch. |
| True Type Option | PIC 9(5) | TrueType® rendering options. |
| Collate | PIC X | True or False. See Printing Multiple Copies (on page 458). |
| ICM Method | PIC 9(10) | System-specific. See Microsoft documentation. |
| ICM Intent | PIC 9(10) | System-specific. See Microsoft documentation. |
| Media Type | PIC 9(10) | System-specific. See Microsoft documentation. |
| Dither Type | PIC 9(10) | System-specific. See Microsoft documentation. |

## Printing Multiple Copies

Creating multiple copies generally requires that the printer driver support printing multiple copies. If the COBOL application is not prepared to generate multiple copies of each page, the PD-UseDevModeCopiesFlag should be set to TRUE to indicate that the application is depending on the printer driver to print multiple copies. For more information, see the definitions of the copy file PRINTDLG.CPY (on page 498).

Not all printers, however, can print multiple copies. If the printer driver does not support printing multiple copies, the "Copies" edit control on the standard Windows Print dialog box (see Figure 42 on page 451) will be disabled. Similarly, if the printer driver does not support collation, the "Collate" check box will be disabled. After the printer is opened or the standard Windows Print dialog box is displayed, the application may use P$GetDialog (on page 461) to retrieve values set by the user in the Windows Print dialog box. The application developer should first check the return information provided by P$DisplayDialog (on page 460) or by P$GetDialog to determine whether the user canceled the Print dialog box. If the application sets PD-UseDevModeCopiesFlag to TRUE, P$GetDialog will return PD-Copies with a value of one and PD-CollateFlag set to FALSE.

If the application is prepared to generate multiple copies of each page, the PD-UseDevModeCopiesFlag should be set to FALSE to indicate that the application will handle multiple copies if requested by the user. Similarly, if the PD-UseDevModeCopiesFlag is set to FALSE, the application is responsible for collating multiple copies.

Regardless of how the PD-UseDevModeCopiesFlag is set, an application can determine from PD-Copies and PD-CollateFlag (obtained from P$GetDialog) how many copies to generate and whether to simulate collation by generating the complete report multiple times.

To preset the number of copies in the Windows Print dialog box to a number other than one, the application should set DM-Copies to the desired number and, if not using the *ParameterName/Value* pairs method, set DM-CopiesField to TRUE.

After the printer is opened or the Windows Print dialog box is displayed, the DM-Copies and DM-CollateValue data items (obtained from P$GetDialog) contain the copies and collate information used by the printer driver. If the PD-UseDevModeCopiesFlag is set to FALSE, DM-Copies will contain the number of copies the printer will print, and DM-CollateValue will be zero (FALSE). If the PD-UseDevModeCopiesFlag is set to TRUE and the printer driver supports multiple copies, DM-Copies will contain the number of copies requested by the user and, if the printer driver supports collation, DM-CollateValue will indicate whether the user wants collation.

Remember, fields in the DEVMODE portion of the PRINTDLG.CPY copy file (that is, those fields that begin with the DM- prefix) are meaningful only if the associated item in DM-Fields is set to TRUE. Therefore, it is necessary to set the appropriate item in DM-Fields in addition to setting such values as DM-Copies. Similarly, the application should check the appropriate item in DM-Fields before referencing the associated item after calling P$GetDialog. When using the *ParameterName/Value* pairs method, although it is not necessary to set DM-Fields before using P$SetDialog (on page 462), it is necessary to check the Validity-Flag when using P$GetDialog.

The example code fragment, Presetting the Print Dialog Box (on page 518), illustrates the proper way to set fields before calling P$SetDialog.

## Printing Partial Reports

It is possible to print less than a full report, but in order to do this, the application must do all the work. Neither the printer driver nor the RM/COBOL runtime will print partial reports. The COBOL application must generate only the pages selected by the user.

If the application does not support generating partial reports, the PD-NoPageNumbersFlag should be set to TRUE to disable the “Pages” option button and the associated edit controls on the Windows Print dialog box. See the definitions of the copy file PRINTDLG.CPY (on page 498). Similarly, the PD-NoSelectionFlag may be set to TRUE to disable the “Selection” option button.

If the application does support generating partial reports (that is, the PD-NoPageNumbersFlag is set to FALSE), the application should set PD-MinPage and PD-MaxPage to specify the minimum and maximum values, respectively, allowed for the page range specified in the “From Page” and “To Page” edit controls. If PD-MinPage and PD-MaxPage have the same value, the “Pages” option button and the starting and ending page edit controls are disabled. The application may specify the initial starting and ending pages with PD-FromPage and PD-ToPage. These values must be within the range of PD-MinPage and PD-MaxPage. If the user selects the “Pages” option button, then the PD-PageNumberFlag will be set to TRUE when the application calls P$GetDialog to determine what the user selected. Remember to first check the return information provided by P$DisplayDialog (on page 460) or by P$GetDialog (on page 461) to determine whether the user canceled the Print dialog box. If the user selected a range of pages to print, the range will be returned by P$GetDialog in the PD-FromPage and PD-ToPage fields. It is then the application’s responsibility to generate only those pages.

## P$ClearDialog

P$ClearDialog is used to clear the standard Windows Print dialog box values back to their default (unset) state.

For an example that includes P$ClearDialog, see Presetting the Print Dialog Box (on page 518).

### Calling Sequence

```
CALL "P$ClearDialog"
```

## P$DisableDialog

P$DisableDialog is used to control the automatic invoking of the standard Windows Print dialog box when opening a "PRINTER?" device. Normally, the Print dialog box is presented the first time a dynamic printer is opened. Calling P$DisableDialog causes the Print dialog box not to be displayed the next time a "PRINTER?" device is opened. This feature can be quite useful when P$SetDialog (on page 462) has been called to preset the desired printer, obtained from P$EnumPrinterInfo (on page 483) or by other methods, and the application does not want the Print dialog box to appear.

### Calling Sequence

```
CALL "P$DisableDialog"
```

## P$DisplayDialog

P$DisplayDialog is used to invoke the standard Windows Print dialog box. After choosing a printer with this dialog box, the next open of a "PRINTER?" device will use the selected printer (and no dialog box will appear at that time). See also the discussion of "PRINTER?" in Windows Printers (on page 306).

For an example that includes P$DisplayDialog, see Checking the Return Code After Displaying the Print Dialog Box (on page 519).

### Calling Sequence

```
CALL "P$DisplayDialog" GIVING DialogReturn
```

*DialogReturn* is a COBOL data item that receives the results of the Windows Print dialog box return value. This allows the application to determine whether the dialog was dismissed using the OK or Cancel button, or by an error condition. Possible values are contained in the 78-level entries in the copy file PRINTDLG.CPY (on page 498) under the heading, "P$DisplayDialog Return Values."

## P$EnableDialog

P$EnableDialog is used to control the automatic invoking of the standard Windows Print dialog box when opening a “PRINTER?” device. Normally, the Print dialog box is presented only the first time a dynamic printer is opened. Calling P$EnableDialog causes the Print dialog box to appear the next time a “PRINTER?” device is opened. See also the discussion of “PRINTER?” in Windows Printers (on page 306).

For an example that includes P$EnableDialog, see Opening and Writing to Separate Printers (on page 521).

### Calling Sequence

```
CALL "P$EnabledDialog"
```

## P$GetDialog

P$GetDialog retrieves fields from the standard Windows Print dialog box.

The two calling sequences for this subprogram allow parameters to be retrieved either individually or collectively. You can retrieve parameters individually by using *ParameterName/Value/Validity-Flag* triplets. Retrieve parameters collectively using the *PrinterDialogDescription* group data item.

### Calling Sequences

```
CALL "P$GetDialog" USING PrinterDialogDescription

CALL "P$GetDialog" USING ParameterName-1 Value-1
      Validity-Flag-1 [ParameterName-n Value-n
      Validity-Flag-n...]
```

*PrinterDialogDescription* is a group data item, as defined in the copy file PRINTDLG.CPY (on page 498).

*ParameterName* is an alphanumeric data item that contains the name of the printer dialog or device mode parameter to get (see Table 47 on page 456).

*Value* is a COBOL data item used to get the value of the parameter named by *ParameterName*. Possible values are contained in the 78-level entries in the copy file, PRINTDLG.CPY.

*Validity-Flag* is a returned numeric data item that indicates the validity of the returned parameter value. A non-zero value indicates a valid parameter.

**Note** For a more complete discussion of printer dialog and device mode parameters, see the Microsoft Windows documentation for the PRINTDLG and DEVMODE structures. For website information, see page 498.

## P$SetDialog

P$SetDialog initializes fields for the standard Windows Print dialog box.

The two calling sequences for this subprogram allow parameters to be set either individually or collectively. Setting parameters individually using *ParameterName/Value* pairs allows multiple calls to the subprogram to accumulate values for the standard Windows Print dialog box. Setting parameters collectively, using the *PrinterDialogDescription* group data item, sets all values, after which the *ParameterName/Value* method can be used to modify values.

For examples that include P$SetDialog, see Presetting the Print Dialog Box (on page 518) and Checking the Return Code After Displaying the Print Dialog Box (on page 519).

### Calling Sequences

```
CALL "P$SetDialog" USING PrinterDialogDescription

CALL "P$SetDialog" USING ParameterName-1 Value-1
     [ParameterName-n Value-n...]
```

*PrinterDialogDescription* is a group data item, as defined in the copy file PRINTDLG.CPY (on page 498).

*ParameterName* is an alphanumeric data item that contains the name of the printer dialog or device mode parameter to set (see Table 47 on page 456).

*Value* is a COBOL data item used to set the value of the parameter named by *ParameterName*. Possible values are contained in the 78-level entries in the copy file, PRINTDLG.CPY.

**Note 1** If the DM-DeviceName of the group call or the “Device Name” *ParameterName* of the pairs call is more than 31 characters, the pre-selected printer in the Windows Print dialog box will not reflect the printer name set in the P$SetDialog call. If the Print dialog box is not displayed (for example, by using the P$DisableDialog call), the correct printer will be displayed. If the printer name is more than 80 characters, then the second form of the function calling sequence must be used.

**Note 2** For a more complete discussion of printer dialog and device mode parameters, see the Microsoft Windows documentation for the PRINTDLG and DEVMODE structures. For website information, see page 498.

# Drawing Subprograms

The following subprograms control the drawing of objects on a printed page:

- P$DrawBitmap (see below)
- P$DrawBox (on page 464)
- P$DrawLine (on page 464)
- P$GetPosition (on page 465)
- P$LineTo (on page 465)
- P$MoveTo (on page 466)
- P$SetBoxShade (on page 466)
- P$SetPen (on page 467)
- P$SetPosition (on page 467)

## P$DrawBitmap

P$DrawBitmap is used to print a bitmap file from an existing Windows bitmap file.

For an example that includes this subprogram, see Printing a Bitmap (on page 520).

### Calling Sequence

```
CALL "P$DrawBitmap" USING Filename [XPosition
      YPosition] [PositionMode] [PositionUnits]
      [SizeWidth SizeHeight] [SizeUnits]
      [GIVING ReturnCode]
```

*Filename* can be any alphabetic or alphanumeric COBOL data type. It specifies the pathname of the Windows bitmap file to draw.

*XPosition/YPosition*. See *Position* in Common P$ Subprogram Arguments (on page 452).

*PositionMode*. See *Mode* in Common P$ Subprogram Arguments.

*PositionUnits/SizeUnits*. See *Units* in Common P$ Subprogram Arguments.

*SizeWidth* and *SizeHeight* are the values used to determine the size of the bitmap. If you specify a value for *SizeWidth*, you must also specify a value for *SizeHeight*. A value of 0,0 indicates that the new bitmap should be the same size as the original bitmap. A value of 0 in one position but not in the other indicates that the new bitmap should be scaled to match the proportions of the original bitmap. For example, a non-zero *SizeWidth* value specifies the width and requests that *SizeHeight* be determined by the original width/height ratio.

*ReturnCode* is a COBOL numeric data item that indicates the success or failure of the P$DrawBitmap call. A value of “0” indicates a failure, “1” indicates success. Failure will be returned if the bitmap file cannot be found or is not a valid bitmap.

## P$DrawBox

P$DrawBox is used to draw a box. For examples that include P$DrawBox, see Drawing Shaded Boxes with Colors (on page 516) and Drawing a Box Around Text (on page 516).

### Calling Sequence

```
CALL "P$DrawBox" USING [XPosition YPosition]
      [PositionMode] [PositionUnits] [SizeWidth
      SizeHeight] [SizeUnits] [ShadeYesNo]
```

*XPosition/YPosition*. See *Position* in Common P$ Subprogram Arguments (on page 452).

*PositionMode*. See *Mode* in Common P$ Subprogram Arguments.

*PositionUnits/SizeUnits*. See *Units* in Common P$ Subprogram Arguments.

*SizeWidth/SizeHeight*. See *Size* in Common P$ Subprogram Arguments.

*ShadeYesNo* is an alphanumeric data item that specifies a yes/no value (see *Yes/No* in Common P$ Subprogram Arguments). It specifies whether to shade the interior of the box using the current box shading color, set with P$SetBoxShade (on page 466).

## P$DrawLine

P$DrawLine is used to draw a line.

For an example that includes P$DrawLine, see Drawing a Ruler (on page 517).

### Calling Sequence

```
CALL "P$DrawLine" USING [X1Point Y1Point]
      [Point1Mode] [Point1Units] [X2Point
      Y2Point] [Point2Mode] [Point2Units]
```

*XnPoint/YnPoint*. See *Position* in Common P$ Subprogram Arguments (on page 452).

*PointnMode*. See *Mode* in Common P$ Subprogram Arguments.

*PointnUnits*. See *Units* in Common P$ Subprogram Arguments.

## P$GetPosition

P$GetPosition is used to retrieve the ending position of the last print operation. For instance, P$GetPosition could be used to return to a previous position after printing a bitmap elsewhere on the page.

For an example that includes P$GetPosition, see Setting Text Position (on page 524).

### Calling Sequence

```
CALL "P$GetPosition" USING XPosition YPosition [Units]
```

*XPosition/YPosition*. See *Position* in Common P$ Subprogram Arguments (on page 452).

*Units*. See *Units* in Common P$ Subprogram Arguments.

## P$LineTo

P$LineTo is used to draw a line starting at the current position.

For an example that includes P$LineTo, see Drawing a Ruler (on page 517).

### Calling Sequence

```
CALL "P$LineTo" USING [XPoint YPoint] [Mode] [Units]
```

*XPoint/YPoint*. See *Position* in Common P$ Subprogram Arguments (on page 452).

*Mode*. See *Mode* in Common P$ Subprogram Arguments.

*Units*. See *Units* in Common P$ Subprogram Arguments.

## P$MoveTo

P$MoveTo is used to reposition the line-draw pen without drawing a line.

For an example that includes P$MoveTo, see Drawing a Ruler (on page 517).

### Calling Sequence

```
CALL "P$MoveTo" USING [XPoint YPoint] [Mode] [Units]
```

*XPoint/YPoint*. See *Position* in Common P$ Subprogram Arguments (on page 452).

*Mode*. See *Mode* in Common P$ Subprogram Arguments.

*Units*. See *Units* in Common P$ Subprogram Arguments.

## P$SetBoxShade

P$SetBoxShade is used to set color and density of the color used in P$DrawBox (on page 464) calls.

For an example that includes P$SetBoxShade, see Drawing Shaded Boxes with Colors (on page 516).

### Calling Sequence

```
CALL "P$SetBoxShade" USING [Color]
```

*Color*. See *Color* in Common P$ Subprogram Arguments (on page 452).

## P$SetPen

P$SetPen is used to set the style, width, and color of the pen used in P$DrawBox (on page 464), P$DrawLine (on page 464), and P$LineTo (on page 465) calls.

For an example that includes P$SetPen, see Drawing Shaded Boxes with Colors (on page 516).

### Calling Sequence

```
CALL "P$SetPen" USING Style [Width] [Color]
```

*Style* can be any COBOL numeric data type. It specifies the style of the pen. Possible values are contained in the copy file WINDEFS.CPY (on page 512).

*Width* can be any COBOL numeric data type. It specifies the pen width in logical units. If *Width* is zero, the pen is a single pixel wide. The default value is 1.

**Note** If you specify a *Width* value greater than 1 for the pen styles, Dash, Dot, DashDot, or DashDotDot, Windows will force *Style* to a value of Solid.

*Color*. See *Color* in Common P$ Subprogram Arguments (on page 452).

## P$SetPosition

P$SetPosition is used to set a position for the next print operation.

For examples that include P$SetPosition, see Drawing a Box Around Text (on page 516) and Setting Text Position (on page 524).

### Calling Sequence

```
CALL "P$SetPosition" USING [XPosition YPosition]
      [Mode] [Units]
```

*XPosition/YPosition*. See *Position* in Common P$ Subprogram Arguments (on page 452).

*Mode*. See *Mode* in Common P$ Subprogram Arguments.

*Units*. See *Units* in Common P$ Subprogram Arguments.

# Text Manipulation Subprograms

The following subprograms control the manipulation of text on a printed page:

- P$ClearFont (see below)
- P$GetFont (see below)
- P$GetTextExtent (on page 470)
- P$GetTextMetrics (on page 470)
- P$GetTextPosition (on page 473)
- P$SetDefaultAlignment (on page 473)
- P$SetFont (on page 473)
- P$SetLineExtendMode (on page 476)
- P$SetPitch (on page 477)
- P$SetTabStops (on page 477)
- P$SetTextColor (on page 478)
- P$SetTextPosition (on page 478)
- P$TextOut (on page 479)

## P$ClearFont

P$ClearFont clears the font description values that were set using P$SetFont (on page 473) and returns them to their default (unset) state. This subprogram can be used to clear previous values before calling P$SetFont using the *ParameterName/Value* method to set information for a new font.

### Calling Sequence

```
CALL "P$ClearFont"
```

## P$GetFont

P$GetFont is used to retrieve the characteristics (or values) of the current font. The format of these values matches those used by P$SetFont (on page 473) instead of the format used by P$GetTextMetrics. The P$GetFont subprogram may be used only after the printer is opened.

The two calling sequences for this subprogram allow parameters to be retrieved either collectively or individually.

The P$GetFont subprogram may be used after calling the P$SetFont subprogram to determine whether the font attributes for the font chosen by Windows are acceptable. This is particularly important when multiple calls to P$SetFont are made using the *ParameterName/Value* method to change font attributes.

**WARNING** If no DEFINE-DEVICE configuration record is specified for a printer, calling P$GetFont on a Windows NT-class operating system after opening a printer but before using P$SetFont to select a font may return a Face Name value of "System", a Height of 16, and a Width of 7. Such information is not useful to the COBOL application. Attempts to use this information as values for the P$SetFont subprogram will produce undesirable results, including, possibly, text that is too small to read or an incorrect font. (This problem does not occur on a Windows 9x-class operating system. The value of fields returned refers to the current/normal

font.) To avoid this problem on a Windows NT-class operating system, use either the PATH keyword of the DEFINE-DEVICE configuration record (on page 304) to specify a font name and size or P$SetFont to select a font before calling P$GetFont. If you use P$SetFont to select the font, you should first use the INITIALIZE statement or P$ClearFont (on page 468) to set a known initial state before setting values for the new font.

### Calling Sequence

```
CALL "P$GetFont" USING LogicalFontDescription

CALL "P$GetFont" USING ParameterName-1 Value-1
      [ParameterName-n Value-n...]
```

*LogicalFontDescription* is a group data item as defined in the copy file LOGFONT.CPY (on page 494).

*ParameterName* is an alphanumeric data item that contains the name of the font parameter to get (see Table 49 on page 475).

*Value* is the COBOL data item used to receive the value of the parameter named by *ParameterName*. Possible values are contained in the 78-level entries in the copy file named LOGFONT.CPY.

**Note 1** The COBOL data types for the *Value* data items are listed in Table 45 on page 448. The numeric fields must be integer, no decimal places are allowed, and the minimum required field size is five (PIC 9(5)). For the alphabetic/alphanumeric fields, the minimum field size is one except for “Face Name”, which must be at least 31 characters.

**Note 2** For a more complete discussion of font attribute parameters, see the Microsoft Windows documentation for the LOGFONT structure. For website information, see page 494.

## P$GetTextExtent

P$GetTextExtent is used to retrieve the bounding rectangle size for text passed to the subprogram, calculated using the current font size. The returned values can be used to draw boxes around text or determine whether the text will fit within a desired region (such as the current line).

For examples that include P$GetTextExtent, see the following:

- Drawing a Box Around Text (on page 516)
- Printing Text at the Top of a Page (on page 522)
- Printing Text at the Corners of a Page (on page 522)

### Calling Sequence

```
CALL "P$GetTextExtent" USING Text, SizeWidth,
      SizeHeight [Units]
```

*Text* may be any alphabetic or alphanumeric data item or nonnumeric literal of nonzero length.

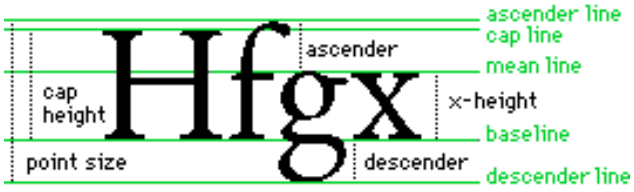
*SizeWidth/SizeHeight.* See *Size* in Common P$ Subprogram Arguments (on page 452).

*Units*. See *Units* in Common P$ Subprogram Arguments.

## P$GetTextMetrics

P$GetTextMetrics is used to retrieve the characteristics of the current font. Figure 43 illustrates some of these characteristics.

**Figure 43: Text Metrics**




For examples that include P$GetTextMetrics, see Drawing a Box Around Text (on page 516) and Setting the Point Size for a Font (on page 523).

The two calling sequences for this subprogram allow parameters to be retrieved either individually or collectively. You can retrieve parameters individually by using *ParameterName/Value* pairs. Retrieve parameters collectively using the *TextMetricDescription* group data item.

**Note** The values retrieved by P$GetTextMetrics are available after opening a P$ printer. All values returned by P$GetTextMetrics are in device units.

### Calling Sequences

```
CALL "P$GetTextMetrics" USING TextMetricDescription
      GIVING Validity-Flag

CALL "P$GetTextMetrics" USING ParameterName-1 Value-1
      [ParameterName-n Value-n...] GIVING Validity-Flag
```

*TextMetricDescription* is a group data item as defined in the copy file TXTMTRIC.CPY (on page 508).

*ParameterName* is an alphanumeric data item that contains the name of the font parameter to get (see Table 48).

*Value* is a COBOL data item used to receive the value of the parameter named by *ParameterName*. Possible values are contained in the 78-level entries in the copy file, TXTMTRIC.CPY.

*Validity-Flag* is a returned numeric data item that indicates the validity of the returned parameter values. A non-zero value indicates valid value(s). This argument is optional.

**Note** For a more complete discussion of text metric parameters, see the Microsoft Windows documentation for the TEXTMETRIC structure. For website information, see page 508.

**Table 48: Text Metric Parameters**

| Parameter Name | PICTURE Clause | Description |
|---|---|---|
| Height | PIC 9(10) | Ascender line to descender line. |
| Ascent | PIC 9(10) | Ascender line to baseline. |
| Descent | PIC 9(10) | Baseline to descender line. |
| Internal Leading | PIC 9(10) | Point size of a font minus the physical size of the font. |
| External Leading | PIC 9(10) | Extra space to be added between lines. |
| Average Character Width | PIC 9(10) | Average character width for a font. |
| Maximum Character Width | PIC 9(10) | Width of the widest character. |
| Weight | PIC 9(10) | Font weight. |
| Overhang | PIC 9(10) | Extra width added to some synthesized fonts. |
| Digitized Aspect X | PIC 9(10) | Horizontal aspect of the device for which the font was designed. |
| Digitized Aspect Y | PIC 9(10) | Vertical aspect of the device for which the font was designed. The ratio of Digitized Aspect X and Digitized Aspect Y is the aspect ratio of the device for which the font was designed. |
| First Character | PIC X | First character defined in the font. |
| Last Character | PIC X | Last character defined in the font. |
| Default Character | PIC X | Character that will be substituted for characters that are not in the font. |
| Break Character | PIC X | Character that will be used for word breaks in text justification. |
| Italic | PIC X | ‘Y’ if italic font; ‘N’ otherwise. |
| Underlined | PIC X | ‘Y’ if underlined font; ‘N’ otherwise. |
| Struck Out | PIC X | ‘Y’ if “struck-out” font; ‘N’ otherwise. |
| Pitch | PIC 9(10) | Pitch of the font. |
| Family | PIC 9(10) | Family of the font. |
| Character Set | PIC 9(10) | Character set of the font. |

## P$GetTextPosition

P$GetTextPosition is used to retrieve the ending position the last print operation adjusted to the top or bottom of the current font.

For examples that include P$GetTextPosition, see Drawing a Box Around Text (on page 516) and Setting Text Position (on page 524).

### Calling Sequence

```
CALL "P$GetTextPosition" USING XPosition YPosition
     [Alignment] [Units]
```

*XPosition/YPosition*. See *Position* in Common P$ Subprogram Arguments (on page 452).

*Alignment*. See *Alignment* in Common P$ Subprogram Arguments.

*Units*. See *Units* in Common P$ Subprogram Arguments.

**Note** P$GetPosition (on page 465) returns the ending "baseline" position. P$GetTextPosition should be used when you need the ascender line (top) or descender line (bottom) position of the current font.

## P$SetDefaultAlignment

P$SetDefaultAlignment is used to set default alignment used in text positioning. Possible values are "Top" and "Bottom". The initial value is "Top".

### Calling Sequence

```
CALL "P$SetDefaultAlignment" USING Alignment
```

*Alignment*. See *Alignment* in Common P$ Subprogram Arguments (on page 452).

## P$SetFont

P$SetFont is used to change fonts. Selecting a font with this subprogram call causes the next COBOL WRITE operation on a P$ printer to use that newly selected font. The P$SetFont subprogram may be used only after the printer is opened.

The two calling sequences for this subprogram allow parameters to be set either individually or collectively. Setting parameters individually, using *ParameterName/Value* pairs, allows multiple calls to the subprogram to accumulate values for the desired font. Setting parameters collectively, using the *LogicalFontDescription* group data item, sets all values, after which the *ParameterName/Value* method can be used to modify values.

The application should use the INITIALIZE statement to set the *LogicalFontDescription* to zeroes and spaces before starting to set values for a P$SetFont call using the entire *LogicalFontDescription* group. Failure to use the INITIALIZE statement may result in undesirable and unpredictable results.

Similarly, the application should use P$ClearFont (on page 468) to set an initial known state for the font before making a P$SetFont call using the *ParameterName/Value* method. Alternatively, if not running on a Windows NT-class operating system, the application may call P$GetFont (on page 468) to retrieve information about the current font.

**WARNING** Because each separate call to P$SetFont results in a call to a Windows API to set the new font information, applications should be coded to call P$SetFont with either the entire *LogicalFontDescription* group or to make a single call to P$SetFont using as many *ParameterName/Value* pairs as desired. Making multiple calls to P$SetFont using the pairs method may result in unpredictable results because Windows must choose an acceptable font after every P$SetFont call.

The P$GetFont subprogram may be used after calling the P$SetFont subprogram to determine whether the font attributes for the font chosen by Windows are acceptable. This is particularly important when multiple calls to P$SetFont are made using the *ParameterName/Value* method to change font attributes.

For examples that include this subprogram, see the following:

- Printing a Watermark (on page 516)
- Changing a Font While Printing (on page 520)
- Using the COBOL WRITE Statement to Print Multiple Text Outputs on the Same Line (on page 520)
- Setting the Point Size for a Font (on page 523)
- Setting Text Position (on page 524)

**Note** The values set by P$SetFont are available to P$GetTextMetrics (on page 470) after the Open operation for the printer. You can use P$ClearFont (on page 468) to clear the existing font description before calling P$SetFont using the *ParameterName/Value* method to set information for a new font.

### Calling Sequences

```
CALL "P$SetFont" USING LogicalFontDescription

CALL "P$SetFont" USING ParameterName-1 Value-1
      [ParameterName-n Value-n...]
```

*LogicalFontDescription* is a group data item as defined in the copy file LOGFONT.CPY (on page 494).

*ParameterName* is an alphanumeric data item that contains the name of the font parameter to set (see Table 49).

*Value* is the COBOL data item used to set the value of the parameter named by *ParameterName*. Possible values are contained in the 78-level entries in the copy file, LOGFONT.CPY.

**Note** For a more complete discussion of font attribute parameters, see the Microsoft Windows documentation for the LOGFONT structure. For website information, see page 494.

**Table 49: Font Parameters**

| Parameter Name | COBOL Data Type | Description |
|---|---|---|
| Height | Any signed numeric | Font height in logical units. |
| Width | Any numeric | Average font width in logical units. |
| Escapement | Any numeric | Angle, in tenths of degrees, for a string of characters (relative to the X-axis of the device). |
| Orientation | Any numeric | Angle, in tenths of degrees, for individual characters (relative to the X-axis of the device). |
| Weight | Any numeric | Font weight in the range 0 through 1000. For example, 400 is normal and 700 is bold. If this value is zero, a default weight is used. |
| Italic | Any alphabetic or alphanumeric | 'Y' if italic font; 'N' otherwise. |
| Underline | Any alphabetic or alphanumeric | 'Y' if underlined font; 'N' otherwise. |
| Strike Out | Any alphabetic or alphanumeric | 'Y' if "struck-out" font; 'N' otherwise. |
| Char Set | Any numeric | Specifies the character set. |
| Out Precision | Any numeric | Specifies the output precision for font matching. |
| Clip Precision | Any numeric | Specifies the clipping precision for font matching. |
| Quality | Any numeric | Specifies the output quality for font matching. |
| Pitch | Any numeric | Pitch of the font. |
| Family | Any numeric | Family of the font. |
| Face Name | Any alphabetic or alphanumeric | Windows typeface name. |

## P$SetLineExtendMode

P$SetLineExtendMode is used to concatenate output from two COBOL WRITE statements on the same line. This is useful for mixing fonts or styles. Specifically, by using the COBOL WRITE statement, then calling P$SetLineExtendMode, and using another WRITE statement with the ADVANCING phrase specifying 0 lines, the output from the second WRITE statement will appear on the same line as the output from the first WRITE statement.

For an example that includes P$SetLineExtendMode, see Using the COBOL WRITE Statement to Print Multiple Text Outputs on the Same Line (on page 520).

### Calling Sequence

```
CALL "P$SetLineExtendMode" USING SpaceAmount [Units]
```

*SpaceAmount* can be any COBOL numeric data type. It specifies the amount of space to leave between the two sets of output. The default value is 0.

*Units*. See *Units* in Common P$ Subprogram Arguments (on page 452).

## P$SetLineSpacing

P$SetLine Spacing is used to set the number of lines per inch.

For an example that includes P$SetLineSpacing, see Changing Orientation, Pitch, and Line Spacing (on page 521).

### Calling Sequence

```
CALL "P$SetLineSpacing" USING Value
```

*Value* can be any numeric COBOL data type. It specifies the number of lines per inch.

## P$SetPitch

P$SetPitch is used to set normal, compressed, or expanded font pitch.

For an example that includes P$SetPitch, see Changing Orientation, Pitch, and Line Spacing (on page 521).

### Calling Sequence

```
CALL "P$SetPitch" USING Type [Factor]
```

*Type* can be any alphabetic or alphanumeric COBOL data type. It specifies "Normal", "Expanded", or "Compressed" font pitch. Only the first letter of the value is relevant, and it is case-insensitive. Possible values are contained in the 78-level entries in the copy file WINDEFS.CPY (on page 512).

*Factor* can be any numeric COBOL data type. It specifies the compression and/or expansion ratio to apply to the current font pitch. For a *Type* value of "Compressed", the default compression ratio is 1.65, which specifies 1.65 times as many characters per inch. For a *Type* value of "Expanded", the default expansion ratio is 2.00, which specifies double-size characters (that is, half as many characters per inch). For a *Type* value of "Normal", the *Factor* argument is not allowed (that is, "Normal" is the normal pitch of the current font).

## P$SetTabStops

P$SetTabStops sets the increment used for computing the next tab stop location. Tabs are sent using the Horizontal Tab escape sequence (see Table 53 on page 526).

### Calling Sequence

```
CALL "P$SetTabStops" USING Increment [Units]
```

*Increment*. See *Increment* in Common P$ Subprogram Arguments (on page 452).

*Units*. See *Units* in Common P$ Subprogram Arguments.

## P$SetTextColor

P$SetTextColor is used to set the color of text for subsequent P$TextOut (on page 479) and COBOL WRITE statements.

For an example that includes P$SetTextColor, see Printing a Watermark (on page 516).

### Calling Sequence

```
CALL "P$SetTextColor" USING Color
```

*Color*. See *Color* in Common P$ Subprogram Arguments (on page 452).

## P$SetTextPosition

P$SetTextPosition is used to set a new position for the next print operation adjusted from the top or bottom of the current font.

For examples that include P$SetTextPosition, see the following:

- Printing Text at the Top of a Page (on page 522)
- Printing Text at the Corners of a Page (on page 522)
- Setting Text Position (on page 524)

### Calling Sequence

```
CALL "P$SetTextPosition" USING XPosition YPosition
     [Alignment] [Mode] [Units]
```

*XPosition/YPosition*. See *Position* in Common P$ Subprogram Arguments (on page 452).

*Alignment*. See *Alignment* in Common P$ Subprogram Arguments.

*Mode*. See *Mode* in Common P$ Subprogram Arguments.

*Units*. See *Units* in Common P$ Subprogram Arguments.

**Note 1** P$SetPosition (on page 467) should be used to set the "baseline" for the next text print operation.

**Note 2** To print a line of text in the top left corner of a page, without cutting off the top of the characters, call P$SetTextPosition specifying a value of "Top" for *Alignment*. To print a line of text in the bottom left corner of a page without cutting off the bottom of the characters, call P$SetTextPosition specifying a value of "Bottom" for *Alignment*.

# P$TextOut

P$TextOut is an alternative to using the COBOL WRITE statement to print text. It allows the program to control the position of the text. The COBOL WRITE statement has no positioning capabilities.

For examples that include P$TextOut, see the following:

- Drawing a Box Around Text (on page 516)
- Changing a Font While Printing (on page 520)
- Changing Orientation, Pitch, and Line Spacing (on page 521)
- Opening and Writing to Separate Printers (on page 521)
- Printing Text at the Top of a Page (on page 522)
- Printing Text at the Corners of a Page (on page 522)
- Setting Text Position (on page 524)

## Calling Sequence

```
CALL "P$TextOut" USING Text [XPosition YPosition]
      [Mode] [Units] [BoxYesNo] [ShadeYesNo]
```

*Text* can be any COBOL alphanumeric data item. It specifies the text to be printed.

*XPosition/YPosition*. See *Position* in Common P$ Subprogram Arguments (on page 452).

*Mode*. See *Mode* in Common P$ Subprogram Arguments.

*Units*. See *Units* in Common P$ Subprogram Arguments.

*BoxYesNo* can be any COBOL alphanumeric data item that specifies a yes/no value (see *Yes/No* in Common P$ Subprogram Arguments). It specifies whether to draw a box around the text.

*ShadeYesNo* can be any COBOL alphanumeric data item that specifies a yes/no value (see *Yes/No* in Common P$ Subprogram Arguments). It specifies whether to shade the interior of the box using the current box shading color, which is set with P$SetBoxShade (on page 466).

# Common Drawing and Text Manipulation Subprograms

The following subprograms are common to both drawing and text manipulation activities:

- P$SetDefaultMode (see below)
- P$SetDefaultUnits (see below)
- P$SetLeftMargin (on page 481)
- P$SetTopMargin (on page 481)

## P$SetDefaultMode

P$SetDefaultMode is used to control the default mode used in positioning and sizing parameters. Possible values are “Relative” and “Absolute”. The initial value is “Absolute”.

### Calling Sequence

```
CALL "P$SetDefaultMode" USING Mode
```

*Mode*. See *Mode* in Common P$ Subprogram Arguments (on page 452).

## P$SetDefaultUnits

P$SetDefaultUnits is used to control the default unit of measurement in position and sizing parameters. Possible values are “Inches”, “Metric”, “Characters”, and “Device Units”. The initial value is “Inches”.

For an example that includes P$SetDefaultUnits, see Drawing a Ruler (on page 517).

### Calling Sequence

```
CALL "P$SetDefaultUnits" USING Units
```

*Units*. See *Units* in Common P$ Subprogram Arguments (on page 452).

## P$SetLeftMargin

P$SetLeftMargin is used to set a left margin (offset from left side of paper) for subsequent COBOL WRITE statements. This margin will be cleared to zero at the next page boundary. This subprogram is useful for generating columns of text.

### Calling Sequence

```
CALL "P$SetLeftMargin" USING SizeWidth [Units]
```

*SizeWidth*. See *Size* in Common P$ Subprogram Arguments (on page 452).

*Units*. See *Units* in Common P$ Subprogram Arguments.

## P$SetTopMargin

P$SetTopMargin is used to set a top margin (offset from top edge of paper) for subsequent pages.

### Calling Sequence

```
CALL "P$SetTopMargin" USING SizeHeight [Units]
```

*SizeHeight*. See *Size* in Common P$ Subprogram Arguments (on page 452).

*Units*. See *Units* in Common P$ Subprogram Arguments.

# Printer Control Subprograms

The following subprograms are used to control various properties of a P$ printer:

## P$ChangeDeviceModes

P$ChangeDeviceModes changes the device mode (DEVMODE) values for the standard Windows Print dialog box. The new values take effect beginning with the next page. This subprogram is used for such tasks as changing the paper source or the orientation of paper.

The two calling sequences for this subprogram allow parameters to be set either individually or collectively. Setting parameters individually using *ParameterName/Value* pairs allows multiple calls to the subprogram to accumulate values for the standard Windows Print dialog box, illustrated in Figure 42 on page 451. Setting parameters collectively, using the *PrinterDialogDescription* group data item, sets all values, after which the *ParameterName/Value* method can be used to modify values.

**Note** Only DEVMODE fields of the *PrinterDialogDescription* can be changed by this subprogram. These fields are defined in the PRINTDLG.CPY copy file and have a DM- prefix.

For an example that includes P$ChangeDeviceModes, see Changing Orientation, Pitch, and Line Spacing (on page 521).

### Calling Sequences

```
CALL "P$ChangeDeviceModes" USING PrinterDialogDescription

CALL "P$ChangeDeviceModes" USING ParameterName-1 Value-1
      [ParameterName-n Value-n...]
```

*PrinterDialogDescription* is a group data item, as defined in the copy file PRINTDLG.CPY (on page 498).

*ParameterName* is an alphanumeric data item that contains the name of the device mode parameter to set (see Table 47 on page 456 starting with the Device Name parameter).

*Value* is the COBOL data item used to set the value of the parameter named by *ParameterName*. Possible values are contained in the 78-level entries in the copy file, PRINTDLG.CPY.

**Note** For a more complete discussion of device mode parameters, see the Microsoft Windows documentation for the DEVMODE structure. For website information, see page 498.

## P$EnableEscapeSequences

P$EnableEscapeSequences is used to enable RM/COBOL-specific escape sequences (on page 525) for the current P$ printer. The current P$ printer, when there is more than one P$ printer open at the same time, is determined by P$SetHandle (on page 491). The escape sequences are enabled until the printer is closed. The ESCAPE-SEQUENCES keyword (see page 305) of the DEFINE-DEVICE configuration record or the Printer Enable Escape Sequences property (on page 79) may also be used to enable these escape sequences.

### Calling Sequence

```
CALL "P$EnableEscapeSequences"
```

## P$EnumPrinterInfo

P$EnumPrinterInfo is used to retrieve detailed information about all of the printers on a system. It is not necessary to open a printer to obtain this information.

Each call to P$EnumPrinterInfo obtains information about a single printer determined by the *PrinterIndex* parameter. If you want to obtain information about all of the available printers, continue to call P$EnumPrinterInfo, advancing *PrinterIndex* until the *ReturnCode* is zero.

The two calling sequences for this subprogram allow parameters to be retrieved either individually or collectively. You can retrieve parameters individually using *ParameterName/Value* pairs. Retrieve parameters collectively using the *PrinterInfoDescription* group data item.

### Calling Sequences

```
CALL "P$EnumPrinterInfo" USING PrinterIndex
      PrinterInfoDescription
      GIVING ReturnCode

CALL "P$EnumPrinterInfo" USING PrinterIndex
      ParameterName-1 Value-1 [ParameterName-n Value-n...]
      GIVING ReturnCode
```

*PrinterIndex* is a numeric data item (1 relative) that specifies the index of the printer for which the information is requested.

*PrinterInfoDescription* is a group data item, as defined in the copy file PRINTINF.CPY (on page 506).

*ParameterName* is an alphanumeric data item that contains the name of the device capability parameter to retrieve. See Table 51 on page 489, which is associated with the description of P$GetPrinterInfo (on page 488).

*Value* is the COBOL data item used to receive the value of the parameter named by *ParameterName*. Possible values are contained in the 78-level entries in the copy file, PRINTINF.CPY.

*ReturnCode* is a COBOL numeric data item that indicates whether the current *PrinterIndex* parameter corresponds to an available printer. A value of "1" indicates that *PrinterIndex* is valid and printer information data has been returned. A value of "0" indicates that *PrinterIndex* is invalid and no data has been returned.

## P$GetDefineDeviceInfo

P$GetDefineDeviceInfo is used to retrieve the define device information as specified in the DEFINE-DEVICE configuration record (on page 304) for the current P$ printer. The current P$ printer, when there is more than one P$ printer open at the same time, is determined by P$SetHandle (on page 491).

The one calling sequence for this subprogram allows parameters to be retrieved collectively. No method is provided to retrieve the parameters individually. Retrieve parameters collectively using the *DefineDeviceDescription* group data item.

**Note** The values retrieved by P$GetDefineDeviceInfo are available only after opening a P$ printer.

### Calling Sequence

```
CALL "P$GetDefineDeviceInfo" USING DefineDeviceDescription
     GIVING Validity-Flag
```

*DefineDeviceDescription* is a group data item, as defined in the copy file DEFDEV.CPY (on page 492).

*Validity-Flag* is a returned numeric data item that indicates the validity of the returned *DefineDeviceDescription*. A non-zero value indicates that a DEFINE-DEVICE configuration record exists for the current P$ printer. This argument is optional.

## P$GetDeviceCapabilities

P$GetDeviceCapabilities is used to retrieve the device capabilities of a P$ printer. This subprogram can be used to compute the printable area of the page to be printed.

The two calling sequences for this subprogram allow parameters to be retrieved either individually or collectively. You can retrieve parameters individually using *ParameterName/Value* pairs. Retrieve parameters collectively using the *DeviceCapabilitesDescription* group data item.

**Note** The values retrieved by P$GetDeviceCapabilities are available after opening a P$ printer.

For examples that include P$GetDeviceCapabilities, see the following:

- Setting the Point Size for a Font (on page 523)
- Printing Text at the Top of a Page (on page 522)
- Printing Text at the Corners of a Page (on page 522)

### Calling Sequences

```
CALL "P$GetDeviceCapabilities" USING TextMetricDescription

CALL "P$GetDeviceCapabilities" USING ParameterName-1
      Value-1 [ParameterName-n Value-n...]
```

*DeviceCapabilitesDescription* is a group data item, as defined in the copy file DEVCAPS.CPY (on page 493).

*ParameterName* is an alphanumeric data item that contains the name of the device capability parameter (see Table 50).

*Value* is the COBOL data item used to receive the value of the parameter named by *ParameterName*. Possible values are contained in the 78-level entries in the copy file, DEVCAPS.CPY.

**Note** For a more complete discussion of device capability parameters, see the Microsoft Windows documentation for the GetDeviceCaps function. For website information, see page 493.

**Table 50: Device Capability Parameters**

| Parameter Name | PICTURE Clause | Description |
|---|---|---|
| Driver Version | PIC 9(10) | Device driver version. |
| Technology | PIC 9(10) | Device technology. |
| Horizontal Size | PIC 9(10) | Width of printable area, in millimeters. |
| Vertical Size | PIC 9(10) | Height of printable area, in millimeters. |
| Horizontal Resolution | PIC 9(10) | Width of printable area, in dots. |
| Vertical Resolution | PIC 9(10) | Height of printable area, in dots. |
| Logical Pixels X | PIC 9(10) | Horizontal dots-per-inch. |
| Logical Pixels Y | PIC 9(10) | Vertical dots-per-inch. |
| Aspect X | PIC 9(10) | Length of horizontal sides of a square. |
| Aspect Y | PIC 9(10) | Length of vertical sides of the same square. |
| Aspect XY | PIC 9(10) | Length of diagonal lines connecting opposite vertices of the same square. |
| Physical Width | PIC 9(10) | Width of the printable area, in device units. |
| Physical Height | PIC 9(10) | Height of the printable area, in device units. |
| Physical Offset X | PIC 9(10) | Width of unprintable area (left and right edges) in device units. |
| Physical Offset Y | PIC 9(10) | Height of unprintable area (top and bottom edges) in device units. |
| Scaling Factor X | PIC 9(10) | X-axis scale factor. |
| Scaling Factor Y | PIC 9(10) | Y-axis scale factor. |

# P$GetHandle

P$GetHandle is used to retrieve the handle of the current P$ printer. This allows the developer to use P$SetHandle (on page 491) to change the printer for subsequent P$ print operations. These calls are not needed unless multiple “PRINTER?” devices are open at the same time. See also the discussion of “PRINTER?” in Windows Printers (on page 306).

Optionally, P$GetHandle can be used to retrieve the true Windows printer handle. The Windows handle can be used by a non-COBOL subprogram to add information (such as special graphics or a bar code) to a page printed on P$ printer. If you do not plan to use a non-COBOL program to enhance the printed output, the Windows handle is not required.

For an example that includes P$GetHandle, see Opening and Writing to Separate Printers (on page 521).

## Calling Sequence

```
CALL "P$GetHandle" USING Handle [Win-Handle]
```

*Handle* is a PICTURE 9(2) numeric data item. It specifies the variable to receive the handle of the current P$ printer.

*Win-Handle* is a PICTURE 9(10) numeric data item. This argument is optional. If present, this argument specifies the variable to receive the true Windows handle of the current P$ printer. The Windows printer handle can be passed to a developer-written non-COBOL subprogram to allow the program to perform GDI (Graphics Device Interface) calls to the printer. The Windows handle must be not be passed to P$SetHandle.

## P$GetPrinterInfo

P$GetPrinterInfo is used to retrieve detailed information about a P$ printer.

The two calling sequences for this subprogram allow parameters to be retrieved either individually or collectively. You can retrieve parameters individually using *ParameterName/Value* pairs. Retrieve parameters collectively using the *PrinterInfoDescription* group data item.

### Calling Sequences

```
CALL "P$GetPrinterInfo" USING PrinterInfoDescription

CALL "P$GetPrinterInfo" USING ParameterName-1
      Value-1 [ParameterName-n Value-n...]
```

*PrinterInfoDescription* is a group data item, as defined in the copy file PRINTINF.CPY (on page 506).

*ParameterName* is an alphanumeric data item that contains the name of the device capability parameter to get (see Table 51).

*Value* is the COBOL data item used to receive the value of the parameter named by *ParameterName*. Possible values are contained in the 78-level entries in the copy file, PRINTINF.CPY.

**Note** For a more complete discussion of printer information parameters, see the Microsoft Windows documentation for the PRINTER_INFO_2 structure. For website information, see page 498.

**Table 51: Printer Information Parameters**

| Parameter Name | PICTURE Clause | Description |
|---|---|---|
| Server Name | PIC X(80) | Name of the server that controls the printer. |
| Printer Name | PIC X(80) | Name of the printer. |
| Share Name | PIC X(80) | Shared name of the printer |
| Port Name | PIC X(80) | Port(s) used to transmit data to the printer. |
| Driver Name | PIC X(80) | Name of the printer driver. |
| Comment | PIC X(80) | A brief description of the printer. |
| Location | PIC X(80) | Physical location of the printer. |
| Sep File | PIC X(80) | Name of file used to create the separator page. |
| Print Processor | PIC X(80) | Name of the print processor used by the printer. |
| Data Type | PIC X(80) | Data type used to record the print job. |
| Parameters | PIC X(80) | Default print-processor parameters. |
| Attributes | Group | **Note** This parameter is not available when using the *ParameterName/Value* pairs calling sequence. It is available only when using the *PrinterInfoDescription* group data item calling sequence. |
| Priority | PIC 9(10) | Print spooler priority. |
| Default Priority | PIC 9(10) | Default spooler priority. |
| Start Time | PIC 9(10) | Earliest allowed print time, in minutes, after midnight GMT. |
| Until Time | PIC 9(10) | Latest allowed print time, in minutes, after midnight GMT. |
| Status | Group | **Note** This parameter is not available when using the *ParameterName/Value* pairs calling sequence. It is available only when using the *PrinterInfoDescription* group data item calling sequence. |
| Jobs | PIC 9(10) | Number of jobs in the print queue. |
| Average PPM | PIC 9(10) | Average print speed in pages-per-minute. |

## P$NewPage

P$NewPage is used to force the next printer output to a new page and, if desired, change the page orientation. The P$NewPage subprogram may only be used after the printer is opened.

### Calling Sequence

```
CALL "P$NewPage" [USING Orientation]
```

*Orientation* can be any alphabetic or alphanumeric COBOL data type. It specifies "Portrait" or "Landscape" page orientation. Only the first letter of the value is relevant, and it is case-insensitive. Possible values are contained in the 78-level entries in the copy file WINDEFS.CPY (on page 512).

## P$ResetPrinter

P$ResetPrinter is used to perform the same functions to control printing as the Reset escape sequence (see Table 53 on page 526); that is, clear the margins, clear line spacing, clear line extend mode, reset text length, set page orientation back to portrait, set paper source back to DMBIN_ONLYONE, and set the font back to normal font.

### Calling Sequence

```
CALL "P$ResetPrinter"
```

## P$SetDocumentName

P$SetDocumentName is used to set the name of the document as it appears in the Windows printer status window. The P$SetDocumentName subprogram must be called before opening the printer for which you want to set the document name.

The default document name is "RM/COBOL". The name set by P$SetDocumentName remains in effect until P$SetDocumentName is called without an argument. The name will then revert to "RM/COBOL".

### Calling Sequence

```
CALL "P$SetDocumentName" [USING DocumentName]
```

*DocumentName* is an alphanumeric data item that contains the desired name of the print document.

## P$SetHandle

P$SetHandle is used to change the current P$ printer. This call is not needed unless multiple "PRINTER?" devices are open at the same time. See also the discussion of "PRINTER?" in Windows Printers (on page 306).

For an example that includes P$SetHandle, see Opening and Writing to Separate Printers (on page 521).

### Calling Sequence

```
CALL "P$SetHandle" USING Handle
```

*Handle* can be any COBOL numeric data item. It specifies the variable that contains the handle of the printer to use in subsequent P$ calls.

## P$SetRawMode

P$SetRawMode is used to set a raw mode output when the next printer is opened. This subprogram allows completely raw byte-stream I/O in applications that require it. It is intended to be used when a user is having problems sending escape sequences to networked printers on a Windows NT-class server. The RAW keyword (see page 306) of the DEFINE-DEVICE configuration record or the Printer Enable Raw Mode property (on page 80) may also be used to set raw mode output for a printer.

Only the following P$ subprograms can be used for a raw mode printer:

- P$ClearDialog (on page 459)
- P$DisableDialog (on page 460)
- P$DisplayDialog (on page 460)
- P$EnableDialog (on page 461)
- P$EnumPrinterInfo (on page 483)
- P$GetDialog (on page 461)
- P$SetDialog (on page 462)
- P$GetDeviceCapabilities (on page 485)
- P$GetHandle (on page 487)
- P$GetPrinterInfo (on page 488)
- P$SetHandle (on page 491)

### Calling Sequence

```
CALL "P$SetRawMode"
```

# Copy Files

The RM/COBOL development system supplies COBOL copy files to facilitate Windows printing program development using P$ subprograms.

The following copy files are supplied:

- DEFDEV.CPY (on page 492) is associated with P$GetDefineDeviceInfo (on page 484).
- DEVCAPS.CPY (on page 493) is associated with P$GetDeviceCapabilities (on page 485).
- LOGFONT.CPY (on page 494) is associated with P$SetFont (on page 473).
- PRINTDLG.CPY (on page 498) is associated with the following subprograms:
  - P$ChangeDeviceModes (on page 482)
  - P$GetDialog (on page 461)
  - P$SetDialog (on page 462)
- PRINTINF.CPY (on page 506) is associated with P$GetPrinterInfo (on page 488).
- TXTMTRIC.CPY (on page 508) is associated with P$GetTextMetrics (on page 470).
- WINDEFS.CPY (on page 512) contains miscellaneous items used by several P$ subprograms. This copy file is also used with C$PlaySound (on page 555).

**WARNING** We strongly recommend that you do not change these Liant-supplied copy files, as unpredictable results may occur if the copy files are changed incorrectly. If you must alter the files, please be aware that the names of the data items are the only thing that can be changed. Do not alter the pictures, types, sizes, or order of the data items.

# DEFDEV.CPY

DEFDEV.CPY contains the following definitions.

```
*
*   Define Device Information Definitions
*
 01 DefineDeviceInformation.
    02 DDI-DeviceName               Picture X(80).
    02 DDI-PortName                 Picture X(10).
    02 DDI-FontName                 Picture X(80).
    02 DDI-PointSize                Picture 9(5) Binary(2).
    02 DDI-RawModeValue             Picture X.
       88 DDI-RawMode                Value 'Y'  When False 'N'.
    02 DDI-EscapeModeValue          Picture X.
       88 DDI-EscapeMode             Value 'Y'  When False 'N'.
```

# DEVCAPS.CPY

Information regarding the Microsoft Windows GetDeviceCaps function can be found on the Internet at http://msdn.microsoft.com/library/. Periodically, Microsoft reorganizes the MSDN information on the website. Use the search capability to find information on the requested topic.

The following search tree was accurate when this document was produced:

```
Graphics and Multimedia
 GDI
  SDK Documentation
   Windows GDI
    Device Contexts
     Device Context Reference
      Device Context Functions
       GetDeviceCaps
```

DEVCAPS.CPY contains the following definitions.

```
*
*   Device Capabilities Definitions
*
 01 DeviceCapabilities.
    02 DC-DriverVersion              Picture 9(10) Binary(4).
    02 DC-TechnologyValue            Picture 9 Binary(4).
       88 DC-TechnologyIsPlotter      Value 0.
       88 DC-TechnologyIsRASDisplay   Value 1.
       88 DC-TechnologyIsRASPrinter   Value 2.
       88 DC-TechnologyIsRASCamera    Value 3.
       88 DC-TechnologyIsCharStream   Value 4.
       88 DC-TechnologyIsRASMetafile  Value 5.
       88 DC-TechnologyIsRASDispfile  Value 6.
    02 DC-HorzSize                   Picture 9(10) Binary(4).
    02 DC-VertSize                   Picture 9(10) Binary(4).
    02 DC-HorzRes                    Picture 9(10) Binary(4).
    02 DC-VertRes                    Picture 9(10) Binary(4).
    02 DC-LogPixelsX                 Picture 9(10) Binary(4).
    02 DC-LogPixelsY                 Picture 9(10) Binary(4).
    02 DC-AspectX                    Picture 9(10) Binary(4).
    02 DC-AspectY                    Picture 9(10) Binary(4).
    02 DC-AspectXY                   Picture 9(10) Binary(4).
    02 DC-PhysicalWidth              Picture 9(10) Binary(4).
    02 DC-PhysicalHeight             Picture 9(10) Binary(4).
    02 DC-PhysicalOffsetX            Picture 9(10) Binary(4).
    02 DC-PhysicalOffsetY            Picture 9(10) Binary(4).
    02 DC-ScalingFactorX             Picture 9(10) Binary(4).
    02 DC-ScalingFactorY             Picture 9(10) Binary(4).
*
*   Technology Values
*
 78 DC-TechnologyPlotter              Value 0.
 78 DC-TechnologyRASDisplay           Value 1.
 78 DC-TechnologyRASPrinter           Value 2.
 78 DC-TechnologyRASCamera            Value 3.
```

```
 78 DC-TechnologyCharStream          Value 4.
 78 DC-TechnologyRASMetafile         Value 5.
 78 DC-TechnologyRASDispfile         Value 6.
*
*   Parameter Name Values
*
 78 DC-DriverVersionParam            Value "Driver Version".
 78 DC-TechnologyParam               Value "Technology".
 78 DC-HorizontalSizeParam           Value "Horizontal Size".
 78 DC-VerticalSizeParam             Value "Vertical Size".
 78 DC-HorizontalResolutionParam     Value "Horizontal Resolution".
 78 DC-VerticalResolutionParam       Value "Vertical Resolution".
 78 DC-LogicalPixelsXParam           Value "Logical Pixels X".
 78 DC-LogicalPixelsYParam           Value "Logical Pixels Y".
 78 DC-AspectXParam                  Value "Aspect X".
 78 DC-AspectYParam                  Value "Aspect Y".
 78 DC-AspectXYParam                 Value "Aspect XY".
 78 DC-PhysicalWidthParam            Value "Physical Width".
 78 DC-PhysicalHeightParam           Value "Physical Height".
 78 DC-PhysicalOffsetXParam          Value "Physical Offset X".
 78 DC-PhysicalOffsetYParam          Value "Physical Offset Y".
 78 DC-ScalingFactorXParam           Value "Scaling Factor X".
 78 DC-ScalingFactorYParam           Value "Scaling Factor Y".
```

# LOGFONT.CPY

Information regarding the Microsoft Windows LOGFONT structure can be found on the Internet at http://msdn.microsoft.com/library/. Periodically, Microsoft reorganizes the MSDN information on the website. Use the search capability to find information on the requested topic.

The following search tree was accurate when this document was produced:

```
Graphics and Multimedia
 GDI
  SDK Documentation
   Windows GDI
    Fonts and Text
     Font and Text Reference
      Font and Text Structures
       LOGFONT
```

LOGFONT.CPY contains the following definitions.

```
*
*   Logical Font Definitions
*
 01 LogicalFont.
    02 LF-Height                    Picture S9(5) Binary(2).
    02 LF-Width                     Picture 9(5) Binary(2).
    02 LF-Escapement                Picture 9(5) Binary(2).
    02 LF-Orientation               Picture 9(5) Binary(2).
    02 LF-WeightValue               Picture 9(3) Binary(2).
       88 LF-WeightIsDontCare        Value 0.
       88 LF-WeightIsThin            Value 100.
```

```
   88 LF-WeightIsExtraLight       Value 200.
   88 LF-WeightIsUltraLight       Value 200.
   88 LF-WeightIsLight            Value 300.
   88 LF-WeightIsNormal           Value 400.
   88 LF-WeightIsRegular          Value 400.
   88 LF-WeightIsMedium           Value 500.
   88 LF-WeightIsSemiBold         Value 600.
   88 LF-WeightIsDemiBold         Value 600.
   88 LF-WeightIsBold             Value 700.
   88 LF-WeightIsExtraBold        Value 800.
   88 LF-WeightIsUltraBold        Value 800.
   88 LF-WeightIsHeavy            Value 900.
   88 LF-WeightIsBlack            Value 900.
02 LF-ItalicValue                Picture X.
   88 LF-Italic                   Value 'Y'  When False 'N'.
02 LF-UnderlineValue             Picture X.
   88 LF-Underline                Value 'Y'  When False 'N'.
02 LF-StrikeoutValue             Picture X.
   88 LF-Strikeout                Value 'Y'  When False 'N'.
02 LF-CharSetValue                Picture 9(3) Binary(2).
   88 LF-CharSetIsANSI            Value 0.
   88 LF-CharSetIsDefault         Value 1.
   88 LF-CharSetIsSymbol          Value 2.
   88 LF-CharSetIsMAC             Value 77.
   88 LF-CharSetIsShiftJIS        Value 128.
   88 LF-CharSetIsHangeul         Value 129.
   88 LF-CharSetIsJohab           Value 130.
   88 LF-CharSetIsChineseBig5     Value 136.
   88 LF-CharSetIsGreek           Value 161.
   88 LF-CharSetIsTurkish         Value 162.
   88 LF-CharSetIsHebrew          Value 177.
   88 LF-CharSetIsArabic          Value 178.
   88 LF-CharSetIsBaltic          Value 186.
   88 LF-CharSetIsRussian         Value 204.
   88 LF-CharSetIsThai            Value 222.
   88 LF-CharSetIsEastEurope      Value 238.
   88 LF-CharSetIsOEM             Value 255.
02 LF-OutPrecisValue              Picture 9 Binary(2).
   88 LF-OutPrecisIsDefault       Value 0.
   88 LF-OutPrecisIsString        Value 1.
   88 LF-OutPrecisIsStroke        Value 3.
   88 LF-OutPrecisIsTrueType      Value 4.
   88 LF-OutPrecisIsDevice        Value 5.
   88 LF-OutPrecisIsRaster        Value 6.
   88 LF-OutPrecisIsTruTypeOnly   Value 7.
   88 LF-OutPrecisIsOutline       Value 8.
02 LF-ClipPrecisValue             Picture 9(3) Binary(2).
   88 LF-ClipPrecisIsDefault      Value 0.
   88 LF-ClipPrecisIsStroke       Value 2.
   88 LF-ClipPrecisIsLHAngles     Value 16.
   88 LF-ClipPrecisIsEmbedded     Value 128.
02 LF-QualityValue                Picture 9 Binary(2).
   88 LF-QualityIsDefault         Value 0.
   88 LF-QualityIsDraft           Value 1.
```

```
           88 LF-QualityIsProof          Value 2.
        02 LF-PitchValue                 Picture 9 Binary(2).
           88 LF-PitchIsDefault          Value 0.
           88 LF-PitchIsFixed            Value 1.
           88 LF-PitchIsVariable         Value 2.
        02 LF-FamilyValue                Picture 9 Binary(2).
           88 LF-FamilyIsDontCare        Value 0.
           88 LF-FamilyIsRoman           Value 1.
           88 LF-FamilyIsSwiss           Value 2.
           88 LF-FamilyIsModern          Value 3.
           88 LF-FamilyIsScript          Value 4.
           88 LF-FamilyIsDecorative      Value 5.
        02 LF-FaceName                   Picture X(31).
      *
      *   Font Weight Values
      *
       78 LF-WeightDontCare               Value 0.
       78 LF-WeightThin                   Value 100.
       78 LF-WeightExtraLight             Value 200.
       78 LF-WeightUltraLight             Value 200.
       78 LF-WeightLight                  Value 300.
       78 LF-WeightNormal                 Value 400.
       78 LF-WeightRegular                Value 400.
       78 LF-WeightMedium                 Value 500.
       78 LF-WeightSemiBold               Value 600.
       78 LF-WeightDemiBold               Value 600.
       78 LF-WeightBold                   Value 700.
       78 LF-WeightExtraBold              Value 800.
       78 LF-WeightUltraBold              Value 800.
       78 LF-WeightHeavy                  Value 900.
       78 LF-WeightBlack                  Value 900.
      *
      *   Font Character Set Values
      *
       78 LF-CharSetANSI                  Value 0.
       78 LF-CharSetDefault               Value 1.
       78 LF-CharSetSymbol                Value 2.
       78 LF-CharSetMAC                   Value 77.
       78 LF-CharSetShiftJIS              Value 128.
       78 LF-CharSetHangeul               Value 129.
       78 LF-CharSetJohab                 Value 130.
       78 LF-CharSetChineseBig5           Value 136.
       78 LF-CharSetGreek                 Value 161.
       78 LF-CharSetTurkish               Value 162.
       78 LF-CharSetHebrew                Value 177.
       78 LF-CharSetArabic                Value 178.
       78 LF-CharSetBaltic                Value 186.
       78 LF-CharSetRussian               Value 204.
       78 LF-CharSetThai                  Value 222.
       78 LF-CharSetEastEurope            Value 238.
       78 LF-CharSetOEM                   Value 255.
      *
      *   Font Output Precision Values
      *
```

```
 78 LF-OutPrecisDefault              Value 0.
 78 LF-OutPrecisString               Value 1.
 78 LF-OutPrecisStroke               Value 3.
 78 LF-OutPrecisTrueType             Value 4.
 78 LF-OutPrecisDevice               Value 5.
 78 LF-OutPrecisRaster               Value 6.
 78 LF-OutPrecisTruTypeOnly          Value 7.
 78 LF-OutPrecisOutline              Value 8.
*
*   Font Clipping Precision Values
*
 78 LF-ClipPrecisDefault             Value 0.
 78 LF-ClipPrecisStroke              Value 2.
 78 LF-ClipPrecisLHAngles            Value 16.
 78 LF-ClipPrecisEmbedded            Value 128.
*
*   Font Quality Values
*
 78 LF-QualityDefault                Value 0.
 78 LF-QualityDraft                  Value 1.
 78 LF-QualityProof                  Value 2.
*
*   Font Pitch Values
*
 78 LF-PitchDefault                  Value 0.
 78 LF-PitchFixed                    Value 1.
 78 LF-PitchVariable                 Value 2.
*
*   Font Family Values
*
 78 LF-FamilyDontCare                Value 0.
 78 LF-FamilyRoman                   Value 1.
 78 LF-FamilySwiss                   Value 2.
 78 LF-FamilyModern                  Value 3.
 78 LF-FamilyScript                  Value 4.
 78 LF-FamilyDecorative              Value 5.
*
*   Parameter Name Values
*
 78 LF-HeightParam                   Value "Height".
 78 LF-WidthParam                    Value "Width".
 78 LF-EscapementParam               Value "Escapement".
 78 LF-OrientationParam              Value "Orientation".
 78 LF-WeightParam                   Value "Weight".
 78 LF-ItalicParam                   Value "Italic".
 78 LF-UnderlineParam                Value "Underline".
 78 LF-StrikeOutParam                Value "Strike Out".
 78 LF-CharSetParam                  Value "Char Set".
 78 LF-OutPrecisionParam             Value "Out Precision".
 78 LF-ClipPrecisionParam            Value "Clip Precision".
 78 LF-QualityParam                  Value "Quality".
 78 LF-PitchParam                    Value "Pitch".
 78 LF-FamilyParam                   Value "Family".
 78 LF-FaceNameParam                 Value "Face Name".
```

# PRINTDLG.CPY

Information regarding the Microsoft Windows PRINTDLG and DEVMODE structures can be found on the Internet at http://msdn.microsoft.com/library/. Periodically, Microsoft reorganizes the MSDN information on the website. Use the search capability to find information on the requested topic.

The following search trees were accurate when this document was produced:

**For PRINTDLG:**

```
User Interface Design and Development
 Windows Management
  User Input
   Common Dialog Box Library
    Common Dialog Box Reference
     Common Dialog Box Structures
      PRINTDLG
```

**For DEVMODE:**

```
Graphics and Multimedia
 GDI
  SDK Documentation
   Windows GDI
    Printing and Print Spooler
     Printing and Print Spooler Reference
      Printing and Print Spooler Structures
       DEVMODE
```

PRINTDLG.CPY contains the following definitions.

```
*
*   Print Dialog Definitions
*
 01 PrintDialog.
    02  PD-ReturnValue                Picture X.
        88  PD-OKReturn                Value 'Y' When False 'N'.
    02  PD-ExtendedErrorValue          Picture 9(5) Binary(2).
        88  PD-ExtErrIsCanceled        Value 0.
        88  PD-ExtErrIsStructSize      Value 1.
        88  PD-ExtErrIsInitialization  Value 2.
        88  PD-ExtErrIsNoTemplate      Value 3.
        88  PD-ExtErrIsNoHInstance     Value 4.
        88  PD-ExtErrIsLoadStrFailure  Value 5.
        88  PD-ExtErrIsFindResFailure  Value 6.
        88  PD-ExtErrIsLoadResFailure  Value 7.
        88  PD-ExtErrIsLockResFailure  Value 8.
        88  PD-ExtErrIsMemAllocFailure Value 9.
        88  PD-ExtErrIsMemLockFailure  Value 10.
        88  PD-ExtErrIsNoHook          Value 11.
        88  PD-ExtErrIsRegisterMsgFail Value 12.
        88  PD-ExtErrIsSetupFailure    Value 4097.
        88  PD-ExtErrIsParseFailure    Value 4098.
        88  PD-ExtErrIsRetDefFailure   Value 4099.
```

```
    88  PD-ExtErrIsLoadDrvFailure   Value 4100.
    88  PD-ExtErrIsGetDevModeFail   Value 4101.
    88  PD-ExtErrIsInitFailure      Value 4102.
    88  PD-ExtErrIsNoDevices        Value 4103.
    88  PD-ExtErrIsNoDefaultPrn     Value 4104.
    88  PD-ExtErrIsDNDMMismatch     Value 4105.
    88  PD-ExtErrIsCreateICFailure  Value 4106.
    88  PD-ExtErrIsPrinterNotFound  Value 4107.
    88  PD-ExtErrIsDefaultDifferent Value 4108.
    88  PD-ExtErrIsDialogFailure    Value 65535.
02  PD-Flags.
    03  PD-AllPagesFlagValue       Picture X.
        88  PD-AllPagesFlag         Value 'Y' When False 'N'.
    03  PD-SelectionFlagValue      Picture X.
        88  PD-SelectionFlag        Value 'Y' When False 'N'.
    03  PD-PageNumbersFlagValue    Picture X.
        88  PD-PageNumbersFlag      Value 'Y' When False 'N'.
    03  PD-NoSelectionFlagValue    Picture X.
        88  PD-NoSelectionFlag      Value 'Y' When False 'N'.
    03  PD-NoPageNumbersFlagValue  Picture X.
        88  PD-NoPageNumbersFlag    Value 'Y' When False 'N'.
    03  PD-CollateFlagValue        Picture X.
        88  PD-CollateFlag          Value 'Y' When False 'N'.
    03  PD-PrintSetupFlagValue     Picture X.
        88  PD-PrintSetupFlag       Value 'Y' When False 'N'.
    03  PD-PrintToFileFlagValue    Picture X.
        88  PD-PrintToFileFlag      Value 'Y' When False 'N'.
    03  PD-NoWarningFlagValue      Picture X.
        88  PD-NoWarningFlag        Value 'Y' When False 'N'.
    03  PD-UseDevModeCopiesFlagValue  Picture X.
        88  PD-UseDevModeCopiesFlag Value 'Y' When False 'N'.
    03  PD-DisablePrintToFileFlagValue Picture X.
        88  PD-DisablePrintToFileFlag Value 'Y' When False 'N'.
    03  PD-HidePrintToFileFlagValue Picture X.
        88  PD-HidePrintToFileFlag  Value 'Y' When False 'N'.
    03  PD-NoNetworkButtonFlagValue Picture X.
        88  PD-NoNetworkButtonFlag  Value 'Y' When False 'N'.
02  PD-FromPage                    Picture 9(5) Binary(2).
02  PD-ToPage                      Picture 9(5) Binary(2).
02  PD-MinPage                     Picture 9(5) Binary(2).
02  PD-MaxPage                     Picture 9(5) Binary(2).
02  PD-Copies                      Picture 9(5) Binary(2).
02  DM-DeviceName                  Picture X(80).
02  DM-Fields.
    03  DM-OrientationFieldValue   Picture X.
        88  DM-OrientationField     Value 'Y' When False 'N'.
    03  DM-PaperSizeFieldValue     Picture X.
        88  DM-PaperSizeField       Value 'Y' When False 'N'.
    03  DM-PaperLengthFieldValue   Picture X.
        88  DM-PaperLengthField     Value 'Y' When False 'N'.
    03  DM-PaperWidthFieldValue    Picture X.
        88  DM-PaperWidthField      Value 'Y' When False 'N'.
    03  DM-ScaleFieldValue         Picture X.
        88  DM-ScaleField           Value 'Y' When False 'N'.
```

```
           03  DM-CopiesFieldValue        Picture X.
               88  DM-CopiesField          Value 'Y' When False 'N'.
           03  DM-PaperSourceFieldValue   Picture X.
               88  DM-PaperSourceField     Value 'Y' When False 'N'.
           03  DM-PrintQualityFieldValue  Picture X.
               88  DM-PrintQualityField    Value 'Y' When False 'N'.
           03  DM-ColorFieldValue         Picture X.
               88  DM-ColorField           Value 'Y' When False 'N'.
           03  DM-DuplexFieldValue        Picture X.
               88  DM-DuplexField          Value 'Y' When False 'N'.
           03  DM-YresolutionFieldValue   Picture X.
               88  DM-YresolutionField     Value 'Y' When False 'N'.
           03  DM-TrueTypeOptionFieldValue Picture X.
               88  DM-TrueTypeOptionField  Value 'Y' When False 'N'.
           03  DM-CollateFieldValue       Picture X.
               88  DM-CollateField         Value 'Y' When False 'N'.
           03  DM-ICMMethodFieldValue     Picture X.
               88  DM-ICMMethodField       Value 'Y' When False 'N'.
           03  DM-ICMIntentFieldValue     Picture X.
               88  DM-ICMIntentField       Value 'Y' When False 'N'.
           03  DM-MediaTypeFieldValue     Picture X.
               88  DM-MediaTypeField       Value 'Y' When False 'N'.
           03  DM-DitherTypeFieldValue    Picture X.
               88  DM-DitherTypeField      Value 'Y' When False 'N'.
       02  DM-OrientationValue            Picture 9 Binary(2).
           88  DM-OrientationIsPortrait    Value 1.
           88  DM-OrientationIsLandscape   Value 2.
       02  DM-PaperSizeValue              Picture 9(2) Binary(2).
           88  DM-PaperSizeIsLetter        Value 1.
           88  DM-PaperSizeIsLetterSmall   Value 2.
           88  DM-PaperSizeIsTabloid       Value 3.
           88  DM-PaperSizeIsLedger        Value 4.
           88  DM-PaperSizeIsLegal         Value 5.
           88  DM-PaperSizeIsStatement     Value 6.
           88  DM-PaperSizeIsExecutive     Value 7.
           88  DM-PaperSizeIsA3            Value 8.
           88  DM-PaperSizeIsA4            Value 9.
           88  DM-PaperSizeIsA4Small       Value 10.
           88  DM-PaperSizeIsA5            Value 11.
           88  DM-PaperSizeIsB4            Value 12.
           88  DM-PaperSizeIsB5            Value 13.
           88  DM-PaperSizeIsFolio         Value 14.
           88  DM-PaperSizeIsQuarto        Value 15.
           88  DM-PaperSizeIs10x14         Value 16.
           88  DM-PaperSizeIs11x17         Value 17.
           88  DM-PaperSizeIsNote          Value 18.
           88  DM-PaperSizeIsEnv9          Value 19.
           88  DM-PaperSizeIsEnv10         Value 20.
           88  DM-PaperSizeIsEnv11         Value 21.
           88  DM-PaperSizeIsEnv12         Value 22.
           88  DM-PaperSizeIsEnv14         Value 23.
           88  DM-PaperSizeIsCSheet        Value 24.
           88  DM-PaperSizeIsDSheet        Value 25.
           88  DM-PaperSizeIsESheet        Value 26.
```

```
    88  DM-PaperSizeIsEnvDl         Value 27.
    88  DM-PaperSizeIsEnvC5         Value 28.
    88  DM-PaperSizeIsEnvC3         Value 29.
    88  DM-PaperSizeIsEnvC4         Value 30.
    88  DM-PaperSizeIsEnvC6         Value 31.
    88  DM-PaperSizeIsEnvC65        Value 32.
    88  DM-PaperSizeIsEnvB4         Value 33.
    88  DM-PaperSizeIsEnvB5         Value 34.
    88  DM-PaperSizeIsEnvB6         Value 35.
    88  DM-PaperSizeIsEnvItaly      Value 36.
    88  DM-PaperSizeIsEnvMonarch    Value 37.
    88  DM-PaperSizeIsEnvPersonal   Value 38.
    88  DM-PaperSizeIsFanFoldUS     Value 39.
    88  DM-PaperSizeIsFanFoldStdGerman Value 40.
    88  DM-PaperSizeIsFanFoldLglGerman Value 41.
02  DM-PaperLength                 Picture 9(5) Binary(2).
02  DM-PaperWidth                  Picture 9(5) Binary(2).
02  DM-Scale                       Picture 9(5) Binary(2).
02  DM-Copies                      Picture 9(5) Binary(2).
02  DM-PaperSourceValue            Picture 9(2) Binary(2).
    88  DM-PaperSourceIsUpper       Value 1.
    88  DM-PaperSourceIsOnlyOne     Value 1.
    88  DM-PaperSourceIsLower       Value 2.
    88  DM-PaperSourceIsMiddle      Value 3.
    88  DM-PaperSourceIsManual      Value 4.
    88  DM-PaperSourceIsEnvelope    Value 5.
    88  DM-PaperSourceIsEnvManual   Value 6.
    88  DM-PaperSourceIsAuto        Value 7.
    88  DM-PaperSourceIsTractor     Value 8.
    88  DM-PaperSourceIsSmallFmt    Value 9.
    88  DM-PaperSourceIsLargeFmt    Value 10.
    88  DM-PaperSourceIsLargeCapacity Value 11.
    88  DM-PaperSourceIsCassette    Value 14.
    88  DM-PaperSourceIsFormSource  Value 15.
02  DM-ResolutionValue             Picture S9 Binary(2).
    88  DM-ResolutionIsDraft        Value -1.
    88  DM-ResolutionIsLow          Value -2.
    88  DM-ResolutionIsMedium       Value -3.
    88  DM-ResolutionIsHigh         Value -4.
02  DM-ColorValue                  Picture 9 Binary(2).
    88  DM-ColorIsMonochrome        Value 1.
    88  DM-ColorIsColor             Value 2.
02  DM-DuplexValue                 Picture 9 Binary(2).
    88  DM-DuplexIsSimplex          Value 1.
    88  DM-DuplexIsVertical         Value 2.
    88  DM-DuplexIsHorizontal       Value 3.
02  DM-Yresolution                 Picture 9(5) Binary(2).
02  DM-TrueTypeValue               Picture 9 Binary(2).
    88  DM-TrueTypeIsBitmap         Value 1.
    88  DM-TrueTypeIsDownload       Value 2.
    88  DM-TrueTypeIsSubDev         Value 3.
02  DM-CollateValue                Picture 9 Binary(2).
    88  DM-CollateIsFalse           Value 0.
    88  DM-CollateIsTrue            Value 1.
```

```
           02  DM-ICMMethodValue              Picture 9 Binary(4).
               88  DM-ICMMethodIsNone          Value 1.
               88  DM-ICMMethodIsSystem        Value 2.
               88  DM-ICMMethodIsDriver        Value 3.
               88  DM-ICMMethodIsDevice        Value 4.
           02  DM-ICMIntentValue              Picture 9 Binary(4).
               88  DM-ICMIntentIsSaturate      Value 1.
               88  DM-ICMIntentIsContrast      Value 2.
               88  DM-ICMIntentIsColorMetric   Value 3.
           02  DM-MediaTypeValue              Picture 9 Binary(4).
               88  DM-MediaTypeIsStandard      Value 1.
               88  DM-MediaTypeIsTransparency  Value 2.
               88  DM-MediaTypeIsGlossy        Value 3.
           02  DM-DitherTypeValue             Picture 99 Binary(4).
               88  DM-DitherTypeIsNone         Value 1.
               88  DM-DitherTypeIsCoarse       Value 2.
               88  DM-DitherTypeIsFine         Value 3.
               88  DM-DitherTypeIsLineArt      Value 4.
               88  DM-DitherTypeIsErrorDiffusion Value 5.
               88  DM-DitherTypeIsGrayScale    Value 10.
      *
      •   Print Dialog Extended Error Values
      *
       78  PD-ExtErrCanceled                  Value 0.
       78  PD-ExtErrStructSize                Value 1.
       78  PD-ExtErrInitialization            Value 2.
       78  PD-ExtErrNoTemplate                Value 3.
       78  PD-ExtErrNoHInstance               Value 4.
       78  PD-ExtErrLoadStrFailure            Value 5.
       78  PD-ExtErrFindResFailure            Value 6.
       78  PD-ExtErrLoadResFailure            Value 7.
       78  PD-ExtErrLockResFailure            Value 8.
       78  PD-ExtErrMemAllocFailure           Value 9.
       78  PD-ExtErrMemLockFailure            Value 10.
       78  PD-ExtErrNoHook                    Value 11.
       78  PD-ExtErrRegisterMsgFail           Value 12.
       78  PD-ExtErrSetupFailure              Value 4097.
       78  PD-ExtErrParseFailure              Value 4098.
       78  PD-ExtErrRetDefFailure             Value 4099.
       78  PD-ExtErrLoadDrvFailure            Value 4100.
       78  PD-ExtErrGetDevModeFail            Value 4101.
       78  PD-ExtErrInitFailure               Value 4102.
       78  PD-ExtErrNoDevices                 Value 4103.
       78  PD-ExtErrNoDefaultPrn              Value 4104.
       78  PD-ExtErrDNDMMismatch              Value 4105.
       78  PD-ExtErrCreateICFailure           Value 4106.
       78  PD-ExtErrPrinterNotFound           Value 4107.
       78  PD-ExtErrDefaultDifferent          Value 4108.
       78  PD-ExtErrDialogFailure             Value 65535.
      *
      •   Device Mode Orientation Values
      *
       78  DM-OrientationPortrait             Value 1.
       78  DM-OrientationLandscape            Value 2.
```

```
*
*   Device Mode Paper Size Values
*
 78  DM-PaperSizeLetter                Value 1.
 78  DM-PaperSizeLetterSmall           Value 2.
 78  DM-PaperSizeTabloid               Value 3.
 78  DM-PaperSizeLedger                Value 4.
 78  DM-PaperSizeLegal                 Value 5.
 78  DM-PaperSizeStatement             Value 6.
 78  DM-PaperSizeExecutive             Value 7.
 78  DM-PaperSizeA3                    Value 8.
 78  DM-PaperSizeA4                    Value 9.
 78  DM-PaperSizeA4Small               Value 10.
 78  DM-PaperSizeA5                    Value 11.
 78  DM-PaperSizeB4                    Value 12.
 78  DM-PaperSizeB5                    Value 13.
 78  DM-PaperSizeFolio                 Value 14.
 78  DM-PaperSizeQuarto                Value 15.
 78  DM-PaperSize10x14                 Value 16.
 78  DM-PaperSize11x17                 Value 17.
 78  DM-PaperSizeNote                  Value 18.
 78  DM-PaperSizeEnv9                  Value 19.
 78  DM-PaperSizeEnv10                 Value 20.
 78  DM-PaperSizeEnv11                 Value 21.
 78  DM-PaperSizeEnv12                 Value 22.
 78  DM-PaperSizeEnv14                 Value 23.
 78  DM-PaperSizeCSheet                Value 24.
 78  DM-PaperSizeDSheet                Value 25.
 78  DM-PaperSizeESheet                Value 26.
 78  DM-PaperSizeEnvDl                 Value 27.
 78  DM-PaperSizeEnvC5                 Value 28.
 78  DM-PaperSizeEnvC3                 Value 29.
 78  DM-PaperSizeEnvC4                 Value 30.
 78  DM-PaperSizeEnvC6                 Value 31.
 78  DM-PaperSizeEnvC65                Value 32.
 78  DM-PaperSizeEnvB4                 Value 33.
 78  DM-PaperSizeEnvB5                 Value 34.
 78  DM-PaperSizeEnvB6                 Value 35.
 78  DM-PaperSizeEnvItaly              Value 36.
 78  DM-PaperSizeEnvMonarch            Value 37.
 78  DM-PaperSizeEnvPersonal           Value 38.
 78  DM-PaperSizeFanFoldUS             Value 39.
 78  DM-PaperSizeFanFoldStdGerman      Value 40.
 78  DM-PaperSizeFanFoldLglGerman      Value 41.
*
*   Device Mode Paper Source Values
*
 78  DM-PaperSourceUpper               Value 1.
 78  DM-PaperSourceOnlyOne             Value 1.
 78  DM-PaperSourceLower               Value 2.
 78  DM-PaperSourceMiddle              Value 3.
 78  DM-PaperSourceManual              Value 4.
 78  DM-PaperSourceEnvelope            Value 5.
 78  DM-PaperSourceEnvManual           Value 6.
```

```
 78  DM-PaperSourceAuto                Value 7.
 78  DM-PaperSourceTractor             Value 8.
 78  DM-PaperSourceSmallFmt            Value 9.
 78  DM-PaperSourceLargeFmt            Value 10.
 78  DM-PaperSourceLargeCapacity       Value 11.
 78  DM-PaperSourceCassette            Value 14.
 78  DM-PaperSourceFormSource          Value 15.
*
•   Device Mode Resolution Values
*
 78  DM-ResolutionDraft                Value -1.
 78  DM-ResolutionLow                  Value -2.
 78  DM-ResolutionMedium               Value -3.
 78  DM-ResolutionHigh                 Value -4.
*
•   Device Mode Color Values
*
 78  DM-ColorMonochrome                Value 1.
 78  DM-ColorColor                     Value 2.
*
•   Device Mode Duplex Values
*
 78  DM-DuplexSimplex                  Value 1.
 78  DM-DuplexVertical                 Value 2.
 78  DM-DuplexHorizontal               Value 3.
*
•   Device Mode True Type Values
*
 78  DM-TrueTypeBitmap                 Value 1.
 78  DM-TrueTypeDownload               Value 2.
 78  DM-TrueTypeSubDev                 Value 3.
*
•   Device Mode Collate Values
*
 78  DM-CollateFalse                   Value 0.
 78  DM-CollateTrue                    Value 1.
*
•   Device ICM Method Values
*
 78  DM-ICMMethodNone                  Value 1.
 78  DM-ICMMethodSystem                Value 2.
 78  DM-ICMMethodDriver                Value 3.
 78  DM-ICMMethodDevice                Value 4.
*
•   Device ICM Type Values
*
 78  DM-ICMTypeSaturate                Value 1.
 78  DM-ICMTypeContrast                Value 2.
 78  DM-ICMTypeColorMetric             Value 3.
*
•   Device Mode Media Type Values
*
 78  DM-MediaTypeStandard              Value 1.
 78  DM-MediaTypeTransparency          Value 2.
```

```
 78  DM-MediaTypeGlossy                Value 3.
*
*   Device Mode Dither Type Values
*
 78  DM-DitherTypeNone                 Value 1.
 78  DM-DitherTypeCoarse               Value 2.
 78  DM-DitherTypeFine                 Value 3.
 78  DM-DitherTypeLineArt              Value 4.
 78  DM-DitherTypeErrorDiffusion       Value 5.
 78  DM-DitherTypeGrayScale            Value 10.
*
*   P$DisplayDialog Return Values
*
 78  PD-ReturnParam                    Value 0.
 78  PD-ReturnCancelled                Value 1.
 78  PD-ReturnError                    Value 2.
*
*   Parameter Name Values
*
 78 PD-ReturnParam                     Value "Return".
 78 PD-ExtendedErrorParam              Value "Extended Error".
 78 PD-AllPagesFlagParam               Value "All Pages Flag".
 78 PD-SelectionFlagParam              Value "Selection Flag".
 78 PD-PageNumbersFlagParam            Value "Page Numbers Flag".
 78 PD-NoSelectionFlagParam            Value "No Selection Flag".
 78 PD-NoPageNumbersFlagParam          Value "No Page Numbers Flag".
 78 PD-CollateFlagParam                Value "Collate Flag".
 78 PD-PrintSetupFlagParam             Value "Print Setup Flag".
 78 PD-PrintToFileFlagParam            Value "Print To File Flag".
 78 PD-NoWarningFlagParam              Value "No Warning Flag".
 78 PD-UseDevModeCopiesFlagParam       Value
                                       "Use Device Mode Copies Flag".
 78 PD-DisablePrintToFileFlagParam     Value
                                       "Disable Print To File Flag".
 78 PD-HidePrintToFileFlagParam        Value
                                       "Hide Print To File Flag".
 78 PD-NoNetworkButtonFlagParam        Value "No Network Button Flag".
 78 PD-FromPageParam                   Value "From Page".
 78 PD-ToPageParam                     Value "To Page".
 78 PD-MinPageParam                    Value "Min Page".
 78 PD-MaxPageParam                    Value "Max Page".
 78 PD-PrintDialogCopiesParam          Value "Print Dialog Copies".
 78 DM-DeviceNameParam                 Value "Device Name".
 78 DM-OrientationParam                Value "Orientation".
 78 DM-PaperSizeParam                  Value "Paper Size".
 78 DM-PaperLengthParam                Value "Paper Length".
 78 DM-PaperWidthParam                 Value "Paper Width".
 78 DM-ScaleParam                      Value "Scale".
 78 DM-DeviceModeCopiesParam           Value "Device Mode Copies".
 78 DM-DefaultSourceParam              Value "Default Source".
 78 DM-PrintQualityParam               Value "Print Quality".
 78 DM-ColorParam                      Value "Color".
 78 DM-DuplexParam                     Value "Duplex".
 78 DM-YResolutionParam                Value "Y Resolution".
```

```
 78 DM-TrueTypeOptionParam            Value "True Type Option".
 78 DM-CollateParam                   Value "Collate".
 78 DM-ICMMethodParam                 Value "ICM Method".
 78 DM-ICMIntentParam                 Value "ICM Intent".
 78 DM-MediaTypeParam                 Value "Media Type".
 78 DM-DitherTypeParam                Value "Dither Type".
```

# PRINTINF.CPY

Information regarding the Microsoft Windows PRINTER_INFO_2 structure can be found on the Internet at http://msdn.microsoft.com/library/. Periodically, Microsoft reorganizes the MSDN information on the website. Use the search capability to find information on the requested topic.

The following search tree was accurate when this document was produced:

```
Graphics and Multimedia
 GDI
  SDK Documentation
   Windows GDI
    Printing and Print Spooler
     Printing and Print Spooler Reference
      Printing and Print Spooler Structures
       PRINTER_INFO_2
```

PRINTINF.CPY contains the following definitions.

```
*
*   Printer Information Definitions
*
01 PrinterInformation.
    02  PI-ServerName                Picture X(80).
    02  PI-PrinterName               Picture X(80).
    02  PI-ShareName                 Picture X(80).
    02  PI-PortName                  Picture X(80).
    02  PI-DriverName                Picture X(80).
    02  PI-Comment                   Picture X(80).
    02  PI-Location                  Picture X(80).
    02  PI-SepFile                   Picture X(80).
    02  PI-PrintProcessor            Picture X(80).
    02  PI-DataType                  Picture X(80).
    02  PI-Parameters                Picture X(80).
    02  PI-Attribute.
        03  PI-QueuedAttributeValue   Picture X.
            88  PI-QueuedAttribute     Value 'Y' When False 'N'.
        03  PI-DirectAttributeValue   Picture X.
            88  PI-DirectAttribute     Value 'Y' When False 'N'.
        03  PI-DefaultAttributeValue  Picture X.
            88  PI-DefaultAttribute    Value 'Y' When False 'N'.
        03  PI-SharedAttributeValue   Picture X.
            88  PI-SharedAttribute     Value 'Y' When False 'N'.
        03  PI-NetworkAttributeValue  Picture X.
            88  PI-NetworkAttribute    Value 'Y' When False 'N'.
        03  PI-HiddenAttributeValue   Picture X.
            88  PI-HiddenAttribute     Value 'Y' When False 'N'.
```

```
    03  PI-LocalAttributeValue     Picture X.
        88  PI-LocalAttribute       Value 'Y' When False 'N'.
    03  PI-EnableDEVQAttributeValue Picture X.
        88  PI-EnableDEVQAttribute  Value 'Y' When False 'N'.
    03  PI-KeepPrintedAttributeValue Picture X.
        88  PI-KeepPrintedJobsAttribute Value 'Y' When False 'N'.
    03  PI-DoComplete1stAttributeValue Picture X.
        88  PI-DoCompleteFirstAttribute Value 'Y' When False 'N'.
    03  PI-WorkOfflineAttributeValue Picture X.
        88  PI-WorkOfflineAttribute Value 'Y' When False 'N'.
    03  PI-EnableBIDIAttributeValue Picture X.
        88  PI-EnableBIDIAttribute Value 'Y' When False 'N'.
02  PI-Priority                    Picture 9(10) Binary(4).
02  PI-DefaultPriority            Picture 9(10) Binary(4).
02  PI-StartTime                  Picture 9(10) Binary(4).
02  PI-UntilTime                  Picture 9(10) Binary(4).
02  PI-Status.
    03  PI-PausedStatusValue       Picture X.
        88  PI-PausedStatus         Value 'Y' When False 'N'.
    03  PI-ErrorStatusValue        Picture X.
        88  PI-ErrorStatus          Value 'Y' When False 'N'.
    03  PI-PendingDeletionStatusValue Picture X.
        88  PI-PendingDeletionStatus Value 'Y' When False 'N'.
    03  PI-PaperJamStatusValue     Picture X.
        88  PI-PaperJamStatus       Value 'Y' When False 'N'.
    03  PI-PaperOutStatusValue     Picture X.
        88  PI-PaperOutStatus       Value 'Y' When False 'N'.
    03  PI-ManualFeedStatusValue   Picture X.
        88  PI-ManualFeedStatus     Value 'Y' When False 'N'.
    03  PI-PaperProblemStatusValue Picture X.
        88  PI-PaperProblemStatus   Value 'Y' When False 'N'.
    03  PI-OfflineStatusValue      Picture X.
        88  PI-OfflineStatus        Value 'Y' When False 'N'.
    03  PI-IOActiveStatusValue     Picture X.
        88  PI-IOActiveStatus       Value 'Y' When False 'N'.
    03  PI-BusyStatusValue         Picture X.
        88  PI-BusyStatus           Value 'Y' When False 'N'.
    03  PI-PrintingStatusValue     Picture X.
        88  PI-PrintingStatus       Value 'Y' When False 'N'.
    03  PI-OutputBinFullStatusValue Picture X.
        88  PI-OutputBinFullStatus  Value 'Y' When False 'N'.
    03  PI-NotAvailableStatusValue Picture X.
        88  PI-NotAvailableStatus   Value 'Y' When False 'N'.
    03  PI-WaitingStatusValue      Picture X.
        88  PI-WaitingStatus        Value 'Y' When False 'N'.
    03  PI-ProcessingStatusValue   Picture X.
        88  PI-ProcessingStatus     Value 'Y' When False 'N'.
    03  PI-IntializingStatusValue  Picture X.
        88  PI-InitializingStatus   Value 'Y' When False 'N'.
    03  PI-WarmingUpStatusValue    Picture X.
        88  PI-WarmingUpStatus      Value 'Y' When False 'N'.
    03  PI-ToneLowStatusValue      Picture X.
        88  PI-TonerLowStatus       Value 'Y' When False 'N'.
    03  PI-NoTonerStatusValue      Picture X.
```

```
           88  PI-NoTonerStatus        Value 'Y' When False 'N'.
       03  PI-PagePuntStatusValue     Picture X.
           88  PI-PagePuntStatus       Value 'Y' When False 'N'.
       03  PI-UserInterventionStatusValue Picture X.
           88  PI-UserInterventionStatus Value 'Y' When False 'N'.
       03  PI-OutOfMemoryStatusValue  Picture X.
           88  PI-OutOfMemoryStatus    Value 'Y' When False 'N'.
       03  PI-DoorOpenStatusValue     Picture X.
           88  PI-DoorOpenStatus       Value 'Y' When False 'N'.
       03  PI-ServerUnknownStatusValue Picture X.
           88  PI-ServerUnknownStatus  Value 'Y' When False 'N'.
       03  PI-PowerSaveStatusValue    Picture X.
           88  PI-PowerSaveStatus      Value 'Y' When False 'N'.
   02  PI-Jobs                        Picture 9(10) Binary(4).
   02  PI-AveragePPM                  Picture 9(10) Binary(4).
*
*  Parameter Name Values
*
 78  PI-ServerNameParam                Value "Server Name".
 78  PI-PrinterNameParam               Value "Printer Name".
 78  PI-ShareNameParam                 Value "Share Name".
 78  PI-PortNameParam                  Value "Port Name".
 78  PI-DriverNameParam                Value "Driver Name".
 78  PI-CommentParam                   Value "Comment".
 78  PI-LocationParam                  Value "Location".
 78  PI-SepFileParam                   Value "Sep File".
 78  PI-PrintProcessorParam            Value "Print Processor".
 78  PI-DataTypeParam                  Value "Data Type".
 78  PI-ParametersParam                Value "Parameters".
 78  PI-PriorityParam                  Value "Priority".
 78  PI-DefaultPriorityParam           Value "Default Priority".
 78  PI-StartTimeParam                 Value "Start Time".
 78  PI-UntilTimeParam                 Value "Until Time".
 78  PI-JobsParam                      Value "Jobs".
 78  PI-AveragePPMParam                Value "Average PPM".
```

# TXTMTRIC.CPY

Information regarding the Microsoft Windows TEXTMETRIC structure can be found on the Internet at http://msdn.microsoft.com/library/. Periodically, Microsoft reorganizes the MSDN information on the website. Use the search capability to find information on the requested topic.

The following search tree was accurate when this document was produced:

```
Graphics and Multimedia
 GDI
  SDK Documentation
   Windows GDI
    Fonts and Text
     Font and Text Reference
      Font and Text Structures
       TEXTMETRIC
```

TXTMTRIC.CPY contains the following definitions.

```
*
*   Text Metric Definitions
*
01 TextMetrics.
    02  TM-Height                    Picture 9(10) Binary(4).
    02  TM-Ascent                    Picture 9(10) Binary(4).
    02  TM-Descent                   Picture 9(10) Binary(4).
    02  TM-InternalLeading           Picture 9(10) Binary(4).
    02  TM-ExternalLeading           Picture 9(10) Binary(4).
    02  TM-AveCharWidth              Picture 9(10) Binary(4).
    02  TM-MaxCharWidth              Picture 9(10) Binary(4).
    02  TM-WeightValue               Picture 9(3) Binary(4).
        88  TM-WeightIsDontCare       Value 0.
        88  TM-WeightIsThin           Value 100.
        88  TM-WeightIsExtraLight     Value 200.
        88  TM-WeightIsUltraLight     Value 200.
        88  TM-WeightIsLight          Value 300.
        88  TM-WeightIsNormal         Value 400.
        88  TM-WeightIsRegular        Value 400.
        88  TM-WeightIsMedium         Value 500.
        88  TM-WeightIsSemiBold       Value 600.
        88  TM-WeightIsDemiBold       Value 600.
        88  TM-WeightIsBold           Value 700.
        88  TM-WeightIsExtraBold      Value 800.
        88  TM-WeightIsUltraBold      Value 800.
        88  TM-WeightIsHeavy          Value 900.
        88  TM-WeightIsBlack          Value 900.
    02  TM-Overhang                  Picture 9(10) Binary(4).
    02  TM-DigitizedAspectX          Picture 9(10) Binary(4).
    02  TM-DigitizedAspectY          Picture 9(10) Binary(4).
    02  TM-ItalicValue               Picture X.
        88  TM-Italic                 Value 'Y' When False 'N'.
    02  TM-UnderlinedValue           Picture X.
        88  TM-Underlined             Value 'Y' When False 'N'.
    02  TM-StruckOutValue            Picture X.
        88  TM-StruckOut              Value 'Y' When False 'N'.
    02  TM-FirstChar                 Picture X.
    02  TM-LastChar                  Picture X.
    02  TM-DefaultChar               Picture X.
    02  TM-BreakChar                 Picture X.
    02  TM-PitchValue                Picture 9 Binary(2).
        88  TM-PitchIsFixedPitch      Value 1.
        88  TM-PitchIsVector          Value 2.
        88  TM-PitchIsTrueType        Value 4.
        88  TM-PitchIsDevice          Value 8.
    02  TM-FamilyValue               Picture 9 Binary(2).
        88  TM-FamilyIsDontCare       Value 0.
        88  TM-FamilyIsRoman          Value 1.
        88  TM-FamilyIsSwiss          Value 2.
        88  TM-FamilyIsModern         Value 3.
        88  TM-FamilyIsScript         Value 4.
        88  TM-FamilyIsDecorative     Value 5.
```

```
           02  TM-CharSetValue                Picture 9(3) Binary(2).
               88  TM-CharSetIsANSI            Value 0.
               88  TM-CharSetIsDefault         Value 1.
               88  TM-CharSetIsSymbol          Value 2.
               88  TM-CharSetIsMAC             Value 77.
               88  TM-CharSetIsShiftJIS        Value 128.
               88  TM-CharSetIsHangeul         Value 129.
               88  TM-CharSetIsJohab           Value 130.
               88  TM-CharSetIsChineseBig5     Value 136.
               88  TM-CharSetIsGreek           Value 161.
               88  TM-CharSetIsTurkish         Value 162.
               88  TM-CharSetIsHebrew          Value 177.
               88  TM-CharSetIsArabic          Value 178.
               88  TM-CharSetIsBaltic          Value 186.
               88  TM-CharSetIsRussian         Value 204.
               88  TM-CharSetIsThai            Value 222.
               88  TM-CharSetIsEastEurope      Value 238.
               88  TM-CharSetIsOEM             Value 255.
      *
      •   Weight Values
      *
       78  TM-WeightDontCare                   Value 0.
       78  TM-WeightThin                       Value 100.
       78  TM-WeightExtraLight                 Value 200.
       78  TM-WeightUltraLight                 Value 200.
       78  TM-WeightLight                      Value 300.
       78  TM-WeightNormal                     Value 400.
       78  TM-WeightRegular                    Value 400.
       78  TM-WeightMedium                     Value 500.
       78  TM-WeightSemiBold                   Value 600.
       78  TM-WeightDemiBold                   Value 600.
       78  TM-WeightBold                       Value 700.
       78  TM-WeightExtraBold                  Value 800.
       78  TM-WeightUltraBold                  Value 800.
       78  TM-WeightHeavy                      Value 900.
       78  TM-WeightBlack                      Value 900.
      *
      •   Pitch Values
      *
       78  TM-PitchFixedPitch                  Value 1.
       78  TM-PitchVector                      Value 2.
       78  TM-PitchTrueType                    Value 4.
       78  TM-PitchDevice                      Value 8.
      *
      •   Family Values
      *
       78  TM-FamilyDontCare                   Value 0.
       78  TM-FamilyRoman                      Value 1.
       78  TM-FamilySwiss                      Value 2.
       78  TM-FamilyModern                     Value 3.
       78  TM-FamilyScript                     Value 4.
       78  TM-FamilyDecorative                 Value 5.
      *
      •   Character Set Values
```

```
*
 78  TM-CharSetANSI                   Value 0.
 78  TM-CharSetDefault                Value 1.
 78  TM-CharSetSymbol                 Value 2.
 78  TM-CharSetMAC                    Value 77.
 78  TM-CharSetShiftJIS               Value 128.
 78  TM-CharSetHangeul                Value 129.
 78  TM-CharSetJohab                  Value 130.
 78  TM-CharSetChineseBig5            Value 136.
 78  TM-CharSetGreek                  Value 161.
 78  TM-CharSetTurkish                Value 162.
 78  TM-CharSetHebrew                 Value 177.
 78  TM-CharSetArabic                 Value 178.
 78  TM-CharSetBaltic                 Value 186.
 78  TM-CharSetRussian                Value 204.
 78  TM-CharSetThai                   Value 222.
 78  TM-CharSetEastEurope             Value 238.
 78  TM-CharSetOEM                    Value 255.
*
•   Parameter Name Values
*
 78  TM-HeightParam                   Value "Height".
 78  TM-AscentParam                   Value "Ascent".
 78  TM-DescentParam                  Value "Descent".
 78  TM-InternalLeadingParam          Value "Internal Leading".
 78  TM-ExternalLeadingParam          Value "External Leading".
 78  TM-AverageCharacterWidthParam    Value "Average Character
Width".
 78  TM-MaximumCharacterWidthParam    Value "Maximum Character
Width".
 78  TM-WeightParam                   Value "Weight".
 78  TM-OverhangParam                 Value "Overhang".
 78  TM-DigitizedAspectXParam         Value "Digitized Aspect X".
 78  TM-DigitizedAspectYParam         Value "Digitized Aspect Y".
 78  TM-FirstCharacterParam           Value "First Character".
 78  TM-LastCharacterParam            Value "Last Character".
 78  TM-DefaultCharacterParam         Value "Default Character".
 78  TM-BreakCharacterParam           Value "Break Character".
 78  TM-ItalicParam                   Value "Italic".
 78  TM-UnderlinedParam               Value "Underlined".
 78  TM-StruckOutParam                Value "Struck Out".
 78  TM-PitchParam                    Value "Pitch".
 78  TM-FamilyParam                   Value "Family".
 78  TM-CharacterSetParam             Value "Character Set".
```

# WINDEFS.CPY

Information regarding the Microsoft Windows PlaySound function and LOGPEN structure can be found on the Internet at http://msdn.microsoft.com/library/. Periodically, Microsoft reorganizes the MSDN information on the website. Use the search capability to find information on the requested topic.

The following search trees were accurate when this document was produced:

**For PlaySound:**

```
Graphics and Multimedia
 Windows Multimedia
  SDK Documentation
   Windows Multimedia
    Multimedia Reference
     Multimedia Functions
      PlaySound
```

**For LOGPEN:**

```
Graphics and Multimedia
 GDI
  SDK Documentation
   Windows GDI
    Pens
     Pen Reference
      Pen Structures
       LOGPEN
```

WINDEFS.CPY contains the following definitions.

```
*
*   Miscellaneous Windows Definitions
*
*
*   C$PlaySound Options
*
 78 SoundSync                          Value 0.
 78 SoundAsync                         Value 2 ** 0.
 78 SoundNoDefault                     Value 2 ** 1.
 78 SoundNoStop                        Value 2 ** 4.
 78 SoundPurge                         Value 2 ** 6.
 78 SoundApplication                   Value 2 ** 7.
 78 SoundNoWait                        Value 2 ** 13.
 78 SoundAlias                         Value 2 ** 16.
 78 SoundFilename                      Value 2 ** 17.
 78 SoundAliasId                       Value (2 ** 16) + (2 ** 20).
*
*   Pen Style Values
*
 78 PenStyleSolid                      Value 0.
 78 PenStyleDash                       Value 1.
 78 PenStyleDot                        Value 2.
 78 PenStyleDashDot                    Value 3.
```

```
 78 PenStyleDashDotDot                  Value 4.
 78 PenStyleNull                        Value 5.
*
*   Position Alignment Argument Values
*
 78 PositionIsTop                       Value "Top".
 78 PositionIsBottom                    Value "Bottom".
*
*   Position Mode Argument Values
*
 78 ModeIsAbsolute                      Value "Absolute".
 78 ModeIsRelative                      Value "Relative".
*
*   Position Unit Argument Values
*
 78 UnitsAreInches                      Value "Inches".
 78 UnitsAreMetric                      Value "Metric".
 78 UnitsAreCharacters                  Value "Characters".
 78 UnitsAreDeviceUnits                 Value "Device Units".
*
*   Yes/No Argument Values
*
 78 DrawBoxWithShading                  Value "Yes".
 78 DrawBoxWithoutShading               Value "No".
 78 DrawBoxNoShading                    Value "No".
 78 TextOutWithBox                      Value "Yes".
 78 TextOutWithoutBox                   Value "No".
 78 TextOutNoBox                        Value "No".
 78 TextOutWithShading                  Value "Yes".
 78 TextOutWithoutShading               Value "No".
 78 TextOutNoShading                    Value "No".
 78 WithBox                             Value "Yes".
 78 WithoutBox                          Value "No".
 78 NoBox                               Value "No".
 78 WithShading                         Value "Yes".
 78 WithoutShading                      Value "No".
 78 NoShading                           Value "No".
*
*   Color Argument Values
*
 78 ColorBlack                          Value "Black".
 78 ColorDarkBlue                       Value "Dark Blue".
 78 ColorDarkGreen                      Value "Dark Green".
 78 ColorDarkCyan                       Value "Dark Cyan".
 78 ColorDarkRed                        Value "Dark Red".
 78 ColorDarkMagenta                    Value "Dark Magenta".
 78 ColorBrown                          Value "Brown".
 78 ColorDarkGray                       Value "Dark Gray".
 78 ColorLightGray                      Value "Light Gray".
 78 ColorBlue                           Value "Blue".
 78 ColorGreen                          Value "Green".
 78 ColorCyan                           Value "Cyan".
 78 ColorRed                            Value "Red".
 78 ColorMagenta                        Value "Magenta".
```

```
 78 ColorYellow                         Value "Yellow".
 78 ColorWhite                          Value "White".
*
*   Pitch Name Values
*
 78 PitchNormal                         Value "Normal".
 78 PitchExpanded                       Value "Expanded".
 78 PitchCompressed                     Value "Compressed".
*
*   Page Orientation Values
*
 78 OrientationPortrait                 Value "Portrait".
 78 OrientationLandscape                Value "Landscape".
```

# Example Code Fragments

The code fragments in this section illustrate the use of P$ subprograms to facilitate Windows printing program development. Table 52 provides a quick reference for the tasks performed in the code fragment examples. These code fragments can be found in **pexample.cbl** in the Samples directory.

**Note** Additional comprehensive examples using P$ subprograms can be found in the Samples directory. These samples may clarify any remaining questions you might have about using the P$ subprogram library.

**Table 52: Task Reference List**

| To | See Example |
|---|---|
| Change a font while printing. | Changing a Font While Printing (on page 520). |
| Change the print orientation. | Presetting the Print Dialog Box (on page 518) and Changing Orientation, Pitch, and Line Spacing (on page 521). |
| Change the pitch of a font. | Changing Orientation, Pitch, and Line Spacing (on page 521). |
| Change the print resolution. | Presetting the Print Dialog Box (on page 518). |
| Check the return code value. | Checking the Return Code After Displaying the Print Dialog Box (on page 519). |
| Draw a box around text. | Drawing a Box Around Text (on page 516). |
| Draw a shaded box with colors. | Drawing Shaded Boxes with Colors (on page 516). |
| Draw a box using “relative” positioning. | Drawing Shaded Boxes with Colors (on page 516). |
| Draw a ruler. | Drawing a Ruler (on page 517). |
| Open three separate printers and write to each one. | Opening and Writing to Separate Printers (on page 521). |
| Print text at corners of a page. | Printing Text at the Corners of a Page (on page 522). |
| Print text at the top of a page. | Printing Text at the Top of a Page (on page 522). |
| Print a bitmap file. | Printing a Bitmap (on page 520). |
| Print multiple copies. | Presetting the Print Dialog Box (on page 518). |
| Print multiple text outputs on the same line. | Using the COBOL WRITE Statement to Print Multiple Text Outputs on the Same Line (on page 520). |
| Print a word in italics. | Using the COBOL WRITE Statement to Print Multiple Text Outputs on the Same Line (on page 520). |
| Print a word in boldface type. | Using the COBOL WRITE Statement to Print Multiple Text Outputs on the Same Line (on page 520). |
| Print a word underlined. | Using the COBOL WRITE Statement to Print Multiple Text Outputs on the Same Line (on page 520). |
| Print a watermark. | Printing a Watermark (on page 516). |
| Set the point size for a font. | Setting the Point Size for a Font (on page 523). |
| Set text position. | Setting Text Position (on page 524). |

## Printing a Watermark

The following code fragment illustrates printing a watermark diagonally across a page. The font is set to a large point size with a 45-degree angle (Escapement). The color is set to a light gray, and the text is positioned with metric measurements.

```
CALL "P$SetFont" USING LF-HeightParam 500, LF-EscapementParam, 450.
CALL "P$SetTextColor" USING ColorLightGray.
CALL "P$TextOut" USING "Example", 5.08, 17.78, "Absolute", "Metric".
```

## Drawing Shaded Boxes with Colors

The following code fragment illustrates drawing two boxes. The first box is drawn with a thick pen, shaded with ten-percent gray, positioned at 2,2 inches on the page, and is one-inch square. The second box is positioned using "relative" positioning so that the lower-right corner of the first box is connected to the upper-left corner of the second box. It is drawn with a thin, dashed red line, no shading, and is 1.5-inches square.

```
CALL "P$SetPen" USING PenStyleSolid, 5.
CALL "P$SetBoxShade" USING ColorBlack, 10.
CALL "P$DrawBox" USING 2.00, 2.00, "Absolute", "Inches", 1.00, 1.00,
   "Inches", WithShading.
CALL "P$SetPen" USING PenStyleDash, 1, ColorRed.
CALL "P$DrawBox" USING 0, 0, "Relative", "Inches", 1.50, 1.50,
   "Inches".
```

## Drawing a Box Around Text

The following code fragment illustrates drawing a box centered around a line of text. The box allows one-half the average character width of the current font before and after the text. The top of the box is positioned at the top alignment of the current font. The height of the box is the height of the current font plus external leading.

```
CALL "P$GetTextMetrics" USING TextMetrics.
CALL "P$SetPosition" USING 0.5, 5.50, "Absolute", "Inches".
CALL "P$GetTextPosition" USING X-POS, Y-POS, "Top", "Device Units".
CALL "P$GetTextExtent" USING "AAaaGGggYYyyTTttH", WIDTH, HEIGHT,
   "Device Units".
COMPUTE X-POS  = X-POS - (TM-AveCharWidth / 2).
COMPUTE WIDTH = WIDTH + TM-AveCharWidth.
COMPUTE HEIGHT = TM-ExternalLeading + TM-Height.
CALL "P$DrawBox" USING X-POS, Y-POS, "Absolute", "Device Units",
   WIDTH, HEIGHT, "Device Units".
CALL "P$TextOut" USING "AAaaGGggYYyyTTttH", 0.5, 5.50,
   "Absolute", "Inches".
```

The following code fragment performs the same task as the previous part of this example, but allows the runtime to compute the size and position of the box as well as draw the box. For top, bottom, left, and right margins, the box allows one-quarter the average character width of the current font.

```
CALL "P$TextOut" USING     "AAaaGGggYYyyTTttH", 0.5, 6.00,
   "Absolute", "Inches", WithBox.
```

## Drawing a Ruler

The following code fragment draws a 5-centimeter ruler.

```
Centimeter-Ruler.
   MOVE 0.0 TO WS-X, WS-Y, WS-Y2.
   MOVE 5.0 TO WS-X2.
   MOVE 0 to WS-CENT-COUNT.
   CALL "P$SetDefaultUnits" USING "Metric".

*Draw a Line 5 Centimeters Long.
   CALL "P$DrawLine" USING WS-X, WS-Y, "Absolute", "Metric",
      WS-X2, WS-Y2.
   PERFORM VARYING WS-X FROM 0 BY 0.1 UNTIL WS-X > 5
      EVALUATE TRUE
        WHEN WS-CENT-COUNT = 5
          MOVE .4 TO WS-Y
        WHEN WS-CENT-COUNT = 10 OR 0
          MOVE .7 TO WS-y
        WHEN OTHER
          MOVE .2 TO WS-Y
      END-EVALUATE
      CALL "P$MoveTo" USING WS-X, WS-Y
      CALL "P$LineTo" USING WS-X, WS-Y2
      IF WS-CENT-COUNT = 10
          MOVE 1 TO WS-CENT-COUNT
      ELSE
          ADD  1 TO WS-CENT-COUNT
      END-IF

   END-PERFORM.
```

## Presetting the Print Dialog Box

The following code fragment causes two copies of the output to be printed with high resolution. The orientation of the print will be in landscape mode. When the Windows Print dialog box appears, the All Pages option button will be selected, the Page Numbers option button and associated edit control will be disabled, the Selection button will be disabled, and the Print to File check box will be hidden.

**Note** The example code fragment shown below sets the collate flag to false, that is, printed pages will not be collated. Some printers do not support printing multiple copies with the pages collated. For more information, see Printing Multiple Copies (on page 458).

```
INITIALIZE PrintDialog.
CALL "P$ClearDialog".
SET PD-AllPagesFlag           TO TRUE.       *>Sets All Pages
                                             *>option on
SET PD-NoPageNumbersFlag      TO TRUE.       *>Disable Pages
                                             *>option button and
                                             *>associated edit
                                             *>controls
SET PD-NoSelectionFlag        TO TRUE.       *>Disables Selection
                                             *>button
SET PD-CollateFlag            TO FALSE.      *>Don't check Collate
                                             *>check box
SET PD-HidePrintToFileFlag    TO TRUE.       *>Hides Print To File
                                             *>checkbox
SET DM-CopiesField            TO TRUE.
MOVE 2                        TO DM-Copies.
SET DM-OrientationField       TO TRUE.
SET DM-OrientationIsLandscape TO TRUE.
SET DM-PrintQualityField      TO TRUE.
SET DM-ResolutionIsHigh       TO TRUE.
CALL "P$SetDialog" USING PrintDialog.
OPEN OUTPUT PFILE.
WRITE FD-RECORD FROM
      "Example 5:  Presetting the Printer Dialog Box Page 1"
      AFTER PAGE.
WRITE FD-RECORD FROM
      "Example 5:  Text After CALLING P$SetDialog".
WRITE FD-RECORD FROM
      "Example 5:  This should be printed in landscape mode.".
WRITE FD-RECORD FROM
      "Example 5:  There should be two copies printed.".
CLOSE PFILE.
```

## Checking the Return Code After Displaying the Print Dialog Box

The following code fragment will set the printer device, display the Windows Print dialog box, and check the return code value. If the value is zero, the user pressed the OK button. If the value is 1, the user pressed the Cancel button or closed the dialog box. If the value is 2, an error occurred while displaying the dialog box and the error code is checked by making a call to P$GetDialog (on page 461).

```
Set PD-NoWarningFlag To True.
CALL "P$SetDialog" USING DM-DeviceNameParam, DM-DeviceName.
CALL "P$DisplayDialog" GIVING Dialog-Return.

Evaluate True
    When Dialog-OK                      *> Value zero
       Display "OK Button Pressed" Line 13 Col 1
       Perform Open-Printer-Para
    When Dialog-Cancel                  *> Value one
       Display "Cancel Button Pressed"  Line 13 Col 1
       Perform Printer-Canceled-Para
    When Dialog-Error                   *> Value two
       Display "Error in Dialog"  Line 13 Col 1
       Perform Printer-Error-Para
    When Other                          *> Value other
       Display "Invalid Value: " Line 13 Col 1 Dialog-Return
       End-Evaluate.
STOP RUN.

Open-Printer-Para.
    Open Output Printer-File.
    Write Fd-Record From
       "Checking the Printer Dialog Box " After Page.
    Close Pfile.

Printer-Canceled-Para.
    Display "Printer Dialog Canceled" Line 22 Col 3 Reverse
    Erase Eol.

Printer-Error-Para.
    CALL "P$GetDialog" USING PD-ExtendedErrorParam,
                             PD-ExtendedErrorValue,
                             Validity-Flag.

If PD-ExtErrIsPrinterNotFound
    Display "Printer Not Found!" Line 22 Col 3 Reverse
    Erase Eol
Else
    Display "Printer Dialog Had Error" Line 22 Col 3 Reverse
                             PD-ExtendedErrorValue Convert
    Erase Eol

End-If.
```

## Printing a Bitmap

The following code fragment prints a bitmap at location 2,2 (inches). The bitmap will be two inches wide, and the height will be scaled to match the original width/height ratio.

```
CALL "P$DrawBitmap" USING "rmlogo.bmp", 2, 2, "Absolute",
   "Inches", 2, 0, "Inches"
                  GIVING Bitmap-Return.
```

## Changing a Font While Printing

The following code fragment prints the text, “Original FONT” in the current font, then switches to Monotype Corsiva and prints the text, “Monotype Corsiva FONT Italic”.

```
CALL "P$TextOut" USING "Original FONT", 0.5, 3.50, "Absolute",
   "Inches".
CALL "P$SetFont" USING LF-FaceNameParam, "Monotype Corsiva",
   LF-ItalicParam, "Y".
CALL "P$TextOut" USING "Monotype Corsiva FONT Italic", 0.5, 4.50,
   "Absolute", "Inches".
```

## Using the COBOL WRITE Statement to Print Multiple Text Outputs on the Same Line

The following code fragment causes multiple outputs to appear on a single line. The COBOL WRITE statement is used with the ADVANCING phrase and P$SetLineExtendMode (on page 476) to produce the following line of text:

Printing a word in *Italic*, Underline, or **Bold** on the same line is no problem with RM/COBOL.

```
INITIALIZE LogicalFont.
CALL "P$SetFont" USING    LF-HeightParam    50,
                          LF-WeightParam    10,
                          LF-FaceNameParam  "Arial".
WRITE PRINT-RECORD FROM "Printing a word in".
CALL "P$SetLineExtendMode" USING 1, "Characters".
CALL "P$SetFont" USING LF-ItalicParam, "Y".
WRITE PRINT-RECORD FROM "Italic", AFTER ADVANCING ZERO.
CALL "P$SetLineExtendMode" USING 1, "Characters".
CALL "P$SetFont" USING LF-ItalicParam, "N", LF-UnderlineParam,
"Y".
WRITE FD-RECORD FROM "Underline," AFTER ADVANCING ZERO.
CALL "P$SetLineExtendMode" USING 1, "Characters".
CALL "P$SetFont" USING    LF-WeightParam, LF-WeightBold,
                          LF-UnderlineParam, "N".
WRITE PRINT-RECORD FROM "or Bold" AFTER ADVANCING ZERO.
CALL "P$SetLineExtendMode" USING 1, "Characters".
CALL "P$SetFont" USING LF-WeightParam, LF-WeightNormal.
WRITE PRINT-RECORD FROM
   "on the same line is no problem with RM/COBOL."
   AFTER ADVANCING ZERO.
```

## Changing Orientation, Pitch, and Line Spacing

The following code fragment demonstrates how to change the print orientation from portrait to landscape, how to change the font pitch from normal to compressed, and how to change the line spacing to 8 lines-per-inch.

```
CALL "P$TextOut" USING "This is Orientation Portrait Normal
    Pitch" 0, 1.50, "Absolute", "Inches".
CALL "P$SetPitch" USING PitchCompressed.
CALL "P$SetLineSpacing" USING 8.
CALL "P$ChangeDeviceModes" USING   DM-OrientationParam,
                                   DM-OrientationLandscape.
WRITE FD-RECORD FROM SPACES AFTER PAGE.
CALL "P$TextOut" USING
                         "This is printed with the Orientation
                         "Landscape with a Compressed Type Size",
                         0, 1.50, "Absolute", "Inches".
CALL "P$SetPitch" USING PitchNormal.

CALL "P$ChangeDeviceModes" USING   DM-OrientationParam,
                                   DM-OrientationPortrait.
WRITE FD-RECORD FROM SPACES AFTER PAGE.
CALL "P$TextOut" USING
    "This is back to Portrait using the Normal Pitch Size",
    0, 1.50, "Absolute", "Inches".
```

## Opening and Writing to Separate Printers

The following code fragment demonstrates how to open three separate printers and write to each one.

```
OPEN OUTPUT PFILE1.
CALL "P$GetHandle" USING HANDLE-1.

CALL "P$EnableDialog".
OPEN OUTPUT PFILE2.
CALL "P$GetHandle" USING HANDLE-2.

CALL "P$EnableDialog".
OPEN OUTPUT PFILE3.
CALL "P$GetHandle" USING HANDLE-3.

CALL "P$SetHandle" USING HANDLE-3.
CALL "P$TextOut"   USING "Text written to PRINTER 3".

CALL "P$SetHandle" USING HANDLE-2.
CALL "P$TextOut"   USING "Text written to PRINTER 2".

CALL "P$SetHandle" USING HANDLE-1.
CALL "P$TextOut"   USING "Text written to PRINTER 1".

CLOSE PFILE1, PFILE2, PFILE3.
```

## Printing Text at the Top of a Page

The following code fragment demonstrates how to position and print text at the top center of the page.

```
    CALL "P$GetDeviceCapabilities" USING   DC-HorizontalSizeParam,
                                           DC-HorzSize.
    CALL "P$GetTextExtent" USING "Top Center",
                                           Ws-Text-Width,
                                           Ws-Text-Height,
                                           UnitsAreMetric.
* Divide by 10 to convert millimeters to centimeters.
    Compute Ws-X-M = ((DC-HorzSize / 10) / 2) - (Ws-Text-Width / 2).
    Move Zero to Ws-Y-M.
    CALL "P$SetTextPosition" USING Ws-X-M, Ws-Y-M,
                                           PositionIsTop,
                                           ModeIsAbsolute,
                                           UnitsAreMetric.
    CALL "P$TextOut" USING "Top Center".
```

## Printing Text at the Corners of a Page

The following code fragment demonstrates how to position and print text at the top left corner and bottom right corner of the page.

```
* Position Text Top Left.
    CALL "P$SetTextPosition" USING 0, 0,
                                   PositionIsTop,
                                   ModeIsAbsolute,
                                   UnitsAreDeviceUnits.

    CALL "P$TextOut" USING "Top Left".

* Position Text Bottom Right.
    CALL "P$GetTextExtent" USING "Bottom Right",
                                 Ws-Text-Width,
                                 Ws-Text-Height,
                                 UnitsAreMetric.

    CALL "P$GetDeviceCapabilities" USING DC-HorizontalSizeParam,
                                         DC-HorzSize,
                                         DC-VerticalSizeParam,
                                         DC-VertSize.

* Divide by 10 to convert millimeters to centimeters.
    Compute Ws-X-M = (DC-HorzSize / 10) - Ws-Text-Width.
    Compute Ws-Y-M = (DC-VertSize / 10).
    CALL "P$SetTextPosition" USING Ws-X-M, Ws-Y-M,
                                   PositionIsBottom,
                                   ModeIsAbsolute,
                                   UnitsAreMetric.

    CALL "P$TextOut" USING "Bottom Right".
```

## Setting the Point Size for a Font

The following code fragment demonstrates how to set the point size for a specified font.

```
   Open Output PrintFile.
   CALL "P$GetDeviceCapabilities" USING DC-LogicalPixelsYParam,
                                        DC-LogPixelsY.

* Compute Font Height for desired Point-Size.
* If Point-Size is 72, this produces a 1-inch high font, including
* Internal Leading.
   Compute LF-Height Rounded = -((Point-Size * Dc-LogPixelsY) / 72).

   Move "Times New Roman" To LF-FaceName.
   CALL "P$SetFont" USING LF-FaceNameParam, LF-FaceName,
                          LF-HeightParam, LF-Height.
   CALL "P$GetTextMetrics" USING TextMetrics.

* Add bias for Internal Leading.
* If Point-Size is 72, this adjustment produces a 1-inch high font
* excluding Internal Leading.
   Compute LF-Height = (LF-Height * TM-Height) /
                       (TM-Height - TM-InternalLeading).

* Set Font with computed LF-Height
   CALL "P$SetFont" USING LF-HeightParam, LF-Height.
   Write Printer-Record from "Greetings" After Advancing 1.
```

## Setting Text Position

The following code fragment will print large "Greetings" followed by three smaller “Greetings” positioned by top alignment, base line alignment and bottom alignment.

```
* Print initial Greetings in a Large Font Size.
   CALL "P$SetFont" USING LF-HeightParam, 330.
   CALL "P$TextOut" USING "Greetings", 0.25, 2, ModeIsAbsolute,
      UnitsAreInches.
* Get the Top Text Position.
   CALL "P$GetTextPosition" USING Ws-X-D, Ws-Top,
PositionIsTop,
      UnitsAreDeviceUnits.
* Get the Bottom Text Position.
   CALL "P$GetTextPosition" USING Ws-XB-D, Ws-Bottom,
      PositionIsBottom,
      UnitsAreDeviceUnits.
* Get the Base Line Position.
   CALL "P$GetPosition" USING Ws-XP-D, Ws-Base,
      UnitsAreDeviceUnits.
   CALL "P$SetFont" USING LF-HeightParam, 50.
* Set the Top Text Position and print, get next X Position.
   CALL "P$SetTextPosition" USING Ws-X-D, Ws-Top,
PositionIsTop,
      ModeIsAbsolute, UnitsAreDeviceUnits.
   CALL "P$TextOut" USING "Greetings".
   CALL "P$GetPosition" USING Ws-X-D, Ws-Dummy,
      UnitsAreDeviceUnits.
* Set the Base Line Position and print, get next X Position.
   CALL "P$SetPosition" USING Ws-X-D, Ws-Base, ModeIsAbsolute,
      UnitsAreDeviceUnits.
   CALL "P$TextOut" USING "Greetings".
   CALL "P$GetPosition" USING Ws-X-D, Ws-Dummy,
      UnitsAreDeviceUnits.
* Set the Bottom Position and print.
   CALL "P$SetTextPosition" USING Ws-X-D, Ws-Bottom,
      PositionIsBottom,
      ModeIsAbsolute, UnitsAreDeviceUnits.
   CALL "P$TextOut" USING "Greetings".
```

# RM/COBOL-Specific Escape Sequences

The typical COBOL application written for Windows makes use of the various Windows printer drivers to control printing functions. Many legacy COBOL programs, however, contain embedded or programmatic escape sequences and thus cannot take advantage of Windows printer drivers. To accommodate these programs, the RM/COBOL runtime has been enhanced to implement a set of RM/COBOL-specific escape sequences to control printing. These escape sequences, which are similar to the Hewlett Packard PCL (Print Command Language) commands, are then mapped by the runtime to the correct printer driver GDI (Graphics Device Interface) calls to accomplish the desired function. Note that some escape sequences will not affect the print output until the next page. For example, changing the print orientation will not affect the current page.

The escape sequences—when enabled—are supported only on Windows. Recognition of the escape sequences can be enabled in one of three ways:

- Set the value of the Printer Enable Escape Sequences property (on page 79) to True.
- Set the ESCAPE-SEQUENCES keyword (see page 305) of the DEFINE-DEVICE configuration record for the printer to YES.
- Call the P$EnableEscapeSequences (on page 483) printer control subprogram after the printer is opened.

Table 53 lists the available printing functions and their escape sequences and options. The “Options” column of the table describes the permissible values for certain variables in an escape sequence that is represented by the characters #, ##, or ###. These variables are always binary values; that is, they are not ASCII characters representing the values.

**Table 53: RM/COBOL-Specific Escape Sequences**

| Function | ASCII | Decimal | Hex | Options (#, ##, or ###) |
|---|---|---|---|---|
| Reset | <Esc>E | 027 069 | 1B 45 | Clears the margins, clears line spacing, clears line extend mode, resets text length, sets page orientation back to portrait, sets paper source back to DMBIN_ONLYONE, and sets the font back to normal font. |
| Left Margin | <Esc>&a#L | 027 038 097 ### 076 | 1B 26 61 ## 4C | Size of left margin, in characters (calculated using current font). |
| Set Tab Stops | <Esc>&k#H | 027 038 107 ### 072 | 1B 26 6B ## 48 | Horizontal increment in characters (calculated using current font). |
| Line Spacing | <Esc>&l#D | 027 038 108 ### 068 | 1B 26 6C ## 44 | Lines per inch. |
| Top Margin | <Esc>&l#E | 027 038 108 ### 069 | 1B 26 6C ## 45 | Size of top margin, in lines (calculated using current font). |
| Text Length | <Esc>&l#F | 027 038 108 ### 070 | 1B 26 6C ## 46 | Length of page, in lines (calculated using current font). |
| Paper Source | <Esc>&l#H | 027 038 108 ### 072 | 1B 26 6C ## 48 | 1 - DMBIN_ONLYONE<br>2 - DMBIN_LOWER<br>3 - DMBIN_MIDDLE<br>4 - DMBIN_MANUAL<br>5 - DMBIN_ENVELOPE<br>6 - DMBIN_ENVMANUAL<br>7 - DMBIN_AUTO<br>8 - DMBIN_TRACTOR<br>9 - DMBIN_SMALLFMT<br>10 - DMBIN_LARGEFMT<br>11 - DMBIN_LARGECAPACITY<br>14 - DMBIN_CASSETTE<br>15 - DMBIN_FORMSOURCE |
| Orientation | <Esc>&l#O | 027 038 108 ### 079 | 1B 26 6c ## 4F | 0 - Portrait<br>1 - Landscape |
| Print Pitch | <Esc>(s#H | 027 040 115 ### 072 | 1B 28 73 ## 48 | Characters per horizontal inch. |
| Style | <Esc>(s#S | 027 040 115 ### 083 | 1B 28 73 ## 53 | 0 - Normal<br>1 - Italic<br>2 - Bold<br>4 - Underline<br>8 - Compressed<br>16 - Expanded |
| Font Height | <Esc>(s#V | 027 040 115 ### 086 | 1B 28 73 ## 56 | Characters per vertical inch. |
| Clear Margins | <Esc>9 | 027 057 | 1B 39 | Clear left and top margin. |
| Null | <NUL> | 000 | 00 | Seethe **Note** following this table. |
| Horizontal Tab | <HT> | 009 | 09 | Position to next tab stop. |
| Shift Out | <SO> | 014 | 0E | Shifts normal font to expanded font. Shifts compressed font to normal. Stays in effect until start of new line. |
| Shift In | <SI> | 015 | 0F | Shifts normal font to compressed font. Shifts expanded font to normal font. Stays in effect until start of new line. |

**Note** RM/COBOL normally changes nulls to spaces before sending them to the printer. This behavior may be changed by setting the value of the Printer Enable Null Esc. Seq. property (on page 80) to True.

# Appendix F: Subprogram Library

This appendix describes the subprograms that are supplied with the RM/COBOL runtime system. It also describes the required RM/COBOL calling sequence and the USING list parameters. Failure to comply with the USING list requirements will halt the run unit with a STOP RUN indication at the line containing the incorrect CALL statement.

**Note** Subprogram names are case-insensitive. For readability, mixed case is used in this document when subprogram names are lengthy. Uppercase letters are used for short subprogram names, in calling sequences, and in code fragments.

## Subprogram Library

Table 54 lists the subprograms alphabetically and gives a brief description.

**Table 54: RM/COBOL Subprogram Library**

| Subprogram | Function |
|---|---|
| C$Bitmap (on page 530) | Sets the name of a bitmap file for a program. |
| C$BTRV (on page 530) | Calls Btrieve directly from an RM/COBOL program. |
| C$CARG (on page 532) | Returns information about a passed argument given the argument name. |
| C$Century (on page 534) | Facilitates updating RM/COBOL programs to handle the year 2000. |
| C$ClearDevelopmentMode (on page 534) | Disables expanded error information reporting (known as “development mode”) for many of the C$ and P$ subprograms. |
| C$CompilePattern (on page 535) | Compiles a runtime-specified pattern regular expression into a compiled pattern buffer. |
| C$ConvertAnsiToOem (on page 536) | Converts ANSI characters to OEM characters. |
| C$ConvertOemToAnsi (on page 537) | Converts OEM characters to ANSI characters. |

**Table 54: RM/COBOL Subprogram Library (*Cont.*)**

| Subprogram | Function |
|---|---|
| C$DARG (on page 537) | Returns information about a passed argument given the argument number. |
| C$Delay (on page 538) | Relinquishes the CPU for a specified length of time. |
| C$Forget (on page 539) | Marks an area of the runtime system's in-memory screen image as unknown. |
| C$GetEnv (on page 540) | Retrieves the value of an environment variable. |
| C$GetLastFileName (on page 540) | Retrieves the last file used in a COBOL I/O statement. |
| C$GetLastFileOp (on page 542) | Retrieves information about the last COBOL I/O operation. |
| C$GetNativeCharset (on page 541) | Retrieves information about the native character set in effect for the current run unit. |
| C$GetRMInfo (on page 543) | Retrieves RM/COBOL runtime system information. |
| C$GetSyn (on page 545) | Retrieves the value of an RM/COBOL synonym from the UNIX resource file or from the Windows registry. |
| C$GetSysInfo (on page 545) | Retrieves operating system information. |
| C$GUICFG (on page 547) | Changes the RM/COBOL graphical user interface (GUI) properties. |
| C$LogicalAnd (on page 548) | Performs a bitwise logical AND operation on two or more nonnumeric or numeric operands. |
| C$LogicalComplement (on page 548) | Performs a bitwise logical ones complement operation on a nonnumeric or numeric operand. |
| C$LogicalOr (on page 549) | Performs a bitwise logical inclusive OR operation on two or more nonnumeric or numeric operands. |
| C$LogicalShiftLeft (on page 550) | Performs a logical shift left operation on a nonnumeric or numeric operand. |
| C$LogicalShiftRight (on page 550) | Performs a logical shift right operation on a nonnumeric or numeric operand. |
| C$LogicalXor (on page 551) | Performs a bitwise logical exclusive OR operation on two or more nonnumeric or numeric operands. |
| C$MBar (on page 552) | Sets the menu bar for a program. |
| C$MemoryAllocate (on page 552) | Allocates dynamic memory. |
| C$MemoryDeallocate (on page 553) | Deallocates (frees) dynamic memory. |
| C$NARG (on page 554) | Returns the number of arguments passed to the called subprogram. |
| C$OSLockInfo (on page 554) | Returns the process ID of the process that performed the last lock operation. |
| C$PlaySound (on page 555) | Plays Windows predefined sound events or sound files. |
| C$RBMenu (on page 556) | Sets the pop-up menu for a program that is displayed when the right mouse button is pressed. |
| C$RERR (on page 557) | Returns expanded completion status for the last I/O operation. |

**Table 54: RM/COBOL Subprogram Library (*Cont.*)**

| Subprogram | Function |
|---|---|
| C$SBar (on page 559) | Sets the status bar for a program. |
| C$SCRD (on page 559) | Allows the contents of the screen to be read into an alphanumeric data item. |
| C$SCWR (on page 560) | Allows the quick display of a large amount of screen data containing various attributes. |
| C$SecureHash (on page 566) | Produces a 20-byte message digest from an input text string using the secure hash algorithm, SHA-1. |
| C$SetDevelopmentMode (on page 567) | Enables expanded error information reporting (known as “development mode”) for many of the C$ and P$ subprograms. |
| C$SetEnv (on page 568) | Sets the value of an environment variable. |
| C$SetSyn (on page 569) | Sets the value of an RM/COBOL synonym in the UNIX resource file or in the Windows registry. |
| C$Show (on page 569) | Sets the show state of the main RM/COBOL window. |
| C$ShowArgs (on page 571) | Displays the list of arguments used to call C$ShowArgs. |
| C$TBar (on page 572) | Sets the toolbar for a program. |
| C$TBarEn (on page 573) | Enables and disables buttons on the toolbar. |
| C$TBarSeq (on page 574) | Sets the bitmap sequence of buttons on the toolbar. |
| C$Title (on page 575) | Sets the title displayed for the RM/COBOL window. |
| C$WRU (on page 575) | Returns the location from which a subprogram was called. |
| DELETE (on page 576) | Deletes a file. |
| RENAME (on page 577) | Renames a file. |
| SYSTEM (on page 578) | Allows an arbitrary operating system command to be executed. |

# C$Bitmap

C$Bitmap is used to display a bitmap file on the RM/COBOL window.

To use this subprogram, the runtime system must be able to locate the **c$bitmap.dll** file. The following example illustrates how to add this DLL to the Runtime Command line with the L Option:

```
runcobol program-name L=C$Bitmap.DLL
```

Other Runtime Command Options can be used. See Chapter 7: *Running* (on page 177).

**Note** This subprogram is supported only under Windows.

### Calling Sequence

```
CALL "C$Bitmap" USING filename
```

*filename* is an alphanumeric data item that contains the name of an existing Windows bitmap (**.bmp**) file. The file will be located using the runtime system search path.

# C$BTRV

C$BTRV is used to call Btrieve directly from an RM/COBOL program.

To use this subprogram, the runtime system must be able to locate the **c$btrv.dll** file. In addition, Btrieve, available from Pervasive Software, must be installed on the computer.

**Note** This subprogram is supported only under Windows.

### Calling Sequence

```
CALL "C$BTRV" USING opcode, status, position-block,
      data-buffer, buffer-length, key-buffer, key-number
```

*opcode* is any unsigned numeric data item that contains the desired Btrieve operation code value.

*status* is any signed numeric data item that receives the Btrieve status code result of the operation.

*position-block* is a 128-byte alphanumeric data item that is used by many of the Btrieve operations and should not be modified by the COBOL program.

*data-buffer* is an alphanumeric data item that contains the data associated with the Btrieve operation. The contents may be input and/or output depending upon the given Btrieve operation.

*buffer-length* is any unsigned numeric data item that contains the length of the data in the data buffer. This value may be input and/or output depending upon the given Btrieve operation.

*key-buffer* is a 256-byte alphanumeric data item that contains the key value associated with the Btrieve operation. The contents may be input and/or output depending upon the given Btrieve operation.

*key-number* is any signed numeric data item into which the key number or special option value is stored depending upon the given Btrieve operation.

**Note** This subprogram uses the CodeBridge parameter conversion facility for the four numeric arguments to allow maximum flexibility in their definition. See the *CodeBridge* manual for more details.

### Examples

The following is a typical DATA DIVISION description of the arguments (the data buffer item can be any size that is appropriate for the operation being performed):

```
01  B-OPCODE            PIC 9(5).
01  B-STATUS            PIC S9(5).
01  B-POSITION-BLOCK    PIC X(128).
01  B-DATA-BUFFER       PIC X(10000).
01  B-BUFFER-LENGTH     PIC 9(5).
01  B-KEY-BUFFER        PIC X(256).
01  B-KEY-NUMBER        PIC S9(3).
```

The following is a typical PROCEDURE DIVISION call of C$BTRV using the arguments described above:

```
CALL "C$BTRV" USING B-OPCODE, B-STATUS, B-POSITION-BLOCK,
                    B-DATA-BUFFER, B-BUFFER-LENGTH,
                    B-KEY-BUFFER, B-KEY-NUMBER.
```

Refer to your *Btrieve Programmer's Guide* for a complete description of the parameters sent from and returned to the application for each Btrieve operation.

The following specific example, using the Btrieve Version (26) operation, also will work:

```
01  B-OP-VERSION        PIC 99 VALUE 26.
01  B-STATUS            PIC S9(5) SIGN LEADING SEPARATE.
01  B-POSITION-BLOCK    PIC X(128).
01  B-VERSION-BUFFER    PIC X(30).
01  B-VER-BUF-LEN       PIC 99 VALUE 30.
01  B-KEY-BUFFER        PIC X(256)
                        VALUE "\\RM1\VOL2\USERS\BTRV.FIL".
01  B-KEY-NUMBER        PIC S9 VALUE 0.

CALL "C$BTRV" USING B-OP-VERSION, B-STATUS, B-POSITION-BLOCK,
                    B-VERSION-BUFFER, B-VER-BUF-LEN,
                    B-KEY-BUFFER, B-KEY-NUMBER.
```

# C$CARG

C$CARG returns information about the actual parameter that corresponds to a formal parameter in the USING or GIVING phrases in the Procedure Division header of a subprogram. This information identifies the type and length of the argument and, when the argument is numeric or numeric edited, the number of digits and scale factor for the argument.

### Calling Sequence

```
CALL "C$CARG" USING okay, argument-name,
      argument-description
```

*okay* is a one-character alphanumeric data item into which the ASCII character Y is stored if C$CARG successfully identifies the argument named by *argument-name*; otherwise, the ASCII character N is stored in the data item.

*argument-name* is the name of a Linkage Section data item named in the Procedure Division header USING list.

*argument-description* is a ten-character group data item into which the desired information about the argument specified by *argument-name* is stored. A typical data description for *argument-description* is as follows:

```
01 ARGUMENT-DESCRIPTION      BINARY(2).
   02 ARGUMENT-TYPE          PIC  9(2).
   02 ARGUMENT-LENGTH        PIC  9(8) BINARY(4).
   02 ARGUMENT-DIGIT-COUNT   PIC  9(2).
   02 ARGUMENT-SCALE         PIC S9(2).
```

**Note** The *argument-description* group item will only have the correct length if ARGUMENT-TYPE, ARGUMENT-DIGIT-COUNT, and ARGUMENT-SCALE are allocated as two-byte binary and ARGUMENT-LENGTH is allocated as four-byte binary. Use of the BINARY-ALLOCATION keyword (see page 288) of the COMPILER-OPTIONS configuration record can change the allocation of binary numeric data items such that this requirement is not met. For example, if BINARY-ALLOCATION=RM1, the default allocation for a data item described with PIC 9(2) is one byte. The example shown specifies a binary allocation override for each binary item to guarantee the right allocation regardless of the configured binary allocation scheme. The binary allocation override is not necessary if BINARY-ALLOCATION=RM. The binary allocation override would also not be necessary for binary allocation schemes RM1 and MF-RM if all three 9(2) entries were changed to 9(3) entries.

A description of the values that may be stored in ARGUMENT-TYPE is found in Table 55. The number of character positions occupied by the argument is stored in ARGUMENT-LENGTH. If ARGUMENT-TYPE indicates that the argument is a numeric or a numeric edited data item, the number of digits defined in the PICTURE character-string for the data item is stored in ARGUMENT-DIGIT-COUNT and the power of 10 scale factor (that is, the position of the implied decimal point) is stored in ARGUMENT-SCALE; otherwise, these fields contain the value zero. The digit count for a numeric or numeric edited data item does not include any positions defined by the PICTURE symbol P, which represents a scaling position.

**Note** If a calling program passes a called program two or more arguments that begin at the same location (either through redefinition, with reference modification, or because one is a group that contains the other), when the called program asks C$CARG for the parameter descriptions, it always receives that of the first actual argument passed that has the same location, regardless of the name specified in *argument-name*. In such cases, the C$DARG (on page 537) subprogram may be used to obtain the distinct descriptions by using *argument-number*.

**Table 55: RM/COBOL Data Types as Numbers**

| Type Number | RM/COBOL Data Type | Type Number | RM/COBOL Data Type |
|---|---|---|---|
| 0 | NSE | 16 | ANS |
| 1 | NSU | 17 | ANS (justified right) |
| 2 | NTS | 18 | ABS |
| 3 | NTC | 19 | ABS (justified right) |
| 4 | NLS | 20 | ANSE |
| 5 | NLC | 21 | ABSE |
| 6 | NCS | 22 | GRP (fixed length) |
| 7 | NCU | 23 | GRPV (variable length) |
| 8 | NPP | 25 | PTR |
| 9 | NPS | 26 | NBSN |
| 10 | NPU | 27 | NBUN |
| 11 | NBS | 32 | OMITTED |
| 12 | NBU | | |

**Note 1** For an explanation of the data type abbreviations and a description of the RM/COBOL data types listed in Table 55, see Table 28: Valid Data Types (on page 248) and Appendix C: *Internal Data Formats* (on page 403).

**Note 2** The data type GRPV (23) does not occur when C$CARG is called with the formal argument name or when C$DARG is called with an actual argument number that corresponds to an argument that is a variable-length group. In all cases, RM/COBOL passes variable-length group actual arguments as if they were a fixed-length group of the maximum length.

# C$Century

C$Century facilitates updating RM/COBOL programs to handle the year 2000 issue. It retrieves the first two digits of the current year. For example, for the year 1999, it will return 19; for the year 2000, it will return 20. For more information on obtaining composite date and time values, see Composite Date and Time (on page 218).

**Note** A number of changes have been made to the Format 2 ACCEPT statement related to improving the way dates and times are handled. These changes provide additional ways of writing Y2K-compliant COBOL. New phrases include CENTURY-DATE, CENTURY-DAY, DATE-AND-TIME, and DAY-AND-TIME. See the *RM/COBOL Language Reference Manual* for more information.

### Calling Sequence

```
CALL "C$Century" USING value-buffer
```

*value-buffer* is a two-byte data item with a format of either unsigned numeric display (NSU) or alphanumeric display (ANS).

# C$ClearDevelopmentMode

C$ClearDevelopmentMode is used to disable expanded error information reporting (known as “development mode”) for many of the C$ and P$ subprograms. The P$ subprograms are discussed in Appendix E: *Windows Printing* (on page 447). When development mode is enabled, as described in C$SetDevelopmentMode (on page 567), more verbose error reporting is performed to assist the COBOL developer in implementing these subprogram calls.

### Calling Sequence

```
CALL "C$ClearDevelopmentMode"
```

# C$CompilePattern

C$CompilePattern compiles a runtime-specified pattern regular expression into a compiled pattern buffer. The use of C$CompilePattern enhances performance when a variable pattern is used multiple times, since the effort to compile the regular expression is significant. It is not necessary to use C$CompilePattern for literal patterns because the RM/COBOL compiler automatically compiles literal patterns used in the LIKE condition when the source program is compiled.

### Calling Sequence

```
CALL "C$CompilePattern" [USING PatternString, PatternStripSpaces,
      [, PatternErrCode [, PatternErrPos [, PatternErrsyntaxPos]]]
      GIVING PatternPointer
```

*PatternString* (input) must refer to an alphanumeric data item, the value of which is the regular expression to be compiled. The *RM/COBOL Language Reference Manual*, in the discussion of the LIKE condition, specifies how pattern regular expressions are written.

*PatternStripSpaces* (input) must refer to a numeric integer data item. When the value of this argument is non-zero, trailing spaces in the value of *PatternString* are stripped before the regular expression is compiled. The value of this data item should be zero if the regular expression contains trailing spaces that are part of the pattern to be matched.

*PatternErrCode* (output) must refer to a numeric integer data item. This argument is optional, but the placeholder OMITTED must be specified if the argument is omitted when either *PatternErrPos* or *PatternErrSyntaxPos* are specified. When the argument is provided, the status of the pattern compilation is stored in the referenced data item. The value 0 indicates success. The values 1 through 26 correspond to RM/COBOL compiler error messages 682 through 707 and have the same meanings, respectively.

*PatternErrPos* (output) must refer to a numeric integer data item. This argument is optional, but the placeholder OMITTED must be specified if the argument is omitted when *PatternErrSyntaxPos* is specified. When the argument is provided, the character position (one-relative) within the pattern where an error was detected is stored in the referenced data item. The value zero is stored if no error occurred.

*PatternErrSyntaxPos* (output) must refer to a numeric integer data item. This argument is optional. When the argument is provided, the character position (one-relative) of the syntax structure within the pattern that is associated with the error is stored in the referenced data item. The value zero is stored if no error occurred. For many errors, this argument will have the same value as *PatternErrPos*. The value is different when the error is associated with the syntax of an escape sequence, character class expression, character range, class subtraction, quantifier, or parenthesized subexpression. For example, in the case of a missing closing parenthesis, this argument would indicate the offset of the corresponding opening parenthesis and *PatternErrPos* would indicate where the missing parenthesis was detected (in this case, the end of the pattern).

*PatternPointer* must refer to a pointer data item. The C$CompilePattern subprogram returns the pointer to the successfully compiled pattern in this data item. This data item may then be used as a pattern specifier for the LIKE condition. If the memory allocation fails because of insufficient memory then a null pointer is returned. If the pattern compilation fails because of a syntax error, the allocated memory is deallocated and a null pointer is returned.

Each call to C$CompilePattern allocates a memory buffer to contain the compiled pattern as if C$MemoryAllocate (on page 552) were called. The pointer to this allocated buffer is stored in *PatternPointer* without regard to the previous value of *PatternPointer*. When the compiled pattern is no longer needed, the program should call C$MemoryDeallocate (on page 553) using *PatternPointer* to deallocate this buffer. In particular, if multiple patterns are compiled using this routine and the same pointer data item for *PatternPointer*, the previously allocated pattern buffer should be deallocated prior to compiling a second or later pattern since the existing compiled pattern buffer will no longer be accessible by the COBOL program unless the pointer value has been copied to another pointer data item. Regardless of the advisability of deallocating no longer needed compiled pattern buffers, the runtime will free any such buffers upon termination of the run unit when these buffers are not explicitly deallocated by the COBOL program.

# C$ConvertAnsiToOem

C$ConvertAnsiToOem is used to convert a buffer containing ANSI characters to a buffer containing the corresponding OEM characters. The runtime's euro character processing is used. See the information on euro support in INTERNATIONALIZATION Configuration Record (on page 309).

**Note** This subprogram is supported only under Windows.

### Calling Sequence

```
CALL "C$ConvertAnsiToOem" USING ansi-buffer, oem-buffer
      [, char-count]
```

*ansi-buffer* is an alphanumeric data item that contains the ANSI characters to be converted to OEM characters.

*oem-buffer* is an alphanumeric data item into which the OEM characters will be stored.

*char-count* is an optional numeric data item that contains the number of characters to be converted. If omitted or if the value is invalid, the actual size of the shorter of *ansi-buffer* and *oem-buffer* is used.

# C$ConvertOemToAnsi

C$ConvertOemToAnsi is used to convert a buffer containing OEM characters to a buffer containing the corresponding ANSI characters. The runtime's euro character processing is used. See the information on euro support in INTERNATIONALIZATION Configuration Record (on page 309).

**Note** This subprogram is supported only under Windows.

### Calling Sequence

```
CALL "C$ConvertOemToAnsi" USING oem-buffer, ansi-buffer
      [, char-count]
```

*oem-buffer* is an alphanumeric data item that contains the OEM characters to be converted to ANSI characters.

*ansi-buffer* is an alphanumeric data item into which the ANSI characters will be stored.

*char-count* is an optional numeric data item that contains the number of characters to be converted. If omitted or if the value is invalid, the actual size of the shorter of *oem-buffer* and *ansi-buffer* is used.

# C$DARG

C$DARG returns information about an actual parameter passed in the USING or GIVING phrases in the CALL statement that called a subprogram. This information identifies the type and length of the argument and, when the argument is numeric or numeric edited, the number of digits and scale factor for the argument.

### Calling Sequence

```
CALL "C$DARG" USING argument-number,
      argument-description
```

*argument-number* is the one-relative ordinal position of the actual argument in the USING phrase of the CALL statement used to call the subprogram that calls C$DARG. The value zero obtains the description of the actual argument in the GIVING phrase of that CALL statement. If the value specified is less than zero or greater than the number of actual arguments passed, an argument-description for an omitted argument will be returned (ARGUMENT-TYPE = 32). The actual number of arguments passed can be obtained with the C$NARG (on page 554) subprogram.

*argument-description* is a ten-character group data item into which the desired information about the argument specified by *argument-number* is stored. A typical data description entry for *argument-description* is as follows:

```
01 ARGUMENT-DESCRIPTION      BINARY(2).
   02 ARGUMENT-TYPE          PIC  9(2).
   02 ARGUMENT-LENGTH        PIC  9(8) BINARY(4).
   02 ARGUMENT-DIGIT-COUNT   PIC  9(2).
   02 ARGUMENT-SCALE         PIC S9(2).
```

**Note** The *argument-description* group item will have the correct length only if ARGUMENT-TYPE, ARGUMENT-DIGIT-COUNT, and ARGUMENT-SCALE are allocated as two-byte binary and ARGUMENT-LENGTH is allocated as four-byte binary. Use of the BINARY-ALLOCATION keyword (see page 288) of the COMPILER-OPTIONS configuration record can change the allocation of binary numeric data items such that this requirement is not met. For example, if BINARY-ALLOCATION=RM1, the default allocation for a data item described with PIC 9(2) is one byte. The example shown specifies a binary allocation override for each binary item to guarantee the right allocation regardless of the configured binary allocation scheme. The binary allocation override is not necessary if BINARY-ALLOCATION=RM. The binary allocation override would also not be necessary for binary allocation schemes RM1 and MF-RM if all three 9(2) entries were changed to 9(3) entries.

A description of the values that may be stored in ARGUMENT-TYPE is found in Table 55 on page 533. The number of character positions occupied by the argument is stored in ARGUMENT-LENGTH. If ARGUMENT-TYPE indicates that the argument is a numeric or numeric edited data item, the number of digits defined in the PICTURE character-string for the data item is stored in ARGUMENT-DIGIT-COUNT and the power of 10 scale factor (that is, the position of the implied decimal point) is stored in ARGUMENT-SCALE; otherwise, these fields contain the value zero. The digit count for a numeric or numeric edited data item does not include any positions defined by the PICTURE symbol P, which represent a scaling position.

# C$Delay

C$Delay is used to relinquish the CPU for a length of time specified in seconds. Calling C$Delay will allow other programs to run while the original program waits. The amount of delay is not exact. It depends upon the particular machine configuration and the load on the machine.

### Calling Sequence

```
CALL "C$Delay" USING seconds
```

*seconds* is a PICTURE 9(*n*) BINARY, where *n* can be a digit from 1 to 9. It specifies the length of time, in seconds, to delay. Delays longer than one day are not normally meaningful and should not be used.

# C$Forget

C$Forget marks an area of the runtime system's in-memory screen image as unknown. The next COBOL output to the unknown area will not be optimized based on the screen contents. This allows COBOL output to be displayed correctly over output produced by a C subprogram in an optional support module. Output from C subprograms is not stored in the in-memory screen image.

**Note** This subprogram is supported only under UNIX.

### Calling Sequence

```
CALL "C$Forget" [USING upper-left-line, upper-left-position,
      lower-right-line, lower-right-position]
```

*upper-left-line* and *upper-left-position* are optional, three-digit, COMP-4 (BINARY) data items that describe the upper-left corner of the area of the screen to be marked as unknown. Valid values range from 0 to the limit of the screen line or position.

*lower-right-line* and *lower-right-position* are optional, three-digit, COMP-4 (BINARY) data items that describe the lower-right corner of the area of the screen to be marked as unknown. Valid values range from 0 to the limit of the screen line or position.

If you call C$Forget with no parameters, the entire screen is marked as unknown. The same result can be achieved by passing parameters with a value of 0. When a pop-up window is displayed and then removed over an unknown area, the original screen contents are replaced with spaces, except for the pop-up window borders and titles, which are not stored in the in-memory screen image.

C subprograms contained in optional support modules also can use C$Forget by calling **RmForget()** with four int parameters specifying the area to be marked as unknown. (See "Runtime Functions for Support Modules" in Appendix H: *Non-COBOL Subprogram Internals for UNIX* of the *CodeBridge* manual.

# C$GetEnv

C$GetEnv is used to retrieve the value of an environment variable. On UNIX, environment variable names are case-sensitive. On Windows, environment variable names are not case-sensitive.

### Calling Sequence

```
CALL "C$GetEnv" USING name, value [, return]
```

*name* is an alphanumeric data item that contains the name of the environment variable to retrieve.

*value* is an alphanumeric data item that contains the value of the environment variable upon return from the call.

*return* is a PICTURE 9(*n*) BINARY, where *n* can be a digit from 1 to 9. The value returned is zero for success and non-zero for failure.

# C$GetLastFileName

C$GetLastFileName is used to retrieve the last file-name and pathname used in a COBOL I-O statement (including OPEN and CLOSE).

### Calling Sequence

```
CALL "C$GetLastFilename" USING filename [, pathname]
```

*filename* is an alphanumeric data item that will contain the COBOL file-name specified in the most recently executed I-O statement. For REWRITE and WRITE statements, the COBOL file-name associated with the specified file record-name is provided. The alphanumeric data item should be 30 characters long. If the COBOL file-name is longer than the length of the data item, it will be truncated on the right.

*pathname* is an alphanumeric data item that will contain the name of the last used pathname (complete file-name). The value SPACES indicates that no name is available (the last file used in an I/O operation was probably closed or no file has been opened). The alphanumeric data item should be at least 64 characters long. If the complete file access name is longer than the length of the data item, it will be truncated on the right.

# C$GetNativeCharset

C$GetNativeCharset is used to retrieve information about the native character set in effect for the current run unit. The native character set specifies how nonnumeric data is encoded in memory and on data files.

The native character set for a run unit on Windows can be either ANSI or OEM. Codepages on Windows (on page 97) explains how this affects the run unit.

The native character set for a run unit on UNIX is determined by the locale settings for the system. Since the runtime system does no implied conversions on UNIX, these settings are not currently needed by an RM/COBOL program. However, the C$GetNativeCharset subprogram may be called on UNIX with the results as described for its calling sequence as shown below.

### Calling Sequence

```
CALL "C$GetNativeCharset" USING charset-name [, codepage-number]
```

*charset-name* must refer to an alphanumeric data item. The specified data item will contain the name of the character set in use for the current run unit after the call. For Windows, the name will have a value of “ANSI” or “OEM”. On UNIX, the value will be “NONE”.

*codepage-number*, if specified, must refer to a numeric data item. If specified, the referenced data item will contain the codepage number of the character set in use for the current run unit after the call. For Windows, the codepage number will be the system ANSI codepage number if *charset-name* contains “ANSI” and will be the system OEM codepage number if *charset-name* contains “OEM”. On UNIX, the value will be 0.

# C$GetLastFileOp

C$GetLastFileOp is used to retrieve information about the last COBOL I/O operation performed. The function returns the COBOL operation performed, and the line number and the intraline number where the operation was done. The intended use of this library subprogram is within a declarative procedure after an I/O error has occurred.

### Calling Sequence

```
CALL "C$GetLastFileOp" USING operation
      [, prog-line [, prog-intraline]]
```

*operation* is an alphanumeric data item that will contain the name of the last COBOL I/O operation performed (see the list below for possible values). The value SPACES indicates that no operation is available (the file was probably never opened). The alphanumeric data item should be 20 characters long. If the operation value is longer than the length of the data item, it will be truncated on the right.

- "Close"
- "CloseUnit"
- "Delete"
- "DeleteFile"
- "DeleteRandom"
- "Open"
- "ReadNext"
- "ReadPrevious"
- "ReadRandom"
- "Rewrite"
- "RewriteRandom"
- "Start"
- "Unlock"
- "Write"
- "WriteRandom"

*prog-line* is a BINARY data item with the picture PIC 9(6), which will contain the line number of the most recent COBOL I/O operation. If the program that contains the I/O operation was compiled with the Q Compile Command Option (see page 148), the value stored in *prog-line* is the segment offset of the statement. Use the values from the Debug heading of the program listing to locate the statement.

*prog-intraline* is a BINARY data item with the picture PIC 9(2), which will contain the intraline number of the I/O operation. For more information, see Debug Values (on page 247) and Line and Intraline Numbers (on page 247). If the program that contains the I/O operation was compiled with the Q Option, the value stored in *prog-intraline* is zero.

**Note** The PROGRAM-ID special register may be used to obtain the program-name of the COBOL program that performed the COBOL I/O operation. For more information about the PROGRAM-ID special register, see Chapter 1: *Language Structure* of the *RM/COBOL Language Reference Manual*.

# C$GetRMInfo

C$GetRMInfo is used to retrieve information about the RM/COBOL runtime system.

### Calling Sequence

```
CALL "C$GetRMInfo" USING RMInfoGroup
```

*RMInfoGroup* is a group data item as defined in the following copy file, RMINFO.CPY. The RMINFO.CPY copy file is supplied with an RM/COBOL development system.

```
*
*    RM/COBOL Information Definitions
*
  01 RMInformation.
     02  RM-BinaryVersionNumber.
         03  RM-MajorVersion          Picture 9(5) Binary(2).
         03  RM-MinorVersion          Picture 9(5) Binary(2).
         03  RM-PointVersion          Picture 9(5) Binary(2).
     02  RM-VersionNumber             Picture X(8).
     02  RM-RegistrationNumber.
         03  RM-ProductCode           Picture XX.
         03  RM-Filler                Picture X.
         03  RM-ADRNumber             Picture 9(4).
         03  RM-Filler                Picture X.
         03  RM-SerialNumber          Picture 9(5).
         03  RM-Filler                Picture X.
         03  RM-UseCount              Picture 9(4).
     02  RM-UseCountLimit             Picture 9(5) Binary(2).
     02  RM-UseCountInUse             Picture 9(5) Binary(2).
     02  RM-MainProgramName           Picture X(30).
     02  RM-ApplRegistrationNumber    Picture X(16).
     02  RM-BuildPlatform             Picture X(80).
     02  RM-InterfaceValue            Picture 9 Binary(2).
         88  RM-InterfaceIsTermcap      Value 1.
         88  RM-InterfaceIsTerminfo     Value 2.
         88  RM-InterfaceIsVMS          Value 3.
         88  RM-InterfaceIsDOS          Value 4.
         88  RM-InterfaceIsBIOS         Value 5.
         88  RM-InterfaceIsRAM          Value 6.
         88  RM-InterfaceIsJBIOS        Value 7.
         88  RM-InterfaceIsJRAM         Value 8.
         88  RM-InterfaceIsGUI          Value 9.
     02  RM-TerminalInterface         Picture X(10).
     02  RM-ConfigurationValue        Picture X.
         88  RM-ConfigurationIsBuiltIn  Value 'Y'
                                   When False 'N'.
     02  RM-WindowMangerValue         Picture X.
         88  RM-WindowManagerIsPresent  Value 'Y'
```

```
                                When False 'N'.
02  RM-EnterpriseCodeBnchValue    Picture X.
    88  RM-EnterpriseCodeBnchIsPresent Value 'Y'
                                   When False 'N'.
02  RM-VGIBValue                  Picture X.
    88  RM-VGIBIsPresent            Value 'Y'
                                When False 'N'.
02  RM-INFOXValue                 Picture X.
    88  RM-INFOXIsPresent           Value 'Y'
                                When False 'N'.
02  RM-plusDBValue                Picture X.
    88  RM-plusDBIsPresent          Value 'Y'
                                When False 'N'.
02  RM-MCBAValue                  Picture X.
    88  RM-MCBAIsPresent            Value 'Y'
                                When False 'N'.
02  RM-FlexGenValue               Picture X.
    88  RM-FlexGenIsPresent         Value 'Y'
                                When False 'N'.
02  RM-RPCValue                   Picture X.
    88  RM-RPCIsPresent             Value 'Y'
                                When False 'N'.
02  RM-CGIXValue                  Picture X.
    88  RM-CGIXIsPresent            Value 'Y'
                                When False 'N'.
02  RM-OFMValue                   Picture X.
    88  RM-OFMIsPresent             Value 'Y'
                                When False 'N'.
02  RM-Reserved                   Picture X(21).
02  RM-LicenseProduct             Picture X(80).
02  RM-LicenseIssuedBy            Picture X(80).
02  RM-LicenseIssuedTo            Picture X(80).
02  RM-LicenseType                Picture X(20).
02  RM-LicenseValidThru           Picture X(20).
```

# C$GetSyn

C$GetSyn is used to retrieve a value of an RM/COBOL synonym from the UNIX resource file (on page 28) or from the Windows registry (on page 66).

### Calling Sequence

```
CALL "C$GetSyn" USING name, value, program
```

*name* is an alphanumeric data item that contains the name of the synonym to retrieve.

*value* is an alphanumeric data item that contains the value of the synonym upon return from the call.

*program* is an alphanumeric data item that indicates the name of the program whose synonym properties are being retrieved. A value of SPACES indicates the "Default Properties" in the Windows registry or the [Default Synonyms] section in the UNIX resource file.

# C$GetSysInfo

C$GetSysInfo is used to retrieve information about the operating system on which the RM/COBOL runtime system is running. Under UNIX, this information is retrieved directly from the operating system. Under Windows, information that is not available from the operating system is instead retrieved from the environment.

### Calling Sequence

```
CALL "C$GetSysInfo" USING SystemInfoGroup
```

*SystemInfoGroup* is a group data item as defined in the following copy file, SYSINFO.CPY. The SYSINFO.CPY is supplied with an RM/COBOL development system.

SYSINFO.CPY contains the following definitions.

```
*
*    System Information Definitions
*
  01 SystemInformation.
     02  Sys-Name                    Picture X(20).
     02  Sys-Version.
         03  Sys-MajorVersion        Picture 9(5) Binary(2).
         03  Sys-MinorVersion        Picture 9(5) Binary(2).
     02  Sys-NodeName                Picture X(20).
     02  Sys-Machine                 Picture X(20).
     02  Sys-UserName                Picture X(20).
```

```
02  Sys-UserID                  Picture 9(10) Binary(4).
02  Sys-GroupName               Picture X(20).
02  Sys-GroupID                 Picture 9(10) Binary(4).
02  Sys-StationName             Picture X(20).
02  Sys-ProcessID               Picture 9(10) Binary(4).
02  Sys-IsMultiUserValue        Picture X.
    88  Sys-IsMultiUser          Value 'Y' When False 'N'.
02  Sys-Reserved                Picture X(23).
```

| Value | Meaning |
|---|---|
| Sys-Name | Contains the name of the operating system. Under Windows, it may be “Windows 95”, “Windows 98”, “Windows Me”, “Windows NT”, “Windows 2000”, “Windows XP”, "Windows Server 2003", or “Windows Unknown”. Under UNIX, it is the name of the operating system. |
| Sys-Version | Contains the major and minor version number of the operating system. |
| Sys-NodeName | Contains the NETBIOS computer name established at system startup. |
| Sys-Machine | Contains the identity of the processor. It is set to “Intel 386”, “Intel 486”, “Intel Pentium”, “MIPS R4000”, “Alpha 21064”, or “Unknown *nnnnn*”, where *nnnnn* is the raw processor value returned by the operating system. |
| Sys-UserName | Contains the name of the user. This is obtained by querying the following three sources, in the following order:<br>1. The name of the currently logged-in user.<br>2. The NAME environment variable.<br>3. The USER environment variable.<br>If none of the above is set, the field is set to “USER”. |
| Sys-UserID | Contains the numeric ID assigned to this user. Under Windows, this item is set to the numeric contents of the USERID environment variable, or 0 if this variable is not set or is non-numeric. |
| Sys-GroupName | Contains the name of the group to which the user belongs. Under Windows, this item is set to the contents of the GROUP environment variable, or “GROUP” if this variable is not set. |
| Sys-GroupID | Contains the numeric ID assigned to the group to which this user belongs. Under Windows, this item is set to the numeric contents of the GROUPID environment variable, or 0 if this variable is not set or is non-numeric. |
| Sys-StationName | Contains the name of the terminal. Under Windows, this item is set to the contents of the STATION environment variable, or “CON” if not set. Under UNIX, this item is set to the name of the current tty. |
| Sys-ProcessID | Contains the numeric ID of the current process. |
| Sys-IsMultiUserValue | Indicates if this runtime is running in single- or in multi-user mode. This will normally be multi-user mode unless the FORCE-USER-MODE configuration option has been specified. |
| Sys-Reserved | Reserved for future expansion. |

# C$GUICFG

C$GUICFG is used to dynamically manipulate certain graphical user interface (GUI) settings. The settings changes are temporary until the next access of the Windows registry file. The registry entries remain unchanged. To use this subprogram, the runtime system must be able to locate the **c$guicfg.dll** file.

**Note** This subprogram is supported only under Windows.

### Calling Sequence

```
CALL "C$GUICFG" USING settingstr [, settingstr
      ...  ], exit-code
```

*settingstr* is an alphanumeric data item that contains the settings modification information. The settings are in the form of item=value where item is one of the properties listed below. The descriptions of the items and values are described in Setting Control Properties (on page 70).

- Auto Paste
- Auto Scale
- Enable Close
- Enable Properties Dialog
- Full OEM To ANSI Conversions
- Icon File
- Mark Alphanumeric
- Paste Termination
- Persistent
- Printer Dialog Always
- Printer Dialog Never
- Remove Trailing Blanks
- Screen Read Line Draw
- Sizing Priority
- Status Bar
- SYSTEM Window Type
- Toolbar
- Toolbar Prompt
- Update Timeout

In *settingstr*, the item identifiers from the above list may include or exclude the spaces between words and case is not significant. If the *value* is misspelled, ignoring case, a default value is used. For example, for True/False values, anything other than True is considered to be False.

*exit-code* must refer to a data item described as PICTURE 9(*n*) BINARY, where *n* can be a digit from 1 to 9 such that a two- or four-byte binary data item is referenced. The value returned in *exit-code* is zero for success and non-zero for failure.

# C$LogicalAnd

C$LogicalAnd is used to perform a bitwise logical AND operation on two or more nonnumeric or numeric operands.

### Calling Sequence

```
CALL "C$LogicalAnd"
   [GIVING Result]
   USING Operand1 {Operand2} ...
```

*Result*, if specified, must be an identifier that references a numeric data item.

*Operand1* may reference a nonnumeric or numeric data item. If nonnumeric, all the USING operands must reference nonnumeric data items. If numeric, all the USING operands must reference numeric data items.

*Operand2* must be nonnumeric if *Operand1* is nonnumeric and numeric otherwise. This is true for all iterations of *Operand2*, if any. If any nonnumeric *Operand2* is shorter than *Operand1*, it is assumed to be padded on the right with binary zeroes.

For nonnumeric USING operands, the bitwise logical AND of all the operands replaces the value of *Operand1*. The value of *Result* is set to a nonzero value if any character of *Operand1* is nonzero after the operation completes and zero otherwise.

For numeric USING operands, each operand is converted, if necessary, to a 32-bit binary integer. These 32-bit binary values are logically ANDed together. If the GIVING phrase is specified, the result of this operation is stored in *Result* and the value of *Operand1* is not modified. If the GIVING phrase is not specified, the result of this operation is stored in *Operand1*.

# C$LogicalComplement

C$LogicalComplement is used to perform a bitwise logical ones complement operation on a nonnumeric or numeric operand.

### Calling Sequence

```
CALL "C$LogicalComplement"
   [GIVING Result]
   USING Operand
```

*Result*, if specified, must be an identifier that references a numeric data item.

*Operand* may reference a nonnumeric or numeric data item.

If *Operand* refers to a nonnumeric data item, the bitwise logical ones complement of the value of *Operand* replaces the value of *Operand*. The value of *Result* is set to a

nonzero value if any character of *Operand* is nonzero after the operation completes and zero otherwise.

If *Operand* refers to a numeric data item, the operand is converted, if necessary, to a 32-bit binary integer. The 32-bit binary value is logically ones complemented. If the GIVING phrase is specified, the result of this operation is stored in *Result* and the value of *Operand* is not modified. If the GIVING phrase is not specified, the result of this operation is stored in *Operand*.

# C$LogicalOr

C$LogicalOr is used to perform a bitwise logical inclusive OR operation on two or more nonnumeric or numeric operands.

### Calling Sequence

```
CALL "C$LogicalOr"
   [GIVING Result]
   USING Operand1 {Operand2} ...
```

*Result*, if specified, must be an identifier that references a numeric data item.

*Operand1* may reference a nonnumeric or numeric data item. If nonnumeric, all the USING operands must reference nonnumeric data items. If numeric, all the USING operands must reference numeric data items.

*Operand2* must be nonnumeric if *Operand1* is nonnumeric and numeric otherwise. This is true for all iterations of *Operand2*, if any. If any nonnumeric *Operand2* is shorter than *Operand1*, it is assumed to be padded on the right with binary zeroes.

For nonnumeric USING operands, the bitwise logical inclusive OR of all the operands replaces the value of *Operand1*. The value of *Result* is set to a nonzero value if any character of *Operand1* is nonzero after the operation completes and zero otherwise.

For numeric USING operands, each operand is converted, if necessary, to a 32-bit binary integer. These 32-bit binary values are logically inclusive ORed together. If the GIVING phrase is specified, the result of this operation is stored in *Result* and the value of *Operand1* is not modified. If the GIVING phrase is not specified, the result of this operation is stored in *Operand1*.

# C$LogicalShiftLeft

C$LogicalShiftLeft is used to perform a logical shift left operation on a nonnumeric or numeric operand.

### Calling Sequence

```
CALL "C$LogicalShiftLeft"
   [GIVING Result]
   USING Operand [ShiftCount]
```

*Result*, if specified, must be an identifier that references a numeric data item.

*Operand* may reference a nonnumeric or numeric data item.

*ShiftCount*, if specified, must be an identifier that references a numeric data item. If *ShiftCount* is not specified, a shift count of 1 is assumed.

If *Operand* refers to a nonnumeric data item, the value of the data item is shifted left by the number of bit positions specified by *ShiftCount*. Any bits shifted off the left end are lost and zero valued bits are shifted into the right end. The value of *Result* is set to a nonzero value if any character of *Operand* is nonzero after the operation completes and zero otherwise.

If *Operand* refers to a numeric data item, the operand is converted, if necessary, to a 32-bit binary integer. The 32-bit binary value is logically shifted left by the number of bit positions specified by *ShiftCount*. If the GIVING phrase is specified, the result of this operation is stored in *Result* and the value of *Operand* is not modified. If the GIVING phrase is not specified, the result of this operation is stored in *Operand*.

# C$LogicalShiftRight

C$LogicalShiftRight is used to perform a logical shift right operation on a nonnumeric or numeric operand.

### Calling Sequence

```
CALL "C$LogicalShiftRight"
   [GIVING Result]
   USING Operand [ShiftCount]
```

*Result*, if specified, must be an identifier that references a numeric data item.

*Operand* may reference a nonnumeric or numeric data item.

*ShiftCount*, if specified, must be an identifier that references a numeric data item. If *ShiftCount* is not specified, a shift count of 1 is assumed.

If *Operand* refers to a nonnumeric data item, the value of the data item is shifted right by the number of bit positions specified by *ShiftCount*. Any bits shifted off the right end are lost and zero valued bits are shifted into the left end. The value of

*Result* is set to a nonzero value if any character of *Operand* is nonzero after the operation completes and zero otherwise.

If *Operand* refers to a numeric data item, the operand is converted, if necessary, to a 32-bit binary integer. The 32-bit binary value is logically shifted right by the number of bit positions specified by *ShiftCount*. If the GIVING phrase is specified, the result of this operation is stored in *Result* and the value of *Operand* is not modified. If the GIVING phrase is not specified, the result of this operation is stored in *Operand*.

# C$LogicalXor

C$LogicalXor is used to perform a bitwise logical exclusive OR operation on two or more nonnumeric or numeric operands.

### Calling Sequence

```
CALL "C$LogicalXor"
   [GIVING Result]
   USING Operand1 {Operand2} ...
```

*Result*, if specified, must be an identifier that references a numeric data item.

*Operand1* may reference a nonnumeric or numeric data item. If nonnumeric, all the USING operands must reference nonnumeric data items. If numeric, all the USING operands must reference numeric data items.

*Operand2* must be nonnumeric if *Operand1* is nonnumeric and numeric otherwise. This is true for all iterations of *Operand2*, if any. If any nonnumeric *Operand2* is shorter than *Operand1*, it is assumed to be padded on the right with binary zeroes.

For nonnumeric USING operands, the bitwise logical exclusive OR of all the operands replaces the value of *Operand1*. The value of *Result* is set to a nonzero value if any character of *Operand1* is nonzero after the operation completes and zero otherwise.

For numeric USING operands, each operand is converted, if necessary, to a 32-bit binary integer. These 32-bit binary values are logically exclusive ORed together. If the GIVING phrase is specified, the result of this operation is stored in *Result* and the value of *Operand1* is not modified. If the GIVING phrase is not specified, the result of this operation is stored in *Operand1*.

# C$MBar

C$MBar is used to display a menu bar in the RM/COBOL window.

To use this subprogram, the runtime system must be able to locate the **rmbars.dll** file. The following example illustrates how to add this DLL to the Runtime Command line with the L Option:

```
runcobol program-name L=RMBARS.DLL
```

Other Runtime Command Options can be used. See Chapter 7: *Running* (on page 177).

**Note** This subprogram is supported only under Windows.

### Calling Sequence

```
CALL "C$MBar" [USING  menustring [, menustring
      ...  ], exit-code]
```

*menustring* is an alphanumeric data item that contains the menu text, as described in Setting Menu Bar Properties (on page 91).

*exit-code* is a PICTURE 9(*n*) BINARY, where *n* can be a digit from 1 to 9. The *exit-code* parameter must be two- or four-byte binary and the value returned is zero for success and non-zero for failure.

**Note** Calling C$MBar with no arguments turns off the menu bar.

# C$MemoryAllocate

C$MemoryAllocate is used to allocate dynamic memory.

### Calling Sequence

```
CALL "C$MemoryAllocate" USING  memory-pointer,
      memory-size
```

*memory-pointer* must be a pointer data item (USAGE POINTER) that will contain the address of the allocated memory area upon successful completion of the call. A null pointer value is returned if the call is not successful.

*memory-size* must be a numeric data item that specifies the size of the area to allocate in bytes. The maximum value for *memory-size* is approximately 2,147,483,611, depending on the size of overhead structures. If the maximum value is exceeded, the allocation request will be unsuccessful and a null pointer will be returned. A null pointer will likely be returned for much smaller values since the operating system will be unable to satisfy the request. If the COBOL data item used to specify *memory-size* supports 10 or more decimal digits, a large value in *memory-size* may be truncated upon conversion internal to

C$MemoryAllocate. In this case, an area may be allocated that is smaller than the requested size or the run unit terminated with an error in C$MemoryAllocate.

If the memory allocation is successful, the allocated memory is initialized to spaces.

The allocated memory belongs to the run unit and may be accessed by any program in the run unit that has access to the pointer data item *memory-pointer* or a copy of that pointer data item. Upon termination of the run unit, all dynamically allocated memory will be freed.

The pointer returned by C$MemoryAllocate may be used to set the base address of a based linkage item in a Format 5 SET statement. The memory area may then be accessed by references to the based linkage item or data items subordinate to the based linkage item. References to based linkage items are slower than references to Working-Storage items, so if the program makes frequent references to the based linkage item (for example, in a PERFORM loop), it is a good idea to move the based linkage item to a Working-Storage item.

# C$MemoryDeallocate

C$MemoryDeallocate is used to deallocate (free) dynamic memory allocated by a previous call to C$MemoryAllocate.

### Calling Sequence

```
CALL "C$MemoryDeallocate" USING memory-pointer
```

*memory-pointer* must be a pointer data item (USAGE POINTER) that points to a memory area previously allocated by a call to C$MemoryAllocate. If the pointer does not point to such a memory area, the call does nothing. If the pointer does point to such a memory area and the memory is successfully freed, then the value of *memory-pointer* is set to a null pointer value.

After memory has been deallocated, the program should not make any references to based linkage items whose base address was set to the area of memory that was deallocated until those based linkage items are reassigned to a new valid base address in a Format 5 SET statement. If the program does reference the deallocated memory, an error may occur or undefined data may be accessed, either from the original memory area while it is still on the free memory list or after the memory has been reallocated for a different use. It is the programmer's responsibility to manage pointer data items and the setting of based linkage base addresses to avoid such conditions.

It is not necessary to call C$MemoryDeallocate. The runtime will free any dynamically allocated memory upon termination of the run unit.

# C$NARG

C$NARG returns the number of parameters passed in the CALL statement USING list to the subprogram that contains the call to C$NARG. The GIVING argument is not included in the count. Arguments specified explicitly as OMITTED in the USING list of the CALL statement are included in the count. An RM/COBOL subprogram may be called with a variable number of actual parameters if it does not attempt, during its execution, to reference formal parameters for which no actual parameters exist.

### Calling Sequence

```
CALL "C$NARG" USING parameter-count
```

*parameter-count* is a BINARY, COMPUTATIONAL-4 or COMPUTATIONAL-1 data item with the PICTURE 9(3) into which is stored the number of parameters in the USING list on the CALL statement that called the subprogram that called C$NARG.

**Note** The restriction on a reference to a formal argument for which no corresponding actual argument exists does not apply to a reference in an ADDRESS OF *identifier-1* special register. Thus, the calling program's omission of the GIVING argument or omission of an embedded USING argument (by use of the OMITTED keyword) can be detected by using IF ADDRESS OF *identifier-1* IS EQUAL TO NULL, where *identifier-1* refers to the appropriate formal argument. The restriction also does not apply to Linkage Section 01 or 77 data items specified in the USING or GIVING phrase of a CALL statement.

# C$OSLockInfo

C$OSLockInfo returns the process ID of the process that has the record locked when a lock request fails. This subprogram should be called immediately after a lock request has failed.

**Note** This subprogram is available only under UNIX.

### Calling Sequence

```
CALL "C$OSLockinfo" USING processid
```

*processid* is a four-byte, unsigned COMP-4 numeric item.

# C$PlaySound

C$PlaySound is used to play Windows predefined sound events or sound files, that is, files with the **.wav** extension.

**Note** This subprogram is supported only under Windows.

### Calling Sequence

```
CALL "C$PlaySound" USING sound [, flags]
```

*sound* is an alphanumeric data item that contains the name of a Windows sound event or **.wav** sound file to play.

*flags* is a numeric data item that contains flags to use when playing the sound. The possible values are provided below and in the 78-level entries in the copy file WINDEFS.CPY (on page 512), and may be combined by adding them together. If the *flags* parameter is omitted, the flag SoundSync is assumed.

| Value | Meaning |
|---|---|
| SoundSync | Synchronous playback of a sound event. |
| SoundAsync | Sound is played asynchronously. |
| SoundNoDefault | No default sound event is used. |
| SoundNoStop | The specified sound event will yield to another sound event that is already playing. |
| SoundPurge | Sounds are to be stopped for the calling task. |
| SoundApplication | The sound is played using an application-specific association. |
| SoundNoWait | If the driver is busy, return immediately without playing the sound. |
| SoundAlias | *sound* is a system-event alias in the Windows registry file or the win.ini file. |
| SoundFilename | *sound* is a filename. |
| SoundAliasId | *sound* is a predefined sound identifier. |

### Example

The following code fragment plays **chimes.wav**:

```
CALL "C$PlaySound" USING "Chimes.wav".
```

# C$RBMenu

C$RBMenu is used to display a pop-up menu in the RM/COBOL window when the right mouse button is pressed.

To use this subprogram, the runtime system must be able to locate the **rmbars.dll** file. The following example illustrates how to add this DLL to the Runtime Command line with the L Option:

```
runcobol program-name L=RMBARS.DLL
```

Other Runtime Command Options can be used. See Chapter 7: *Running* (on page 177).

**Note** This subprogram is supported only under Windows.

### Calling Sequence

```
CALL "C$RBMenu" [USING menustring [, menustring
      ...  ], exit-code]
```

*menustring* is an alphanumeric data item that contains the menu text, as described in Setting Pop-Up Menu Properties (on page 93).

*exit-code* is a PICTURE 9(*n*) BINARY, where *n* can be a digit from 1 to 9. The *exit-code* parameter must be two- or four-byte binary and the value returned is zero for success and non-zero for failure.

**Note** Calling C$RBMenu with no arguments turns off the pop-up menu.

# C$RERR

C$RERR returns the expanded I-O completion status , as shown in Input/Output Errors (on page 370). It returns either a four-character or an eleven-character extended status code depending upon the length of the data item specified in the USING phrase. This status is for the last attempted I/O operation. When the COBOL I-O status for the last operation is represented differently under ANSI COBOL 1985 and 1974, the value returned conforms to ANSI COBOL 1974 when the calling program is compiled in 1974 mode (that is, when the 7 Compile Command Option is specified, as described on page 149). The value returned conforms to ANSI COBOL 1985 when the calling program is compiled in 1985 mode.

### Calling Sequence

```
CALL "C$RERR" USING extended-status
```

*extended-status* is either a four-character or an eleven-character alphanumeric data item into which the expanded I/O completion status is stored in ASCII characters.

If *extended-status* is four characters in length, the first two character positions contain the same digits as would the file status data item on completion of the I/O operation. The last two character positions provide additional information about the file status. In cases where Appendix A: *Error Messages* shows only two digits for a status, the last two character positions will contain ASCII zeroes. Although most statuses contain only the decimal digits 0 through 9, note that the hexadecimal digits A through F are possible in some character positions (for example, 39,3A through 39,3F, as shown beginning on page 376).

If *extended-status* is eleven characters in length, the first two character positions (positions one and two) contain the same digits as would the file status data item on completion of the I/O operation. In cases where Appendix A shows only two digits for a status, the remaining nine character positions contain ASCII blanks. In cases where Appendix A shows four digits for a status, character position three contains an ASCII comma, character positions four and five contain the last two digits of the status, and the remaining six character positions contain ASCII blanks. For permanent errors, that is, when the first two digits are 30 as shown in Input/Output Errors (on page 370) and in the discussion of the 30, *OS* error *code* on page 373, character position three contains an ASCII comma, character positions four and five contain a two-digit OS code (see Table 56), character position six contains an ASCII comma, and character positions seven through eleven contain a five-digit, OS-specific error code. Although most statuses contain only the decimal digits 0 through 9, note that the hexadecimal digits A through F are possible in some character positions (for example, 39,3A through 39,3F as shown beginning on page 376).

**Table 56: Two-Digit OS Codes**

| Code | Description |
|---|---|
| 00 | Unknown OS error. |
| 01 | File Manager Detected error. |
| 02 | MS-DOS error. |
| 03 | OS/2 error. |
| 04 | UNIX error. |
| 05 | RM/COS error. |
| 06 | Btrieve error. |
| 07 | Informix error. |
| 08 | Oracle error. |
| 09 | AmigaDos error. |
| 10 | Open File Manager error. |
| 11 | C Library error. |
| 12 | MS-Windows error. |
| 13 | NetWare error. |
| 14 | VMS error. |
| 15 | RM/InfoExpress Server error. |
| 16 | RM/InfoExpress Client error. |
| 17 | RM/InfoExpress TLI error. |
| 18 | RM/InfoExpress TLISYS error. |
| 19 | RM/InfoExpress NetBIOS error. |
| 20 | RM/InfoExpress SPX error. |
| 21 | RM/InfoExpress WinSock error. |

## Examples

The following examples illustrate the difference between four-character and eleven-character expanded I-O completion status codes.

| Four-character | Eleven-character | Meaning |
|---|---|---|
| `0000` | `00` | Successful. |
| `1000` | `10` | At end. |
| `0405` | `04,05` | Record read shorter than minimum. |
| `395A` | `39,5A` | Key length mismatch (alt. key #10). |
| `3010` | `30,06,01002` | Btrieve error 1002 (memory). |
| `3010` | `30,21,10054` | Windows Sockets error 10054 (reset). |

# C$SBar

C$SBar is used to display a status bar in the RM/COBOL window.

To use this subprogram, the runtime system must be able to locate the **rmbars.dll** file. The following example illustrates how to add this DLL to the Runtime Command line with the L Option:

```
runcobol program-name L=RMBARS.DLL
```

Other Runtime Command Options can be used. See Chapter 7: *Running* (on page 177).

**Note** This subprogram is supported only under Windows.

### Calling Sequence

```
CALL "C$SBar" [USING status-text, exit-code]
```

*status-text* is an alphanumeric data item that contains the text to be displayed on the status line. For more information, see Status Bar Text property (on page 82).

*exit-code* is a PICTURE 9(*n*) BINARY, where *n* can be a digit from 1 to 9. The *exit-code* parameter must be two- or four-byte binary and the value returned is zero for success and non-zero for failure.

**Note** Calling C$SBar with no arguments turns off the status bar.

# C$SCRD

C$SCRD allows the contents of the screen to be read into an alphanumeric data item.

### Calling Sequence

```
CALL "C$SCRD" USING screen-buffer [, buffer-size [,
      screen-line [, screen-position]]]
```

*screen-buffer* is an alphanumeric data item that will receive the characters read from the terminal display screen.

*buffer-size* is an optional COMP-1 data item that specifies the number of characters to be read. If the value is 0 or the parameter is omitted, the actual size of *screen-buffer* is used.

*screen-line* is a COMP-1 data item that specifies the line where the cursor is to be placed prior to the screen read. If omitted, a value of 1 is used. If a pop-up window is active, *screen-line* is window-relative, not screen-relative.

*screen-position* is a COMP-1 data item that specifies the position where the cursor is to be placed prior to the screen read. If omitted, a value of 1 is used. If a pop-up window is active, *screen-position* is window-relative, not screen-relative.

**Note** The three optional arguments, *buffer-size*, *screen-line*, and *screen-position*, may be explicitly omitted by specifying the keyword OMITTED in the corresponding position in the USING list.

The cursor position after the call obeys the rules for the ACCEPT and DISPLAY statements. No errors are returned.

If this function is requested to read characters past the end of the screen or window, as many actual characters as possible are returned. The remainder of the buffer is set to spaces.

It is not possible to retrieve attribute information from the display. Only the actual character values are returned.

If line draw graphic characters have been written to the display using the GRAPHICS keyword of the CONTROL phrase of the ACCEPT and DISPLAY statements, and this call is used to read such characters, the characters returned are plus, hyphen, and vertical bar unless, on Windows, the Screen Read Line Draw property is set to True, in which case the characters returned are DOS line draw characters (for example, $D9, "┘", for lower-right corner). For more information, see the description of the GRAPHICS keyword of the CONTROL phrase and the line draw characters in Table 22: System-Specific Line Draw Characters on page 197.

# C$SCWR

C$SCWR allows a COBOL program to display quickly a large amount of information containing multiple display attributes.

### Calling Sequence

```
CALL "C$SCWR" USING display-description, text-characters
      [, attribute-codes, palette-table]
```

*display-description* is a required, 8- to 14-character group data item into which the location and size of the display are stored. The COBOL description is as follows:

```
01 DISPLAY-DESCRIPTION        USAGE BINARY (2).
   02 DISPLAY-VERSION         PIC  9(4) VALUE 0.
   02 DISPLAY-UNIT            PIC  9(4) VALUE 0.
   02 DISPLAY-LINE            PIC  9(4).
   02 DISPLAY-POSITION        PIC  9(4).
   02 DISPLAY-LENGTH          PIC  9(4).
   02 DISPLAY-EXCEPTION-CODE  PIC  9(4).
   02 DISPLAY-EXCEPTION-VALUE PIC  9(4).
```

DISPLAY-VERSION (required, input) is reserved for future use and must be set to a value of 0.

DISPLAY-UNIT (required, input) is the unit number of the terminal to which the display is directed. Under UNIX, specifying a value of 0 causes output to be written to the terminal from which the runtime system was started. Under Windows, this value must be set to 0.

DISPLAY-LINE (required, input) is the one-relative line number in the current window where the text is to be displayed. If set to a value of 0, the display begins on the current line (as described in the "Determining Line and Position" section of the DISPLAY statement in Chapter 6: *Procedure Division Statements* of the *RM/COBOL Language Reference Manual*).

DISPLAY-POSITION (required, input) is the one-relative column number in the current window where the text is to be displayed. If set to a value of 0, the display begins at the current column (as described in the "Determining Line and Position" of the DISPLAY statement in Chapter 6: *Procedure Division Statements* of the *RM/COBOL Language Reference Manual*).

DISPLAY-LENGTH (optional, input) is the number of characters of text to display. If set to a value of 0 or omitted, the actual length of *text-characters* is used (see *text-characters* below).

DISPLAY-EXCEPTION-CODE (optional, output) is set to a value of 0 if this function succeeds. Otherwise, one of the exception codes (on page 564) is returned. Notice that some exception codes are merely warnings.

DISPLAY-EXCEPTION-VALUE (optional, output) is set to a value of 0 if this function succeeds. Otherwise, it contains a value that provides more details on the exception that occurred. For more information, see exception codes (on page 564).

**Note** *display-description* will only have the correct length if four-digit binary data items are allocated with two bytes of storage. The example shown specifies a binary allocation override to guarantee the right allocation regardless of the configured binary allocation scheme. The binary allocation override is not necessary if BINARY-ALLOCATION=RM, RM1, or MF-RM.

*text-characters* is a required alphanumeric data item that contains the characters to be displayed. The number of characters to be displayed is controlled by DISPLAY-LENGTH. If DISPLAY-LENGTH is set to a value of 0, all of *text-characters* is displayed. If DISPLAY-LENGTH is less than the length of *text-characters*, the first DISPLAY-LENGTH characters are displayed. Otherwise, *text-characters* is displayed padded with spaces to DISPLAY-LENGTH characters.

*attribute-codes* is an optional alphanumeric data item that contains the attribute codes used to display *text-characters*. Each attribute code occupies a single character and controls the character at the same relative position in *text-characters*. The value of each attribute code is a one-relative indicator of an entry in *palette-table* (described below). An attribute code of X'00' causes the preceding code in *attribute-codes* to be used again. If there is no preceding code, the colors currently in effect (set by the preceding ACCEPT, DISPLAY, or C$SCWR) are used with all PALETTE-ATTRIBUTE-VALUES set to off.

If *attribute-codes* is omitted, or the length of *attribute-codes* is less than the actual DISPLAY-LENGTH (as described previously), *attribute-codes* is internally padded with X'00'. If *attribute-codes* is longer than the actual DISPLAY-LENGTH, the excess attribute codes are ignored.

If *attribute-codes* is specified, *palette-table* must also be specified.

*palette-table* is an optional group data item that is used to interpret *attribute-codes*. The table consists of 1 to 255 entries. Each *palette-table* entry describes a combination of colors and attributes.

Here are two possible COBOL descriptions for *palette-table*:

```
01 PALETTE-TABLE-1.
   03 PALETTE-TABLE-ENTRIES.
      05 PALETTE-TABLE-ENTRY-1.
         07 FOREGROUND-COLOR-1   PIC X.
         07 BACKGROUND-COLOR-1   PIC X.
         07 ATTRIBUTE-VALUE-1    PIC 9(4) BINARY(2).
         07 FILL-CHARACTER-1     PIC X.
      05 PALETTE-TABLE-ENTRY-2.
         07 FOREGROUND-COLOR-2   PIC X.
         07 BACKGROUND-COLOR-2   PIC X.
         07 ATTRIBUTE-VALUE-2    PIC 9(4) BINARY(2).
         07 FILL-CHARACTER-2     PIC X.
                    ⋮
      05 PALETTE-TABLE-ENTRY-255.
         07 FOREGROUND-COLOR-255 PIC X.
         07 BACKGROUND-COLOR-255 PIC X.
         07 ATTRIBUTE-VALUE-255  PIC 9(4) BINARY(2).
         07 FILL-CHARACTER-255   PIC X.

01 PALETTE-TABLE-2.
   03 PALETTE-TABLE-ENTRIES.
      05 PALETTE-TABLE-ENTRY         OCCURS 255 TIMES.
         07 FOREGROUND-COLOR-VALUE   PIC X.
         07 BACKGROUND-COLOR-VALUE   PIC X.
         07 ATTRIBUTE-VALUE          PIC 9(4) BINARY(2).
         07 FILL-CHARACTER           PIC X.
```

In the following description of color values, FOREGROUND-COLOR-VALUE contains a value from PALETTE-COLOR-VALUES that indicates the text color; BACKGROUND-COLOR-VALUE contains a value from PALETTE-COLOR-VALUES that indicates the background color. Both FOREGROUND-COLOR-VALUE and BACKGROUND-COLOR-VALUE are ignored with a warning unless a color monitor is being used or the USE-COLOR keyword (see page 331) of the TERM-ATTR configuration record is set to YES.

The permitted color values are listed below. Using a value not included in this list results in undefined behavior.

```
01 PALETTE-COLOR-VALUES          USAGE DISPLAY.
   03 PALETTE-UNSPECIFIED-1      PIC X VALUE SPACE.
   03 PALETTE-UNSPECIFIED-2      PIC X VALUE X"00".
   03 PALETTE-BLACK              PIC X VALUE "0".
   03 PALETTE-RED                PIC X VALUE "1".
   03 PALETTE-GREEN              PIC X VALUE "2".
   03 PALETTE-YELLOW             PIC X VALUE "3".
   03 PALETTE-BLUE               PIC X VALUE "4".
   03 PALETTE-MAGENTA            PIC X VALUE "5".
   03 PALETTE-CYAN               PIC X VALUE "6".
   03 PALETTE-WHITE              PIC X VALUE "7".
```

PALETTE-UNSPECIFIED-1 or PALETTE-UNSPECIFIED-2 cause the last color output to be used.

ATTRIBUTE-VALUE contains a value that specifies what attributes are to be applied. The value is produced by summing together the desired values from PALETTE-ATTRIBUTE-VALUES, listed below. Values omitted from the following table are reserved and must not be set.

```
01 PALETTE-ATTRIBUTE-VALUES      USAGE BINARY(2).
   03 INTENSITY-HIGH             PIC 9(4) VALUE 1.
   03 BLINK-ON                   PIC 9(4) VALUE 2.
   03 REVERSE-ON                 PIC 9(4) VALUE 4.
   03 UNDERLINE-ON               PIC 9(4) VALUE 8.
   03 GRAPHICS-ON                PIC 9(4) VALUE 16.
   03 FILL-CHARACTER-ON          PIC 9(4) VALUE 32.
```

With the exception of FILL-CHARACTER-ON, the meaning of each attribute is the same as if the attribute were specified in the CONTROL phrase of a DISPLAY statement. To combine multiple attributes, sum the values together. The FILL-CHARACTER-ON attribute specifies a character to be used instead of the character specified in the *text-characters*.

FILL-CHARACTER contains a character to be displayed if PALETTE-ATTRIBUTE-VALUES indicates FILL-CHARACTER-ON. The value in FILL-CHARACTER is ignored unless FILL-CHARACTER-ON attribute has been set.

Because the *palette-table* contains both alphanumeric and numeric fields, use the INITIALIZE statement to remove all colors and attributes from the table. To reset the example palette tables, enter:

```
INITIALIZE PALETTE-TABLE-1, PALETTE-TABLE-2.
```

## Usage Notes

If this function is requested to write characters past the end of the screen (or window, if pop-up windows are active), the screen or window is scrolled.

The current line and current position after the call obey the rules for the ACCEPT and DISPLAY statements.

Colors set by this function also affect subsequent ACCEPT and DISPLAY operations if those operations do not specify an FCOLOR or BCOLOR keyword, as described in the CONTROL phrase (on page 195).

## Fatal Errors

The runtime system terminates if the C$SCWR function is called with any of the following four conditions:

1. C$SCWR is called with other than two or four parameters, or any one of the parameters is a simple numeric parameter.
2. The length of DISPLAY-DESCRIPTION is less than eight characters.
3. The length of text-characters is zero.
4. The size of any passed data item exceeds 65535 bytes.

## Exception Codes

The following exception codes are stored in the DISPLAY-EXCEPTION-CODE variable. The C$SCWR function reports three types of exception codes: error, warning, and informational. Lower-numbered exception codes are reported before higher-numbered exception codes. These codes and their associated definitions are listed in Table 57.

**Note** Error codes take precedence over warning and informational codes. If multiple exceptions occur, the first occurring exception at a given level is returned. If warnings are returned, the display is performed, but the results may not be as expected.

**Table 57: C$SCWR Exception Codes**

| Code | Type | Description |
|---|---|---|
| 0 | | No error detected. |
| 1 | Error | An invalid DISPLAY-VERSION was specified. This data item must be set to a value of 0. |
| 2 | Error | An I/O error occurred while the write operation was being performed. DISPLAY-EXCEPTION-VALUE contains the RM/COBOL I/O error code. |
| 3 | Error | DISPLAY-LINE is greater than the number of lines on the window or screen. The display was not performed. DISPLAY-EXCEPTION-VALUE contains the number of lines on the window or on the screen. An out-of-range line specification is diagnosed before an out-of-range position specification. |
| 4 | Error | DISPLAY-POSITION is greater than the number of columns on the screen. The display was not performed. DISPLAY-EXCEPTION-VALUE contains the number of columns on the window or on the screen. |
| 5 | Error | The specified palette is invalid. DISPLAY-EXCEPTION-VALUE indicates either (a) the palette table does not contain complete palette entries, or (b) the table contains more than 255 entries. |
| 6 | Error | An invalid UNIT was specified. This data item must be set to a value of 0. |
| 20 | Warning | An out-of-range palette index was specified. DISPLAY-EXCEPTION-VALUE contains the offending palette index. The palette specification is ignored and treated as if unspecified. |
| 21 | Warning | The palette contains an invalid foreground color specification. DISPLAY-EXCEPTION-VALUE contains the offending palette index. The color is ignored and treated as if unspecified. |
| 22 | Warning | The palette contains an invalid background color specification. DISPLAY-EXCEPTION-VALUE contains the offending palette index. The color is ignored and treated as if unspecified. |
| 23 | Warning | The palette contains an invalid foreground and background color specification. DISPLAY-EXCEPTION-VALUE contains the offending palette index. The color is ignored and treated as if unspecified. |
| 24 | Warning | The palette contains an invalid attribute value. DISPLAY-EXCEPTION-VALUE contains the offending palette index. The entire attribute code is ignored and treated as if unspecified. |
| 40 | Informational | A monochrome display is in use, or USE-COLOR=N was configured. The specified foreground and background colors are not validated and are ignored. This condition cannot be detected under all circumstances. However, in all cases, DISPLAY-EXCEPTION-VALUE contains the palette index of the offending attribute. |

# C$SecureHash

C$SecureHash is used to produce a 20-byte message digest from an input text string using the secure hash algorithm (SHA-1).

### Calling Sequence

```
CALL "C$SecureHash"
   USING MessageText [MessageLength]
   GIVING MessageDigest
```

*MessageText* must be an identifier that references a nonnumeric data item. Its value is the input text string to the secure hash algorithm. While the secure hash algorithm supports messages of length 2**64 or less bits (2**61 or less bytes), this implementation is limited to messages of length 2**32 or less bits (2**29 or less bytes).

*MessageLength*, if specified, must reference a numeric integer data item. Its value specifies the number of bytes of *MessageText* to be considered when producing the message digest. Thus, the value must be less than or equal to the length of data item referenced by *MessageText*. If *MessageLength* is omitted, the entire value of the data item referenced by *MessageText* is used, as if LENGTH OF *MessageText* had been specified for *MessageLength*.

*MessageDigest* must be an identifier that references a nonnumeric data item of exactly 20 bytes in length. The message digest result from the secure hash algorithm is returned in the referenced data item. The message digest value is stored in the form most significant byte at lowest address to least significant byte at highest address regardless of the memory architecture of the machine on which C$SecureHash is called. When there is insufficient memory for C$SecureHash to do its work, the contents of *MessageDigest* are set to all binary zeroes. This only occurs when a memory area slightly larger than the size of the message text cannot be allocated.

The secure hash algorithm used by C$SecureHash, other than the length limitation, is the one defined as the secure hash standard by Federal Information Processing Standard (FIPS) Publication 180-1, which is often referred to as SHA-1. More information on SHA-1 can be obtained by reading FIPS Pub 180-1, which is available at http://www.itl.nist.gov/fipspubs/fip180-1.htm.

One example of the usefulness of a message digest is storing a password in a secure form. Since the message digest is produced using a one-way hash of the password, it is computationally infeasible to recover the password from the message digest value. (However, if the password is easy to guess or find in a dictionary, a computer program can be used to search for a password that hashes to a given message digest value.)

**Note** The input text string “abc” (length = 3 bytes) produces the hash value:

```
x"A9993E364706816ABA3E25717850C26C9CD0D89D"
```

Since this is a well-known test result for the secure hash algorithm (documented in FIPS Pub 180-1), “abc” is not recommended as a password value.

Message digests are also often used to verify that a message has not been changed from its original value. This involves computing the message digest of the original message text and transmitting the message digest in a secure manner, either on a separate secure channel or by using encryption of the message digest to guarantee that the message digest is not modified during transmission. The receiver of the message can then compute the message digest from the received message text and verify that the resulting message digest matches the supplied message digest. If the message digests match, it is extremely unlikely that the message text has been modified during transmission.

# C$SetDevelopmentMode

C$SetDevelopmentMode is used to enable expanded error information reporting (known as "development mode") for many of the C$ and P$ subprograms. The P$ subprograms are discussed in Appendix E: *Windows Printing* (on page 447). When development mode is enabled, more verbose error reporting is performed to assist the COBOL developer in implementing these subprogram calls. See also C$ClearDevelopmentMode (on page 534).

### Calling Sequence

```
CALL "C$SetDevelopmentMode"
```

Development mode also may be set at program startup with the RM_DEVELOPMENT_MODE environment variable:

```
RM_DEVELOPMENT_MODE=[Y|y|N|n]
```

Specify Y or y to enable development mode. Specify N or n, or omit the variable, to disable development mode. All other values are undefined.

# C$SetEnv

C$SetEnv is used to set or clear the value of an environment variable. On UNIX, environment variable names are case-sensitive. On Windows, environment variable names are not case-sensitive.

Setting the value of an environment variable with C$SetEnv updates the corresponding environment variable immediately in the process space of the current run unit. Thus, when the RM/COBOL runtime system uses environment variables for such actions as file access name resolution, the call to C$SetEnv will have an immediate effect on that run unit. However, to affect a different run unit, C$SetSyn (on page 569) should be used instead of, or together with, C$SetEnv.

### Calling Sequence

```
CALL "C$SetEnv" USING name, value [, return]
```

*name* is an alphanumeric data item that contains the name of the environment variable to set or clear.

*value* is an alphanumeric data item that contains the value to which the environment variable is set. A value of SPACES indicates that the environment variable should be deleted.

*return* is a PICTURE 9(*n*) BINARY, where *n* can be a digit from 1 to 9. The value returned is zero for success and non-zero for failure.

# C$SetSyn

C$SetSyn is used to set the value of an RM/COBOL synonym in the UNIX resource file (on page 28) or in the Windows registry (on page 66).

Setting the value of a synonym with C$SetSyn does not update the corresponding environment variable until the next time the RM/COBOL runtime system is started, or the registry or resource file is scanned. Thus, when the runtime system uses environment variables for such actions as file access name resolution, the call to C$SetSyn will not have an immediate effect on the run unit. When an immediate effect on the run unit is desired, an additional call to C$SetEnv (on page 568) is required to update the corresponding environment variable.

### Calling Sequence

```
CALL "C$SetSyn" USING name, value, program
```

*name* is an alphanumeric data item that contains the name of the synonym to set or clear.

*value* is an alphanumeric data item that contains the value to which the synonym is set. A value of SPACES indicates that the synonym should be deleted.

*program* is an alphanumeric data item that indicates the name of the program whose synonym properties are being changed. A value of SPACES indicates the "Defaults Properties" in the Windows registry or the [Default Synonyms] section in the UNIX resource file.

# C$Show

C$Show is used to set the show state of the main RM/COBOL window.

**Note** This subprogram is supported only under Windows.

### Calling Sequence

```
CALL "C$Show" USING flag
```

*flag* is a numeric data item that contains one of the values described below:

```
01 WIN-SHOW-STYLES.
   03 SW-HIDE               PIC 9(4) BINARY VALUE 0.
   03 SW-MINIMIZE           PIC 9(4) BINARY VALUE 6.
   03 SW-RESTORE            PIC 9(4) BINARY VALUE 9.
   03 SW-SHOW               PIC 9(4) BINARY VALUE 5.
   03 SW-SHOWMAXIMIZED      PIC 9(4) BINARY VALUE 3.
   03 SW-SHOWMINIMIZED      PIC 9(4) BINARY VALUE 2.
   03 SW-SHOWMINNOACTIVE    PIC 9(4) BINARY VALUE 7.
   03 SW-SHOWNA             PIC 9(4) BINARY VALUE 8.
   03 SW-SHOWNOACTIVATE     PIC 9(4) BINARY VALUE 4.
   03 SW-SHOWNORMAL         PIC 9(4) BINARY VALUE 1.
```

| Value | Meaning |
|---|---|
| SW-HIDE | Hides the window and passes activation to another window. |
| SW-MINIMIZE | Minimizes the specified window and activates the top-level window in the system's list. |
| SW-RESTORE | Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position (same as SW-SHOWNORMAL). |
| SW-SHOW | Activates a window and displays it in its current size and position. |
| SW-SHOWMAXIMIZED | Activates a window and displays it as a maximized window. |
| SW-SHOWMINIMIZED | Activates a window and displays it as an icon. |
| SW-SHOWMINNOACTIVE | Displays a window as an icon. The window that is currently active remains active. |
| SW-SHOWNA | Displays a window in its current state. The window that is currently active remains active. |
| SW-SHOWNOACTIVATE | Displays a window in its most recent size and position. The window that is currently active remains active. |
| SW-SHOWNORMAL | Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position (same as SW-RESTORE. |

### Examples

The following code fragment hides the main window:

```
CALL "C$SHOW" USING SW-HIDE.
```

The following code fragment shows the main window:

```
CALL "C$SHOW" USING SW-SHOW.
```

# C$ShowArgs

C$ShowArgs displays the list of arguments that were used to call C$ShowArgs. This facility is useful during development of non-COBOL subprograms.

**Note** On UNIX, the argument information is written to stdout. On Windows, the argument information is appended to the file **showargs.log** in the current directory.

### Calling Sequence

```
CALL "C$ShowArgs" [USING argument [, argument [, ...  ]]]
```

*argument* is any data item. C$ShowArgs will print its own name, the number of arguments, and the initial state flag. Then, for each argument, C$ShowArgs prints the address of the argument using hexadecimal notation, the size of the argument, the numeric type of the argument, three or four character-strings indicating the type of the argument, and finally, the number of digits and the scale of the argument.

**Note** This subprogram is most useful when it is used to replace another subprogram temporarily, thereby allowing the actual arguments that are being passed to be inspected.

### Example

```
Name="C$SHOWARGS", Args= 1, Initial=0
    Giving  OMITTED
    Arg # 1 Ptr=0041DBA8 Size=12  Type=16 ANS Digits=  0 Scale=  0
Name="C$SHOWARGS", Args= 3, Initial=65535
    Giving  Ptr=0041DBBE Size= 4  Type=11 NBS Digits=  8 Scale=  0
    Arg # 1 Ptr=0041DBA8 Size=12  Type=16 ANS Digits=  0 Scale=  0
    Arg # 2 Ptr=0041DBB4 Size=10  Type= 3 NTC Digits= 10 Scale= -2
    Arg # 3 Ptr=0041DBBE Size= 4  Type=11 NBS Digits=  8 Scale=  0
```

# C$TBar

C$TBar is used to display a toolbar in the RM/COBOL window.

To use this subprogram, the runtime system must be able to locate the **rmbars.dll** file. The following example illustrates how to add this DLL to the Runtime Command line with the L Option:

```
runcobol program-name L=RMBARS.DLL
```

Other Runtime Command Options can be used. See Chapter 7: *Running* (on page 177).

**Note** This subprogram is supported only under Windows.

### Calling Sequence

```
CALL "C$TBar" [USING buttonstr [, buttonstr ...  ], exit-code]
```

*buttonstr* is an alphanumeric data item that contains the button definition. The syntax is as follows:

```
buttonname["prompt"]=string
```

*buttonname* is the name of the icon stored in the filename specified by the Icon File property (on page 75).

*prompt* is an optional text string that is displayed when the mouse cursor is placed over the toolbar icon that is specified by *buttonname*. The setting of the Toolbar Prompt property (on page 84) determines how the text string is displayed. The text may be displayed in the status bar if the status bar is on, as described in C$SBar (on page 559) and Status Bar property (on page 82). The text may also or alternatively be displayed as a tooltip. The text string may contain one of the separator characters newline (x'0a'), colon (“:”), or vertical bar (“|”) to divide it into separate status bar and tooltip text. The appropriate separator character is determined by the Toolbar Prompt property.

*string* is an ASCII text string. However, it also can contain special characters for the Return, Tab, Escape, or Function keys. If the first character is a greater than character (>), then the characters that follow are executed as a command. The special characters are described in Table 11: Special Characters for the Button Character-String in Setting Toolbar Properties (on page 88). These characters are interpreted by the COBOL ACCEPT statement, as described in Table 19: Keys that Generate Field Termination Codes on page 191.

*exit-code* is a PICTURE 9(*n*) BINARY, where *n* can be a digit from 1 to 9. The *exit-code* parameter must be two- or four-byte binary and the value returned is zero for success and non-zero for failure.

**Note** Calling C$TBar with no arguments turns off the toolbar.

For additional information on toolbars, see Setting Toolbar Properties (on page 88) and Toolbar Editor (on page 94).

# C$TBarEn

C$TBarEn is used to enable and disable buttons on the toolbar.

To use this subprogram, the runtime system must be able to locate the **rmbars.dll** file. The following example illustrates how to add this DLL to the Runtime Command line with the L Option:

```
runcobol program-name L=RMBARS.DLL
```

Other Runtime Command Options can be used. See Chapter 7: *Running* (on page 177).

If the toolbar button bitmap contains three frames instead of the normal two, the third bitmap will be displayed while the button is disabled.

**Note** This subprogram is supported only under Windows.

### Calling Sequence

```
CALL "C$TBarEn" USING buttonname, flag,
      buttonname, flag, ...
```

*buttonname* is the name of the icon stored in the filename specified by the Icon File property (on page 75). If an equal sign is contained in the value of *buttonname*, the equal sign and any text following the equal sign is removed before the value of *buttonname* is used. Therefore, the same value as was used for the *buttonstr* argument to C$TBar (on page 572) may be used as long as a *prompt* value was not specified.

*flag* is a numeric data item with the value of 1 for enable and 0 for disable.

For additional information on toolbars, see Setting Toolbar Properties (on page 88) and Toolbar Editor (on page 94).

# C$TBarSeq

C$TBarSeq is used to set the bitmap sequence to use for buttons on the toolbar.

To use this subprogram, the runtime system must be able to locate the **rmbars.dll** file. The following example illustrates how to add this DLL to the Runtime Command line with the L Option:

```
runcobol program-name L=RMBARS.DLL
```

Other Runtime Command Options can be used. See Chapter 7: *Running* (on page 177).

**Note** This subprogram is supported only under Windows.

### Calling Sequence

```
CALL "C$TBarSeq" USING buttonname, seq,
      buttonname, seq, ...
```

*buttonname* is the name of the icon stored in the filename specified by the Icon File property (on page 75). If an equal sign is contained in the value of *buttonname*, the equal sign and any text following the equal sign is removed before the value of *buttonname* is used. Therefore, the same value as was used for the *buttonstr* argument to C$TBar (on page 572) may be used as long as a *prompt* value was not specified.

*seq* is a numeric data item that contains a value of 0 through 9 indicating the sequence number to use for displaying the named button.

For more information, see Setting Toolbar Properties (on page 88) and Toolbar Editor (on page 94).

# C$Title

C$Title is used to set the window title for the RM/COBOL window.

To use this subprogram, the runtime system must be able to locate the **c$title.dll** file. The following example illustrates how to add this DLL to the Runtime Command line with the L Option:

```
runcobol program-name L=C$Title.DLL
```

Other Runtime Command Options can be used. See Chapter 7: *Running* (on page 177).

**Note** This subprogram is supported only under Windows.

### Calling Sequence

```
CALL "C$Title" USING string
```

*string* is the text to be placed in the window title.

**Note** The Title Text property (on page 83) also can be used to set the text of the title for the RM/COBOL window.

# C$WRU

C$WRU returns the program identification, line number, and intraline number of the CALL statement, which called the subprogram that contains the call to C$WRU.

### Calling Sequence

```
CALL "C$WRU" USING program-name, prog-line,
      prog-intraline
```

*program-name* is a 30-character alphanumeric data item into which the program-id of the calling subprogram is stored. If the first subprogram of the run unit (that is, the main program) calls C$WRU, the value "RUNCOBOL" will be stored in PROGRAM-NAME, and zeroes are stored in *prog-line* and *prog-intraline*.

*prog-line* is a BINARY data item with the picture PIC 9(6) into which the line number containing the CALL statement is stored. If the program that contains the call to the subprogram that called "C$WRU" was compiled with the Q Compile Command Option (see page 148), the value stored in *prog-line* is the segment offset of the original CALL statement. Use the values from the Debug heading of the program listing to locate the statement.

*prog-intraline* is a BINARY data item with the picture PIC 9(2) into which the intraline number of the CALL statement is stored. For more information, see

Debug Values (on page 247) and Line and Intraline Numbers (on page 247). If the program that contains the call to the subprogram that called “C$WRU” was compiled with the Q Option, the value stored in *prog-intraline* is zero.

# DELETE

DELETE deletes a file.

### Calling Sequence

```
CALL "DELETE" USING file-name [exit-code]
```

*file-name* is the full or relative pathname of the file to be deleted. The name may be quoted with double quotes (") or single quotes ('). When the name is quoted, the quotes are removed, but the name is not otherwise modified. If the name is not quoted, the first control character terminates the name on Windows and the first white space character terminates the name on UNIX. On Windows, trailing spaces are removed from unquoted names. The *file-name* data item must be less than 1024 characters in length.

*exit-code* is an optional numeric data item that contains the exit code of the command upon return from the operating system. *exit-code* must be declared as PIC S9(4) BINARY. The value returned in *exit-code* is dependent on the underlying operating system. A value of 0, however, indicates success and a non-zero value indicates an error.

**Note** The values ANSI and OEM specified in the ALLOW-EXTENDED-CHARS-IN-FILENAMES keyword of the RUN-FILES-ATTR configuration record will affect the filenames passed to this subprogram.

# RENAME

RENAME renames a file.

### Calling Sequence

```
CALL "RENAME" USING old-name new-name [exit-code]
```

*old-name* is the source (old) filename. The name may be quoted with double quotes (") or single quotes ('). When the name is quoted, the quotes are removed, but the name is not otherwise modified. If the name is not quoted, the first control character terminates the name on Windows and the first white space character terminates the name on UNIX. On Windows, trailing spaces are removed from unquoted names. The *old-name* data item must be less than 1024 characters in length.

*new-name* is the target (new) filename. The name may be quoted with double quotes (") or single quotes ('). When the name is quoted, the quotes are removed, but the name is not otherwise modified. If the name is not quoted, the first control character terminates the name on Windows and the first white space character terminates the name on UNIX. The *new-name* data item must be less than 1024 characters in length.

*exit-code* is an optional numeric data item that contains the exit code of the command upon return from the operating system. *exit-code* must be declared as PIC S9(4) BINARY. The value returned in *exit-code* is dependent on the underlying operating system. A value of 0, however, indicates success and a non-zero value indicates an error.

**Note** The values ANSI and OEM specified in the ALLOW-EXTENDED-CHARS-IN-FILENAMES keyword of the RUN-FILES-ATTR configuration record will affect the filenames passed to this subprogram.

# SYSTEM

SYSTEM allows an arbitrary operating system command to be executed.

### Calling Sequence

```
CALL "SYSTEM" USING command-line [repaint-screen]
      [exit-code]
```

*command-line* is an alphanumeric data item that contains the command line to be passed to the operating system.

**Note** Under Windows, the command line is restricted to 130 characters. On most UNIX systems, the longest command line that can be executed is 4096 characters.

*repaint-screen* is an optional, one-byte, alphanumeric data item that controls whether the screen is redrawn after execution of the command. A value of Y or y causes the screen to be redrawn. A value of N or n does not redraw the screen. A blank or any other value defaults to the TERM-ATTR configuration record REDRAW-ON-CALL-SYSTEM value (see page 330).

**Note** This parameter is ignored under Windows.

*exit-code* is an optional numeric data item that contains the exit code of the command upon return from the operating system. *exit-code* must be declared as PIC S9(4) BINARY. The value returned in *exit-code* is dependent on the underlying operating system. A value of 0, however, indicates success and a non-zero value indicates an error.

If the argument count is incorrect or the arguments are of the wrong type, the run unit is stopped with the message “COBOL STOP RUN at line ? in SYSTEM ...”.

## UNIX Considerations

A *command-line* that contains either a single NULL character or all blanks starts a new instance of the shell.

The command is executed with the **system()** library function call. For a null or all space *command-line*, the environment variable SHELL is used to locate the shell processor. To return to the calling COBOL program, type:

```
exit or ctrl+d
```

The runtime system automatically calls **resetunit()** before executing the command to place the terminal in a “normal state” (the state before the runtime system was executed). Following the execution of the command, the runtime system automatically calls **setunit()** to return the terminal to the state that it requires for terminal I/O, and also causes the terminal screen to be redrawn. However, redrawing the screen may not always be desirable.

For example, the command executed might not change the screen. If all of the calls to SYSTEM in a COBOL program do not change the screen, the REDRAW-ON-CALL-SYSTEM keyword (see page 330) of the TERM-ATTR configuration record

may be set to NO and the runtime system will not redraw the screen after every call to SYSTEM. If some of the calls to SYSTEM in a COBOL program change the screen contents, the redrawing of the screen can be controlled through the use of the optional second argument, *repaint-screen*.

**Note** If the value of the second argument is N, or the REDRAW-ON-CALL-SYSTEM configuration keyword value is NO, and the called program changes the screen contents, unpredictable results may occur on the next DISPLAY statement because of the changed cursor position. CALL "SYSTEM" output is not stored in the runtime system's in-memory screen image.

See program **subtest.cbl** (included with the RM/COBOL development system) for examples of how to use the SYSTEM subprogram.

## Windows Considerations

A *command-line* that contains either a single NULL character or all blanks starts a new instance of the command processor.

The *command-line* parameter may be used to specify either a DOS or Windows program. The execution of the calling COBOL program is suspended until the called program terminates. For a null or all space *command-line*, the environment variable COMSPEC is used to locate the command processor. To return to the calling COBOL program, type:

```
EXIT
```

The style of the window used for Windows programs is controlled with the SYSTEM Window Type property (on page 83). The SYSTEM Window Type property also can be set with the C$GUICFG (on page 547) non-COBOL subprogram.

In order to execute a DOS internal command, it is necessary to specify completely the command processor on the command line. For example, to execute a **dir** command, enter the following:

- On Windows 98 and Windows Me

  ```
  COMMAND.COM /C DIR
  ```

- On Windows NT, Windows XP, Windows 2000, and Windows Server 2003

  ```
  CMD.EXE /C DIR
  ```

These examples assume that the command processor can be located through the PATH environment variable.

You can configure whether a DOS program runs in a full screen or in a window by modifying the MS-DOS Prompt Properties. This can be done by right-clicking the MS-DOS Prompt icon and selecting Properties from the pop-up menu.

**Note 1** When DOS internal commands such as **dir**, **mkdir**, **copy**, or **type** are executed via command.com, the returned *exit-code* is always zero. This is a DOS limitation. If a non-zero status is needed on failure, use an equivalent external command if one exists, such as **xcopy** instead of **copy**, or use CodeBridge to generate a COBOL-callable subprogram that executes an equivalent Windows or C library function. For example, CodeBridge samples include an example showing how to call the Win32 function **CreateDirectory** or the C library function **_mkdir**

from a COBOL program and obtain any resulting error code by building a DLL from a CodeBridge template file.

**Note 2** The values ANSI and OEM, if specified in the ALLOW-EXTENDED-CHARS-IN-FILENAMES keyword (see page 316) of the RUN-FILES-ATTR configuration record, will affect filenames passed to this subprogram in *command-line*. For example, CALL "SYSTEM" USING "c:\command.com /c mkdir *dir-name*", if *dir-name* contains extended characters, they will be interpreted as ANSI or OEM based on the specified configuration option value.

# Appendix G: Utilities

RM/COBOL provides a wide range of file conversion, management, and manipulation facilities. The majority of these utilities allow you to specify the required parameters either in the initial invocation command line or interactively during the course of execution.

This appendix describes the set of utilities provided for file conversion, management, and manipulation.

## Organization

The utilities described in this appendix are as follows:

1. The Combine Program (rmpgmcom) utility (on page 583) creates a program file by combining other program files and eliminating duplicate programs. It is used to create object program libraries.
2. The Map Program File (rmmappgm) utility (on page 586) produces a report describing the contents of an object program file created by the RM/COBOL compiler or the **rmpgmcom** utility.
3. The Map Indexed File (rmmapinx) utility (on page 589) produces a report describing the structure of an indexed file created by an RM/COBOL program or by the **rmdefinx** utility (described in the following item).
4. The Define Indexed File (rmdefinx) utility (on page 594) pre-creates an indexed file for use by RM/COBOL programs, or modifies some of the characteristics of an existing indexed file.
5. The Indexed File Recovery (recover1) utility (on page 598) is used to recover the data in an indexed file.

   **Note** The Indexed File Unload (**recover2**) utility is used to unload an indexed file to a sequential file. This utility is no longer needed to recover indexed files.
6. The Attach Configuration (rmattach) utility (on page 613), available only under Windows, is used to attach configuration files to **runcobol.exe**, **rmcobol.exe**, and **recover1.exe**.

7. The Initialization File to Windows Registry Conversion (ini2reg) utility (on page 614), available only under Windows, converts an RM/COBOL for Windows initialization file and places its contents into the Windows registry database.
8. The RM/COBOL Configuration (rmconfig) utility (on page 615), available only under Windows, is used to modify the configuration options for one or more RM/COBOL programs.

# Utilities Delivered on Media

The programs and files required to execute the utilities described in this appendix are provided with your RM/COBOL product. The actual number of files and programs depends on the specific version of the product you purchased and whether you purchased a development or a runtime system. The delivered media contains one or more README files that list the actual files and programs delivered. Please check these README files after you have installed the product to make sure that you have received all of the appropriate files and programs.

The utility programs and files may be placed in one directory. Be sure that the compiler and runtime system directory search sequences include the directory in which these files are placed. For more information, see Directory Search Sequences on UNIX (on page 24) and Directory Search Sequences on Windows (on page 61).

# General Considerations

Files with an extension of **.cob** were created with the RM/COBOL compiler and are executed with the RM/COBOL Runtime Command.

The programs **rmmapinx**, **rmmappgm**, **rmpgmcom**, and **recover1** generate reports that are written to a file named **PRINTER**. Under UNIX, the line printer spooler is used. Under Windows, the Windows default printer is used.

If you want to discard the report under UNIX, set the PRINTER environment variable to the value /dev/null with a resource file synonym or by using the following commands:

```
PRINTER=/dev/null ; export PRINTER
```

If you want to discard the report under Windows, set the PRINTER environment variable to the value NUL (with a synonym or other means) as follows:

```
PRINTER=NUL
```

If you want the report to go to a disk file under UNIX, set the PRINTER environment variable to the value of the filename with a resource file synonym or by using the following commands:

```
PRINTER=filename ; export filename
```

If you want the report to go to a disk file under Windows, set the PRINTER environment variable to the value of the filename (with a synonym or other means) as follows:

```
PRINTER=filename
```

If you want the report to go to the console under UNIX, set the PRINTER environment variable to the value **/dev/tty** with a resource file synonym or by using the following commands:

```
PRINTER=/dev/tty ; export PRINTER
```

If you want the report to go to the console under Windows, set the PRINTER environment variable to the value CON (with a synonym or other means) as follows:

```
PRINTER=CON
```

## Installing the Utility Programs

The RM/COBOL installation sections of this user's guide contain complete instructions on installing the utility programs. If you did not install the utilities when you installed RM/COBOL, refer to these instructions. For the appropriate installation and system considerations information for your specific operating system, see either Chapter 2: *Installation and System Considerations for UNIX* (on page 17) or Chapter 3: *Installation and System Considerations for Microsoft Windows* (on page 49).

# Combine Program (rmpgmcom) Utility

The **rmpgmcom** utility combines multiple RM/COBOL object files into a single program file library. RM/COBOL allows programs to be placed in the same file. This simplifies software distribution and provides a more consistent and logical representation of software. The **rmpgmcom** utility builds a Table of Contents (TOC) at the end of the output program file library. A Table of Contents speeds up **runcobol** library initialization while very slightly increasing the size of the library file.

The compiler allows several source programs to be contained in the same file. The program file generated has as many object files as the source file defines. **rmpgmcom**, on the other hand, allows source modules to be contained in separate files, and the contents of the resulting object files to be combined into one file.

**rmpgmcom.cob** is required to use this utility. This file is delivered in object form.

## Using the Utility

The Combine Program utility is executed with this command:

```
runcobol rmpgmcom [A='[PRINTER,] [STRIP,] out-file,
       file-1 [, file-2, ...  , file-n]']
```

PRINTER, if present, requests a copy of the report to be written to the printer specified with the environment variable, PRINTER.

STRIP, if present, removes COBOL symbol table and debug line table information, produced by the compiler Y Option, from object files. This is useful to reduce the size of a program library after debugging is complete.

*out-file* is the name of the new program file being created. If a filename extension is not specified, **.cob** is used.

*file-1*, *file-2*, . . ., *file-n* are the names of the program files being combined to form *out-file*. For each name, if a filename extension is not specified, **.cob** is used. If the file does not exist with a filename extension of **.cob**, the name is used with no filename extension.

If the argument list is not specified, **rmpgmcom** prompts first for whether or not to copy the report to the PRINTER, second for the STRIP option, third for the output filename, and then for the input filenames. **rmpgmcom** adds the programs in each input file to the output file, and then prompts for another input file. To terminate the program, press Enter without typing a name when prompted for an input file.

If the filenames are specified through the command line, the command line argument is limited to no more than 100 characters. Combining more than 100 characters of filenames requires direct operator input, use of input redirection, or multiple executions of the program.

As a precaution, if the output file exists before execution, **rmpgmcom** terminates, preventing accidental erasure of a good program file.

In most cases, an error does not terminate the program. If the program terminates abnormally, assume that the output file is not in a valid state and cannot be used to execute programs.

The same program-name can occur in more than one input program file. If this is the case, **rmpgmcom** uses the first one encountered and ignores subsequent programs with the same names. This can be very useful when you want to replace one program in a program file with a new version. Instead of having to recreate the file, you could use commands similar to the following.

**Note** **rmpgmcom** can combine more than 500 programs, but only the first 500 are guaranteed to have unique names. You may want to limit your program libraries to no more than 500 programs for this reason.

For UNIX, enter:

```
mv rmutil.lib, rmutil.bak
```

For Windows, use the Windows Explorer or open an MS-DOS window and enter:

```
RENAME rmutil.lib rmutil.bak
```

For all operating systems, enter:

```
runcobol rmpgmcom A='rmutil.lib, rmmappgm.cob, rmutil.bak'
```

***rmutil.bak*** is the name of a program file containing multiple programs (including ***rmmappgm.cob***).

***rmmappgm.cob*** is the name of the program file containing the program to be updated.

***rmmappgm.cob*** is loaded into ***rmutil.lib***, after which the programs in ***rmutil.bak*** (which is the previous version of ***rmutil.lib***) are loaded into ***rmutil.lib***. The previous version of ***rmmappgm.cob*** in ***rmutil.bak*** is ignored.

## Execution of Programs within Libraries

To execute programs within the created library, the L Runtime Command Option is used. For example:

```
runcobol rmmappgm L=rmutil.lib
```

Refer to Chapter 7: Running (on page 177) for additional information.

### Examples

Here is sample input to **rmpgmcom**:

```
Copy to PRINTER (y/n)?  Y
Output File:  rmutil.lib
Input File:  rmmappgm

The following programs are copied from rmmappgm.cob:
  MAP-PGM 2004/03/21  14:42:05
Input File:  rmpgmcom

The following programs are copied from rmpgmcom.cob:
  PGM-COM 2004/03/21  14:37:15
Input File:  rmmapinx

The following programs are copied from rmmapinx.cob:
  MAP-INX 2004/03/21  14:40:26
Input File:  rmdefinx

The following programs are copied from rmdefinx.cob:
  DEF-INX 2004/03/21  14:39:36
Input File:  (press ENTER)
```

Here is sample output from **rmpgmcom**.

```
The following programs are copied from rmmappgm.cob:
  MAP-PGM 2004/03/21  14:42:05

The following programs are copied from rmpgmcom.cob:
  PGM-COM 2004/03/21  14:37:15

The following programs are copied from rmmapinx.cob:
  MAP-INX 2004/03/21  14:40:26

The following programs are copied from rmdefinx.cob:
  DEF-INX 2004/03/21  14:39:36
```

The **rmpgmcom** utility allows for redirected input under UNIX. To redirect the input, follow these steps:

1. At the command line under UNIX, enter:

   ```
   ls *.cob > file.txt
   ```

   where *file.txt* is a valid file access name.

2. Edit *file.txt* using any text editor and enter Y (Yes) or N (No) as the first line of the file. (A response of Y requests a copy of the report to be written to the PRINTER.)

   Enter Y (Yes) or N (No) as the second line of the file. (A response of Y requests that symbol table and debug line table information be removed.)

   Add the library name as the third line of the file.

3. At the command line prompt, enter:

   ```
   runcobol rmpgmcom < file.txt
   ```

# Map Program File (rmmappgm) Utility

The **rmmappgm** utility reports information related to the object file created by the RM/COBOL compiler and the program library created by the **rmpgmcom** utility. This utility reports on all unnested programs contained in an object file and reports when a program library Table of Contents (TOC) is present. It also reports when an object program is a demonstration version (RM/DEMOV), an evaluation version (RM/EVALV), or an educational version (RM/EDUCV) object.

**rmmappgm.cob** is required to use the mapping utility. The file is delivered in object form.

## Using the Utility

The Map Program File utility is executed with this command:

```
runcobol rmmappgm [A='file-name [,option]…']
```

*file-name* is the name of the program file to be processed. If no filename extension is specified, **.cob** is used. If the file does not exist with a filename extension of **.cob**, the name is used with no filename extension.

*option* may be chosen from the following selections:

- PRINTER, if present, requests a copy of the report to be written to the printer specified with the environment variable, PRINTER. NOPRINTER can be specified to suppress a printer report file. The default is NOPRINTER.
- SCREEN, if present, requests a screen display of the report. NOSCREEN can be specified to suppress screen display of the report. The default is SCREEN.
- DETAILS, if present, requests a detailed report for each program instead of just a report with just a summary line for each program. NODETAILS can be specified to suppress details and produce the summary line report instead. The default is NODETAILS.

If the argument list is not specified, **rmmappgm** prompts first for the name of a file to be processed and then prompts for whether or not to copy the report to the printer. This continues after each report until you press Enter without typing a name.

The report generated by this program is always written to the screen and is also written to the printer only if the PRINTER option is specified. The information in the report is as follows:

| | |
|---|---|
| PROGRAM NAME | Corresponds to the PROGRAM-ID value of the program. |
| SIZES | Indicates the size in bytes of the memory needed for the fixed procedure area, the data area, and the overlay area, as well as the total of these three. |
| # ARG | Indicates the number of arguments required by the program. |
| # FILE | Indicates the number of data files defined in the program, including any nested programs. |
| COMPILED | Indicates the date and time the program was compiled, the compiler options chosen, as well as the version of the compiler used and the operating system under which the program was compiled. |
| OBJ VERS | Indicates the object version of the program. For more details, see Appendix H: *Object Versions* (on page 617). |

### Examples

Here is sample input to **rmmappgm**.

```
Object File:  rmmappgm.cob
Copy to PRINTER (y/n)?  Y
Object File:  (press ENTER)
```

Here is an example of a summary line report from the **rmmappgm** utility.

```
                ----------SIZES----------   #    #   --------------------COMPILED--------------------------OBJ
    PROGRAM NAME     PROCEDURE  DATA  OVLY  TOTAL ARG  FILE   DATE      TIME       VERSION        SYSTEM   OPTS    VERS
File:  rmmappgm.cob
RMMAPPGM                2170  1778     0   3948   1    2  2004/03/21 12:10:15  RM/COBOL 8.0n.00 UNIX     E        6
```

Here is an example of a detailed report from the **rmmappgm** utility.

```
Map RM/COBOL object files - Version 9.00
File: rmmappgm.COB
Details for program-name:  rmmappgm
   Object sizes:  Procedure =   9416, Data =    4538,
                  Overlay   =      0, Total =  13954
   Arguments accepted:    1;  Files used:    2;  CDs used:    0
   Items with external attribute:     0
   Low value:   X"00"
   High value:  X"FF"
   Compilation date & time:   2004/08/27 11:03:16
   Compilation system:  32-Bit Windows
   Compilation options:
      A - Allocation map listing
   Compiler version:  RM/COBOL 9.00.00
   Object version:    9
   Program code-set:  ASCII
   Features flags:
      ACCEPT/DISPLAY (low volume IO)
      CALL
      INSPECT
      STRING
      UNSTRING
      Sequential I/O
      LINAGE
   Native character set:  Windows OEM
```

# Map Indexed File (rmmapinx) Utility

The **rmmapinx** utility reports information related to an indexed file.

**Note** In order to report information related to a Btrieve file, use the Btrieve File Manager (either the Windows interactive version, **wbmanage.exe**, or the command line version, **butil.exe**). Refer to the appropriate Btrieve installation and operation manual for complete information about the utility.

## Using the Utility

The Map Indexed File utility is executed with the following command:

```
runcobol rmmapinx [A='file-name [,DETAIL][,PRINTER]']
```

*file-name* is the name of the file to be mapped.

DETAIL, if present, requests a more detailed report.

PRINTER, if present, requests a copy of the report to be written to the printer specified with the environment variable, PRINTER.

If the optional information is not specified, you are prompted for *file-name*, DETAIL, and PRINTER, as follows:

```
Indexed File:
Detail Information (y/n)?
Copy to PRINTER (y/n)?
```

Enter the name of the file to be used in response to the first prompt line. A response of Y to the second prompt requests detailed information for the file, as described in Detailed Information Report (on page 591). A response of N requests only the basic information (see the following paragraphs). When in prompt mode, the user is prompted again after each file report is processed. To exit the program when in prompt mode, press Enter without typing a name when prompted for a filename. A response of Y to the third prompt requests a copy of the report to be written to the printer.

## Basic File Information Display

Basic file information is always reported for each existing file entered, without respect to the presence or absence of the DETAIL option.

- **File status.** If the file was created by the Define Indexed File (rmdefinx) utility (on page 594) and has never been opened for output, a file status line indicating this fact appears. Also, if this status is indicated, only the disk block size and data and key compression status (with compression characters) appear. Detail information, described in the next section, is reported if the detailed report is requested.
- **Record length.** Record length is reported in one of two formats. For fixed-length records, it is shown as a specific value:

  ```
  Records are fixed length = size
  ```

  *size* is the actual record size. If variable-length records are defined for the file, this is shown as a range of values:

  ```
  Records vary in length from min to max bytes
  ```

  *min* and *max* are the minimum and maximum record lengths.
- **Disk block size.** Disk block size is the number of bytes actually allocated to one block on disk.
- **User block size.** User block size is the number of records or bytes specified by the user in the FD statement used when creating this file.
- **Data compression status.** Data compression status may be either compressed or uncompressed. If compressed, the data space and zero characters also appear.
- **Key compression status.** Key compression status may be either compressed or uncompressed. If compressed, the key space character also appears.
- **Number of keys.** This is the actual number of keys (including the prime key) defined for the file.
- **Number of segments.** This is the total number of segments in all keys defined for the file. If the file has no split keys, the number of segments is equal to the number of keys.
- **Number of records.** This is the actual number of data records contained in the file.
- **Number of blocks.** This is the actual number of logical blocks allocated to the file.
- **Number of empty blocks.** This is the number of unused blocks allocated to the file.
- **Number of blocks required for a rewrite.** This is the maximum number of empty blocks required if a rewrite were to relocate the modified record in the file.
- **Atomic I/O log.** If the file is a version 4 indexed file with atomic I/O enabled, then the number of atomic I/O log blocks and the atomic I/O log state are reported.
- **Key information.** See Key Descriptor Information Display (on page 592).

- **Last error.** This is the last error received when accessing the file. It includes the date and time the error occurred. Only errors with class of 30, or a class of 98 with a suberror greater than 2, are remembered.
- **Open For Modify Count.** This is a count, held in the file, of the number of times the file is currently open. If this count is not zero, and there are actually no current opens, it is an indication that the file may be corrupted because of an incomplete close operation.

## Detailed Information Report

The following information is reported when the detailed report is requested, either by specifying DETAIL in the command line when invoking the utility, or by responding Y to the detailed report prompt.

- **File version information.** The file version number indicates any advanced features used in the file. The minimum read version number and the minimum write version number are used to prevent previous versions of RM/COBOL from reading or modifying files with features unavailable to them. For more information on indexed file version numbers, see Indexed File Version Levels (on page 243).
- **First unused empty block.** The first unused empty block will be displayed only for files with a version number of 2 or greater. It is the block number of an empty block that has never been used and that is followed only by other unused empty blocks.
- **File lock limit.** The largest address where a lock may be applied to this file is displayed only for files with a version number of 3 or greater. See the description of the LARGE-FILE-LOCK-LIMIT keyword on page 318 of the RUN-FILES-ATTR configuration record for more information.
- **Disk block increment.** Disk block increment is the difference in the disk addresses of consecutive blocks of the file. It is always greater than or equal to the disk block size.
- **Allocation increment.** Allocation increment is the number of disk blocks that are allocated when the file is first created and whenever the file increases in its physically allocated size.
- **Version 4 information.** If the file is a version 4 indexed file, then the number of KIB blocks and duplicate KIB blocks are reported. If the version 4 indexed file has atomic I/O enabled, then various information related to the atomic I/O log are reported. This information is generally not useful to the user unless contacting Liant technical support services.
- **Recoverability/Performance Strategy summary.** This is a summary of the recoverability and performance strategy options, either set by the runtime system when the file was created, or set when the file was created or modified by the Define Indexed File (rmdefinx) utility (on page 594).
- **Recover1 last run time.** If a version 7.5 or later **recover1** utility has ever been run on this file, then the time and date of the most recent **recover1** run are reported.

## Key Descriptor Information Display

If the file has been opened for output, the attributes of each key are reported one key segment per line. The information reported is as follows:

- **Key number.** The first key is labeled prime, with alternate keys numbered starting with 1.
- **Segment number.** The segment number within the key. The first segment of a key is number 1.
- **Starting position.** This is the position in the record where the key segment starts. The leftmost position in a record is position 1.
- **Segment length.** This is the number of bytes that the key segment occupies in the record.
- **Key length.** This is the total number of bytes that the entire key occupies in the record. It is the sum of the segment lengths of all of the segments of the key. This value is shown only on the final segment line of a split key since it applies to the entire key.
- **Tree height.** This is the maximum number of disk accesses that could be required to locate a record containing the key. Additional disk accesses may be required to read or modify the record. This value is shown only on the final segment line of a split key since it applies to the entire key.
- **Duplicates permitted.** This value is either yes or no. Yes indicates that the key allows duplicate key values to be present. This value is shown only on the final segment line of a split key since it applies to the entire key.

Additionally, a blank line is displayed between keys when any of the keys of the file is a split key. If the file has no split keys, there is one line per key with no intervening blank lines.

### Example

Here is sample input to **rmmapinx**.

```
Indexed File:  file1
Detail Information (y/n)?  y
Copy to PRINTER (y/n)?  Y
Indexed File:  (press ENTER)
```

Here are the file control entry and file description entry for file1.

```
SELECT file1 ASSIGN TO DISK "file1"
    ORGANIZATION IS INDEXED
    RECORD KEY f1-prime-key = f1-pkey-s1, f1-pkey-s2
    ALTERNATE RECORD KEY f1-alt-key1 = f1-akey1-s1
    ALTERNATE RECORD KEY f1-alt-key2 = f1-akey2-s1,
        f1-akey2-s2, f1-akey2-s3 WITH DUPLICATES.

FD file1     RECORD CONTAINS 134 CHARACTERS.
01 f1-rec.
   02 f1-pkey-s1        PIC X(30).
   02 f1-akey1-s1       PIC X(3).
   02 f1-akey2-s1       PIC X(10).
   02 data1             PIC X(40).
   02 f1-akey2-s2       PIC X(6).
```

```
           02 f1-pkey-s2        PIC X(5).
           02 f1-akey2-s3       PIC X(20).
           02 data2             PIC X(20).
```

Here is sample output from **rmmapinx**.

```
RM/COBOL Map Key Utility - 8.00  03/20/2004  12:59:17    Page   1
File Information:
  file1 is an Indexed File.
  Records are fixed length = 134 Bytes.
  Disk Block Size = 1024 Bytes, User Block Size = not specified.
  Data Records are compressed, Keys are compressed.
    Data Block Space Character Value = 32.
    Data Block Zero Character Value = 48.
    Key Block Space Character Value = 32.
  File has 3 Keys and 6 Segments.
  File contains 1 Record and occupies 128 Blocks.
  There are 12 empty Blocks.
  7 empty Blocks may be needed for a Write operation.
  There are 110 Atomic I/O Log Blocks.
  The Atomic I/O log state is 1 (inactive).
  Open For Modify Count = 0.

Detail Information:
  File version number = 4.
  Minimum read version number = 4.
  Minimum write version number = 4.
  First unused empty block = 116.
  File Lock Limit = 2 GB.
  Disk Block Increment Size = 1024 Bytes.
  Allocation Increment = 8 Blocks.
  There is 1 KIB block and 1 duplicate KIB block.
  There is 1 Log Map block and 109 Log Area blocks.
  Atomic I/O Log Overhead Blocks:
      First map = 1, First log = 2,
      First used = 2, Next available = 2,
      First duplicate KIB = 111.
    Log blocks per operation = 93.
    Log Operation Numbers:  First = 1, Next = 1.
  Recoverability/Performance Strategy:
    Data and Index blocks are logged and forced to disk during
        each update.
      Force Write Data Blocks = No.
      Force Write Index Blocks = No.
      Force to Disk = No.
      Force File Closed = No.
      Atomic I/O Enabled = Yes.
  Recover1 was last run on this file at 17:34:45 on 03/20/2004.

Key Information:
     Key   Segment  Starting  Segment   Key     Tree   Duplicates
   Number  Number   Position  Length   Length  Height  Permitted?
    Prime      1          1      30
    Prime      2         90       5       35       1        No

       1       1         31       3        3       1        No

       2       1         34      10
       2       2         84       6
       2       3         95      20       36       1        Yes
```

# Define Indexed File (rmdefinx) Utility

The **rmdefinx** utility pre-creates an indexed file in order to alter the default characteristics assigned by RM/COBOL programs. The utility also can modify some of those characteristics on files created by RM/COBOL programs.

**rmdefinx.cob** is required to use the Define Indexed File utility. The file is delivered in object form.

**Note** In order to pre-create a Btrieve file, use the Btrieve File Manager (either the Windows interactive version, **wbmanage.exe**, or the command line version, **butil.exe**). Refer to the appropriate Btrieve installation and operation manual for complete information about the utility.

## Using the Utility

The Define Indexed File utility is executed by issuing the following command:

```
runcobol rmdefinx [A='file-name [,CONVERT4] [,ATOMICIO]
       [,[NO]RECOVER1]']
```

*file-name* is the name of the file to be defined or altered.

CONVERT4, if present, requests that **rmdefinx** be run in batch mode (that is, no interactive questions) to convert *file-name* to indexed file version 4. For more information, see File Version Level 4 (on page 244).

ATOMICIO, if present, requests that **rmdefinx** be run in batch mode to enable atomic I/O on *file-name*. ATOMICIO implies CONVERT4 if the indexed file is currently less than file version 4. Specifying CONVERT4 without specifying ATOMICIO disables atomic I/O.

RECOVER1, if present, requests that **recover1** be run on the indexed file after **rmdefinx** has altered it. This is the default behavior if **rmdefinx** is run in batch mode (that is, if CONVERT4 or ATOMICIO is specified).

NORECOVER1, if present, requests that **recover1** not be run on the indexed file after **rmdefinx** has altered it. This might be used with CONVERT4 or ATOMICIO if the user wants to run **recover1** a special way or wants to run **recover1** to do the conversion after all of the files have been marked for conversion by **rmdefinx**.

You may omit the argument list. In this case, the program prompts for *file-name* in the following manner:

```
Indexed File:
```

The name of the file to be used is supplied in response to this prompt.

You may wish to run **rmdefinx** in batch mode (by specifying CONVERT4 and/or ATOMICIO) in order to convert a large number of indexed files to file version 4. To do so, you must construct a batch stream or command script that runs **rmdefinx** on each file to be converted. Be certain that you have a current backup of all files being converted. The **rmdefinx** utility automatically runs the **recover1** utility and terminates with return code 1 if any error occurs. Also, note that the **recover1** utility,

as well as **runcobol** and **rmdefinx**, must be specified in your PATH environment variable. The following general example could be used on Windows via the Command Prompt (MS-DOS Prompt) box:

```
start /w runcobol rmdefinx a="d:\dat\file1,convert4"
if errorlevel 1 echo "error on file1" >>errlog
start /w runcobol rmdefinx a="d:\dat\file2,convert4"
if errorlevel 1 echo "error on file2" >>errlog
start /w runcobol rmdefinx a="d:\dat\file3,convert4"
if errorlevel 1 echo "error on file3" >>errlog
   . . .
start /w runcobol rmdefinx a="d:\dat\file<n>,convert4"
if errorlevel 1 echo "error on file<n>" >>errlog
```

## File Pre-creation

The following prompts are issued when a file is being pre-created:

```
Disk Block Size (in bytes):
Disk Block Increment (in bytes):
Allocation Increment (in blocks):
```

These queries deal with the manner in which space is allocated and used in the file. Disk block size is the actual number of bytes used in each physical block. A physical block size is indicated by the disk block increment size. These two sizes should be identical. The number of blocks allocated each time more space is required in the file is controlled by the allocation increment.

The next prompt is as follows:

```
Data Compression (y/n)?
```

This controls the amount of space taken up by logical records in the file. If you want logical records to be stored without compression, enter N. Otherwise, enter Y, and then respond to the following two prompts:

```
Space Character Value:
Zero Character Value:
```

These control the manner in which data compression takes place. Respond with the decimal value of the characters to be interpreted by the compression algorithm as a space and a zero. For example, the ASCII space and zero have decimal values of 32 and 48, respectively; the EBCDIC space and zero have decimal values of 64 and 240, respectively.

The next two prompts are as follows:

```
Key Compression (y/n)?
Space Character Value:
```

These queries control key compression in a manner similar to data compression, except that only trailing spaces are compressed.

The next prompts are as follows:

```
Force Write Data Blocks (y/n)?
Force Write Index Blocks (y/n)?
Force to Disk (y/n)?
Force File Closed (y/n)?
```

These prompts determine recoverability/performance strategies for the file being processed. To select a strategy option, enter Y in response to the appropriate prompt. To omit a strategy option, enter N. See the discussion of data recoverability (on page 231) of indexed files.

The next prompt is as follows:

```
Version number (0, 2, 3, or 4)? 4
```

This prompt sets the version number of the file. Valid version numbers are shown in the following chart.

| Feature | Version Number 0 | Version Number 2 | Version Number 3 | Version Number 4 |
|---|---|---|---|---|
| Improved Empty Block List Algorithm | | ✓ | ✓ | ✓ |
| Use LARGE-FILE-LOCK-LIMIT | | | ✓ | ✓ |
| Block underfoot, duplicate KIB, atomic I/O | | | | ✓ |

The next prompt is as follows:

```
Enable Atomic I/O (y/n)?
```

This prompt is shown only if the file version number has been set to 4. The default (**Y)** allows support for the atomic I/O capability, which provides for more reliable indexed files.

The next prompt is as follows:

```
File Lock Limit (in GB)?
```

This prompt is shown only if the file version number has been set to 3 or 4. It allows you to specify the location of the largest lock to be placed on this file. For more information, see the descriptions of the LARGE-FILE-LOCK-LIMIT keyword of the RUN-FILES-ATTR configuration record (on page 316) and the DEFAULT-FILE-VERSION-NUMBER keyword of the RUN-INDEX-FILES configuration record (on page 320).

## File Modification

When the file specified already exists, the utility allows you to change the allocation increment, described previously, and offers you the option to alter the key compression, the recoverability/performance strategy, the file version number, and the file lock limit.

The prompt to change the key compression is as follows:

```
Change key compression (y/n)?
```

If you enter Y, you are prompted for whether the key compression should be enabled and for the value of the key compression space character, as described on the previous page. If the key compression or the key compression space character is changed, the file is marked as needing recovery. After you run the Indexed File Recovery (recover1) utility (on page 598), the indexed trees are built with the requested key compression. If the current key compression and key compression space character are unchanged, you are not forced to recover the index structure.

The prompt to change the recoverability or performance strategy is as follows:

```
Change Recoverability/Performance Strategy (y/n)?
```

If you enter Y, you are prompted for the four options discussed previously. If you enter N, the strategy is unchanged.

The prompt to change the file version number is as follows:

```
Change file version number (y/n)?
```

If you enter Y, you are prompted for the file version number discussed on page 596. If you specify the file version number as 3, you are prompted for the file lock limit. If you specify the file version as 4, you are prompted for whether to enable atomic I/O and then for the file lock limit. If the version number is changed from 0 to another version, or changed from another version to 0, the file is marked as needing recovery, and the Indexed File Recovery (**recover1**) utility must be run before the file can be used. After you run the Indexed File Recovery (**recover1**) utility, the empty block list is built with the correct algorithm. If the file version number is unchanged, you are not forced to recover the empty block list.

Similarly, if you change the version number to 4 from another version or from 4 to another version, the file is marked as needing recover, and the Indexed File Recovery (**recover1**) utility must be run before the file can be used.

If you attempt to change the file version number to 0 of a file that contains split keys or duplicate prime keys, a message is displayed indicating that you cannot do so; files using these features cannot be converted to version number 0 files. Also, files with a version number of 3 may have grown too large to be changed back to version 0 or 2. For more information on indexed file version numbers, see Indexed File Version Levels (on page 243).

After all questions have been answered, the following prompt appears:

```
OK (y/n)?
```

If you enter Y, the file is updated and the program either terminates (if a filename was specified on the command line) or prompts you for another filename.

When the file specified already exists, the utility attempts to lock the file at the file's currently set File Lock Limit value. If this lock attempt fails, the following message and prompt are displayed:

```
File lock attempt at n GB failed, possibly because
   this limit is too large for this system.
If you want to continue with the file NOT LOCKED,
   make certain nobody else has it open!
Continue (y/n)?
```

If you enter Y, the utility displays the normal file modification prompts. You should enter a new File Lock Limit value, as described above, that is valid for the system on

which you are running. If you enter N, the utility terminates with a "Lock Error 30" error message.

After **rmdefinx** has successfully altered the file, you are prompted to run **recover1** now if any changes you made require **recover1** to be run. The prompt is as follows:

```
Do you want to run recover1 now (y/n)?
```

If you enter **Y**, **rmdefinx** runs **recover1** and then displays a success or failure message. The **recover1** drop file argument is specified as **r1-drop**, which will be created in the current directory. The **rmdefinx** utility terminates with return code 1 if the **recover1** utility fails. You must not run **rmdefinx** a second time on a file that **rmdefinx** told you to run **recover1** on without running the intervening **recover1**.

**Note** The **rmdefinx** utility does not cause the file to "exist" in the COBOL sense. An OPEN OUTPUT (or OPEN I-O or EXTEND if SELECT OPTIONAL is used) must be successfully executed before other open modes become valid.

### Example

Here is sample input to **rmdefinx**.

```
Indexed File: file1
New File.
Disk Block Size (in bytes): 1024
Disk Block Increment (in bytes): 1024
Allocation Increment (in blocks): 10
Data Compression (y/n)? n
Key Compression (y/n)? n
Define Recoverability/Performance Strategy
   Force Write Data Blocks (y/n)? y
   Force Write Index Blocks (y/n)? y
   Force to Disk (y/n)? y
   Force File Closed (y/n)? y
Set file version number
   Version number (0,2,3, or 4)? 4
   Enable Atomic I/O (y/n)? y
   File Lock Limit (in GB)? 2
OK (y/n)? y
```

# Indexed File Recovery (recover1) Utility

The **recover1** utility recovers data stored in an RM/COBOL indexed file. It is a standalone program; that is, it does not require use of the Runtime Command (**runcobol**) to be executed. The **recover1** utility is also used by (or run after) the **rmdefinx** utility to convert an indexed file between some file versions or to change other fixed file attributes.

**Note 1** In order to recover data stored in a Btrieve file, use the Btrieve File Manager (either the Windows interactive version, **wbmanage.exe**, or the command line version, **butil.exe**). Refer to the appropriate Btrieve installation and operation manual for complete information about the utility.

**Note 2** Unless specifically stated otherwise, the name **recover1** refers to both the UNIX (**recover1**) and Windows (**recover1.exe**) versions of the **recover1** program.

**Note 3** If the output window of the Windows version of the **recover1** program disappears upon successful completion and you want that window to remain visible, set the Persistent property (on page 78) to True for the **recover1.exe** program.

**Note 4** The **recover1** utility does not use the environment variable RUNPATH to locate files. For more information, see Locating RM/COBOL Files on UNIX (on page 24) and Locating RM/COBOL Files on Windows (on page 61).

## Recovery Command

The Indexed File Recovery (**recover1**) utility is executed by issuing the following command:

```
recover1 indexed-file drop-file [options] ...
```

*indexed-file* is the filename of the indexed file to be recovered. The name is not resolved through any environment variables.

*drop-file* is the name of the file where **recover1** places any unrecoverable records found in the indexed file, as discussed in Recovery Process Description (on page 602). If *drop-file* specifies an environment variable name, the environment variable value will be resolved before opening the dropped record file.

*option* is zero or more command line options, as described in the following section. Options are specified with letters that must be preceded with a hyphen (-) or a slash (/). Option letters may be specified in upper or lower case. Certain option letters allow an optional pathname as part of the option format. The presence or absence of the pathname is determined by whether or not the next non-white space character following the option letter is a hyphen or slash, whichever one was used preceding the option letter.

**Note** The option introducer character slash is supported for Windows compatibility and should not be used on UNIX, where it can be confused with an absolute pathname; that is, a pathname that begins with a slash. Nevertheless, either the hyphen or the slash may be used to introduce option letters on Windows and UNIX. In the option formats given in this document, only the hyphen is shown, but the hyphen may be replaced with a slash.

**WARNING** Because of several changes in the RM/COBOL 7.5 runtime system, it is possible that an indexed file created by performing an OPEN OUTPUT in a COBOL program using the RM/COBOL 7.5 or later runtime system may have a different block size than a file you are attempting to recover. This may happen even though you specify the same file control entry and file description entry for the template file as when you initially created the file with an earlier version of RM/COBOL. Attempting to use this new file with a different block size as a template file may result in a loss of a large percentage of the recoverable records. You should verify that the block size of your template file is correct by using the Map Indexed File (rmmapinx) utility (on page 589). You can also avoid this problem by creating the template file with a version of RM/COBOL prior to 7.5 or by using a backup copy of the undamaged file. The MINIMUM-BLOCK-SIZE and ROUND-TO-NICE-BLOCK-SIZE keywords of the RUN-INDEX-FILES configuration record (on page 320) may also be used to cause the runtime to create a file with a block size that matches releases prior to 7.5.

## Recovery Command Options

Recovery Command options can be specified in either of the following two ways:

1. Depending on the operating system, they can be placed into the Windows registry (on page 66) or the UNIX resource file (on page 28).
   - In the Windows registry, the Command Line Options property (on page 72) provides command line options for the Indexed File Recovery utility when Recovery is selected on the Select File tab of the RM/COBOL Properties dialog box.
   - In the UNIX resource file, the Options keyword, described in Command Line Options (on page 29), provides command line options for the Indexed File Recovery utility in the global resource file **/etc/default/recover1rc** and the local resource file **~/.recover1rc**.
2. They can be specified in the Recovery Command itself.

The following options may be specified to modify the behavior of the Indexed File Recovery (**recover1**) utility.

**I** Use the I option to cause **recover1** to test only the file integrity and then stop. The file will not be modified in any way. Specifying the I option causes both the T and Z options to be ignored. If no problems are discovered, the return code is set to 0. If a problem is discovered, the return code is set to 1. The I option has the following format:

```
-I
```

The default is for recover1 to do a complete recovery of the indexed file if the file is marked as needing recovery. See the Y and Z options in this topic for additional options that modify the behavior of the Indexed File Recovery utility.

**Note** The integrity scan is a quick test of the file and is not comprehensive. Some problems, such as records with invalid duplicate keys, will not be detected. Indexed files with no errors detected by the integrity scan may still receive “98” errors or other I/O errors.

**K** Use the K option to indicate that the Key Information Block (KIB) should be assumed to be invalid and, optionally, to specify a template file for recovering the KIB. The K option has the following format:

```
-K [template-file]
```

If no *template-file* is specified, the user will be prompted either for a template file or for enough information to rebuild the KIB. If *template-file* is specified, it should be the name of a valid indexed file with the same format as the file being recovered. This file will be used as a template. The required KIB information is read from the KIB of the template file. The template file can be a backup copy of the file being recovered, if the backup occurred before the file was damaged, or, it can be a file created by performing an OPEN OUTPUT in a COBOL program with the proper file control entry and file description entry for the file being recovered. An OPEN OUTPUT must have been performed on the template file, but it need not contain any records. A template file must be specified if the KIB is corrupt and the file uses either an enumerated code set or an enumerated collating sequence. The default is to check the KIB for validity and, if it is found to be invalid, prompt for either a template file or information

to rebuild the KIB. The name of the template file is not resolved through any environment variables.

**WARNING** A template file with the wrong block size can cause the loss of a large percentage of the recoverable records in your file. This can happen because of changes introduced in RM/COBOL version 7.5 in the method used by the runtime system for calculating the block size. For additional information, see the warning that is noted in Recovery Command (on page 599).

**L** Use the L option to write information about errors encountered while recovering the file to a log file. The L option has the following format:

```
-L [log-file]
```

Only the first 100 errors will be logged. In addition to errors, a number of informational lines about the indexed file and its recovery are written to the log file, including information about sort memory (see the M option regarding sort memory). If *log-file* specifies an environment variable name, the environment variable value will be resolved before opening the log file; this allows the use of the name PRINTER to send the log information to the print device. If *log-file* is omitted in the L option, the default value of *log-file* is PRINTER. If the L option is not specified, the default is not to write a log file.

**Note** Environment variables can be set using synonyms set in the Windows registry (on page 66) or the UNIX resource file (on page 28).

**M** Use the M option to specify the number of megabytes of memory to allocate to the sort algorithm used in phase 4, build node blocks (see page 603). The M option has the following format:

```
-M [MB-of-memory]
```

where *MB-of memory* is a number in the range 0 to 2000. Allocating more memory generally results in faster execution of **recover1** and causes fewer node blocks to be built. If this option is not specified, a suitable number will be computed; in this case, sort memory is limited to no more than 40 million bytes. When a log file is written (see the L option), a line is written into the log file to show the maximum effective sort-memory size. If the M option is specified without a number of megabytes, the default value of 50 is used.

**Note** Specifying a number for *MB-of-memory* that is too large for your system may result in very poor system performance.

**Q** Use the Q option to cause **recover1** to perform its work without displaying information or asking the operator questions. The Q option has the following format:

```
-Q
```

If the file is marked as needing recovery, or has a non-zero Open For Modify Count, as discussed in Basic File Information Display (on page 590), then it will be recovered. Otherwise, no action occurs. This behavior can be modified by using the Y option. The default is to display information and ask questions, which must be answered by the operator.

**T** Use the T option to indicate that unused space should be truncated, and returned to the operating system. The T option has the following format:

```
-T
```

Specifying the T option will result in a minimal size indexed file, but may reduce performance if records are subsequently added to the indexed file. The default is not to truncate the file. When the file is not truncated, any empty blocks remain part of the file and are available for use in adding new records to the file.

**Note** Some versions of UNIX do not support the operating system call required to truncate a file.

**Y** Use the Y option to cause **recover1** to assume that the operator wants to answer "y" to all possible questions and therefore not stop to wait for a response. The Y option has the following format:

```
-Y
```

Using the Y option will cause a file to be recovered even if it is not marked for recovery, including the case of when the Q option is also specified. The default is to wait for a response from the operator after a question is displayed.

**Z** Use the Z option to reset the Open For Modify Count to zero, as discussed in Basic File Information Display (on page 590), without performing a full recovery. The Z option has the following format:

```
-Z
```

If the file is marked as needing recovery, the Z option is ignored. The default is to treat a non-zero Open For Modify Count as indicating that the file needs recovery.

**Note** Use the Z option with caution. Resetting the Open For Modify Count to zero without performing a full recovery may leave the file in a corrupted state.

## Recovery Process Description

If the **recover1** program is successful, the exit code is set to 0. If the **recover1** program is canceled by the operator, the exit code is set to 2. Otherwise, the exit code is set to 1.

You may produce a list of the support modules loaded by the **recover1** program by defining the environment variable RM_DYNAMIC_LIBRARY_TRACE. The listing will indicate which Terminal Interface support module is used and whether the Automatic Configuration File module is present. This information is most helpful when attempting to diagnose a problem with support modules.

**Note** The information will be visible only if you enter the **recover1** command without any parameters. In this case, **recover1** will show the proper form for the command and the list of support modules.

The **recover1** program attempts to recover the indexed file in place; that is, the program rebuilds the internal file structure in the actual file being recovered. If necessary, the Key Information Block (KIB) is rebuilt and any corrupted data blocks are repaired. Corrupt data blocks may result in loss of some data records. Because

of this feature, it is strongly recommended that you either backup the file or copy the indexed file to be recovered to some other directory or pathname as additional security. Any records that cannot be successfully reindexed into the file due to invalid duplicate key values, or invalid record sizes, are decompressed (if compression is selected for the file), converted to the native code set, and then written to *drop-file*. **recover1** should be able to handle most kinds of indexed file corruption problems, but some fatal errors may still cause the recovery to fail. Any fatal error is displayed and causes the program to terminate. Broken hardware should be suspected in many of these cases.

*drop-file* can be in fixed- or variable-length format; this is set by **recover1** based on whether *indexed-file* is fixed- or variable-length format. Records placed in *drop-file* were those undergoing change at the time of the system failure that required recovery or have invalid record sizes. Investigate any records appearing in *drop-file* and make the appropriate corrections to *indexed-file*.

The **recover1** program's processing consists of up to four separate phases, which are run in the following order:

1. **Integrity Scan.** If the *-q* option or *-y* option is specified, the Integrity Scan phase is disregarded unless it is forced to occur by the specification of the *-i* option or *-l* option. This phase reads the entire file in a forward direction checking for simple errors, and produces a summary report showing the state of the file and an estimate of the number of records **recover1** can recover. The indexed file is not modified during this phase.
2. **Repair Blocks.** The Repair Blocks phase, which is always run, reads and writes the file in a backward direction repairing corrupt data blocks, converting non-data blocks to empty blocks, and rebuilding some internal file structures.
3. **Move Data Blocks.** The Move Data Blocks phase is run only when the truncate file option (*-t*) is specified. This phase reads and writes parts of the file moving high-numbered data blocks (near the end of the file) to lower-numbered available blocks to maximize the amount of space at the end of the file that can be truncated and returned to the operating system when **recover1** finishes.
4. **Build Node Blocks.** The Build Node Blocks phase, which is always run, reads data blocks and writes node blocks in the file in a forward direction rebuilding the entire node structure for each key of the file.

**Note 1** After the Integrity Scan phase, if the Estimated Recoverable records value is zero or very low, and the number of corrupt data blocks is very close to the total number of data blocks found, the number of keys that allow duplicates may be incorrect, either because the KIB is corrupt or the user provided incorrect key information to **recover1**.

**Note 2** After the Integrity Scan phase, if most of the blocks are invalid, the Disk Block Size or the Disk Block Increment may have been incorrectly specified or the KIB may be corrupt.

**Note 3** During the Repair Blocks phase, a count of blocks that could be read but not written may be displayed. This count may indicate the presence of a hardware problem with your disk.

## Recovery Support Module Version Errors

During initialization, the recovery utility locates and loads various support modules, including the automatic configuration module, and, on UNIX, either the terminfo or the termcap Terminal Interface support module. Also, at initialization, the recovery utility verifies that each support module is the correct version for the recovery utility. If a support module is not the correct version, the following message is displayed:

```
RM/COBOL:  module-name version mismatch, expected 8.0n.nn,
   found n.nn.nn.
```

When the previous message is displayed, the recovery utility terminates with the following message:

```
Recover1:  Error invoking mismatched recover1 and
   support module.
```

## Recovery Example

An example run through the Indexed File Recovery utility is described in Figure 44 through Figure 47. The recovery session is started in this example by the following command:

```
recover1 master.inx dropout1
```

Figure 44 shows information about the file **master.inx**.

Under the name of the file to be recovered, a description of the state of the file is displayed. Any of the following messages may appear:

- This file has not been marked as needing recovery!
- The Open For Modify Count for this file is not zero: *count*
- File has been marked as corrupted due to a previous error.
- KIB is corrupt. Using template file: *template-file*
- KIB is corrupt. Enter a template filename (press Enter for manual entry).

**WARNING** Because of several changes in the RM/COBOL 7.5 runtime system, it is possible that an indexed file created by performing an OPEN OUTPUT in a COBOL program using the RM/COBOL 7.5 or later runtime system may have a different block size than a file you are attempting to recover. This may happen even though you specify the same file control entry and file description entry for the template file as when you initially created the file with an earlier version of RM/COBOL. Attempting to use this new file with a different block size as a template file may result in a loss of a large percentage of the recoverable records. You should verify that the block size of your template file is correct by using the Map Indexed File (rmmapinx) utility (on page 589). You can also avoid this problem by creating the template file with a version of RM/COBOL prior to 7.5 or by using a backup copy of the undamaged file. The MINIMUM-BLOCK-SIZE and ROUND-TO-NICE-BLOCK-SIZE keywords of the RUN-INDEX-FILES configuration record (on page 320) may also be used to cause the runtime to create a file with a block size that matches releases prior to 7.5.

If the KIB is corrupt, and a template filename is not entered, **recover1** will prompt the user for the required KIB information before continuing.

If more keys exist than can appear on this screen, as many as possible appear, after which you are asked if you want to see the remaining key descriptors. This continues until all keys are shown. You are then asked to verify that this is the file you want to recover. Entering N terminates the program. Entering Y continues the program.

**Figure 44: Indexed File Recovery Utility: File Recovery Verification**

```
                Indexed File Recovery Utility
                  Recover1 Version 9.0n.00

Indexed File: master.inx
This file has not been marked as needing recovery!

Disk Block Size:        1024      Minimum Record Length:    80
Disk Block Increment:   1024      Maximum Record Length:    80
Number of Index Blocks:  170      Number of Records:       150

   Key Position Size  Remarks
 PRIME      1      8
     1      9      8
     2     17      8  duplicates allowed



Is this the file you wish to recover (y/n)?
```

Figure 45 shows a summary of the information that is gathered during the file integrity scan. You are then asked if you would like to proceed with the recovery process. Entering N terminates the program. Entering Y continues the program.

The "Average record length" is computed by adding the length of all the records in the file and dividing by the number of records. The "Average data size" is computed by adding the size that the record actually occupies in the file and dividing by the number of records. This size allows you to determine how much your data can be compressed.

**Figure 45: Indexed File Recovery Utility: recover1 Summary**

```
                Indexed File Recovery Utility
                  Recover1 Version 9.0n.00

Indexed File: master.inx
Drop  File: dropout1
This file has not been marked as needing recovery!

Disk Block Size:        1024    Minimum Record Length:      80
Disk Block Increment:   1024    Maximum Record Length:      80
Number of Index Blocks:  170    Number of Records:         150
Phase:  Integrity Scan          Estimated Recoverable:     150

             |   Total |   Total |   First |    Last |
Block Type   |   Found | Corrupt | Corrupt | Corrupt |
  KIB        |       1 |       0 |         |         |
  Data       |     102 |       0 |         |         |
  Node       |      61 |       0 |         |         |
  Empty      |       6 |       0 |         |         |
  Invalid    |       0 |       0 |         |         |
  Unreadable |       0 |       0 |         |         |

Average data size: 14, Average record length: 80

Do you wish to proceed with recovery (y/n)?
```

Figure 46 shows the information that is displayed while **recover1** is rebuilding the node blocks for the prime key.

**Figure 46: Indexed File Recovery Utility: recover1 Statistics**

```
                Indexed File Recovery Utility
                  Recover1 Version 9.0n.00

Indexed File: master.inx
Drop  File: dropout1
This file has not been marked as needing recovery!

Disk Block Size:        1024       Minimum Record Length:   80
Disk Block Increment:   1024       Maximum Record Length:   80
Number of Index Blocks:  170       Number of Records:      150
Phase:  Build Node Blocks          Estimated Recoverable:  150

Key being processed:                          PRIME
Records recovered:                            100
Records written to drop file:
Block being processed:                         13
Number of data blocks moved (for truncate):     5
```

Figure 47 shows the information that is displayed after **recover1** terminates successfully. The two lines regarding truncation are shown only when the *-t* option is specified.

**Figure 47: Indexed File Recovery Utility: recover1 Finished Successfully**

```
                Indexed File Recovery Utility
                  Recover1 Version 9.0n.00

Indexed File: master.inx
Drop  File: dropped
This file has not been marked as needing recovery!

Disk Block Size:        1024       Minimum Record Length:  126
Disk Block Increment:   1024       Maximum Record Length:  126
Number of Index Blocks:  120       Number of Records:      100
Phase:  Build Node Blocks          Estimated Recoverable:  100

Key being processed:              PRIME
Records recovered:                  100
Records written to drop file:
Block being processed:              120
Truncate option specified - number of data blocks moved:   4
Truncate action successful - new Number of Index Blocks: 112

Recovery successful.
```

In the example shown in Figure 48, the KIB of the file has been corrupted, and key information must be entered for the file to be recovered. Key information can be obtained from the output of the Map Indexed File (rmmapinx) utility (on page 589). Underlined characters have been entered by the user.

The recovery session is started by the following command:

```
recover1 master.inx dropout1 -k
```

**Note** Entering incorrect information about how many keys, or which keys, can have duplicate values may cause unpredictable results.

**Figure 48: Indexed File Recovery Utility: Entering Key Information**

```
                  Indexed File Recovery Utility
                    Recover1 Version 9.0n.00
Indexed File: master.inx

Last error was 98,38 at 9:29 on 03-21-2004

Are any of the keys in this file segmented (split) (y/n)? y
Key #: PRIME Segment #: 2  Starting Position? 10    Length? 5
            Another Segment (y/n)? n
            Total Key Length =  13  Duplicates Permitted (y/n)? n
Another Key (y/n)? n
```

Figure 49 shows an example of entering the remainder of the KIB information. Underlined characters have been entered by the user.

**Figure 49: Indexed File Recovery Utility: Entering KIB Information**

```
                        Indexed File Recovery Utility
                          Recover1 Version 9.0n.00
Indexed File: master.inx

Last error was 98,38 at 9:29 on 03-21-2004
Minimum Record Length (in bytes)? 80
Maximum Record Length (in bytes)? 80
Disk Block Size (in bytes)? 1024
User Block Size (1=none/2=in bytes/3=in records)? 1
Data Compression (y/n)? y   Space Character Value? 32   Zero Character Value? 48
Key Compression (y/n)? y    Space Character Value? 32
File Version Number (0/2/3/4)? 4        Atomic I/O Enabled (y/n) y
File Lock Limit (in GB)? 2
Disk Block Increment (in bytes)? 1024
Allocation Increment (in blocks)? 8
Force Write Data Blocks (y/n)? n        Force Write Index Blocks (y/n)? n
Force to Disk (y/n)? n                  Force File Closed (y/n)? n
Code Set (1=none/2=ASCII/3=EBCDIC)? 1
Collating Sequence (1=none/2=ASCII/3=EBCDIC)? 1

Is this information correct (proceed with recovery) (y/n)? y
```

After the key and KIB information has been successfully entered, the recovery process proceeds the same as before, beginning with Figure 44, as illustrated on page 605. If a template file had been specified on the command line or a template filename had been entered when prompted, the screens prompting for the key and KIB information would not have been displayed. A template file must be specified if

the KIB is corrupt and the file uses either an enumerated code set or an enumerated collating sequence.

## Recovery Program Error Messages

### Error *status* initializing file manager

**recover1** was unable to initialize the RM/COBOL file management system for the reason indicated by *status*. The usual cause for this error is that a buffer pool has been configured that is too large to be allocated. See the BUFFER-POOL-SIZE keyword of the RUN-FILES-ATTR configuration record (on page 316) for instructions on changing the buffer pool size.

### Truncate option not supported

**recover1** detected that the truncated function was not supported on the system when the user requested file truncation. Truncation of the file is not possible.

### recovery terminating - no records recoverable!

**recover1** detected corruption in the indexed file and no records could be recovered. In this case, **recover1** terminates at the end of the integrity scan to protect the user from erroneously deleting all the records from the file. This error may indicate that the block size, the block size increment, or the number of keys that allow duplicates has been incorrectly specified, or the KIB may be corrupt.

### Error *status* on template file

**recover 1** detected an error in the KIB of the template file specified by the user. The user may enter another template file, may enter the KIB information manually, or may enter a Ctrl-C to terminate **recover 1**.

### Cannot write near end of file - check "ulimit"

**recover1** detected that blocks near the end of the file can be read but not written, but other blocks of the file may be both read and written. This error may indicate that the operating system file size limit (ulimit) may be smaller than the size of the file. Set the file size limit correctly or use an account with sufficient privileges and run **recover1** again.

## Standalone Use of the Recover2 Program

The **recover2** utility program can be used to unload an indexed file to a sequential file. The **recover2** program is no longer needed to recover indexed files. It is invoked by entering the command:

```
runcobol recover2 K [A='file-1 [,[file-2] [,option]']
```

*file-1* is the filename of the indexed file to be unloaded. **recover2** does not use the directory search sequence to locate *file-1*.

*file-2* is the filename of the sequential file into which **recover2** unloads the indexed file records.

*option* can be either SUB or NOSUB, depending on whether or not you want **recover2** to trust the overhead information in the file. NOSUB indicates that the overhead information in the indexed file (record size, block size, and so forth) is correct and can be used to unload the file. SUB indicates that the overhead information may not be correct and needs to be verified and possibly altered. The overhead information appears and you are given the opportunity to alter the information before processing of the file begins. You must enter the value for the "Number of Keys that allow Duplicates", as described in Basic File Information Display (on page 590).

*file-1*, *file-2* and *option* can be omitted, and **recover2** prompts you for a value.

For example, entering the command:

```
runcobol recover2 K A='file-1, option'
```

causes **recover2** to prompt you for the *file-2* filename. Entering the command:

```
runcobol recover2 K A='file-1, file-2'
```

causes **recover2** to prompt you for the option to use.

**recover2** also produces a log duplicating the overhead information appearing on the screen to the printer specified with the environment variable, PRINTER.

If an error occurs, **recover2** displays a message, then continues to attempt to recover the file. The exit code is set to 1.

If no errors occur, the recovery is successful and the exit code is set to 0.

Figure 50 shows the main screen associated with the data unload program (**recover2**). In the first attempt, the NOSUB Option is specified and all of the fields shown are filled in by the program. If the file cannot be unloaded with the NOSUB Option, an error message is displayed and a second attempt is made with the SUB Option specified, causing **recover2** to prompt you for the values by first displaying what is in the file. To select the displayed value, press Enter or Tab. Otherwise, type a new value and press Enter.

**Note** When using the SUB Option, the value for the number of keys that allow duplicates must be entered by the user since there is no default.

**Figure 50: Indexed File Recovery Utility: recover2 Main Screen**

```
         Copy all data records to dropped record file

Indexed File: master.inx
 Drop File: drop
    Option: nosub

     Disk Block Size: 498           Maximum Record Length: 80
Disk Block Increment: 498           Minimum Record Length: 80
                      Number of Keys that allow Duplicates: 1
Data Record Compression (y/n)? Y
        SPACE Character Value: 32
         ZERO Character Value: 48
Records Written to Drop File:
        Block being Processed:  8
```

Figure 51 shows the OK prompt you see during the second attempt. Responding N to this prompt causes the program to restart the prompts for file information. Responding Y indicates that secondary data recovery should start.

**Figure 51: Indexed File Recovery Utility: Secondary Recovery**

```
         Copy all data records to dropped record file

Indexed File: master.inx
 Drop File: drop
    Option: sub

     Disk Block Size: 498            Maximum Record Length: 80
Disk Block Increment: 498            Minimum Record Length: 80
                      Number of Keys that allow Duplicates: 1
Data Record Compression (y/n)? Y
        SPACE Character Value: 32
         ZERO Character Value: 48
Record Written to Drop File:
      Block being Processed:
OK (y/n)?
```

## Recover2 Program Error Messages

**File:** *index-filename* **- Open Error** *status***.**

**recover2** was unable to open the indexed file for the reason indicated by *status*. The execution of **recover2** terminates.

**File:** *index-filename* **- Input Error** *status***.**

**recover2** encountered the error indicated by *status* while reading the indexed file. If the NOSUB Option was chosen, execution terminates. If the SUB Option was chosen, the operator is given the option to continue execution.

**File:** *index-filename* **- Premature end of file encountered.**

**recover2** encountered the end of file in an unexpected place while reading the indexed file, and **recover2** may reasonably recover. If the SUB Option was chosen, the operator may be given the option to continue. Otherwise, execution terminates.

**File:** *index-filename* **- File has never been opened for output.**

The indexed file has never been opened for output and therefore cannot contain any data records.

**File:** *index-filename* **- may not be an Indexed file.**

The overhead structures in the indexed file are not consistent. If the NOSUB Option was chosen, execution terminates. If the SUB Option was chosen, the operator may continue execution. Values that appear for block size and record size should be carefully verified.

**File:** *index-filename* **- Block Size is too big for recover2 program.**

The block size specified is larger than 65280 bytes, which is the largest block size supported by **recover2**. Execution terminates.

**File:** *index-filename* **- Invalid compressed data in block/record** *block number/record-label***.**

**recover2** has encountered a compressed data record that is inconsistent with the version of **recover2** being executed or with the compressed data record length. The indicated record and subsequent records in the block are not written to the unload file. Processing continues with the next block.

**File:** *index-filename* **- Record length mismatch.**

The minimum record size is greater than the maximum record size or the block size is too small for the worst-case record size (due to data compression). It is checked when the NOSUB Option is chosen. Execution terminates.

**File:** *index-filename* **- Bad block overhead in block** *block number***.**

**recover2** has encountered a data block with inconsistent overhead structures. No records from the block are written to the unload file. Processing continues with the next block.

**File:** *index-filename* **- Bad record overhead after block/record** *block number/record label***.**

**recover2** has encountered a data record with inconsistent overhead structures. Any records in the block subsequent to the indicated record are not written to the unload file. Processing continues with the next block.

**File:** *index-filename* **- Record size {< minimum | > maximum} for block/record** *block number/record label***.**

**recover2** has encountered a data record that does not conform to the constraints of minimum or maximum record length but is otherwise consistent. The indicated record is written to the unload file. Processing continues with the next record.

**File:** *unload-filename* **- Record Size is too big for recovery program.**

The maximum record length specified is too large for **recover2**. The maximum record length is 65280 bytes. Execution terminates.

**File:** *unload-filename* **Error:** *status***.**

The indicated error was encountered during an I/O operation on the unload file. Execution terminates.

# Attach Configuration (rmattach) Utility

The Attach Configuration (**rmattach**) utility attaches a configuration file, as discussed in Automatic and Attached Configuration Files (on page 282), to an RM/COBOL command processor **.exe** file and creates a new command processor file. The following processors can be configured in this manner:

- runtime system (**runcobol.exe**)
- compiler (**rmcobol.exe**)
- Indexed File Recovery utility (**recover1.exe**)

**Note** The **rmattach** utility will work only with the Windows version of RM/COBOL. The use of an attached configuration file is no longer recommended. Attached configurations cannot be easily determined to be present or reviewed when present, are applicable to Windows only, and may be disallowed by future versions of the Windows operating system for security reasons. The Automatic Configuration File (on page 282) module has none of these problems and is recommended for all future development of RM/COBOL applications.

## Using the Utility

The Attach Configuration utility is executed with this command:

```
rmattach input-exe-file output-exe-file config-file
```

*input-exe-file* is the name of the command processor file to be configured. This file will not be modified.

**Note** The RUNPATH directory search sequence is not used to open this file. If *input-exe-file* is not in the current directory, the entire pathname must be entered.

*output-exe-file* is the name of the newly configured **.exe** file. If a file with the same name already exists, it is overwritten. *output-exe-file* must not have the same name as that specified for *input-exe-file*. *output-exe-file* should have a filename extension of **.exe**.

*config-file* is the name of a line sequential file containing the desired configuration records, as described in Chapter 10: *Configuration* (on page 281). If a configuration file is already attached to *input-exe-file*, it will be replaced by the contents of *config-file*.

**rmattach** does no syntax checking of the configuration file. The **.exe** file will diagnose errors during its initialization.

Here is an example of an **rmattach** command:

```
rmattach runcobol.exe runcfg.exe setting.cfg
```

# Initialization File to Windows Registry Conversion (ini2reg) Utility

The Initialization File to Windows Registry Conversion (**ini2reg**) utility converts an RM/COBOL Windows initialization file (**.ini**) and places its contents into the Windows registry database.

**Note** When using this utility, several Windows registry issues must be considered if the RM/COBOL for Windows runtime executable has been renamed. For more details, see Windows Registry Considerations (on page 67).

Previous versions of RM/COBOL for Windows stored program configuration information for the compiler, runtime, and Indexed File Recovery utility (**recover1.exe**) programs in separate **.ini** files. Beginning with version 6.5, RM/COBOL uses the Windows registry database to store this information. With this utility, earlier **.ini** files can be converted and current initialization information can be distributed to end-users by using a text file (with the **.reg** extension) that can be exported from the Windows registry database.

**Note** This utility is available only under Windows.

## Using the Utility

The Initialization File to Windows Registry Conversion utility is executed by clicking on the INI to Registry icon or typing the command:

```
ini2reg [-q] [file-name]
```

By default, the **ini2reg** utility program converts text in the initialization file from the OEM character set to the ANSI character set. If the initialization file already used an ANSI character set, use the *-q* option to disable this conversion.

*file-name* is the name of the initialization file to be merged into the Windows registry database. If no file is specified, a File Open dialog box appears in order to browse the file system for the proper file. *file-name* must be specified with a proper path. This utility does not search the PATH or RUNPATH environment variables.

# RM/COBOL Configuration (rmconfig) Utility

The RM/COBOL Configuration (**rmconfig**) utility provides a way to modify the configuration options for one or more COBOL programs. These options are specific to the RM/COBOL system running under Windows.

A modified version of the **rmconfig** property sheet can be displayed for a single COBOL program by right-clicking the program icon, Registry Configuration, and choosing Properties.

**Note** This utility is available only under Windows.

## Using the Utility

The RM/COBOL Configuration utility is executed by clicking on the Registry Configuration icon or typing the command:

```
rmconfig [-r|-c|-y] [-k key] [file]
```

*-r* indicates that you initially will be configuring properties to be used while running programs. This is the default if neither *–r*, *-c*, or *–y* are specified.

*-c* indicates that you initially will be configuring properties to be used while compiling programs.

*-y* indicates that you initially will be configuring properties to be used while recovering data files with the Indexed File Recovery (recover1) utility (on page 598), **recover1.exe**. *–1* is also accepted for backward compatibility.

*-k* key sets a custom key for the Windows registry. Setting a custom registry key is normally required only if you renamed the compiler, runtime system, or recovery utility. A description of how the key is selected if this option is not specified is given below.

**Note** Several Windows registry issues must be considered if the RM/COBOL for Windows runtime executable has been renamed. For more details, see Windows Registry Considerations (on page 67).

*file* is the optional name of the file that you initially wish to configure.

Figure 52 shows the Select File tab of the Properties dialog box. The Select File tab allows the specification of configuration options for a selected COBOL program (Individual File option) or for all programs (Default Properties option). Changes made on the other Properties tabs will affect the configuration of the program selected here. For descriptions of the other Properties tabs, see the topics under Setting Properties (on page 68).

If a key is not specified, the name of the program that is stored in the following location in the registry is used as the default key. The keys listed below are created during installation and are used to determine the default action that occurs when you double-click on a COBOL program or source file in Windows Explorer.

- *-r*: HKEY_CLASSES_ROOT\RMCOBOL.Object\shell\open\command
- *-c*: HKEY_CLASSES_ROOT\RMCOBOL.Source\shell\open\command
- *-y*: The default key is **recover1**, that is, the default key is not obtained from the registry in this case.

**Note** For a default installation, the default key for *–r* is **runcobol** and for *–c* is **rmcobol**.

**Figure 52: Select File Tab**

# Appendix H: Object Versions

This appendix describes the new object features that are incompatible with earlier releases of RM/COBOL-8X and RM/COBOL.

## Level Numbers

The object version level number in a RM/COBOL object file identifies the earliest release of the RM/COBOL product that supports the features required by the program. The set of features available in the first release of the product, RM/COBOL-8X, has been assigned object version 1. When new features have been added in subsequent releases of RM/COBOL, these have been assigned successive object version numbers. The RM/COBOL compiler marks each object file with the object version number of the latest features actually used in the source program, but not less than version 7.

Every release of the RM/COBOL runtime system supports features up to some object version level. Object files with a higher object version level number cannot be run. When a program is named in the Runtime Command or in a CALL statement, the runtime system searches for an object file containing a program with the specified name. If during this search, the runtime system finds an object program that has an object version level number which exceeds that accepted by the runtime system, that object program is not considered valid and the runtime system continues its search. If no valid program with the specified name is found, the ON EXCEPTION phrase of the CALL statement is taken or the Runtime Command fails.

The RM/COBOL compiler has an object version level option to control the object version level number placed in the object file. When the option is specified, any language features used in the source program requiring a higher object version cause a source diagnostic, and the program is not compiled. The object version level number placed in the resulting object file is no higher than the value specified in the compiler option. If the object version level option is not specified, the default value is the highest value accepted by the compiler, thus allowing all features supported by the compiler. Since the compiler marks the object program with the value of the latest feature actually used in the source program, the resulting object program may still be executable on earlier releases of the runtime system.

Table 58 enumerates past RM/COBOL product releases and the highest object version level number they accept.

**Table 58: Object Version Numbers by Product**

| Product | Platform | Releases | Object Level Number |
|---|---|---|---|
| RM/COBOL-8X | DOS | 1.*nn* | 1 |
| RM/COBOL | DOS<br>UNIX | 2.*nn*<br>2.0*n* | 2<br>2 |
| RM/COBOL | AS/400 | | 3 |
| RM/COBOL | DOS<br>UNIX | 4.*nn*<br>4.0*n* | 4<br>4 |
| RM/COBOL | DOS<br>UNIX | 5.*nn*<br>5.*nn* | 5<br>5 |
| RM/COBOL | DOS<br>UNIX | 5.2*n*<br>5.2*n* | 6<br>6 |
| RM/COBOL | DOS<br>UNIX<br>Windows | 6.*nn*<br>6.*nn*<br>6.*nn* | 7<br>7<br>7 |
| RM/COBOL | UNIX<br>Windows | 7.*nn*<br>7.*nn* | 8<br>8 |
| RM/COBOL | UNIX<br>Windows | 7.5*n*<br>7.5*n* | 9<br>9 |
| RM/COBOL | UNIX<br>Windows | 7.50.01*n*<br>7.50.01*n* | 10<br>10 |
| RM/COBOL | UNIX<br>Windows | 8.0*n*<br>8.0*n* | 11<br>11 |
| RM/COBOL | UNIX<br>Windows | 9.0*n*<br>9.0*n* | 12<br>12 |

In most cases, the object version is of no concern to the user. However, when compiling programs intended for distribution to other users, the object version may be of concern. If these other users do not have the current release of the runtime system, the Z Compile Command Option (see page 149) should be specified to restrict the object version level. The object version level number specified to the compiler should be the highest value that does not exceed the level accepted by any of the runtime systems used by the intended recipients. When the object version level number is limited by use of the Z Option, the compiler suppresses any optimizations and diagnoses all source language features not supported by earlier runtime systems.

The features associated with each object version are described in the following sections.

# Object Version 1

The RM/COBOL-8X compiler and runtime system versions 1.*nn* support features in object version 1. These product releases implement the high subset of ANSI COBOL 1974.

If the object version level number is limited to 1 by use of the Z Compile Command Option, the resulting object program is executable on any released RM/COBOL-8X or RM/COBOL runtime system.

# Object Version 2

The RM/COBOL compiler and runtime system versions 2.*nn* support object version 2. These product releases implement the intermediate subset of ANSI COBOL 1985. The RM/COBOL version 2.*nn* runtime systems support the language features of both object version 1 and object version 2.

Several new language features were added in object version 2 that are not supported by previous versions. Unless the object version is limited to 1 by the Z Compile Command Option, programs with the following features will not execute on runtime systems with a version number less than 2:

1. A source program with a simple INSPECT statement.

   A simple INSPECT statement is one with single-character control operands and only a single CHARACTERS, ALL, LEADING or FIRST phrase per TALLYING or REPLACING phrase. A simple INSPECT statement may have both a TALLYING and REPLACING phrase and both a BEFORE and AFTER INITIAL phrase.

   The RM/COBOL version 2.*nn* and later compilers generate optimized code for simple INSPECT statements. The optimized code is not supported by earlier runtime systems. If the Compile Command options specify object version 1, the previous unoptimized code is generated for a simple INSPECT statement.

2. A source program with a NUMERIC class condition that has an unsigned numeric display operand.

   The RM/COBOL version 2.*nn* and later compilers generate optimized code for a NUMERIC class condition the operand of which is an unsigned numeric display data item. The optimized code is not supported by earlier runtime systems. If the Compile Command options specify object version 1, the previous unoptimized code is generated for such a class condition.

3. In COBOL-85 compatibility mode, FILE STATUS clauses imply that ANSI COBOL 1985 I-O status values are expected. RM/COBOL-8X version 1.*nn* runtime systems never produce ANSI COBOL 1985 I-O status values.

   If the Compile Command options specify object version 1 and do not specify ANSI COBOL 1974 compatibility mode, FILE STATUS clauses are diagnosed as an object version incompatibility and are ignored.

4. The following RM/COBOL new language features require new runtime system support for their implementation. If the Compile Command options specify object version 1, these language features are diagnosed as an object version incompatibility and are compiled:

   a. Reference modification.

   b. The PADDING CHARACTER clause.

   c. The RECORD DELIMITER clause.

   d. The VALUE clause with or subordinate to OCCURS clauses.

   e. A class-name, ALPHABETIC-LOWER, or ALPHABETIC-UPPER class condition.

   f. A CD FOR I-O referenced in a DISABLE, ENABLE, RECEIVE or SEND statement.

   g. An ACCEPT . . . FROM DAY-OF-WEEK phrase.

h. A CALL . . . USING phrase that references a subscripted identifier.

i. A DISPLAY . . . WITH NO ADVANCING phrase.

j. An INSPECT . . . CONVERTING phrase.

k. A MERGE . . . GIVING phrase that specifies two or more files.

l. An OPEN EXTEND phrase that references a relative or indexed organization file.

m. The PURGE statement.

n. A SEND . . . REPLACING LINE phrase.

o. A SORT . . . WITH DUPLICATES IN ORDER phrase.

p. A SORT . . . GIVING phrase that specifies two or more files.

**Note** Several important language features added in the version 2.0*n* releases of RM/COBOL do not generate object version 2 code for their implementation. Examples of such features are EVALUATE, INITIALIZE and the NOT conditional phrases (NOT AT END, NOT ON SIZE ERROR, and so forth). These new language features may be used whether or not the object version is restricted to object version 1 without affecting the object version of the resulting object program.

# Object Version 3

The RM/COBOL compiler and runtime system versions 3.*nn* support object version 3. These product releases implement the high subset of ANSI COBOL 1985. The RM/COBOL version 3.*nn* runtime systems support the language features of object versions 1, 2, and 3.

Two language features were added in object version 3 that are not supported by the earlier versions. These are as follows:

1. The EXTERNAL clause in file-description-entries and Working-Storage Section record-description entries.
2. Nested programs, including the PROGRAM IS COMMON clause, the GLOBAL file description entry clause, the GLOBAL data description entry clause and the GLOBAL phrase of the USE statement.

**Note** Programs with either of these two features will not execute on runtime version 1.*nn* or 2.*nn* systems. Since these features required new runtime system support, you cannot compile them if the object version level is restricted to 1 or 2 with the Z Compile Command Option. If the object version level is restricted to 1 or 2, the RM/COBOL compiler diagnoses these features as an object version incompatibility.

The new language feature CALL . . . USING BY CONTENT does not generate new object code. Programs using this feature can be compiled when the object version level is restricted to 1 or 2 on the Z Compile Command Option.

When compiling a sequence of programs not separated by END PROGRAM headers, you must restrict the object version level to 1 or 2. Versions 3.00 and later of the RM/COBOL compiler interpret such a sequence as nested programs.

# Object Version 4

The RM/COBOL compiler and runtime system versions 4.*nn* support object version 4. These product releases extend the high subset of ANSI COBOL 1985 with the X/Open Screen Section. The RM/COBOL version 4.*nn* runtime systems support the language features of object versions 1 through 4.

Four language features were added in object version 4 that are not supported by the earlier versions. These are as follows:

1. ACCEPT or DISPLAY statements that reference screen-names defined in the new Screen Section of the Data Division.
2. ACCEPT statements that specify the FROM ESCAPE KEY or the FROM EXCEPTION STATUS phrase.
3. CALL PROGRAM statement.
4. DELETE FILE statement.

**Note** Programs with any of these four features will not execute on runtime version 3.*nn* or earlier systems. Since these features required new runtime system support, you cannot compile them if the object version level is restricted to 3 or less with the Z Compile Command Option. If the object version level is restricted to 3 or less, the RM/COBOL compiler diagnoses these features as an object version incompatibility.

# Object Version 5

The RM/COBOL compiler and runtime system versions 5.*nn* support object version 5. The RM/COBOL version 5.*nn* runtime systems support the language features of object versions 1 through 5.

Two language features were added in object version 5 that are not supported by the earlier versions. These are as follows:

1. A READ statement that specifies the PREVIOUS phrase for a relative or indexed organization file.
2. A START statement that specifies a KEY IS LESS, a KEY IS NOT GREATER, or a KEY IS LESS OR EQUAL relation.

**Note** Programs with either of these two features will not execute on runtime version 4.*nn* or earlier systems. Since these features required new runtime system support, you cannot compile them if the object version level is restricted to 4 or less with the Z Compile Command Option. If the object version level is restricted to 4 or less, the RM/COBOL compiler diagnoses these features as an object version incompatibility.

# Object Version 6

The RM/COBOL compiler and runtime system versions 5.2*n* support object version 6. The RM/COBOL version 5.2*n* runtime systems support the language features of object versions 1 through 6.

Two language features were added in object version 6 that are not supported by earlier versions. These are as follows:

1. An ACCEPT statement that specifies the TIME phrase.
2. A START statement that specifies an identifier in the SIZE phrase. Even though the SIZE phrase is supported only by the 5.2*n* compiler, a literal specified in the SIZE phrase of the START statement is supported by all RM/COBOL runtime system.

**Note** Programs with either of these two features will not execute on runtime version 5.1*n* or earlier systems. Since these features require new runtime system support, you cannot compile them if the object version level is restricted to 5 or less with the Z Compile Command Option. If the object version level is restricted to 5 or less, the RM/COBOL compiler diagnoses these features as an object version incompatibility.

In addition, performance enhancements for certain existing language features require runtime systems that support object version 6. Most programs compiled with the version 5.2*n* compiler result in object files that require object version 6 support unless the object version is restricted to 5 or less. The performance enhancements requiring object version 6 are the following:

1. Adding a literal value in the range –128 to +127 to an integer binary data item that is within the first 65280 bytes of the program local data area. The addition can be the result of an ADD statement, an INSPECT statement TALLYING phrase, a PERFORM statement VARYING or AFTER phrase, a SEARCH statement VARYING phrase or an UNSTRING statement TALLYING phrase.
2. Subtracting a literal value in the range –127 to +128 from an integer binary data item that is within the first 65280 bytes of the program local data area. The subtraction can be the result of a SUBTRACT statement or a PERFORM statement VARYING or AFTER phrase.
3. PERFORM statement with the TIMES phrase when the local data area for the program, including compiler-generated temporary data items, is less than 65280 bytes in length.
4. Subscripted operands 255 characters or less in length that are elements of tables within the first 65280 bytes of the program local data area that are specified in INITIALIZE, MOVE, READ, RELEASE, RETURN, REWRITE, SET or WRITE statements.
5. Operands, subscripted or not, 255 characters or less in length that are within the first 65280 bytes of the program local data area specified in ACCEPT, CALL, CALL PROGRAM, CANCEL, DISABLE, DISPLAY, ENABLE, INITIALIZE, INSPECT, MOVE, READ, RECEIVE, RELEASE, RETURN, REWRITE, SEND, SET, START, STOP, STRING, UNSTRING and WRITE statements or in class or relation conditions.
6. Nonnumeric relations where the subject and object operands are different length data items, neither operand is subscripted or reference modified other than by literals, both operands are 255 characters or less in length, and both operands are within the first 65280 bytes of the program local data area.

7. GO TO statements in the fixed permanent segments of a program that generates between 32512 and 65280 bytes of object code for the fixed permanent segments.

**Note** The program local data area is the area of storage reserved for File Section, Working-Storage, Communication, and Screen Section data items not described with the external attribute. Linkage Section data items and data items described with the EXTERNAL clause are not part of the program local data area.

# Object Version 7

The RM/COBOL compiler and runtime system versions 6.*nn* support object version 7. The RM/COBOL version 6.*nn* runtime systems support the language features of object versions 1 through 7.

New language features were added in object version 7 that are not supported by earlier versions. These are as follows:

1. A compiler option allows for computational sign representations that are compatible with COBOL-74 data types.
2. Thirty digits of precision are available for numeric data items.
3. The compiler and runtime system provide support for the START statement to specify the FIRST or LAST KEY phrase. For example, START *file-name* KEY IS FIRST *key-name*.
4. The compiler and runtime system provide support for duplicate prime record keys (WITH DUPLICATES may be specified for the RECORD KEY phrase).
5. The compiler and runtime system provide support for split keys. The RECORD KEY phrase and the ALTERNATE RECORD KEY phrase may define split keys.
6. The compiler and runtime system provide support for multiple record locks in the same file. The LOCK MODE clause of the file control entry may now specify the LOCK ON MULTIPLE RECORDS phrase.

**Note** Programs that use any of these features will not execute on runtime versions 5.*n* or earlier systems. Since these features require new runtime system support, you cannot compile them if the object version level is restricted to 6 or less with the Z Compile Command Option. If the object version level is restricted to 6 or less, the RM/COBOL compiler diagnoses these features as an object version incompatibility.

# Object Version 8

The RM/COBOL compiler and runtime system versions 7.*nn* support object version 8. The RM/COBOL version 7.*nn* runtime systems support the language features of object versions 1 through 8.

New language features were added in object version 8 that are not supported by earlier versions. These are as follows:

1. BINARY, COMPUTATIONAL-4 and COMP-4 usage data items allocated as other than two- four-, eight-, or sixteen-bytes as a result of the BINARY-ALLOCATION keyword specification in the COMPILER-OPTIONS configuration record or the binary allocation override specification in the source. If a binary data item is allocated as sixteen bytes because of these new features, object version 7 will be required and generated since it was the first object version that supported sixteen-byte binary.
2. Pointer data items (USAGE POINTER), the figurative constant NULL (NULLS), the ADDRESS special register, and Formats 5 and 6 of the SET statement for manipulating pointer data items.
3. The GIVING or RETURNING phrase in the Procedure Division header or in a CALL statement.
4. The CENTURY-DATE, CENTURY-DAY, DATE-AND-TIME, or DAY-AND-TIME options in the ACCEPT statement.
5. The OMITTED option for an argument in the USING phrase of a CALL statement.
6. A source program that refers to linkage records (01 or 77 level data items defined in the Linkage Section) in the USING phrase of a CALL statement or with reference modification.

   The RM/COBOL version 7.00 and later compilers generate code to reference the data item on which the linkage record is based, that is, the actual argument passed by the calling program or the area of memory referenced by a SET ADDRESS OF statement. This new code is not supported by earlier runtime systems. If the Compile Command options specify object version 7 or lower, then the previous code is generated that uses the description of the data item in the linkage section of the called program.
7. A source program may now use more than 64K of name space (unique spellings of user-defined words), but object versions less than 8 support a maximum of 64K of name space for the object symbol table. If the Y Compile Command Option is specified to place the symbol table in the object file for debugging purposes and more than 64K of name space has been used, object version 8 is required and will be generated regardless of the maximum object version setting specified in the Z Compile Command Option. The compiler generates a diagnostic message in this event.

**Note 1** Programs that use any of these features will not execute on runtime versions 6.*n* or earlier systems. Since these features require new runtime system support, you cannot compile them if the object version level is restricted to 7 or less with the Z Compile Command Option. If the object version level is restricted to 7 or less, the RM/COBOL compiler diagnoses these features as an object version incompatibility.

**Note 2** Several important language features added in the version 7.0 release of RM/COBOL do not generate object version 8 code for their implementation. These

include level-number 78 constant-name declarations, constant-name references, the DATE-COMPILED option in the ACCEPT statement, in-line comments, a numeric literal specified in the VALUE clause for a numeric edited data item, and the COUNT, COUNT-MAX, COUNT-MIN, LENGTH, and PROGRAM-ID special registers. A binary allocation override that specifies two-, four-, or eight-byte allocation may be used for any object version. A binary allocation override that specifies sixteen-byte allocation may be used for object version 7.

# Object Version 9

The RM/COBOL compiler and runtime system versions 7.5*n* support object version 9. The RM/COBOL version 7.5*n* runtime systems support the language features of object versions 1 through 9.

New language features were added in object version 9 that are not supported by earlier versions. These are as follows:

1. The LIKE condition.
2. Operators, other than the equal operator, in format 3 VALUE clauses when the symbol table is being generated into the object (Y Compile Command Option is specified).

**Note 1** Programs that use any of these features will not execute on runtime versions prior to version 7.5. Since these features require new runtime support, you cannot compile them if the object version level is restricted to 8 or less with the Z Compile Command Option. If the object version level is restricted to 8 or less, the RM/COBOL compiler diagnoses these features as an object version incompatibility.

**Note 2** Several language features added in the version 7.5 release of RM/COBOL do not generate object version 9 code for their implementation. These include the new four-digit year formats for the ACCEPT statement (but these do require object version 8), the NOT OPTIONAL phrase in the SELECT clause, the new formats of the EXIT statement, the enhancements to the INITIALIZE statement, and an OPEN mode series in the USE statement.

# Object Version 10

The RM/COBOL compiler and runtime system versions 7.50.01 and later support object version 10. The RM/COBOL version 7.50.01 runtime systems support the language features of object versions 1 through 10.

A new language features was added in object version 10 that is not supported by earlier versions. This is as follows:

- Specification of a variable (non-literal) reference modifier for the pattern of a LIKE condition.

# Object Version 11

The RM/COBOL compiler and runtime system versions 8.0*n* support object version 11. The RM/COBOL version 8.0*n* runtime systems support the language features of object versions 1 through 11.

New language features were added in object version 11 that are not supported by earlier versions. These are as follows:

1. COMPUTATIONAL-5 and COMP-5 usage, that is, machine native binary data format.
2. Empty groups declared when the object symbol table is produced (Compiler Command Option Y) or use of empty groups in the Procedure Division in cases where the compiler does not eliminate them. The compiler eliminates references to empty groups when used as the receiving operand in a MOVE statement.

# Object Version 12

The RM/COBOL compiler and runtime system versions 9.0*n* support object version 12. The RM/COBOL version 9.0*n* runtime systems support the language features of object versions 1 through 12.

New language features were added in object version 12 that are not supported by earlier versions. These are as follows:

1. A source program with more than 65535 lines of Procedure Division or a Procedure Division header with a line number greater than 65535 now produces object version 12 with 32-bit debugging line numbers where needed. Previously, debugging line numbers after 65535 lines of Procedure Division were modulo 65536 (debugging line numbers in the object were a 16-bit offset from the Procedure Division header line number) and the Procedure Division header line number was modulo 65536 in the object. Runtimes (and thus, CodeWatch) prior to version 9 did not support the new code for line numbers with an offset greater than 65535 from the Procedure Division header line or a Procedure Division header line number greater than 65535. This does not apply if the Q Compile Command Option is specified or configured, since debugging line numbers are not generated in this case. If the Z Compile Command Option is specified or configured to restrict the object version to less than 12, more than 65535 lines of Procedure Division or a Procedure Division header line number greater than 65535 causes an object version conflict error followed by a program overflow termination.
2. The CURSOR IS clause in the Special-Names paragraph requires object version 12.
3. The SECURE phrase in a Format 3 ACCEPT statement is interpreted the same as the SECURE clause in a screen section data description entry, that is, input characters are displayed as asterisks for object version 12. If the object version is restricted to less than 12, the prior interpretation of SECURE in a Format 3 ACCEPT statement as a synonym of OFF is provided, that is, input characters are not displayed. The old interpretation of SECURE as OFF makes it an intensity specification, of which there may only be one. The new interpretation allows intensity and SECURE as independent options.

4. Support has been added for an extremely large number of file parameters (FILE STATUS, RELATIVE KEY, PADDING CHARACTER, LINAGE, etc.). The new limit is about four times what was supported in prior object versions. When the compiler detects that the old limit has been exceeded, object version 12 is generated. If the object version is restricted to less than 12, a program overflow occurs if the old limit is exceeded.
5. Relaxed reference modification at runtime will occur only for runtimes that support object version 12 and later, since prior runtimes enforced the strict reference modification rules; there is no compiler diagnostic for this runtime dependency issue. Compile time reference modification, that is, with literals, can be relaxed or strict per the configuration with no effect on the object version.

**Note** Several language features added in the version 9 release of RM/COBOL do not generate object version 12 code for their implementation. These include the SUPPRESS phrase in the COPY statement, WHEN-COMPILED special register, CONSOLE IS CRT clause in the Special-Names paragraph, CRT STATUS clause in the Special-Names paragraph, user-defined words longer than 30 characters, and the ACCEPT and DISPLAY statement syntax enhancements.

# Appendix I: Extension, Obsolete, and Subset Language Elements

RM/COBOL supports the extension, obsolete, and subset language elements discussed in this appendix. Each language element is only briefly described in order to identify it. For further information on each language element, refer to the *RM/COBOL Language Reference Manual*.

The extension language elements are RM/COBOL extensions to the American National Standard COBOL X3.23-1985. Extensions such as the ACCEPT and DISPLAY screen control syntax simplify the use of COBOL in an interactive environment. Other extensions relax some of the rules of COBOL to simplify program writing.

The obsolete language elements are language features declared obsolete in ANSI COBOL 1985. The standard has declared certain features as obsolete to indicate that they will be removed in the next full revision of COBOL. The features declared obsolete are ones, such as the ALTER statement, that have been identified as contributing to poor programming practices. Obsolete features should be avoided in new programs and removed from existing programs when they are revised.

The subset language elements are language features that are required only when more than the minimum COBOL language is implemented. Above minimum COBOL, there are two additional subsets defined by the standard: intermediate and high. In addition, RM/COBOL supports two standard optional modules: segmentation and communication. Each of these optional modules is further divided into a level 1 and level 2 subset, where the level 2 subset includes the level 1 subset.

The F Compile Command Option (see page 150) contains a flagging option to cause flagging of the occurrence of any of the items in the following lists. Each of the lists is preceded by an explanation of the flagging message produced for items on that list.

# Extension Elements

The warning message:

```
W   69:   FIPS NONCONFORMING NONSTANDARD
```

is produced for each of the following language elements if they appear in a source program compiled with the F=EXTENSION Compile Command Option (see page 150). Each item on the list is an RM/COBOL extension to the American National Standard COBOL X3.23-1985 language features and may, therefore, not be available in other COBOL implementations.

Many extensions noted for RM/COBOL-8X do not appear below. These extensions have not been deleted, but have been incorporated as standard features in the American National Standard COBOL X3.23-1985.

The extensions are as follows:

1. User-defined word with more than 30 characters.
2. More than seven subscripts.
3. In-line comments (*>*comment-entry*).
4. *integer-1* + *integer-2* as a subscript (literal subscript with relative offset) or zero used for relative offset in a subscript.
5. Numeric literals with more than 18 digits.
6. Numeric data items with more than 18 digits of precision.
7. Nonnumeric literals greater than 160 characters in length.
8. Text-names and library-names specified as nonnumeric literals.
9. COPY statement within a copied file.
10. COPY statement with the SUPPRESS PRINTING phrase.
11. User-defined word ending in a hyphen.
12. Reserved words used as system-names (the ASSIGN clause in the file control entry, the VALUE OF clause in the file description entry, and the ENTER statement).
13. Use of an index-name in subscripting a table other than the one with which it is associated.
14. Apostrophe used as a delimiter for nonnumeric literals.
15. ALL [ALL] . . . *literal* form of a figurative constant.
16. Procedure-name that is the same as a data-name or index-name.
17. NULL or NULLS figurative constants.
18. Hexadecimal literal.
19. ADDRESS special register.
20. COUNT, COUNT-MAX, and COUNT-MIN special registers.
21. LENGTH special register.
22. PROGRAM-ID special register.

23. WHEN-COMPILED special register.
24. constant-name reference (a constant-name is defined in a level-number 78 data description entry).
25. RETURN-CODE special register.
26. ID abbreviation for IDENTIFICATION.
27. Identification Division paragraphs out of order.
28. Program-name specified as a nonnumeric literal in the PROGRAM-ID paragraph.
29. REMARKS paragraph in the Identification Division.
30. OBJECT-COMPUTER paragraph optional clauses out of order.
31. Special-Names paragraph clauses out of order.
32. Three or more ON or OFF STATUS phrases for a switch clause in the Special-Names paragraph after two non-duplicating ON or OFF phrases.
33. ALPHABET keyword missing in the SPECIAL-NAMES paragraph when an alphabet-name appears.
34. Repeated character in an ALPHABET literal.
35. ALPHABET literal THRU phrase on ALSO phrase.
36. ALPHABET literal ALSO phrase on THRU phrase.
37. SYMBOLIC CHARACTERS clause specified as SYMBOLIC CHARACTER, that is, CHARACTER used as a synonym for CHARACTERS.
38. CURRENCY SIGN literal specified as a figurative constant.
39. NUMERIC SIGN clause in the Special-Names paragraph.
40. CONSOLE IS CRT clause in the Special-Names paragraph.
41. CRT STATUS clause in the Special-Names paragraph.
42. CURSOR clause in the Special-Names paragraph.
43. ALTERNATE RECORD KEY clause that specifies that split-key-name option.
44. ASSIGN TO clause with data-name specified for file-access-name.
45. CODE-SET clause in the file control entry.
46. CODE-SET clause that refers to an alphabet-name defined with the literal phrase.
47. CODE-SET clause specified for a relative or indexed file.
48. COLLATING SEQUENCE clause in the file control entry.
49. LOCK MODE clause in the file control entry.
50. ORGANIZATION clause that specifies LINE or BINARY.
51. RESERVE clause that specifies NO or ALTERNATE.
52. RECORD KEY clause that specifies the DUPLICATES phrase.
53. RECORD KEY clause that specifies the split-key-name option.
54. SELECT clause that contains the NOT OPTIONAL phrase.
55. Optional word IS in POSITION phrase of MULTIPLE FILE TAPE clause.

56. SCREEN SECTION in the Data Division.
57. LINAGE clause integer operand with a positive sign explicitly specified.
58. Qualification of the data-name in the RECORD IS VARYING DEPENDING ON clause.
59. Level-number with three or more digits.
60. Level-number 78 data description entry.
61. Declaration of an empty group.
62. OCCURS clause specified in an 01 or 77 level-number data description entry in the Working-Storage Section.
63. DEPENDING phrase specified in OCCURS clause that omits [TO *integer-2*].
64. More than 30 characters in a PICTURE character-string.
65. PICTURE clause omitted and thus implied in an elementary data description entry with a VALUE clause (the flag occurs on the following level-number or header since that is what makes the data item elementary in the absence of a PICTURE clause).
66. PICTURE character-string that ends in a comma or period and is not immediately followed by a period space separator.
67. REDEFINES of last name on same level, even though it is also a REDEFINES.
68. REDEFINES not first clause in a data description entry.
69. SYNCHRONIZED clause specified with USAGE INDEX in a data description entry.
70. USAGE COMP-1, COMP-3, COMP-4, COMP-5, COMP-6, COMPUTATIONAL-1, COMPUTATIONAL-3, COMPUTATIONAL-4, COMPUTATIONAL-5, and COMPUTATIONAL-6.
71. USAGE POINTER.
72. USAGE clause that specifies a binary allocation override (*integer-3*).
73. VALUE clause in a data description entry that specifies a relational operator when defining a condition-name.
74. VALUE clause in a data description entry that specifies the WHEN SET TO FALSE phrase.
75. VALUE clause in a data item data description entry in the File Section or Linkage Section or in other than the first record description entry subordinate to a communication description entry in the Communication Section.
76. VALUE clause in a data item data description entry in an external record in the Working-Storage Section record.
77. VALUE clause that specifies a numeric literal for a numeric edited data item.
78. Procedure Division header that specifies the GIVING or RETURNING phrase.
79. END PROGRAM specified without a program-name.
80. END PROGRAM header that specifies a nonnumeric literal for the program-name.
81. Segment-numbers greater than 99.
82. Optional word THEN used as a statement connective.

83. Both operands of a relation being literals.
84. An index-name as one operand in a relation and an arithmetic expression as the other operand.
85. Pointer data item used in a relation condition.
86. LIKE condition.
87. Literal specified in a class condition.
88. Literal specified in a sign condition.
89. Nondisplay data item specified in a NUMERIC class condition.
90. ACCEPT . . . FROM CONSOLE when CONSOLE not defined as a *mnemonic-name* in the SPECIAL-NAMES paragraph.
91. ACCEPT . . . FROM SYSIN when SYSIN not defined as a *mnemonic-name* in the SPECIAL-NAMES paragraph.
92. ACCEPT … FROM CENTURY-DATE or ACCEPT … FROM DATE YYYYMMDD.
93. ACCEPT … FROM CENTURY-DAY or ACCEPT … FROM DAY YYYYDDD.
94. ACCEPT … FROM DATE-AND-TIME.
95. ACCEPT … FROM DATE-COMPILED.
96. ACCEPT … FROM DAY-AND-TIME.
97. ACCEPT . . . FROM ESCAPE KEY statement.
98. ACCEPT . . . FROM EXCEPTION STATUS statement.
99. ACCEPT operand series.
100. ACCEPT statement that specifies a screen-name.
101. ACCEPT with screen control (LINE, POSITION, SIZE, CURSOR, CONTROL, ERASE, TAB, UNIT, PROMPT, UPDATE, ECHO, BLINK, REVERSE, HIGH, LOW, OFF, CONVERT, NO BEEP, ON EXCEPTION, NOT ON EXCEPTION, END-ACCEPT, BEFORE TIME).
102. CALL PROGRAM statement.
103. CALL . . . USING *literal*.
104. CALL … USING pointer data item.
105. CALL … USING OMITTED.
106. CALL … GIVING/RETURNING phrase.
107. CLOSE statement that specifies the NO REWIND phrase with either the REEL or UNIT phrase.
108. DELETE FILE statement.
109. DISABLE statement without INPUT, OUTPUT or I-O phrase.
110. DISPLAY . . . UPON CONSOLE when CONSOLE is not defined as a *mnemonic-name* in the SPECIAL-NAMES paragraph.
111. DISPLAY . . . UPON SYSOUT when SYSOUT not defined as a *mnemonic-name* in the SPECIAL-NAMES paragraph.
112. DISPLAY statement that specifies a screen-name.

113. DISPLAY with screen control (LINE, POSITION, SIZE, CONTROL, ERASE, UNIT, BLINK, REVERSE, HIGH, LOW, CONVERT, BEEP).
114. ENABLE statement without INPUT, OUTPUT or I-O phrase.
115. ENTER statement not ended by a period.
116. EVALUATE statement that specifies an index-name or index data item as a selection subject or selection object.
117. EXIT statement that contains the PARAGRAPH, SECTION, or PERFORM phrases.
118. GOBACK statement.
119. IF statement that specifies END-IF and NEXT SENTENCE.
120. INITIALIZE statement that specifies the FILLER, VALUE, or DEFAULT phrases, the word THEN in the REPLACING phrase, multiple categories in the category-name of the REPLACING phrase, or the DATA-POINTER category in the REPLACING phrase.
121. INITIALIZE statement for which any *identifier-1* refers to a variable-occurrence data item or to a group than contains a variable-occurrence data item.
122. INSPECT . . . TALLYING . . . FOR FIRST phrase.
123. INSPECT statement that refers to an ALL *literal* as a control operand.
124. INSPECT statement that refers to a group data item as a control operand.
125. MOVE CORRESPONDING statement with a receiving operand series.
126. OPEN EXCLUSIVE phrase.
127. OPEN EXTEND statement that refers to a file described with the LINAGE clause.
128. OPEN . . . WITH LOCK phrase.
129. PERFORM *integer-1* TIMES statement where *integer-1* is zero or signed.
130. In-line PERFORM VARYING statement with AFTER phrases.
131. READ . . . PREVIOUS phrase.
132. READ statement that specifies the WITH NO LOCK or WITH LOCK phrase.
133. RELEASE . . . FROM *literal*.
134. REWRITE . . . FROM *literal*.
135. SEARCH statement that specifies END-SEARCH and NEXT SENTENCE.
136. SEND . . . FROM *literal*.
137. SET statement (Format 1) that specifies more than one TO phrase.
138. SET statement (Format 2) that specifies more than one UP/DOWN BY phrase.
139. SET {*condition-name*} . . . TO FALSE statement.
140. SET statement that specifies more than one instance of the TO TRUE phrase.
141. SET statement (Formats 5 and 6) that refers to a pointer data item.
142. START statement with LESS THAN, LESS THAN OR EQUAL, or equivalent relations.
143. START statement that specifies the SIZE phrase.

144. START statement in which the FIRST or LAST option is specified in the KEY phrase.
145. STOP statement that specifies an identifier instead of the *literal* phrase.
146. STOP RUN statement with RETURN-CODE specified.
147. UNLOCK statement.
148. USE statement that specifies more than one OPEN mode or specifies an OPEN mode and one or more file-names.
149. WRITE . . . FROM *literal*.
150. WRITE . . . ADVANCING TO LINE phrase.

# Obsolete Elements

The warning message:

```
W   71:   FIPS OBSOLETE
```

is produced for each of the following language elements if they appear in a source program compiled with the F=OBSOLETE Compile Command Option (see page 150). Each item on the list is identified in the American National Standard COBOL X3.23-1985 as being an obsolete language element that will be deleted from the next full revision of the COBOL standard.

1. ALL *literal*, where *literal* has a length greater than 1 if associated with a numeric or numeric edited data item.
2. AUTHOR, INSTALLATION, DATE-WRITTEN, DATE-COMPILED and SECURITY paragraphs.
3. MEMORY SIZE clause.
4. RERUN clause.
5. MULTIPLE FILE TAPE clause.
6. LABEL RECORDS clause.
7. VALUE OF clause.
8. DATA RECORDS clause.
9. ALTER statement.
10. KEY phrase of the DISABLE statement.
11. KEY phrase of the ENABLE statement.
12. ENTER statement.
13. GO TO statement with omitted *procedure-name-1*.
14. REVERSED phrase of the OPEN statement.
15. STOP *literal* phrase.
16. Segment-numbers and the SEGMENT-LIMIT clause.

# Subset Elements

The warning message:

```
W   70:   FIPS NONCONFORMING STANDARD
```

is produced for each of the following language elements if they appear in a source program compiled with the appropriate F Compile Command Option (see page 150). The keyword that causes the language element to be flagged is shown in parentheses after each element description. Note that HIGH elements will be flagged if F=INTERMEDIATE or F=HIGH is specified, COM2 elements will be flagged if F=COM1 or F=COM2 is specified, and SEG2 elements will be flagged if F=SEG1 or F=SEG2 is specified.

If obsolete element flagging is also enabled, any obsolete element from the following list will only be flagged as obsolete even when the other keyword is also specified in the F Compile Command Option. That is, obsolete flagging takes precedence over subset flagging.

1. Segment-number (SEG1, OBSOLETE).
2. Noncontiguous segments with same segment-number (SEG2, OBSOLETE).
3. Symbolic-character (HIGH).
4. ALL literal figurative constant (HIGH).
5. LINAGE-COUNTER special register (HIGH).
6. Qualification (HIGH).
7. More than three subscripts (HIGH).
8. Reference modification (HIGH).
9. Continuation of a COBOL word, numeric literal or PICTURE character-string (HIGH).
10. IDENTIFICATION DIVISION header of a contained program (HIGH).
11. END PROGRAM header (HIGH).
12. COMMON clause in PROGRAM-ID paragraph (HIGH).
13. INITIAL clause in PROGRAM-ID paragraph (HIGH).
14. DATE-COMPILED paragraph (HIGH, OBSOLETE).
15. COPY statement (INTERMEDIATE).
16. COPY . . . OF/IN *library-name* (HIGH).
17. COPY . . . REPLACING phrase (HIGH).
18. REPLACE statement (HIGH).
19. SEGMENT-LIMIT clause (SEG2, OBSOLETE).
20. ALPHABET clause literal phrase (HIGH).
21. SYMBOLIC CHARACTERS clause (HIGH).
22. OPTIONAL phrase in file control entry (HIGH).
23. ACCESS MODE RANDOM clause (INTERMEDIATE).

24. ACCESS MODE DYNAMIC clause (HIGH).
25. RELATIVE KEY phrase (INTERMEDIATE).
26. ALTERNATE RECORD KEY clause (HIGH).
27. ORGANIZATION RELATIVE clause (INTERMEDIATE).
28. ORGANIZATION INDEXED clause (INTERMEDIATE).
29. PADDING CHARACTER clause (HIGH).
30. RECORD DELIMITER clause (HIGH).
31. RECORD KEY clause (INTERMEDIATE).
32. RESERVE AREA clause (HIGH).
33. MULTIPLE FILE TAPE clause (HIGH, OBSOLETE).
34. SAME RECORD AREA clause (HIGH).
35. SAME SORT/SORT-MERGE AREA clause INTERMEDIATE).
36. SD level indicator (INTERMEDIATE).
37. BLOCK CONTAINS *integer-1* TO *integer-2* RECORDS/CHARACTERS (HIGH).
38. LINAGE clause (HIGH).
39. RECORD VARYING IN SIZE clause (HIGH).
40. VALUE OF clause that specifies a data-name (HIGH, OBSOLETE).
41. COMMUNICATION SECTION header (COM1).
42. CD level indicator (COM1).
43. INITIAL clause in a CD entry (COM2).
44. SYMBOLIC SUB-QUEUE-1, SUB-QUEUE-2 and SUB-QUEUE-3 clauses in a CD entry (COM2).
45. Data-name series in a CD entry (COM2).
46. DESTINATION TABLE clause in a CD entry (COM2).
47. Level-number 66 data description entry (HIGH).
48. Level-number 88 data description entry (HIGH).
49. EXTERNAL clause (HIGH).
50. GLOBAL clause (HIGH).
51. OCCURS clause ASCENDING/DESCENDING KEY phrase (HIGH).
52. OCCURS clause *integer-1* TO *integer-2* TIMES DEPENDING ON phrase (HIGH).
53. REDEFINES clauses nested (HIGH).
54. REDEFINES clause that refers to a table item (HIGH).
55. RENAMES clause (HIGH).
56. Procedure Division header USING phrase with more than five operands (HIGH).
57. Condition-name condition (HIGH).
58. Sign condition (HIGH).

59. Logical operators AND, OR, NOT (HIGH).
60. Arithmetic expression operators + – * / ** (HIGH).
61. ACCEPT statement FROM phrase (HIGH).
62. ACCEPT MESSAGE COUNT statement (COM1).
63. ADD statement CORRESPONDING phrase (HIGH).
64. ALTER statement procedure-name series (HIGH, OBSOLETE).
65. CALL statement with *identifier-1* (HIGH).
66. CALL statement USING phrase with more than five operands (HIGH).
67. CALL statement USING BY CONTENT or BY REFERENCE phrase (HIGH).
68. CALL statement ON OVERFLOW phrase (HIGH).
69. CALL statement ON EXCEPTION phrase (HIGH).
70. CALL statement NOT ON EXCEPTION phrase (HIGH).
71. CANCEL statement (HIGH).
72. CLOSE statement FOR REMOVAL phrase (HIGH).
73. CLOSE statement WITH NO REWIND phrase (HIGH).
74. CLOSE statement WITH LOCK phrase (HIGH).
75. COMPUTE statement (HIGH).
76. DELETE statement (INTERMEDIATE).
77. DISABLE statement (COM2).
78. DISPLAY statement UPON phrase (HIGH).
79. DISPLAY statement WITH NO ADVANCING phrase (HIGH).
80. DIVIDE statement REMAINDER phrase (HIGH).
81. ENABLE statement (COM2).
82. EVALUATE statement (HIGH).
83. GO TO statement with omitted procedure-name (HIGH, (OBSOLETE).
84. IF statement that contains a conditional statement (HIGH).
85. INITIALIZE statement (HIGH).
86. INSPECT statement with multicharacter data item (HIGH).
87. INSPECT statement BEFORE/AFTER phrase series (HIGH).
88. INSPECT statement TALLYING phrase series (HIGH).
89. INSPECT statement REPLACING phrase series (HIGH).
90. INSPECT statement CONVERTING phrase (HIGH).
91. MERGE statement (INTERMEDIATE).
92. MOVE statement CORRESPONDING phrase (HIGH).
93. MOVE statement de-editing of numeric edited items (HIGH).
94. OPEN statement WITH NO REWIND phrase (HIGH).
95. OPEN statement REVERSED phrase (HIGH, OBSOLETE).

96. OPEN statement EXTEND phrase (HIGH).
97. PERFORM statement TEST BEFORE/AFTER phrase (HIGH).
98. PERFORM statement VARYING phrase (HIGH).
99. PURGE statement (COM2).
100. READ statement NEXT phrase (HIGH).
101. READ statement KEY phrase (HIGH).
102. READ statement INVALID KEY phrase (INTERMEDIATE).
103. READ statement NOT INVALID KEY phrase (INTERMEDIATE).
104. RECEIVE statement (COM1).
105. RECEIVE statement SEGMENT phrase (COM2).
106. RELEASE statement (INTERMEDIATE).
107. RETURN statement (INTERMEDIATE).
108. REWRITE statement INVALID KEY phrase (INTERMEDIATE).
109. REWRITE statement NOT INVALID KEY phrase (INTERMEDIATE).
110. SEARCH statement (HIGH).
111. SEND statement (COM1).
112. SEND statement Format 1 (COM2).
113. SEND statement WITH identifier phrase (COM2).
114. SEND statement WITH ESI phrase (COM2).
115. SEND statement BEFORE/AFTER ADVANCING *mnemonic-name* phrase (COM2).
116. SEND statement REPLACING LINE phrase (COM2).
117. SET statement *condition-name* TO TRUE (HIGH).
118. SORT statement (INTERMEDIATE).
119. START statement (HIGH).
120. STRING statement (HIGH).
121. SUBTRACT statement CORRESPONDING phrase (HIGH).
122. UNSTRING statement (HIGH).
123. USE statement GLOBAL phrase (HIGH).
124. USE statement ON file-name series (HIGH).
125. USE statement ON EXTEND (HIGH).
126. WRITE statement BEFORE/AFTER ADVANCING *mnemonic-name* phrase (HIGH).
127. WRITE statement AT END-OF-PAGE/EOP phrase (HIGH).
128. WRITE statement NOT AT END-OF-PAGE/EOP phrase (HIGH).
129. WRITE statement INVALID KEY phrase (INTERMEDIATE).
130. WRITE statement NOT INVALID KEY phrase (INTERMEDIATE).

# Appendix J: Code-Set Translation Tables

## ASCII to EBCDIC Conversion

Table 58 and Table 60 describe the translation between ASCII and EBCDIC character sets. The ASCII to EBCDIC translation is identical to that described by IBM in the document, *Systems Network Architecture Format and Protocol Reference Manual: Architecture Logic* (SC30-3112-0). The EBCDIC to ASCII translation is the inverse of the ASCII to EBCDIC mapping, with the addition that EBCDIC characters with no ASCII equivalent are assigned values in the range X‘80’ to X‘FF’.

Character abbreviations are defined in Table 61 beginning on page 652.

**Table 59: ASCII to EBCDIC Conversion**

| ASCII Code (Decimal) | ASCII Code (Hexadecimal) | U.S. Character | EBCDIC Code (Hexadecimal) | U.S. Character |
|---|---|---|---|---|
| 0 | 00 | NUL | 00 | NUL |
| 1 | 01 | SOH | 01 | SOH |
| 2 | 02 | STX | 02 | STX |
| 3 | 03 | ETX | 03 | ETX |
| 4 | 04 | EOT | 37 | EOT |
| 5 | 05 | ENQ | 2D | ENQ |
| 6 | 06 | ACK | 2E | ACK |
| 7 | 07 | BEL | 2F | BEL |
| 8 | 08 | BS | 16 | BS |
| 9 | 09 | HT | 05 | HT |
| 10 | 0A | LF | 25 | LF |
| 11 | 0B | VT | 0B | VT |
| 12 | 0C | FF | 0C | FF |
| 13 | 0D | CR | 0D | CR |
| 14 | 0E | SO | 0E | SO |
| 15 | 0F | SI | 0F | SI |

**Table 59: ASCII to EBCDIC Conversion *(Cont.)***

| ASCII Code (Decimal) | ASCII Code (Hexadecimal) | U.S. Character | EBCDIC Code (Hexadecimal) | U.S. Character |
|---|---|---|---|---|
| 16 | 10 | DLE | 10 | DLE |
| 17 | 11 | DC1 | 14 | ENP |
| 18 | 12 | DC2 | 24 | INP |
| 19 | 13 | DC3 | 04 | SEL |
| 20 | 14 | DC4 | 15 | NL |
| 21 | 15 | NAK | 3D | NAK |
| 22 | 16 | SYN | 32 | SYN |
| 23 | 17 | ETB | 26 | ETB |
| 24 | 18 | CAN | 18 | CAN |
| 25 | 19 | EM | 19 | EM |
| 26 | 1A | SUB | 3F | SUB |
| 27 | 1B | ESC | 27 | ESC |
| 28 | 1C | FS | 1C | IFS |
| 29 | 1D | GS | 1D | IGS |
| 30 | 1E | RS | 1E | IRS |
| 31 | 1F | US | 1F | IUS |
| 32 | 20 | Space | 40 | Space |
| 33 | 21 | ! | 4F | \| |
| 34 | 22 | ” | 7F | ” |
| 35 | 23 | # | 7B | # |
| 36 | 24 | $ | 5B | $ |
| 37 | 25 | % | 6C | % |
| 38 | 26 | & | 50 | & |
| 39 | 27 | ’ | 7D | ’ |
| 40 | 28 | ( | 4D | ( |
| 41 | 29 | ) | 5D | ) |
| 42 | 2A | * | 5C | * |
| 43 | 2B | + | 4E | + |
| 44 | 2C | , | 6B | , |
| 45 | 2D | - | 60 | - |
| 46 | 2E | . | 4B | . |
| 47 | 2F | / | 61 | / |
| 48 | 30 | 0 | F0 | 0 |
| 49 | 31 | 1 | F1 | 1 |
| 50 | 32 | 2 | F2 | 2 |
| 51 | 33 | 3 | F3 | 3 |
| 52 | 34 | 4 | F4 | 4 |
| 53 | 35 | 5 | F5 | 5 |

**Table 59: ASCII to EBCDIC Conversion *(Cont.)***

| ASCII Code (Decimal) | ASCII Code (Hexadecimal) | U.S. Character | EBCDIC Code (Hexadecimal) | U.S. Character |
|---|---|---|---|---|
| 54 | 36 | 6 | F6 | 6 |
| 55 | 37 | 7 | F7 | 7 |
| 56 | 38 | 8 | F8 | 8 |
| 57 | 39 | 9 | F9 | 9 |
| 58 | 3A | : | 7A | : |
| 59 | 3B | ; | 5E | ; |
| 60 | 3C | < | 4C | < |
| 61 | 3D | = | 7E | = |
| 62 | 3E | > | 6E | > |
| 63 | 3F | ? | 6F | ? |
| 64 | 40 | @ | 7C | @ |
| 65 | 41 | A | C1 | A |
| 66 | 42 | B | C2 | B |
| 67 | 43 | C | C3 | C |
| 68 | 44 | D | C4 | D |
| 69 | 45 | E | C5 | E |
| 70 | 46 | F | C6 | F |
| 71 | 47 | G | C7 | G |
| 72 | 48 | H | C8 | H |
| 73 | 49 | I | C9 | I |
| 74 | 4A | J | D1 | J |
| 75 | 4B | K | D2 | K |
| 77 | 4D | M | D4 | M |
| 78 | 4E | N | D5 | N |
| 79 | 4F | O | D6 | O |
| 80 | 50 | P | D7 | P |
| 81 | 51 | Q | D8 | Q |
| 82 | 52 | R | D9 | R |
| 83 | 53 | S | E2 | S |
| 84 | 54 | T | E3 | T |
| 85 | 55 | U | E4 | U |
| 86 | 56 | V | E5 | V |
| 87 | 57 | W | E6 | W |
| 88 | 58 | X | E7 | X |
| 89 | 59 | Y | E8 | Y |
| 90 | 5A | Z | E9 | Z |
| 91 | 5B | [ | 4A | ¢ |

**Table 59: ASCII to EBCDIC Conversion *(Cont.)***

| ASCII Code (Decimal) | ASCII Code (Hexadecimal) | U.S. Character | EBCDIC Code (Hexadecimal) | U.S. Character |
|---|---|---|---|---|
| 92 | 5C | \ | E0 | \ |
| 93 | 5D | ] | 5A | ! |
| 94 | 5E | ^ | 5F | ¬ |
| 95 | 5F | _ | 6D | _ |
| 96 | 60 | ‘ | 79 | ‘ |
| 97 | 61 | a | 81 | a |
| 98 | 62 | b | 82 | b |
| 99 | 63 | c | 83 | c |
| 100 | 64 | d | 84 | d |
| 101 | 65 | e | 85 | e |
| 102 | 66 | f | 86 | f |
| 103 | 67 | g | 87 | g |
| 104 | 68 | h | 88 | h |
| 105 | 69 | I | 89 | I |
| 106 | 6A | j | 91 | j |
| 107 | 6B | k | 92 | k |
| 108 | 6C | l | 93 | l |
| 109 | 6D | m | 94 | m |
| 110 | 6E | n | 95 | n |
| 111 | 6F | o | 96 | o |
| 112 | 70 | p | 97 | p |
| 113 | 71 | q | 98 | q |
| 114 | 72 | r | 99 | r |
| 115 | 73 | s | A2 | s |
| 116 | 74 | t | A3 | t |
| 117 | 75 | u | A4 | u |
| 118 | 76 | v | A5 | v |
| 119 | 77 | w | A6 | w |
| 120 | 78 | x | A7 | x |
| 121 | 79 | y | A8 | y |
| 122 | 7A | z | A9 | z |
| 123 | 7B | { | C0 | { |
| 124 | 7C | \| | 6A | \| |
| 125 | 7D | } | D0 | } |
| 126 | 7E | ~ | A1 | ~ |
| 127 | 7F | DEL | 07 | DEL |

# EBCDIC to ASCII Conversion

**Table 60: EBCDIC to ASCII Conversion**

| EBCDIC Code (Decimal) | EBCDIC Code (Hexadecimal) | U.S. Character | ASCII Code (Hexadecimal) | U.S. Character |
|---|---|---|---|---|
| 0 | 00 | NUL | 00 | NUL |
| 1 | 01 | SOH | 01 | SOH |
| 2 | 02 | STX | 02 | STX |
| 3 | 03 | ETX | 03 | ETX |
| 4 | 04 | SEL | 13 | DC3 |
| 5 | 05 | HT | 09 | HT |
| 6 | 06 | | 80 | |
| 7 | 07 | DEL | 7F | DEL |
| 8 | 08 | | 81 | |
| 9 | 09 | | 82 | |
| 10 | 0A | | 83 | |
| 11 | 0B | VT | 0B | VT |
| 12 | 0C | FF | 0C | FF |
| 13 | 0D | CR | 0D | CR |
| 14 | 0E | SO | 0E | SO |
| 15 | 0F | SI | 0F | SI |
| 16 | 10 | DLE | 10 | DLE |
| 17 | 11 | | 84 | |
| 18 | 12 | | 85 | |
| 19 | 13 | | 86 | |
| 20 | 14 | ENP | 11 | DC1 |
| 21 | 15 | NL | 14 | DC4 |
| 22 | 16 | BS | 08 | BS |
| 23 | 17 | | 87 | |
| 24 | 18 | CAN | 18 | CAN |
| 25 | 19 | EM | 19 | EM |
| 26 | 1A | | 88 | |
| 27 | 1B | | 89 | |
| 28 | 1C | IFS | 1C | FS |
| 29 | 1D | IGS | 1D | GS |
| 30 | 1E | IRS | 1E | RS |
| 31 | 1F | IUS | 1F | US |
| 32 | 20 | | 8A | |
| 33 | 21 | | 8B | |
| 34 | 22 | | 8C | |

**Table 60: EBCDIC to ASCII Conversion *(Cont.)***

| EBCDIC Code (Decimal) | EBCDIC Code (Hexadecimal) | U.S. Character | ASCII Code (Hexadecimal) | U.S. Character |
|---|---|---|---|---|
| 35 | 23 | 8D | | |
| 36 | 24 | INP | 12 | DC2 |
| 37 | 25 | LF | 0A | LF |
| 38 | 26 | ETB | 17 | ETB |
| 39 | 27 | ESC | 1B | ESC |
| 40 | 28 | | 8E | |
| 41 | 29 | | 8F | |
| 42 | 2A | | 90 | |
| 43 | 2B | | 91 | |
| 44 | 2C | | 92 | |
| 45 | 2D | ENQ | 05 | ENQ |
| 46 | 2E | ACK | 06 | ACK |
| 47 | 2F | BEL | 07 | BEL |
| 48 | 30 | | 93 | |
| 49 | 31 | | 94 | |
| 50 | 32 | SYN | 16 | SYN |
| 51 | 33 | | 95 | |
| 52 | 34 | | 96 | |
| 53 | 35 | | 97 | |
| 54 | 36 | | 98 | |
| 55 | 37 | EOT | 04 | EOT |
| 56 | 38 | | 99 | |
| 57 | 39 | | 9A | |
| 58 | 3A | | 9B | |
| 59 | 3B | | 9C | |
| 60 | 3C | | 9D | |
| 61 | 3D | NAK | 15 | NAK |
| 62 | 3E | | 9E | |
| 63 | 3F | SUB | 1A | SUB |
| 64 | 40 | Space | 20 | Space |
| 65 | 41 | | 9F | |
| 66 | 42 | | A0 | |
| 67 | 43 | | A1 | |
| 68 | 44 | | A2 | |
| 69 | 45 | | A3 | |
| 70 | 46 | | A4 | |
| 71 | 47 | | A5 | |
| 72 | 48 | | A6 | |

**Table 60: EBCDIC to ASCII Conversion *(Cont.)***

| EBCDIC Code (Decimal) | EBCDIC Code (Hexadecimal) | U.S. Character | ASCII Code (Hexadecimal) | U.S. Character |
|---|---|---|---|---|
| 73 | 49 | | A7 | |
| 74 | 4A | ¢ | 5B | [ |
| 75 | 4B | . | 2E | . |
| 76 | 4C | < | 3C | < |
| 77 | 4D | ( | 28 | ( |
| 78 | 4E | + | 2B | + |
| 79 | 4F | \| | 21 | ! |
| 80 | 50 | & | 26 | & |
| 81 | 51 | | A8 | |
| 82 | 52 | | A9 | |
| 83 | 53 | | AA | |
| 84 | 54 | | AB | |
| 85 | 55 | | AC | |
| 86 | 56 | | AD | |
| 87 | 57 | | AE | |
| 88 | 58 | | AF | |
| 89 | 59 | | B0 | |
| 90 | 5A | ! | 5D | ] |
| 91 | 5B | $ | 24 | $ |
| 92 | 5C | * | 2A | * |
| 93 | 5D | ) | 29 | ) |
| 94 | 5E | ; | 3B | ; |
| 95 | 5F | ¬ | 5E | ^ |
| 96 | 60 | - | 2D | - |
| 97 | 61 | / | 2F | / |
| 98 | 62 | | B1 | |
| 99 | 63 | | B2 | |
| 100 | 64 | | B3 | |
| 101 | 65 | | B4 | |
| 102 | 66 | | B5 | |
| 103 | 67 | | B6 | |
| 104 | 68 | | B7 | |
| 105 | 69 | | B8 | |
| 106 | 6A | | 7C | |
| 107 | 6B | , | 2C | , |
| 108 | 6C | % | 25 | % |
| 109 | 6D | _ | 5F | _ |

**Table 60: EBCDIC to ASCII Conversion *(Cont.)***

| EBCDIC Code (Decimal) | EBCDIC Code (Hexadecimal) | U.S. Character | ASCII Code (Hexadecimal) | U.S. Character |
|---|---|---|---|---|
| 110 | 6E | > | 3E | > |
| 111 | 6F | ? | 3F | ? |
| 112 | 70 | | B9 | |
| 113 | 71 | | BA | |
| 114 | 72 | | BB | |
| 115 | 73 | | BC | |
| 116 | 74 | | BD | |
| 117 | 75 | | BE | |
| 118 | 76 | | BF | |
| 119 | 77 | | C0 | |
| 120 | 78 | | C1 | |
| 121 | 79 | ‘ | 60 | ‘ |
| 122 | 7A | : | 3A | : |
| 123 | 7B | # | 23 | # |
| 124 | 7C | @ | 40 | @ |
| 125 | 7D | ’ | 27 | ’ |
| 126 | 7E | = | 3D | = |
| 127 | 7F | ” | 22 | ” |
| 128 | 80 | | C2 | |
| 129 | 81 | a | 61 | a |
| 130 | 82 | b | 62 | b |
| 131 | 83 | c | 63 | c |
| 132 | 84 | d | 64 | d |
| 133 | 85 | e | 65 | e |
| 134 | 86 | f | 66 | f |
| 135 | 87 | g | 67 | g |
| 136 | 88 | h | 68 | h |
| 137 | 89 | i | 69 | i |
| 138 | 8A | | C3 | |
| 139 | 8B | | C4 | |
| 140 | 8C | | C5 | |
| 141 | 8D | | C6 | |
| 142 | 8E | | C7 | |
| 143 | 8F | | C8 | |
| 144 | 90 | | C9 | |
| 145 | 91 | j | 6A | j |
| 146 | 92 | k | 6B | k |
| 147 | 93 | l | 6C | l |

**Table 60: EBCDIC to ASCII Conversion *(Cont.)***

| EBCDIC Code (Decimal) | EBCDIC Code (Hexadecimal) | U.S. Character | ASCII Code (Hexadecimal) | U.S. Character |
|---|---|---|---|---|
| 148 | 94 | m | 6D | m |
| 149 | 95 | n | 6E | n |
| 150 | 96 | o | 6F | o |
| 151 | 97 | p | 70 | p |
| 152 | 98 | q | 71 | q |
| 153 | 99 | r | 72 | r |
| 154 | 9A | | CA | |
| 155 | 9B | | CB | |
| 156 | 9C | | CC | |
| 157 | 9D | | CD | |
| 158 | 9E | | CE | |
| 159 | 9F | | CF | |
| 160 | A0 | | D0 | |
| 161 | A1 | ~ | 7E | ~ |
| 162 | A2 | s | 73 | s |
| 163 | A3 | t | 74 | t |
| 164 | A4 | u | 75 | u |
| 165 | A5 | v | 76 | v |
| 166 | A6 | w | 77 | w |
| 167 | A7 | x | 78 | x |
| 168 | A8 | y | 79 | y |
| 169 | A9 | z | 7A | z |
| 170 | AA | | D1 | |
| 171 | AB | | D2 | |
| 172 | AC | | D3 | |
| 173 | AD | | D4 | |
| 174 | AE | | D5 | |
| 175 | AF | | D6 | |
| 176 | B0 | | D7 | |
| 177 | B1 | | D8 | |
| 178 | B2 | | D9 | |
| 179 | B3 | | DA | |
| 180 | B4 | | DB | |
| 181 | B5 | | DC | |
| 182 | B6 | | DD | |
| 183 | B7 | | DE | |
| 184 | B8 | | DF | |

**Table 60: EBCDIC to ASCII Conversion *(Cont.)***

| EBCDIC Code (Decimal) | EBCDIC Code (Hexadecimal) | U.S. Character | ASCII Code (Hexadecimal) | U.S. Character |
|---|---|---|---|---|
| 185 | B9 | | E0 | |
| 186 | BA | | E1 | |
| 187 | BB | | E2 | |
| 188 | BC | | E3 | |
| 189 | BD | | E4 | |
| 190 | BE | | E5 | |
| 191 | BF | | E6 | |
| 192 | C0 | { | 7B | { |
| 193 | C1 | A | 41 | A |
| 194 | C2 | B | 42 | B |
| 195 | C3 | C | 43 | C |
| 196 | C4 | D | 44 | D |
| 197 | C5 | E | 45 | E |
| 198 | C6 | F | 46 | F |
| 199 | C7 | G | 47 | G |
| 200 | C8 | H | 48 | H |
| 201 | C9 | I | 49 | I |
| 202 | CA | | E7 | |
| 203 | CB | | E8 | |
| 204 | CC | | E9 | |
| 205 | CD | | EA | |
| 206 | CE | | EB | |
| 207 | CF | | EC | |
| 208 | D0 | } | 7D | } |
| 209 | D1 | J | 4A | J |
| 210 | D2 | K | 4B | K |
| 211 | D3 | L | 4C | L |
| 212 | D4 | M | 4D | M |
| 213 | D5 | N | 4E | N |
| 214 | D6 | O | 4F | O |
| 215 | D7 | P | 50 | P |
| 216 | D8 | Q | 51 | Q |
| 217 | D9 | R | 52 | R |
| 218 | DA | | ED | |
| 219 | DB | | EE | |
| 220 | DC | | EF | |
| 221 | DD | | F0 | |

**Table 60: EBCDIC to ASCII Conversion *(Cont.)***

| EBCDIC Code (Decimal) | EBCDIC Code (Hexadecimal) | U.S. Character | ASCII Code (Hexadecimal) | U.S. Character |
|---|---|---|---|---|
| 222 | DE | | F1 | |
| 223 | DF | | F2 | |
| 224 | E0 | \ | 5C | \ |
| 225 | E1 | | F3 | |
| 226 | E2 | S | 53 | S |
| 227 | E3 | T | 54 | T |
| 228 | E4 | U | 55 | U |
| 229 | E5 | V | 56 | V |
| 230 | E6 | W | 57 | W |
| 231 | E7 | X | 58 | X |
| 232 | E8 | Y | 59 | Y |
| 233 | E9 | Z | 5A | Z |
| 234 | EA | | F4 | |
| 235 | EB | | F5 | |
| 236 | EC | | F6 | |
| 237 | ED | | F7 | |
| 238 | EE | | F8 | |
| 239 | EF | | F9 | |
| 240 | F0 | 0 | 30 | 0 |
| 241 | F1 | 1 | 31 | 1 |
| 242 | F2 | 2 | 32 | 2 |
| 243 | F3 | 3 | 33 | 3 |
| 244 | F4 | 4 | 34 | 4 |
| 245 | F5 | 5 | 35 | 5 |
| 246 | F6 | 6 | 36 | 6 |
| 247 | F7 | 7 | 37 | 7 |
| 248 | F8 | 8 | 38 | 8 |
| 249 | F9 | 9 | 39 | 9 |
| 250 | FA | | FA | |
| 251 | FB | | FB | |
| 252 | FC | | FC | |
| 253 | FD | | FD | |
| 254 | FE | | FE | |
| 255 | FF | | FF | |

# Character Abbreviations

Character abbreviations are defined in Table 61.

**Table 61: Character Abbreviations**

| Abbreviation | Meaning |
|---|---|
| ACK | Acknowledgment |
| BEL | Bell |
| BS | Backspace |
| CAN | Cancel |
| CR | Carriage Return |
| DC1 | Device Control 1 |
| DC2 | Device Control 2 |
| DC3 | Device Control 3 |
| DC4 | Device Control 4 |
| DEL | Delete |
| DLE | Data Link Escape |
| EM | End of Medium |
| ENP | Enable Presentation |
| ENQ | Enquiry |
| EOT | End of Transmission |
| ESC | Escape |
| ETB | End of Transmission Block |
| ETX | End of Text |
| FF | Form Feed |
| FS | File Separator |
| GS | Group Separator |
| HT | Horizontal Tab |
| IFS | Interchange File Separator |
| IGS | Interchange Group Separator |
| INP | Inhibit Presentation |
| IRS | Interchange Record Separator |
| IUS | Interchange Unit Separator |
| LF | Line Feed |
| NAK | Negative Acknowledgment |
| NL | New Line |
| NUL | Null |
| RS | Record Separator |
| SEL | Select |

**Table 61: Character Abbreviations *(Cont.)***

| Abbreviation | Meaning |
|---|---|
| SI | Shift In |
| SO | Shift Out |
| SOH | Start of Heading |
| STX | Start of Text |
| SUB | Substitute |
| SYN | Synchronous Idle |
| US | Unit Separator |
| VT | Vertical Tab |

# Appendix K: Troubleshooting RM/COBOL

You may encounter some common problems when running RM/COBOL on different operating environments. This appendix presents solutions or workarounds for these problems.

## RM/COBOL for Windows Running in a Microsoft Windows or Novell Network Environment

Liant technical support services have received several reports of the RM/COBOL for Windows runtime system returning 30,*xx* and 98,*xx*, or other errors when used with Windows clients connected to Microsoft Windows or Novell NetWare Servers. The following sections identify those problems and provide basic instructions for resolving them. Generally, the types of status codes that are returned to the RM/COBOL application are 30 and 98 errors. Systems experiencing any unusual frequency of 30 or 98 errors, or any other errors, should consider making the changes suggested below. The following table summarizes the problems and describes the platforms on which they occur.

| This problem | Applies to |
|---|---|
| Old vredir.vxd file | Windows 95 clients |
| Network redirector file caching | Windows NT-class Servers and Windows NT-class Workstations |
| Opportunistic locking | Windows NT-class Servers |
| Virus protection software | All Windows environments |
| Novell NetWare Client32 version | Windows 95 |
| Printing to a Novell Print queue using Novell NetWare Client32 | Novell NetWare Client for Windows NT-class Servers |
| File and printer sharing for NetWare networks service | Windows 9x-class clients in a peer-to-peer network |

**Note** The RM/COBOL for Windows installation procedure has been enhanced to check the system configuration automatically for compatibility with running on

Windows NT-class machines and make corrections, if necessary. See Installation Notes for Windows (on page 53).

## Old vredir.vxd File

**Platform:** This problem applies only to Microsoft Windows 95 clients.

When the Microsoft Client for Microsoft Networks is used, files that reside on a server (such as a Microsoft Windows NT-class Server) may become damaged or may contain invalid data if multiple workstations access the file at the same time.

This problem occurs because the network redirector (**vredir.vxd**) caches data locally for files it accesses on the server. If the last modified time or file size does not change within a two-second interval, the redirector reads file data from the local cache rather than from the actual file on the server.

To resolve this issue, Liant recommends that you ensure that each Windows 95 machine has version 4.00.1112 (dated 2/11/97) or later of the file **vredir.vxd**.

For more information about this problem, see the following Microsoft Knowledge Base article located at:

- http://support.microsoft.com/?kbid=148367

**Note** Periodically, Microsoft reorganizes the information on its website. If necessary, use the search capability to find information on a particular topic.

## Network Redirector File Caching

**Platforms:** This problem applies only to a Microsoft Windows NT-class Server and a Windows NT-class Workstation.

By default, when the Windows NT-class redirector opens a file for read or read/write access, the redirector uses the Windows NT-class system cache. As a result, when data is written to the file, it is written to the cache and not immediately flushed to the redirector. The cache manager flushes the data at a later time. If an unrecoverable network error occurs while the data is being transferred to the remote server, it may cause the cache write request to fail, thus possible leaving the file in a corrupted state.

**Note** Microsoft warns that this change will slow down network I/O. In addition, improper Windows registry changes can disable your network server. Liant Software recommends that only a qualified technician make these changes to your system.

For more information about disabling redirector file caching, see the Microsoft Knowledge Base article located at:

- http://support.microsoft.com/?kbid=163401

**Note** Periodically, Microsoft reorganizes the information on its website. If necessary, use the search capability to find information on a particular topic.

## Opportunistic Locking

**Platform:** This problem applies only to a Microsoft Windows NT-class Server.

With opportunistic locking, if a file is opened in a non-exclusive mode, the network redirector requests an opportunistic lock of the entire file. As long as no other process has the file open, the server will grant this oplock, giving the redirector exclusive access to the specified file. This action will allow the redirector to perform read-ahead, write-behind, and lock caching, as long as no other process tries to open the file.

Liant Software recommends that opportunistic locking feature on a Windows NT-class Server be disabled.

For more information about disabling opportunistic locking on Windows NT Servers, see the following Microsoft Knowledge Base article located at:

- http://support.microsoft.com/?kbid=129202

For more information about disabling opportunistic locking on Windows 2000 Servers, see the following Microsoft Knowledge Base article located at:

- http://support.microsoft.com/support/kb/articles/q296/2/64.asp

**Note** Periodically, Microsoft reorganizes the information on its website. If necessary, use the search capability to find information on a particular topic.

## Virus Protection Software

**Platforms:** This problem applies to all Microsoft Windows environments.

There have been a few reports of certain virus protection programs interfering with RM/COBOL data files. This can result in file corruption or invalid error messages. If you are experiencing either of these problems, and have virus protection software either enabled on the client or the server, Liant Software recommends adjusting the parameters that control the behavior of the virus protection software.

If possible, configure the virus protection software so that it does not scan the COBOL data files after every modification. If necessary, completely disable scanning of COBOL data files.

## Novell NetWare Client32 Version

**Platforms:** This problem applies to those Windows 95 users who choose to use Novell's NetWare Client32 package for access to NetWare rather than Microsoft's Client for NetWare Networks.

RM/COBOL version 6.5 or higher for Windows 95 requires version 2.11 (or later) of NetWare Client32 for Windows 95 (dated 8/21/96, file size 461,359 bytes). Record locking does not work properly and file corruption may occur with earlier versions of Client32. Additionally, there may be open errors (94,*xx*) or other permanent errors (30,*xx*) with older versions of Client32. Receiving error 30, MS-Windows error 1 (30,12,00001 from C$RERR) may indicate a defective version of Client32.

Other reported errors include:

- 30,05 (Access Denied)
- "Error invoking unauthorized copy of runtime" error messages

There have also been reports of Novell's NetWare Client32 for Windows 95 causing file corruption. Novell recommends turning off the "Opportunistic Locking" and "Packet Burst" options. To adjust these settings, open the Network dialog box by double-clicking the Network icon in the Control Panel. Then select the NetWare Client32 component on the Configuration tab. Click Properties, and then click the Advanced tab.

Contact your Novell support representative for further information about the Novell NetWare Client32 software.

## Printing to a Novell Print Queue Using Novell NetWare Client32

**Platform:** This problem applies to a Microsoft Windows NT-class Server.

When using the Novell NetWare Client for a Windows NT-class Server, the runtime can hang on the open of a print file. There are two known workarounds for this problem:

- Specify "PRINTER?" in the ASSIGN clause of the file control entry or in the DEFINE-DEVICE configuration record. When the RM/COBOL for Windows runtime encounters the open operation, it displays the standard Windows Print dialog box, which allows the user to select the correct printer and print the file.
- When configuring your printer, select "LPT*n*:" (a local printer), not a network printer. Then use the Novell Capture command to redirect output to a Novell print queue.

## File and Printer Sharing for NetWare Networks Service

**Platform:** This problem applies only to Windows 9x-class machines used in a peer-to-peer network with files larger than 2 gigabytes (GB).

The NetWare file sharing service can handle files only up to 2 GB. If a COBOL program tries to write a remote large file, the write operation will fail near the 2 GB boundary with error "30, MS-Windows error 5." If a remote large file already exists and is larger than 2 GB, the open operation will fail with a 37, 07 error. For information useful in understanding this problem, see Large File Locking Issues (on page 108).

In order to support remote large files up to 4 GB, you must install the "File and printer sharing for Microsoft Networks" service on the server machine (that is, the machine on which the large file resides). Note that only one File and printer sharing service may be installed.

**Note** The NetWare file sharing service referred to in this section is not the "Client for NetWare Networks" network component (a Microsoft product) or the "Novell NetWare Client 32" network component (a Novell product). Do not modify these network components. Liant Software recommends that only a qualified technician make these changes to your system.

To check for, or install, the service:

1. In Windows, click Start, and then point to Settings.
2. Click Control Panel.
3. Double-click the Network icon.

   In the Network dialog box, the Configuration tab lists the network components, including file sharing services (if any), which are installed on your computer.
4. If you have “File and printer sharing for NetWare Networks” installed, select it and then click Remove.
5. To add the Microsoft file sharing service, click Add.

   The Select Network Component Type dialog box opens
6. Select Service, and then click Add.

   The Select Network Service dialog opens.
7. Select “File and printer sharing for Microsoft Networks” and click OK. Continue to click OK on all the dialog boxes that appear.
8. After the service is installed, shut down and restart your computer.

# Appendix L: Summary of Enhancements

This appendix provides a history and summary of the enhancements from earlier releases of RM/COBOL, beginning with the most recent previous release.

**Note** The enhancements in the current release are listed and described in What's New (on page 9).

## Version 8.0 Enhancements

The following sections summarize the major enhancements available in RM/COBOL version 8.0 for Windows and UNIX. This summary describes the main features of each enhancement and tells you where to look for more information about them. The *RM/COBOL Language Reference Manual* and this user's guide primarily contain the details regarding these features.

Deficiencies that are version-specific or are discovered after printing are described in the README files contained on the delivered media. See also Appendix B: *Limits and Ranges* (on page 399) for further information.

### New Runtime System Features

The RM/COBOL version 8.0 for Windows and UNIX runtime system has been enhanced with the following new features:

- **Native Binary Data.** The runtime now supports machine-native, binary-format numeric data items. Such data items are denoted in the source program with the COMPUTATIONAL-5 or COMP-5 usage clause in the data description entry. Native, binary-format numeric items are sometimes useful in interfacing with non-COBOL programs, but in most cases, CodeBridge can be used to avoid the need for this non-portable data type. For more information, see Unsigned Numeric COMPUTATIONAL-5 Data (on page 425) and Signed Numeric COMPUTATIONAL-5 Data (on page 426).
- **New Object Version Level.** Object version 11 (on page 626) has been introduced to support COMPUTATIONAL-5 and COMP-5 usage (that is, machine-native binary data format) and the declaration of empty groups when

the object symbol table is produced (Compiler Command Option Y) or in cases when the compiler does not eliminate them in the Procedure Division.

## New Compiler Features

The RM/COBOL version 8.0 for Windows and UNIX compiler has been enhanced with the following new features:

- COMPUTATIONAL-5 (COMP-5) usage for denoting machine-native, binary-format numeric data items is now supported. This non-portable data type should be used only for interfacing with non-COBOL programs where the design of those programs requires native binary support. In most cases, CodeBridge can be used to interface to non-COBOL programs without the need for native binary data since the CodeBridge library will automatically convert from native binary to any COBOL data type. However, support for COMPUTATIONAL-5 data simplifies development in cases where a non-COBOL program saves the address of a COBOL data item passed on a CALL statement and then later stores a binary value at that address. See the "Usage Clause" section in Chapter 4: *Data Division* of the *RM/COBOL Language Reference Manual*.
- The compiler now provides default Working-Storage Section data description entries for file and data parameters that are specified with an unqualified data-name but which are not defined by the end of the Data Division (see Chapter 4 of the *RM/COBOL Language Reference Manual*). This new compiler feature eliminates compilation errors caused by forgetting to define one or more such parameters and thus speeds program development. The following parameters are affected by this feature:
  - File access name specified in the ASSIGN clause.
  - Padding character data item specified in the PADDING CHARACTER clause.
  - Relative key data item specified in the RELATIVE KEY phrase of the ACCESS MODE clause.
  - I/O status data item specified in the FILE STATUS clause.
  - Record size data item specified in the DEPENDING phrase of the RECORD IS VARYING clause.
  - Linage data items specified in the LINAGE clause (the linage lines, footing lines, top lines, and bottom lines data items).
  - Label data item specified in the VALUE OF clause.
  - Occurs-count data item specified in the DEPENDING phrase of the OCCURS clause.
- The compiler now supports concatenation expressions. Concatenation expressions use the ampersand (&) operator to concatenate nonnumeric literals. Besides being an improved method of continuing long nonnumeric literals, concatenation expressions provide a means to construct nonnumeric literal values at compile time from combinations of quoted strings, hexadecimal strings, symbolic-characters, and constant-names. See Chapter 1: *Language Structure* of the *RM/COBOL Language Reference Manual*.
- The OCCURS clause is now allowed on level-number 01 and 77 data description entries in the Working-Storage Section.

- An empty group, that is, a group that does not contain any elementary data items, is now allowed. Empty groups correspond to XML empty elements and thus more closely align the COBOL data model with the XML data model.
- The compiler now supports the NUMERIC SIGN clause in the Special-Names paragraph. This clause allows the source program to specify the default sign convention for signed numeric display data items that are described without the SIGN clause in their data description entry. This feature can be used to eliminate the need to remember to specify the Compile Command Option S (separate sign) for source programs that require this option, such as when a record area size depends on allocating separate signs for signed numeric display data items. For more details, see Chapter 3: *Environment Division* of the *RM/COBOL Language Reference Manual*.
- The ASSIGN clause no longer requires a device-name, even when the file access name is specified with a data-name. The device-name is now required only if no file access name (*data-name-1* or *literal-1*) is specified in the ASSIGN clause. The only restriction is that, if *data-name-1* is specified for the file access name, *data-name-1* must not match one of the device-names built into the compiler as context-sensitive words. See Chapter 3: *Environment Division* of the *RM/COBOL Language Reference Manual* for more information.

# Version 7.5 Enhancements

The following section summarizes the major enhancements available in version 7.5 of RM/COBOL. This summary describes the main features of each enhancement. The *RM/COBOL Language Reference Manual* and this user's guide contain the details regarding these features.

## CodeWatch Application Development Environment Introduced

This release includes the latest version of CodeWatch, a fully integrated development environment for Windows. Starting with version 7.5, CodeWatch now supports the entire development cycle, including editing, compiling, and debugging RM/COBOL applications. CodeWatch can be used to debug and change programs that are independently compiled, without requiring you to build projects—instead, the required knowledge about the structure of your application is built up during debugging sessions. For more information, see the *CodeWatch* manual, which is included with the documentation for an RM/COBOL development system.

## CodeBridge Enhancements

CodeBridge, Liant Software's cross-language call system, has been enhanced to handle 64-bit integers on all UNIX platforms having a C compiler that supports 64-bit integers. For more information about CodeBridge, see the *CodeBridge* manual.

A new runtime callback, GetCallerInfo, has been added to allow CodeBridge non-COBOL subprograms to enhance error messages with additional information about the caller. The new callback provides the calling program name and line number, the object file name, and the date and time the calling program was compiled. The definition and commentary for this new runtime callback and its associated data

structure, CALLER_INFO, are available in **rtcallbk.h**, a header file provided with RM/COBOL systems. Examples of the use of this new callback are included in the **msgbox.c** sample subprogram for Windows and the **usrsub.c** sample subprogram for UNIX.

Two new parameter attributes, called error base attributes, have been added to CodeBridge for retrieving error information set by C library and Windows API functions. The new error base attributes, **errno** and **get_last_error**, allow return of the error information by editing the CodeBridge template instead of the generated code.

## Console-Mode Compiler on Windows

The RM/COBOL compiler can now be run as a console-mode application on Windows with the **rmcobolc** command. The **rmcobolg** command can still be used to start the Windows graphical user interface version (GUI-mode) of the compiler. The console-mode compiler is smaller and faster than the GUI-mode version and is well suited to batch compilations of large numbers of programs. Other than the interface, the two compilers are identical since they both use a common DLL for the compiler implementation. An option at installation allows you to pick either version of the compiler to be invoked with the **rmcobol** command. For additional information, see Batch Compilation on Windows (on page 138).

## Multiple and Batch Compiles Easier and Faster

The Windows RM/COBOL compiler selection dialog now allows more than one file to be selected. You may select additional files by holding down the Ctrl key while clicking on additional files, or you may use Ctrl+A to select all files. Subdirectories are automatically ignored. The compilations stop when all files have been compiled or a compilation returns a non-zero exit code. This mode of compiling is faster than using a script because the compiler does not need to be reloaded between sources.

Wildcard characters on the command line for both the console-mode compiler and the GUI-mode compiler can also be used to select multiple files to compile. Supported wildcard characters are “?” (match any single character) and “*” (match zero or more characters).

For additional information on these new capabilities, see Multiple File Compilation on Windows (on page 139).

## More Reliable Indexed Files

Indexed file support has been made more reliable by adding new integrity features as part of file version level 4 (on page 244). Additionally, version 4 indexed files optionally support the new “atomic I/O” capability, which provides a means for users to avoid almost all 98 errors caused by failures that occur while a file is open. Files created with atomic I/O will almost never need recovery. If a crash occurs during a COBOL I/O operation, the file will be automatically and quickly recovered the next time the file is opened or a write operation is performed. The ENABLE-ATOMIC-IO keyword has been added to the RUN-INDEX-FILES configuration record (on page 320) to determine whether indexed files created by the runtime system use atomic I/O.

The default indexed file version for new files has been changed from 2 to 4 to automatically provide the higher level of reliability to new files. The DEFAULT-

FILE-VERSION-NUMBER keyword of the RUN-INDEX-FILES configuration record may be used to specify a different value.

Version 4 indexed files may, like version 3 files, grow to a larger size than version 0 or 2 files. However, unlike version 3 files, version 4 files may be either large or regular sized files, depending on the new USE-LARGE-FILE-LOCK-LIMIT keyword of the RUN-INDEX-FILES configuration record (on page 320). This new keyword determines whether the LARGE-FILE-LOCK-LIMIT or the FILE-LOCK-LIMIT keyword of the RUN-FILES-ATTR configuration record (on page 316) is used to determine the largest address that can be locked in the file. This, in turn, determines how large the file can be.

## Better Indexed File Performance

Several changes have been made to increase indexed file performance by creating new indexed files with more reasonable block sizes and by increasing the maximum size of the file buffer pool from less than one million bytes to ten million bytes.

The RM/COBOL version 7.5 runtime system now creates new indexed files with a minimum block size of 1024 bytes and ensures that the block size for new indexed files is a multiple of the disk sector size. Indexed file processing is generally more efficient with larger block sizes and with block sizes that are also a multiple of the disk sector size (512 bytes on Windows and, normally, 1024 bytes on UNIX). For additional information about these changes, see BLOCK CONTAINS CLAUSE (Indexed File Description Entry) on page 233.

Two new keywords, MINIMUM-BLOCK-SIZE and ROUND-TO-NICE-BLOCK-SIZE, have been added to the RUN-INDEX-FILES configuration record (on page 320) to allow the block size to be computed in the same manner as prior versions of the RM/COBOL runtime system.

The BUFFER-POOL-SIZE keyword of the RUN-FILES-ATTR configuration record (on page 316) now allows the buffer pool size to be as large as 10,000,000 bytes. Generally, a larger buffer pool size will produce better file performance than a smaller buffer pool size. Some testing may be required to find the optimal size for your application.

## Automatic Configuration File Available for Windows

Configuration files may now be automatically loaded on Windows for the runtime, compiler, and recovery utility in a manner similar to the capability on UNIX. The ability to attach a configuration file to the executable on Windows using the **rmattach** utility, however, is still provided. For more information, refer to Automatic and Attached Configuration Files.

## Tail Comments for Configuration Records

Configuration records may now contain a tail comment, that is, a comment that does not start in column one of the configuration record. More information on configuration records and tail comments can be found on page 282.

## Enhancements for Non-COBOL Subprograms on Windows

The RM/COBOL 7.5 for Windows runtime system has been enhanced to load dynamic-link libraries (DLLs) automatically from a special subdirectory, **RmAutoLd**, of the **runcobol** execution directory without the need to specify the filename with the L (Library) Option on the **runcobol** command. All DLLs in this special subdirectory will be loaded automatically. While it is no longer necessary to specify any non-COBOL libraries on the **runcobol** command, the L (Library) Runtime Command Option is still supported for doing so. For further details, refer to Locating Optional Support Modules (on page 431).

The Windows runtime system has also been enhanced to support the special predefined symbols (entry points and variable names), such as **RM_EntryPoints**, **RM_AddOnInit**, and **RM_AddOnTerminate**, which, previously, were available only on UNIX. While none of these special entry points is required, if present, the DLL can provide a list of COBOL-callable entry points without the need to specify the .EDATA section at link time, and can provide special initialization and termination code that will be called automatically when the runtime system initializes and terminates. Additional information about these special entry points may be found in Appendix G: *Non-COBOL Subprogram Internals for Windows* in the *CodeBridge* manual.

## Additions to the RM/COBOL Subprogram Library

The RM/COBOL subprogram library has been extended with the following new C$ subprograms. For more details, see Appendix F: *Subprogram Library* (on page 527).

- **C$CompilePattern** compiles a variable pattern regular expression for use in the new LIKE condition, which has been added to the RM/COBOL language. More information about the LIKE condition may be found in Chapter 5: *Procedure Division* of the *RM/COBOL Language Reference Manual*.
- **C$ConvertAnsiToOem** may be used to convert a buffer containing ANSI characters to a buffer containing the corresponding OEM characters. The runtime's ANSI/OEM euro character configuration is preserved in the conversion.
- **C$ConvertOemToAnsi** may be used to convert a buffer containing OEM characters to a buffer containing the corresponding ANSI characters. The runtime's ANSI/OEM euro character configuration is preserved in the conversion.
- **C$DARG** may be used to obtain the description of an actual argument by using an argument number to refer to the desired argument.
- **C$SecureHash** may be used to produce a message digest from a message text using the secure hash algorithm (SHA-1).
- Several library subprograms for doing bitwise logical operations have been added. These include **C$LogicalAnd**, **C$LogicalComplement**, **C$LogicalOr**, **C$LogicalShiftLeft**, **C$LogicalShiftRight**, and **C$LogicalXor**. Each of these subprograms can operate either on nonnumeric strings or numeric values.

## Message Files Eliminated

The message files for RM/COBOL executable programs (**runcobol**, **rmcobol** and **recover1**) have been eliminated. The files **RUN.MSG**, **RMC.MSG,** and **REC.MSG**, present in previous versions of RM/COBOL, no longer exist beginning with version 7.5. The messages contained in these files now reside within each executable. Thus, there is no longer any searching for the correct message file and no possibility of having mismatched executable and message file versions.

## Compiler Overlay File Eliminated

The overlay file for the RM/COBOL compiler executable programs (**rmcobol** and **rmcobolc**) has been eliminated. The file **RMCOBOL.OVY**, present in previous versions of RM/COBOL development systems, no longer exists beginning with version 7.5. Thus, there is no longer any searching for the correct overlay file and no possibility of having mismatched executable and overlay file versions.

## New Runtime System Features

In addition to the new C$ subprogram library subprograms supplied in the runtime, the RM/COBOL version 7.5 for Windows and UNIX runtime system has been enhanced with the following new features:

- **Pipe Paths.** For UNIX, where piping print output to a print spooler is common, a file access name that begins with a pipe character ('|') may now be used to create the pipe without having to use a DEFINE-DEVICE configuration record. This allows the spooler options to be constructed dynamically in a variable by the COBOL program. The check for the pipe character is made after the file access name is mapped using any applicable environment variables, so a program also can be caused to pipe its output by setting an environment variable that maps the file access name specified in the program to a value having the pipe character as its first character. For more details, see File Access Names on UNIX (on page 25).
- **Input Pipes.** Input pipes are now supported on UNIX. A pipe is used for input when a file is opened in the input mode and either the path begins with a pipe character ('|') or the file access name refers to a DEFINE-DEVICE configuration record (on page 304) that specifies a pipe with the PIPE=YES keyword. For example, a file opened in the input mode with a file access name having the value "| ls -1 *.txt" will read a list of the text files (assuming text files are identified by the ".txt" extension) in the current directory.
- **Default Use Procedure Configuration.** The action to take when there is no applicable USE procedure for an I/O error on a file can now be configured. Previously, RM/COBOL terminated the run unit with an appropriate error message when there was no applicable USE procedure. Now, the runtime behavior may be configured to cause the program to continue as if an empty USE procedure were applicable by specifying DEFAULT-USE-PROCEDURE=CONTINUE in the RUN-FILES-ATTR configuration record (on page 316).
- **Library Configuration.** RM/COBOL object libraries and non-COBOL subprogram libraries may now be configured with the L keyword, which has been added to the RUN-OPTION configuration record (on page 322), paralleling the L (Library) Runtime Command Option. Additionally, the LIBRARY-PATH

keyword has been added to the RUN-OPTION configuration record to cause loading of all RM/COBOL object libraries in a specified directory. Both the L and LIBRARY-PATH keywords may be specified multiple times in the configuration.

- **Main Program Configuration.** The MAIN-PROGRAM keyword has been added to the RUN-OPTION configuration record (on page 322). This keyword allows specifying a main program-name to override the program-name specified on the command line.
- **Configuration Record Name Enhancement.** Configuration record names have been enhanced to allow the singular or plural forms interchangeably. For example, the RUN-OPTION and RUN-OPTIONS record names are both allowed and either record name supports the same set of keywords. The alternative forms of the record-type identifiers are shown in Table 29: Types of Configuration Records in Configuration Records (on page 285).
- **Euro Support for Windows.** A new configuration record type, with the identifier INTERNATIONALIZATION, has been added to allow configuration of support for the euro symbol (€) in ACCEPT, DISPLAY, and printing operations on Windows. For more information, see INTERNATIONALIZATION configuration record (on page 309). In addition, the DATA-CHARACTERS keyword of the TERM-ATTR configuration record (on page 327) has been enhanced to allow specification of multiple disjoint ranges on Windows, thus matching a capability that was already supported on UNIX. Since the euro symbol is not typically in the default range of characters that are interpreted as text characters, allowing the euro symbol to be entered for an ACCEPT statement requires modifying the range of the data characters. This can be done by using multiple DATA-CHARACTERS keywords with disjoint ranges or by modifying the upper bound for the data characters range to include the euro symbol. An example is provided in the description of the DATA-CHARACTERS keyword. Additionally, the description of the DATA-CHARACTERS keyword has an example that demonstrates how to convert the period on the numeric keypad into a comma when doing numeric input in countries that use the comma as the fractional separator rather than the period.
- **Toolbar Tooltips.** For Windows, the toolbar button prompt value that was displayed in the status bar is now also displayed as a tooltip, which is a small pop-up window containing the prompt text displayed near the toolbar command button when the mouse cursor hovers over the button. The Toolbar Prompt property (on page 84) has been added to control this new behavior. This new property allows choosing the old behavior of displaying the prompt only in the status bar. The property also allows choosing not to display the prompt at all, display the prompt only as a tooltip, and display different values in the status bar and tooltip (the latter requires changing the prompt string set in the toolbar properties in the Windows registry by using **rmconfig** or in the RM/COBOL program by calling C$TBar). The Toolbar Prompt property can be set in the registry with **rmconfig** or temporarily changed at runtime by calling C$GUICFG.
- **SYSTEM Window Types.** For Windows, the values that may be specified for the SYSTEM Window Type property (on page 83) have been expanded to include MinimizedNoActive and ShowNoActivate. These values can be stored in the Windows registry using **rmconfig** or, they can be set at runtime by calling C$GUICFG.

- **C$SCRD Support for Line Draw Characters.** The C$SCRD (on page 559) subprogram has been modified to support line draw characters by returning hyphens, plus signs, and vertical bars for the box characters. On Windows, the Screen Read Line Draw property (on page 81) has been added to allow C$SCRD to return DOS line draw characters (for example, $D9, "Ù" for the lower-right corner of a box).
- **P$GetHandle.** The P$GetHandle (on page 487) subprogram has been modified to provide for the optional return of the true Windows handle of the current P$ printer. This allows a non-COBOL program to add information (such as special graphics or a bar code) to a page printed on a P$ printer.
- **P$DisableDialog.** The P$DisableDialog (on page 460) subprogram is used to control the automatic invoking of the standard Windows Print dialog box when opening a "PRINTER?" device.
- **P$EnumPrinterInfo.** The P$EnumPrinterInfo (on page 483) subprogram is used to retrieve detailed information about all of the printers on a system. It is not necessary to open a printer to obtain this information.
- **Termination Log for UNIX.** The UNIX runtime now allows logging of termination error messages, including traceback information, using the ENABLE-LOGGING=TERMINATION keyword of the RUN-OPTION configuration record (on page 322). The termination log allows error messages to be collected for later analysis. UNIX users can still redirect standard error to collect termination information, but doing so means that the information will not be displayed for the user.
- **Creating Files on an RM/InfoExpress Server.** The new DISABLE-LOCAL-ACCESS-METHOD keyword of the RUN-FILES-ATTR configuration record (on page 316) can be used to prevent files with unqualified, simple names from being created in the current working directory. Specifying UNQUALIFIED-NAMES for the value of the new keyword will allow new files to be created on an RM/InfoExpress Server. On previous versions of the runtime system, it was necessary for the application to specify the server machine on which the file should be created either by including a complete pathname in the COBOL program or by using an environment variable to specify the complete path. Provided the new keyword is specified, it is now possible to create such files in the first directory of the RUNPATH environment variable. Specifying the new keyword has no effect on finding existing files.
- **Setting Toolbar Properties.** New special characters have been added for the toolbar icon string under Windows, as discussed in Setting Toolbar Properties (on page 88).
- **Additional Enhancements to Configuration Records.** These include the following:
  - More TERM-INPUT character equivalents have been added for Windows. See Translation of TERM-INPUT Sequences on Windows (on page 334).
  - The COPY-TO-CLIPBOARD value has been added to the list of values allowed for the ACTION keyword of the TERM-INPUT configuration record. See Field Editing Actions (on page 338).

- **New Object Version Level.** Object version 10 (on page 625) has been introduced to solve a problem with reference modified patterns in LIKE conditions. RM/COBOL v7.50.01 and later runtime systems support object version level 10.
- **Pop-Up Window Positioning.** A new Pop-Up Window Positioning property (on page 78) has been added to control initial positioning of pop-up windows on Windows.

## New Compiler Features

The RM/COBOL version 7.5 for Windows and UNIX compiler has been enhanced with the following new features:

- **Reserved Words.** To support the new language features mentioned below, the reserved word list has been extended with the new words DATA-POINTER, DEFAULT, and LIKE. Also, several words have been removed from the list of reserved words and placed in the new category of context-sensitive words. For more details, see Appendix A: *Reserved Words* in the *RM/COBOL Language Reference Manual*, or see this topic in the *RM/COBOL Syntax Summary*.
- **Context-Sensitive Words.** Some words previously considered to be always reserved have been changed to be reserved only in certain contexts and are thus now in the new category of context-sensitive words. For example, the word UNDERLINE was previously a reserved word, but is now reserved only in the context of a screen description entry. Also, several new words have been added to this category of words to support the new features mentioned below. For additional information, see Appendix A *Reserved Words* in the *RM/COBOL Language Reference Manual*, or see this topic in the *RM/COBOL Syntax Summary*.
- **FILLER Items Entered in Symbol Table.** FILLER data items are now entered in the symbol table and will therefore be displayed in the listing allocation map. Keeping FILLER data items in the symbol table requires additional compile-time memory but allows support for the new WITH FILLER phrase on the INITIALIZE statement. A new compiler configuration option, SUPPRESS-FILLER-IN-SYMBOL-TABLE, has been added to reduce the memory required to compile a program with many FILLER data items. For more information, refer to this keyword in the COMPILER-OPTIONS configuration record (on page 287).
- **SELECT Clause NOT OPTIONAL Phrase.** For compatibility with other COBOL dialects, the NOT OPTIONAL phrase may be specified in the SELECT clause for files that are required to be present at runtime. Since RM/COBOL assumes that files are required at runtime unless the OPTIONAL phrase is specified, the NOT OPTIONAL phrase has no effect, but is accepted in order to ease conversion of programs originally written in other COBOL dialects. For further details, see the topic "File Control Entry" in Chapter 3: *Environment Division* of the *RM/COBOL Language Reference Manual*.
- **PICTURE Clause.** The PICTURE clause may now be omitted for an elementary data item described with a VALUE clause that specifies a nonnumeric literal. The data item defined in this case is as if a PIC X(*n*) clause had been specified, where *n* is the length of the nonnumeric literal specified in the VALUE clause. Refer to the section "Data Description Entry" in Chapter 4: *Data Division* of the *RM/COBOL Language Reference Manual*.

- **Implied PICTURE for Numeric VALUE (by PICTURE Clause).** The compiler now supports an implied PICTURE character-string when the VALUE clause specifies a numeric literal. This feature is in addition to the feature of an implied PICTURE character-string when the VALUE clause specifies a nonnumeric literal, also new in version 7.5. For more information, see the "Data Description Entry" section Chapter 4: *Data Division* of the *RM/COBOL Language Reference Manual*.

- **Format 1 VALUE Clause.** The Format 1 VALUE clause, which defines initialization values for Working-Storage items, now also defines values to be used by the new VALUE phrase of the INITIALIZE statement. Therefore, the clause is now allowed in the File, Linkage, and Communication Sections and also in record descriptions described with the EXTERNAL clause, without the previous RM/COBOL restriction that the VALUE clause could be used only in such situations when it was included in the source program as part of a copied file. Refer to the section "Data Item Initialization (Format 1 VALUE Clause)" in Chapter 4: *Data Division* of the *RM/COBOL Language Reference Manual*.

- **Format 2 VALUE Clause.** The Format 2 VALUE clause, which defines values for level-number 88 condition-names, has been extended to allow relational operators. This allows, in particular, the use of the new LIKE condition to specify valid values for a data item by use of a pattern regular expression. Refer to the section "Condition-Name Rules (Format 2 VALUE Clause)" in Chapter 4: *Data Division* of the *RM/COBOL Language Reference Manual*.

- **LIKE Condition.** The relational operators have been extended to include the LIKE operator and thus provide the special case of relation conditions called the LIKE condition. The LIKE condition specifies a truth-value based on whether a data item matches a pattern value. Pattern values are given as regular expressions in the same form used by XML Schema. You can read more about this topic in the section "LIKE Condition (Special Case of Relation Condition)" in Chapter 5: *Procedure Division* of the *RM/COBOL Language Reference Manual*.

- **ACCEPT Statement Enhancements.** The ACCEPT statement now supports a proposed COBOL standard method of obtaining four-digit years by use of the YYYYMMDD and YYYYDDD optional phrases in the FROM DATE or FROM DAY phrases, respectively. For more information, see "ACCEPT…FROM Statement" in Chapter 6: *Procedure Division Statements* of the *RM/COBOL Language Reference Manual*.

- **EXIT Statement Enhancements.** The EXIT statement now includes formats for exiting a paragraph, a section, or an in-line PERFORM statement. For additional information, see "EXIT Statement" in Chapter 6: *Procedure Division Statements* of the *RM/COBOL Language Reference Manual*.

- **INITIALIZE Statement Enhancements.** The INITIALIZE statement now includes the optional FILLER, VALUE, and DEFAULT phrases. The FILLER phrase causes FILLER data items to be initialized rather than ignored. The VALUE phrase causes initialization to the literal value specified in the VALUE clause associated with the elementary data item being initialized. The DEFAULT phrase causes items to be initialized to default values (spaces, zero, or null) when the VALUE or REPLACING phrases are specified but neither of these values is applicable to the elementary data item to be initialized. The INITIALIZE statement also now includes the new category-name, DATA-POINTER, for initializing data pointer data items and allows initialization of variable-occurrence data items. For more information, see "INITIALIZE

Statement" in Chapter 6: *Procedure Division Statements* of the *RM/COBOL Language Reference Manual*.

- **USE Statement Enhancement.** The USE statement now allows a series of OPEN modes, with or without a series of filenames, so that a single USE procedure may be declared for multiple open modes or specific files. See the section "USE Statement" in Chapter 5: *Procedure Division* of the *RM/COBOL Language Reference Manual* for more information.

- **Literals Passed "BY CONTENT".** The RM/COBOL version 7.5 compiler has been modified to pass literals specified in the USING phrase of CALL statements as if the BY CONTENT phrase applied. This change was made to protect the value of the literal in the calling program from inadvertent changes made to the corresponding Linkage Section data item in the called program. A new keyword in the COMPILER-OPTIONS configuration record (on page 287), SUPPRESS-LITERAL-BY-CONTENT, has been added to override this new behavior until the COBOL source program can be corrected. For additional information, see Argument Considerations (on page 216).

- **Compiler Registration.** The RM/COBOL version 7.5 for Windows compiler now consists of a client (either the console-mode client, **rmcobolc.exe**, or the GUI-mode client, **rmcobolg.exe**, with either being called **rmcobol.exe**) and a server (**rmcbl75c.dll**). The client components are completely compatible with previous versions of the **rmcobol.exe** program; that is, no command line changes are required. The server DLL, however, must be registered with Windows before RM/COBOL programs can be compiled. This is automatically performed during installation and is an issue only if the compiler is moved to a different directory without being installed again. In this case, either client can be used to re-register the server. You can read more about this topic in Registering the RM/COBOL Compiler and Runtime Executables (on page 54).

- **Compiler Z Option Changed.** The Z Compile Command Option (described on page 149) may no longer be used to restrict the object version level of generated code to object levels 6 and below. Object level 7 corresponds to the RM/COBOL 6.*nn* releases. Eliminating the need for the compiler to generate code for these very old versions improves the efficiency and reliability of the compiler and ensures that the compiler does not need to suppress optimizations that the older runtime systems do not support.

- **Listing Now Includes Replaced Statements.** Source lines that have been replaced because of the REPLACE statement or the REPLACING phrase of the COPY statement are now included as comments in the compilation listing. These lines can be suppressed in the listing by specifying either the C=2 or C=3 Compile Command Option on (see page 144), or the SUPPRESS-REPLACED-LINES configuration value for the LISTING-ATTRIBUTES keyword (see page 296) of the COMPILER-OPTIONS configuration record.

- **More Informative Compiler Output.** The copy level indicator in the compilation listing also has been enhanced to provide information about source lines that have been modified, replaced, or inserted as a result of the REPLACE statement or the REPLACING phrase of the COPY statement. In addition, the presentation of replacement text in the compilation listing has been improved. This topic is discussed beginning on page 154.

- **Error-Only Listing with Debug Information.** The E Compile Command Option (see page 145) is no longer ignored when the Y=2 or Y=3 Compile Command Options are specified. Thus, it is now possible to request an error-only listing while at the same time generating complete debugging information in the object file.

## New Utility Features

New features in the RM/COBOL version 7.5 for Windows and UNIX utility programs include the following:

- **Fast Conversion to New Indexed File Format.** The Indexed File Recovery (recover1) utility (on page 598) has been enhanced to convert indexed files to the new, more reliable version 4 format. This conversion is very efficient, taking only slightly longer than a file recovery done to correct an error. To convert an existing file to the new format, use the Define Indexed File (rmdefinx) utility (on page 594) to change to the new file version in the file header and then use **recover1** to perform the conversion. Some indexed files with very small block sizes cannot be converted to a version 4 format. In this case, when you run **rmdefinx** to change the file version in the header, you will receive notification that the block size is too small for successful conversion.
- **Batch Mode for Changing Indexed Files to File Version Level 4.** New parameters, CONVERT4 and ATOMICIO, have been added to the Define Indexed File (rmdefinx) utility (on page 594) to allow a large number of files to be converted easily to file version 4 either with or without the atomic I/O capability. Use of either of these new parameters will cause rmdefinx to run in "batch mode" without the normal, interactive user prompting.

## More Flexible Licensing

The RM/COBOL compiler, RM/COBOL runtime system, and the Indexed File Recovery utility now require the same sort of license certificates that Cobol-WOW (known as WOW Extensions in release 9) and Relativity have been using for some time. These license certificates are customized for each product and allow for specialized banners, product expiration dates, and easier upgrades.

## Automatic Update Check

The RM/COBOL compiler, the RM/COBOL runtime, and CodeWatch now provide information about available product updates automatically. CodeWatch provides the update information when beginning, while the RM/COBOL compiler and runtime provide the information when they are terminating. The new keyword, DISPLAY-UPDATE-MESSAGES, has been added to the COMPILER-OPTIONS configuration record (on page 287) and to the RUN-OPTION configuration record (on page 322) to control which of the update messages are displayed. It is possible to see all the update messages (the default for an RM/COBOL development system) or just the "urgent" messages (the default for an RM/COBOL runtime system). Urgent messages are used to indicate only important problems that users need to be aware of immediately.

# Version 7.1 for UNIX Enhancements

The following section summarizes the major enhancements available in version 7.1 of RM/COBOL. This summary describes the main features of each enhancement. The *RM/COBOL Language Reference Manual* and this user's guide contain the details regarding these features.

**Note** Many—but not all—features that were new in version 7.0 for Windows are now available on UNIX in version 7.1. For example, the compiler new features and the runtime support for those new features are available on UNIX in version 7.1. Windows-specific features, such as Windows printing are not available in RM/COBOL version 7.1 for UNIX.

## Runtime Linking Eliminated

Versions of RM/COBOL prior to 7.1 required that the runtime system be relinked to add new functionality such as the VanGui Interface Builder. Also, there were numerous different versions of the runtime, including the terminfo version, the termcap version, the runtime with the RM/InfoExpress client, and the FlexGen runtime.

RM/COBOL version 7.1 for UNIX eliminates the need to relink the runtime by using optional support modules to add functionality to the runtime. With version 7.1, there will only be a single version of the runtime and various support modules to provide the additional capabilities. These optional support modules are implemented as UNIX "shared objects." Liant provides support modules with the RM/COBOL runtime for the terminfo and termcap terminal interfaces (selectable at installation, as before), the RM/InfoExpress client interface, and the FlexGen interface routines. Other support modules, such as the VanGui Interface Builder Server and Enterprise CodeBench, are available from Liant. The CodeBridge facility will also generate user-written support modules. Liant expects that new custom add-on, non-COBOL libraries will be available in the future. For more information, see Appendix D: *Support Modules (Non-COBOL Add-Ons)* on page 429.

## UNIX Resource File

A resource file capability is provided to support C$GetSyn (on page 545) and C$SetSyn (on page 569) and to provide stored configuration information for the runtime system and the compiler. A UNIX resource file (on page 28), similar in format to a Windows initialization (**.ini**) file, allows for permanent storage of synonym names and values on UNIX in the same way that the registry file does on Windows. You can use the resource files to customize your RM/COBOL application.

## Automatic Configuration File

UNIX versions of RM/COBOL prior to 7.1 allowed a configuration file to be linked into the runtime, compiler, or recovery utility. Version 7.1 of RM/COBOL for UNIX allows for a configuration file to be located automatically by the runtime system, the compiler, and the recovery utility. This new method is described as an "automatic configuration file." For more details, see Automatic and Attached Configuration Files (on page 282).

## Support for UNIX Added to CodeBridge

CodeBridge, Liant Software's cross-language call system, is in the RM/COBOL version 7.1 system. The CodeBridge Builder uses a template file to produce a C source file. The C source file provides the COBOL/C interface that may be used in an optional support module callable from COBOL programs.

The CodeBridge Builder generates C source modules that are platform-independent. Thus, you can use the CodeBridge Builder on a Windows platform to generate C source files that may be used on either a Windows or UNIX system. For more information, refer to the *CodeBridge* manual, included in the RM/COBOL development system documentation set for this release.

## Enhancements to Configuration Records

RM/COBOL version 7.1 for UNIX includes the following additions to the configuration records:

The ENABLE-LOGGING keyword of the RUN-OPTION configuration record (on page 322) has been enhanced with new values to control the generation of various error and information log files.

The TERM-ATTR configuration record (on page 327) includes a new keyword, ALWAYS-USE-CURSOR-POSITIONING, which determines whether optimized cursor positioning is suppressed when positioning the cursor on the screen.

# Version 7.0 for Windows Enhancements

The following section summarizes the major enhancements available in version 7.0 of RM/COBOL. This summary describes the main features of each enhancement. The *RM/COBOL Language Reference Manual* and this user's guide contain the details regarding these features.

## CodeWatch Debugger Introduced

This release includes CodeWatch, a standalone, source-level debugger for Windows. CodeWatch supports the debugging of any applications without requiring that they be built under the RM/CodeBench or Enterprise CodeBench development environments. CodeWatch uses true 32-bit Windows "look-and-feel" and common Windows controls such as native toolbars, tree lists, progress indicators, and spin buttons. For more information, see the *CodeWatch* manual, which is included in the RM/COBOL development system documentation set for this release. CodeWatch uses the enhanced debugging information that is available when the new debugging options are specified during compilation of a source program. For additional information on the new debugging options, refer to the Enhanced Debugging Options paragraph in Compiler New Features (on page 679).

## CodeBridge Cross-Language Call System Introduced

A new facility that simplifies communication between COBOL programs and non-COBOL subprograms (such as those written in C or C++) is included in RM/COBOL version 7.0 for Windows. Known as CodeBridge, this facility allows COBOL programmers to call external APIs or custom-developed subprograms without introducing "foreign" language and data dependencies into their programs. This means that developers can write C functions using C data types as usual, without worrying about the complex ArgEntry structure or COBOL data types. After the function declaration has been augmented for use as a template, the CodeBridge utility automatically produces a C source file for a bridge DLL. This file contains all the logic to interface to the calling COBOL program, the developer's C functions, and the needed calls to a CodeBridge data conversion library. The developer then compiles this source file, along with the C functions to be called, and links everything together with the conversion library to form the completed non-COBOL DLL. In a similar manner, pre-existing DLL libraries also can be called from RM/COBOL applications.

For a complete description of this facility, refer to the *CodeBridge* manual, which is included in the RM/COBOL development system documentation set for this release.

## Enhanced Windows Printing

The RM/COBOL version 7.0 for Windows runtime contains enhancements to provide more capabilities and flexibility when printing with Windows printer drivers. A new predefined printer device, "PRINTER?", has been added to the runtime. See Windows Printers (on page 306) in the DEFINE-DEVICE configuration record in Chapter 10: *Configuration*. When this device is opened, a standard Windows Print Setup dialog box is presented to the user to allow dynamic selection of the Windows printer.

A library of P$ subprograms, described in P$ Subprogram Library (on page 447), and two options in the Windows registry, the Printer Dialog Always property (on page 79) and the Printer Dialog Never property (on page 79), have been added to allow developer control of this dialog box. The RM/COBOL for Windows compiler can use this capability to select the printer for the listing file.

The RM/COBOL for Windows runtime also has been enhanced to support raw mode printing, which is useful when printing with escape sequences to a network printer on a Windows NT-class server. Limited support for Windows printers via escape sequences is provided as well. See RM/COBOL-Specific Escape Sequences (on page 525).

## Additions to the RM/COBOL Subprogram Library

The RM/COBOL subprogram library now includes the following C$ subprograms. For more details, see Appendix F: *Subprogram Library* (on page 527).

- **C$ClearDevelopmentMode** disables expanded error information reporting (known as "development mode") in many of the C$ and P$ subprograms.
- **C$Delay** relinquishes the CPU for a specified length of time (in seconds).
- **C$GetEnv** and **C$SetEnv** retrieve and set, respectively, the value of environment variables.

- **C$GetLastFileName** retrieves the last file-name and pathname used in a COBOL I/O statement.
- **C$GetLastFileOp** retrieves information about the last COBOL I/O file operation performed.
- **C$GetRMInfo** retrieves information about the RM/COBOL runtime system.
- **C$GetSyn** and **C$SetSyn** retrieve and set, respectively, the values of RM/COBOL synonyms used in the UNIX resource file and in Windows registry file.
- **C$GetSysInfo** retrieves information about the operating system on which the RM/COBOL runtime system is running.
- **C$MemoryAllocate** allocates dynamic memory.
- **C$MemoryDeallocate** deallocates (frees) dynamically allocated memory.
- **C$PlaySound** plays Windows predefined sound events or sound files.
- **C$SetDevelopmentMode** enables expanded error information reporting (known as "development mode") in many of the C$ and P$ subprograms.
- **C$Show** sets the show state of the main RM/COBOL window (rmguife.exe).
- **C$ShowArgs** displays a list of the arguments used to call a subprogram.

## Ability to Use Btrieve Interface

An additional subprogram, C$BTRV (on page 530), makes raw Btrieve functions, including transactions, accessible from RM/COBOL using Btrieve's normal interface.

## Runtime New Features

In addition to the new C$ and P$ subprogram libraries supplied in the runtime, RM/COBOL version 7.0 for Windows runtime system includes the following features:

- **Support for Large Files.** The runtime system allows RM/COBOL 7.0 files to grow past limits imposed in previous versions. On the Windows 9x class of operating systems, files up to 4 gigabytes (GB) are supported. The Windows NT class of operating systems support multiple terabyte files on the NT file system. Only files on FAT32 (file system format supported for Windows 9x-class operating systems) or NTFS (file system format supported on Windows NT-class operating systems) may be larger than 2 GB. See Using Large Files on Windows (on page 107).
- **Enhanced Runtime Performance.** The RM/COBOL version 7.0 for Windows runtime typically runs computational tasks 20 to 30 percent faster than RM/COBOL version 6.51 for Windows.
- **Termination Log Added.** On Windows, a log of termination error messages, including traceback information, can be used to collect information about errors not trapped by declaratives. The ENABLE-LOGGING=TERMINATION and LOG-PATH keywords have been added to the RUN-OPTION configuration record (on page 322) to allow error messages to be collected for later analysis.

- **New Runtime Behavior.** When the RM/COBOL runtime loads a non-COBOL subprogram library using CodeBridge, it builds a table of entry points into that library. If the sequence "RMDLL" is found at the beginning of any entry name, it is removed. See the *CodeBridge* documentation for more information.

- **Runtime Registration.** The RM/COBOL version 7.0 for Windows runtime now consists of two components: a client (**runcobol.exe**) and a server (**rmcblrun.dll**). The client component is completely compatible with previous versions of the **runcobol.exe** program—no command line changes are required. The server DLL, however, must be registered with Windows before RM/COBOL programs can be run. This is automatically performed during program installation and is an issue only if the runtime is moved to a different directory without being installed again. In this case, the client can be used to reregister the server. See the procedures that are described in System Installation for Windows (on page 51).

- **More Flexible Filenames.** Two new options have been added to the RUN-FILES-ATTR configuration record (on page 316). The ALLOW-EXTENDED-CHARS-IN-FILENAMES keyword allows extended characters in filenames. The ENABLE-OLD-DOS-FILENAME-HANDLING keyword supports the way in which filenames are handled in the old DOS-style 8.3 format.

- **LINAGE Configuration Options.** The PRINT-ATTR configuration record (on page 311) now includes two new configuration keywords to assist in configuring LINAGE for page printers, such as laser printers and ink jet printers.

- The new LINAGE-INITIAL-FORM-POSITION keyword defines where the form is positioned in the printer when the file is opened. The RM/COBOL implementation of LINAGE has required in the past that the form be aligned in the printer such that the first line printed is line one of the page body of the first logical page. This was a reasonable requirement when COBOL was mainly used with line printers and continuous forms. However, this requirement is not reasonable for page printers. For page printers, the value TOP-OF-FORM would be specified for this new keyword. The value PAGE-BODY-LINE-1 (the default value) would be specified for line printers when the operator adjusts the form to the first line of the logical page.

- The new LINAGE-PAGES-PER-PHYSICAL-PAGE keyword in the PRINT-ATTR configuration record defines the number of logical pages per physical page. LINAGE specifies logical pages, not physical pages. Therefore, the RM/COBOL implementation of LINAGE did not use form feed characters when printing LINAGE files since form feed characters advance the printer to the next physical page. For a page printer, a form feed character is sometimes necessary to advance to the next physical page. When LINAGE-PAGES-PER-PHYSICAL-PAGE is set to a nonzero value, then a physical page break (normally a form feed character) is printed each time that number of logical pages has been printed. When LINAGE-PAGES-PER-PHYSICAL-PAGE is set to 0 (the default value), then physical page breaks are not generated for LINAGE files.

- **Inclusion of RM/Panels Runtime.** The RM/Panels runtime is now included with the RM/COBOL runtime.

## Compiler New Features

While RM/COBOL version 6.61 for UNIX contained numerous changes to the compiler, RM/COBOL version 7.0 for Windows features even more capabilities, including the following:

- **Syntax Summary Online Help File.** The full RM/COBOL language syntax is included in an online help file along with examples, coding templates, and tips. Descriptions of the Compile, Runtime, and Debug Commands are also included.
- **Increased Compiler Capacity.** RM/COBOL version 7.0 for Windows compiler allows 65535 identifiers to be defined in a single program (up from the 8192 allowed in version 6.61). The maximum space for user-defined words has been increased such that all 65535 identifiers could have unique names 30-characters in length (up from the 21-character average limit for version 6.61). The changes that support the increased capacity also eliminate the problem that limited consecutive comment lines to around 800. The limit is now about 18,000 consecutive comment lines.
- **New Reserved Words.** To support some of the features described in the following paragraphs, the reserved words list has been extended with the following new words: ADDRESS, CENTURY-DATE, CENTURY-DAY, COUNT-MAX, COUNT-MIN, DATE-AND-TIME, DAY-AND-TIME, NULL, NULLS, and RETURNING.

  Note that if you use these reserved words as user-defined words, you must either change the spelling of these user-defined words or use the DERESERVE keyword in the COMPILER-OPTIONS configuration record.
- **ACCEPT Statement Enhancements.** A number of changes have been made to the ACCEPT statement related to improving the way dates and times are handled. These changes provide additional ways of writing Y2K-compliant COBOL. New phrases include CENTURY-DATE, CENTURY-DAY, DATE-AND-TIME, DATE-COMPILED, and DAY-AND-TIME. For more information, see the *RM/COBOL Language Reference Manual* and Composite Date and Time (on page 218).
- **Constant-Names.** Compile-time constants can now be defined with constant-names declared in level-number 78 data description entries. Once defined, a constant-name can be used in almost all contexts where a literal or integer is required in the language syntax. When properly used, constant-names greatly simplify the maintenance of COBOL programs.
- **POINTER Data Types.** The POINTER data type has been added. Pointer data items can be used to point to other data items in the program or in allocated memory. Support for pointer data items includes the NULL and NULLS figurative constants, which are pointer literals with a null pointer value. There are two new formats of the SET statement to manipulate pointer data items and an ADDRESS OF special register for obtaining the address of a data item as a pointer value. A non-null pointer refers to an area of memory that may be accessed in COBOL by setting the base address of a level-number 01 or 77 data item described in the Linkage Section of the program to the pointer value. Dynamic memory allocation is supported by two new subprograms in the provided library, C$MemoryAllocate (on page 552) and C$MemoryDeallocate (on page 553).
- **Binary Allocation Configuration.** New configuration options allow binary numeric data items to be allocated in the minimum size necessary to support the specified PICTURE character-string consistent with the configured sizes.

RM/COBOL has traditionally allocated binary numeric data items as two, four, eight, or sixteen bytes. The new configuration options allow binary numeric data items described with one or two digits to be allocated as a single byte. For more details on binary numeric data item allocation configuration, see the description of the BINARY-ALLOCATION and BINARY-ALLOCATION-SIGNED keywords in the COMPILER-OPTIONS configuration record (on page 287).

- **Binary Allocation Override.** In addition to binary allocation configuration, the compiler now supports a binary allocation override specification in the USAGE clause. The binary allocation override specification is an integer, enclosed in parentheses, that follows a binary usage type (COMPUTATIONAL-4, COMP-4, or BINARY). The integer specifies the number of bytes to allocate, overriding the number of bytes that would have been allocated based on the current compiler configuration. The override specification may specify fewer bytes than would be required to support the decimal precision indicated by the PICTURE character-string.

  **Note** The binary allocation override feature also applies to native binary data items (COMPUTATIONAL-5, COMP-5), which were added in version 8.0.

- **COUNT, COUNT-MAX, and COUNT-MIN Special Registers.** The compiler now supports three new special registers, COUNT OF *data-name-1*, COUNT-MIN OF *data-name-1*, and COUNT-MAX OF *data-name-1*, that may be used to obtain the number of occurrences of a table data item. For a fixed occurrence table, COUNT, COUNT-MAX, and COUNT-MIN all return the fixed number of occurrences specified in the OCCURS clause. For a variable occurrence table, COUNT-MIN returns the minimum number of occurrences specified in the OCCURS clause, COUNT returns the current number of occurrences (that is, the current value of the DEPENDING data item specified in the OCCURS clause), and COUNT-MAX returns the maximum number of occurrences specified in the OCCURS clause.

- **LENGTH Special Register.** The compiler now supports a new special register, LENGTH OF *identifier-1*, which may be used to obtain the length in bytes of any data item. The length for most data items is a constant, but the length is a variable for variable size groups and for identifiers that specify reference modification.

- **PROGRAM-ID Special Register.** The compiler now supports a new special register, PROGRAM-ID, which may be used to obtain the program-name of any program that specifies this special register.

- **In-Line Comments.** The compiler now supports the *> symbol as an in-line comment introducer. Any characters on the same source record following *> will be treated as commentary. The *> symbol must be preceded by a space separator.

- **OMITTED Arguments.** The compiler now supports the keyword OMITTED in the USING list of a CALL statement. Since arguments are positional, this feature allows an argument to be omitted from other than the end of the USING list.

- **GIVING/RETURNING Phrase.** The compiler now supports the GIVING/RETURNING phrase in the Procedure Division header and in a CALL statement. This phrase specifies an additional argument intended as the output argument of a called program.

- **Formal Arguments (USING/GIVING).** The compiler now handles as a special case the specification of a formal argument as an actual argument in a

CALL statement or in a reference modified identifier reference. In these two cases, the reference is evaluated according to the description of the actual argument corresponding to the formal argument rather than using the Linkage Section description of the formal argument. This means that a program that is just an intermediary between two programs need not have a Linkage Section data description entry that accurately describes the size of the actual argument being passed through it. For example, calling C$CARG with a formal argument, which is described as longer than the corresponding actual argument, will no longer result in a data reference error. Instead, C$CARG will return the correct length of the actual argument, and because of the reference modification change described here, this length may be successfully used to reference modify the formal argument. This also means that a program can call the supplied subprogram C$CARG with an argument that the calling program omitted without getting a data reference error. In this case, the call to C$CARG will succeed and return an argument descriptor that includes a type of OMITTED and a length of zero.

- In the case of reference modification, an omitted actual argument would cause a data reference error, but for an argument that is not omitted, the reference modification can use any offset and length combination that is consistent with the actual argument. Previous to this enhancement, reference modification that used variables implied a reference to the item as described in the Linkage Section for the formal argument data item and this implied reference, if larger than the corresponding actual argument, would cause a data reference error before the reference modification was applied.

- **New Listing Date Formats.** The compilation listing date format can now be configured to include four-digit years with the new format values DDMMYYYY, MMDDYYYY, YYYYMMDD, and YYYYDDD. The default listing date format has been changed to MMDDYYYY so that a four-digit year will be used by default. This change also included making the DATE-COMPILED paragraph use the same date format as the compilation listing header.

- **Enhanced Debugging Options.** The Y Compile Command Option has two new variations that direct the compiler to embed additional debugging information in the program object file. The additional debugging information allows CodeWatch to display the program source during execution without requiring that the program be compiled under Enterprise CodeBench. See the description of the Y Compile Command Option in Object Program Compile Command Options (on page 147), and the DEBUG-TABLE-OUTPUT keyword in the COMPILER-OPTIONS configuration record (on page 287).

- **Argument Linkage Configuration.** The COMPILER-OPTIONS configuration record now includes the keyword LINKAGE-ENTRY-SETTINGS to configure the behavior of arguments and based linkage items upon subprogram entry during multiple calls to a subprogram in a run unit. The keyword supports values that provide behavior corresponding to Micro Focus COBOL behavior for the various settings of the Micro Focus COBOL compiler directive STICKY-LINKAGE. The keyword also supports behavior matching certain IBM COBOL implementations. For more information, see the LINKAGE-ENTRY-SETTINGS keyword in the COMPILER-OPTIONS configuration record (on page 287).

- Source lines that have been replaced because of the REPLACE statement or the REPLACING phrase of the COPY statement are now included as comments in the compilation listing. These lines can be suppressed in the listing by specifying the C=2 or C=3 Compile Command Option or the SUPPRESS-

REPLACED-LINES configuration value for the LISTING-ATTRIBUTES keyword of the COMPILER-OPTIONS configuration record (on page 287).

- The copy level indicator in the compilation listing also has been enhanced to provide information about source lines that have been modified, replaced or inserted as a result of the REPLACE statement or the REPLACING phrase of the COPY statement. In addition, the presentation of replacement text in the compilation listing has been improved.
- The E Compile Command Option is no longer ignored when the Y=2 or Y=3 Compile Command Options are specified. Thus, it is now possible to request an error-only listing while at the same time generating complete debugging information in the object file.

## Enhanced File Recovery Performance

For large files, the Indexed File Recovery (recover1) utility (on page 598) runs at least twice as fast as previous versions. The -m option has been added to allow the user to specify a larger amount of memory to be used for the recovery. Allocating more memory generally results in much faster recovery.

## New rmpgmcom Utility Option

A new option in the Combine Program (rmpgmcom) utility (on page 583), STRIP, can remove COBOL symbol table and debug line table information produced by the compiler Y Option from object files.

# Version 6.6 Enhancements

The following section summarizes the major enhancements available in version 6.6 of RM/COBOL. This summary describes the main features of each enhancement. The *RM/COBOL Language Reference Manual* and this user's guide contain the details regarding these features.

## Override Date/Time Feature for Year 2000 Testing

Beginning with RM/COBOL version 6.61, a new feature has been added to assist users in testing for Year 2000 and other future date/time problems in their COBOL programs. Users can now set the initial date and time when the runtime starts. This allows the user to test parts of an application without having to change the actual date and time on the machine. For more information, see the ALLOW-DATE-TIME-OVERRIDE keyword in the COMPILER-OPTIONS configuration record (on page 287).

## Increased Compiler Capacity

The ability to compile large programs has been improved by allowing more identifiers and user-defined words during compilation. The maximum number of identifiers has been increased from 3612 to 8192, while the maximum space available for user-defined words has been doubled. The new user-defined word limit

allows for 8192 user-defined words that average 21 characters in length, but these cannot all be identifiers since procedure-names must also fit in this space. The cross reference capability of the compiler was also improved to correctly cross reference source programs up to 65535 lines in length, up from the 16384 lines supported in prior versions.

## Improved Compiler Performance for Large Programs

The compiler performance has been improved for large programs by adjustments to the memory allocation algorithms.

## New Statistics in Compilation Listing File

The compiler now provides additional statistics regarding how much identifier table space and user-defined word space has been consumed to compile a source program. The messages are intended to help project managers avoid surprises in suddenly having a source program exceed one of the limits. The messages are now part of the Program Summary Statistics portion of the listing. Here is an example of the messages:

```
Source program used 4489 (55%) of 8192 available identifiers
   (T28 limit).

Source program used 33004 (50%) of 65536 available user-defined
   word space (T48 limit).
```

The new compiler also offers additional statistics regarding the use of memory during compilation. The message helps the user establish a proper setting of the workspace size compiler option (W command line option and WORKSPACE-SIZE keyword of the COMPILER-OPTIONS configuration record). The message is now part of the Program Summary Statistics portion of the listing. Here is an example of the message:

```
Maximum compilation memory used was 487K bytes (2 presses and
   0 increases required).
```

Memory presses and increases occur in the compiler to help minimize the amount of memory used, but at the cost of compilation speed. Minimizing the number of presses by increasing the workspace size setting yields the fastest compilation. A small number of presses do not affect compiler speed.

## Double-Byte Character Set (DBCS) Support

Double-byte character set (DBCS) characters are now supported by RM/COBOL when running on those versions of UNIX that allow their use. See the description of the DBCS-CHARACTERS keyword in the TERM-ATTR configuration record (on page 327).

## Enhanced Indexed File Recovery Program

The Indexed File Recovery (recover1) utility (on page 598) has been enhanced to improve its ability to repair damaged indexed files. **recover1** is now able to repair not only those problems that, in previous versions, required the use of the **recover2** utility, but other problems as well. The enhancements to **recover1** have made the **recovery** and **recover2** utilities obsolete. However, **recovery** and **recover2** are still included on the distribution media for backward compatibility.

## Masked Input and Output

The RM/COBOL runtime now allows dynamic input based on a template supplied with a new keyword, MASK, that can be specified in the CONTROL phrase in ACCEPT and DISPLAY statements. This capability applies to UNIX only. For more details, see ACCEPT and DISPLAY Phrases (on page 195).

## Support For Large Files

When running under operating systems that support files larger than 2 GB (gigabytes), the runtime system will now allow RM/COBOL files to grow past limits imposed in previous versions. This support is provided by the LARGE-FILE-LOCK-LIMIT keyword of the RUN-FILES-ATTR configuration record (on page 316). In order to use this new limit on relative or sequential files, you must use the USE-LARGE-FILE-LOCK-LIMIT keyword in a RUN-REL-FILES configuration record (on page 325) or a RUN-SEQ-FILES configuration record (on page 326). In order to use this new limit on indexed files, you must use an indexed file version of 3. For more information, see Very Large File Support (on page 221). Additional information about systems that support large files also can be found in Using Large Files on UNIX (on page 47) and Using Large Files on Windows (on page 107).

# Version 6.5 Enhancements

The following section summarizes the major enhancements available in version 6.5 of RM/COBOL. This summary describes the main features of each enhancement. The *RM/COBOL Language Reference Manual* and this user's guide contain the details regarding these features.

## Full 32-Bit Implementation

RM/COBOL is now implemented in 32-bit code across all platforms (UNIX and Windows). One common RM/COBOL runtime system or compiler will execute under the various Windows operating systems. On the Windows 9x class of operating systems, the COBOL programs can make calls to non-COBOL subprograms in either 16-bit or 32-bit dynamic-link libraries (DLLs). On the Windows NT class of operating systems, COBOL programs can make calls only to non-COBOL subprograms in 32-bit DLLs.

## Windows Registry Support

RM/COBOL for Windows makes use of the Windows registry to maintain program properties. This feature eliminates the need for initialization (**.ini**) files. The tabs in the Properties dialog box for the COBOL object program can be used to set values in the registry. See Windows Registry (on page 66) for more information. The new RM/COBOL Configuration (rmconfig) utility (on page 615) also can be used to set these properties by displaying the Properties dialog box.

**Note** The Initialization File to Windows Registry Conversion (ini2reg) utility (on page 614), available only in Windows, converts an RM/COBOL for Windows initialization file and places its contents into the registry database.

## Extensions for 32-bit Windows

RM/COBOL for Windows now supports various Microsoft 32-bit Windows extensions, including long filenames and 3D controls.

## Automated System Installation and Removal

An automated system installation and removal capability is now a part of RM/COBOL for Windows. This feature greatly simplifies the loading and unloading of RM/COBOL. It is especially useful in ensuring that the appropriate programs are included or removed when upgrading to new versions. See System Installation for Windows and System Removal for Windows for more information.

## Right Mouse Button Pop-Up Menu

RM/COBOL now provides the ability to define a pop-up menu that displays when the right mouse button is clicked in the client area of the RM/COBOL window. For more information about setting pop-up menu properties, see Setting Pop-Up Menu Properties (on page 93). A new subprogram, C$RBMenu (on page 556), also can be used to define such a pop-up menu temporarily.

## New Subprograms for Windows

Several other new subprograms, which are supported only under Windows, have been added in version 6.5. C$GUICFG (on page 547) temporarily changes the RM/COBOL graphical user interface. C$TBarEn (on page 573) enables and disables the toolbar buttons in the RM/COBOL window. C$TBarSeq (on page 574) sets the bitmap sequence of a toolbar button in the RM/COBOL window.

## Window Style and the SYSTEM Non-COBOL Subprogram

The Windows version of RM/COBOL now allows you to set the style of the window created when you use the SYSTEM non-COBOL subprogram. For more information, see SYSTEM Window Type property (on page 83).

## Btrieve Adapter Enhancements

RM/COBOL version 6.5 for 32-bit Windows includes an **rmbtrv32.dll**, which is the 32-bit version of the 16-bit **rbadaptr.dll** that was shipped with RM/COBOL version 6.08 for 16-bit Windows.

**rmbtrv32** supports the following RM/COBOL version 6 features: split keys, duplicate prime keys, multiple record locks, record lock timeouts, and START with FIRST or LAST. In addition, **rmbtrv32** supports the RUN-INDEX-FILES DATA-COMPRESSION and BLOCK-SIZE keywords, and **rmbtrv32** returns expanded error codes for better error reporting. **rmbtrv32** also supports selected features of Btrieve version 6 and 6.1 files as well as Btrieve version 6.15 MicroKernel Database Engines. **rmbtrv32** supports the Btrieve maximum of 119 key segments, repeating duplicates, and the no currency change (NCC) option on insert and update operations.

Using repeating duplicates along with the NCC option should eliminate the possible position-lost errors that could occur when a second user deleted records as the first user was reading through them.

**rmbtrv32** allows pre-created Btrieve files that have multiple alternate collating sequences (ACS) defined, although all Btrieve keys that map to RM/COBOL keys must use ACS number zero since COBOL defines one ACS per file.

Setting RUN-INDEX-FILES DATA-COMPRESSION=NO allows the user to have **rmbtrv32** create uncompressed Btrieve files (earlier versions of **rbadaptr** always created compressed Btrieve files, which forced some users to pre-create uncompressed files).

For further information on these features, see Chapter 4: *System Considerations for Btrieve* (on page 111).

## Attached Configuration Files on Windows

The Attach Configuration (rmattach) utility (on page 613) now allows configuration files to be physically attached to **rmcobol.exe**, **runcobol.exe**, and **recover1.exe** under Windows. When attached, a configuration file will be processed prior to any configuration file specified with a command line option.

## Built-In Configuration File under UNIX

The UNIX version of RM/COBOL allows a configuration file to be linked in to the compiler and runtime system. For more information, see Automatic and Attached Configuration Files (on page 282).

## Year 2000 Subprogram

In order to facilitate updating RM/COBOL programs to handle the year 2000 issue, this release provides a non-COBOL subprogram called C$Century (on page 534). This subprogram retrieves the first two digits of the current year. For example, for the year 1999, it will return 19; for the year 2000, it will return 20.

## C$RERR Eleven-Character Extended Status

The C$RERR (on page 557) subprogram has been enhanced to return an eleven-character extended status when called with an eleven-character data item. The four-character extended status returned in a four-character data item remains unchanged from previous versions of RM/COBOL.

## Improved recover1 Utility Program

The Indexed File Recovery (recover1) utility (on page 598) has several new enhancements, including:

- Displaying information on why recovery may be required.
- The ability to display the last 98 or 30 error received when accessing the file, and the date and time it occurred.
- Several options to control the amount of user interaction required.
- An option that allows the Open For Modify Count to be reset without performing a full recovery.

## Enhanced rmmapinx Utility Program

The Map Indexed File (rmmapinx) utility (on page 589) includes two significant improvements. It now displays the Open for Modify Count for an indexed file, and it also displays the value of the last 98 or 30 error received while accessing the file. The date and time that the error occurred is also available. In addition, this utility reports split keys correctly.

## Dynamically Configurable Prompt Character

ACCEPT statements may now specify a prompt character in the CONTROL phrase. The specified character is used for that ACCEPT statement only; the default prompt character is not changed. The PROMPT keyword is described under the CONTROL Phrase (on page 195).

## Building Custom Products Using the customiz Shell Script

The User Makefile that was included in previous versions of the RM/COBOL development system for UNIX has been replaced with a UNIX Bourne Shell script. When executed, this script interactively obtains information about the product to be built from the user and generates the appropriate **Makefile**. The user can then use this **Makefile** to build the product.

## Indexed File Block Sizes After OPEN OUTPUT

The manner in which a block size is chosen for an indexed file when OPEN OUTPUT is performed may differ from previous releases. For more information, see the description of the BLOCK CONTAINS clause under Indexed Files (on page 111).

## DELETE FILE under UNIX

The DELETE FILE operation will now fail if the user does not have write permission for both the file to be deleted and the directory containing the file.

## Resolution of Program-Names

The method used to resolve program-names from environment variables has been changed to the method used in earlier versions. The environment is now searched for a matching name only after appending the extensions. This procedure is described in steps 5 through 7 in Subprogram Loading (on page 214).

## Compiler Support for External Access Methods

The RM/COBOL compiler now supports the use of external access methods (such as RM/InfoExpress) to locate source files and copy files. See the EXTERNAL-ACCESS-METHOD configuration record (on page 308).

# Index

## B

## C

## D

## E

## F

## G



## H



## I

## K



## L

## M

## N



## O

## P

## Q



## R

## S

## T

## U



## V

## W



## X



## Y



## Z