Liant Software Corporation

RM/COBOL® to RM/COBOL-85®

Conversion Guide

LIANT

Release Revision History: Part # 300001

First Release	September 1989
Revision 1	July 1992
Addendum	May 1994
Revision 2	November 2003

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
Web site	http://www.liant.com

Contents

List of Tables

Why Convert to RM/COBOL-85?

It has been almost twenty years since the American National Standard COBOL committee published the ANSI COBOL 1985 standard. It becomes increasingly more important that COBOL applications developers who want to remain competitive make the move from programs conforming to the 1974 standard (known as RM/COBOL-74, but referred to as RM/COBOL in this document) to programs conforming to the 1985 standard (known as RM/COBOL, but referred to as RM/COBOL-85 in this document).

Why then have so many developers not made the move?

For one thing, conversion can be painful; such efforts often become nothing more than an experience of encountering one unpleasant surprise after another. For another, there are still some people who do not know the advantages of the new standard. And still others are not convinced that the new standards provide substantial benefits.

What we have done in this conversion guide is set down everything you need to know to move your RM/COBOL programs over to RM/COBOL-85. And not just RM/COBOL programs, but any program compiled on an RM/COBOL "look-alike", or with an RM/COBOL compatibility switch.

Many of the incompatibilities created by the change in standard are resolved by RM/COBOL-85 through the system design or through a set of compiler options. You will notice in Table 1: *Conversion Effort*, that in many cases, differences in the standard create no compatibility problems at all. In fact, less than one-third of these differences require source modification.

RM/COBOL-85 provides the full power of the ANSI COBOL 1985 language to the developer. By using RM/COBOL-85, you can reduce development time in half because RM/COBOL-85 code compiles nearly twice as fast as previous versions of RM/COBOL and produces extremely compact object code. Also, since no linking is required, you can complete the application much sooner and spend your time creating new application functionality.

When used in conjunction with the power and flexibility of the extensive RM/COBOL product line, RM/COBOL-85 offers the most comprehensive COBOL development and deployment environment in the world today.

Table 1: Conversion Effort

Topic	No Source Conversion Required	Source Conversion Suggested	Source Conversion May Be Required
Compilation Option Defaults		✓	
Reserved Words	✓		
COMP Data Items		✓	
COMP-1 Data Items		✓	
COMP-3 Data Items		✓	
COMP-6 Data Items		✓	
INDEX Data Items		✓	
SYNCHRONIZED Clause	✓		
Implicit EXIT PROGRAM	✓		
Incorrect Use of Index-Names			✓
Null Literals			✓
Generated END PROGRAM	✓		
SELECT OPTIONAL Phrase		✓	
Compilation Error Checking		✓	
Alphabetic Data Items*	✓		
BLOCK CONTAINS Clause			✓
RECORD CONTAINS Clause			✓
File Formats	✓		
File Access Names	✓		
Record Lock Detection			✓
OPEN OUTPUT Statement			✓
ACCEPT Statement		✓	
ACCEPT and DISPLAY with Default Positioning		✓	
An ACCEPT that Extends Beyond the Screen Line Length		✓	

Table 1: Conversion Effort (C*ont.*)

Topic	No Source Conversion Required	Source Conversion Suggested	Source Conversion May Be Required
ACCEPT of a Numeric Without the CONVERT Phrase		✓	
Interactive Debugger and Error Messages	✓		
C Language Calling Sequence			✓
CALL . . . ON OVERFLOW Condition		✓	
Line-Sequential File Blank Stripping			✓
Line-Sequential File Blank Padding			✓
Undefined Arithmetic Operations	✓		
CLOSE . . . WITH NO REWIND Statement			✓
Variable-Length Groups as Destinations			✓
CALL . . . USING Statement			✓
Runtime Error Checking		✓	
ALPHABETIC Class Condition*		✓	
PERFORM . . . VARYING Augmentation/Set Order*		✓	
INSPECT, STRING and UNSTRING Subscripting Evaluation*			✓
I-O Status Values*		✓	

Items marked with an asterisk (*) are changes to both RM/COBOL and RM/COBOL-8X 1.n.

Compilation Differences

There are some areas where RM/COBOL-85 programs and RM/COBOL programs compile differently. Note that in this section, RM/COBOL refers to the system conforming to the 1974 standard and that RM/COBOL-85 refers to the system conforming to the 1985 standard.

Items marked with an asterisk (*) are changes to both RM/COBOL and RM/COBOL-8X 1.*n*. In most of these cases, the -7 Compile Command Option will make these items compatible with RM/COBOL-85.

Note UNIX users specify options as a dash followed by a lowercase letter. DOS users specify options as either an uppercase or lowercase letter with no preceding dash. In order to simplify the presentation, this guide uses only the UNIX style of specifying options.

Compilation Option Defaults

Difference

In the **RM/COBOL** compiler, the -a Option controls two features:

1. The format of sequential files whose source declaration specifies neither LINE nor BINARY. Such files are treated as binary sequential when the -a Option is selected, and as line sequential when the -a Option is not selected.
2. The representation of signs for signed numeric display items for which there is no explicit SIGN specification. Signed numeric display items whose declaration does not include a SIGN clause and is not subordinate to a SIGN clause are treated as TRAILING combined when the -a Option is selected, and as TRAILING separate when the -a Option is not selected.

In the **RM/COBOL-85** environment, two compiler options have been provided to control these two features independently:

1. Sequential files whose declaration does not specify either LINE or BINARY are treated as line sequential when the -v Option is selected, and as binary sequential (unless otherwise configured) when the -v Option is not selected.
2. Signed numeric display items whose declaration does not include a SIGN clause and is not subordinate to a SIGN clause are treated as TRAILING separate when the -s Option is

selected, and as TRAILING combined when the -s Option is not selected.

Resolution

Programs compiled with the RM/COBOL -a Option should not need special treatment, since the RM/COBOL-85 -s and -v Compile Command Options default to the equivalent of the RM/COBOL -a Option.

Other programs may need to be compiled using the RM/COBOL-85 -s Option, -v Option or both, or the source programs may be edited to specify the exact sequential file format and sign representation desired.

Take care to ensure you do not inadvertently mix sign representations for numeric data items passed as parameters between calling and called programs.

Reserved Words

Difference

The **RM/COBOL-85** compiler reserves words not reserved in **RM/COBOL**. Table 2: *Newly Reserved Words*, identifies words reserved in RM/COBOL-85 that were not reserved in RM/COBOL.

Resolution

Programs that use words reserved in RM/COBOL-85 that were not reserved in RM/COBOL require some special treatment in order for them to compile correctly in the RM/COBOL-85 environment. There are several methods to resolve this incompatibility:

1. Edit the source program to alter the spelling of the user-defined words in conflict with the new reserved word list
2. Specify the -2 Compile Command Option; this removes the new reserved words and the language features associated with them.
3. Use the RM/COBOL-85 DERESERVE configuration keyword, which allows you to remove individual words from the reserved word list.

Table 2: Newly Reserved Words

ADDRESS
ALPHABET
ALPHABETIC-LOWER
ALPHABETIC-UPPER
ALPHANUMERIC
ALPHANUMERIC-EDITED
ALSO
ANY
ASCENDING
AUTO
AUTOMATIC
BACKGROUND
BACKGROUND-COLOR
BELL
BOTTOM
CD
CENTURY-DATE
CENTURY-DAY
CF
CH
CLASS
CLOCK-UNITS
COBOL
CODE
COL
COLUMN
COMMON
COMMUNICATION
COMP-4
COMP-5
COMPUTATIONAL-4
COMPUTATIONAL-5
CONTENT
CONTINUE
CONTROL
CONTROLS
CONVERTING
COUNT
COUNT-MAX
COUNT-MIN
CURSOR
DATA-POINTER
DATE-AND-TIME
DATE-COMPILED
DAY-AND-TIME
DAY-OF-WEEK
DE
DEBUG-CONTENTS
DEBUG-ITEM
DEBUG-LINE
DEBUG-NAME
DEBUG-SUB-1
DEBUG-SUB-2
DEBUG-SUB-3
DEBUGGING
DEFAULT
DELIMITED
DELIMITER
DESCENDING
DESTINATION
DETAIL
DISABLE
EGI
EM1
ENABLE
END-ACCEPT
END-ADD
END-CALL
END-COMPUTE
END-DELETE
END-DIVIDE
END-EVALUATE
END-IF
END-MULTIPLY
END-OF-PAGE
END-PERFORM
END-READ
END-RECEIVE
END-RETURN
END-REWRITE
END-SEARCH
END-START

Table 2: Newly Reserved Words *(Cont.)*

END-STRING
END-SUBTRACT
END-UNSTRING
END-WRITE
ENTER
EOP
ESCAPE
ES1
EVALUATE
EVERY
EXCLUSIVE
EXTERNAL
FALSE
FINAL
FIXED
FOOTING
FOREGROUND
FOREGROUND-COLOR
FULL
FUNCTION
GENERATE
GLOBAL
GOBACK
GROUP
HEADING
HIGHLIGHT
ID
INDICATE
INITIALIZE
INITIATE
LAST
LENGTH
LIKE
LIMIT
LIMITS
LINAGE
LINAGE-COUNTER
LINE-COUNTER
LOWLIGHT
MANUAL
MERGE
MESSAGE
MULTIPLE
NEGATIVE
NULL
NULLS
NUMBER
NUMERIC-EDITED
OPTIONAL
ORDER
OTHER
PACKED-DECIMAL
PADDING
PAGE-COUNTER
PF
PH
PLUS
POINTER
POSITIVE
PREVIOUS
PRINTING
PROCEDURES
PURGE
QUEUE
RD
RECEIVE
RECORDING
REFERENCE
REFERENCES
RELEASE
REMARKS
REMOVAL
REPLACE
REPORT
REPORTING
REPORTS
REQUIRED
RERUN
RESET
RETURN
RETURN-CODE
RETURNING
REVERSED
REVERSE-VIDEO

Table 2: Newly Reserved Words *(Cont.)*

RF	SUM
RH	SUPPRESS
SCREEN	SYMBOLIC
SD	TABLE
SEARCH	TAPE
SECURE	TERMINAL
SEGMENT	TERMINATE
SEGMENT-LIMIT	TEST
SEND	TEXT
SORT	THEN
SORT-MERGE	TOP
SOURCE	TRUE
STANDARD-2	TYPE
STRING	UNDERLINE
SUB-QUEUE-l	UNSTRING
SUB-QUEUE-2	UPON
SUB-QUEUE-3	VARIABLE

Computational Data Items

There are differences and incompatibilities between RM/COBOL-85 and RM/COBOL when COMP, COMP-1, COMP-3 and COMP-6 data items are present.

COMP Data Items

Difference

The **RM/COBOL-85** compiler uses the hexadecimal value 0C in the sign data character position to indicate a nonnegative value, and the hexadecimal value 0D to indicate a negative value.

The **RM/COBOL** compiler uses the hexadecimal value 0B in the sign data character position to indicate a nonnegative value, and the hexadecimal value 0D to indicate a negative value.

This difference causes a problem when comparing for equality or inequality between a data item with an RM/COBOL nonnegative sign and a literal or a data item with an RM/COBOL-85 nonnegative sign. This difference also causes a problem when comparing two groups, one containing data items with the RM/COBOL nonnegative sign and the other containing data items with the RM/COBOL-85 nonnegative sign.

Resolution

Data items using the RM/COBOL sign convention only occur in existing data files written under RM/COBOL. To fix the sign of positive COMP

data values, copy the file performing an ADD 0 TO statement on every signed COMP field of every record.

When the -2 Option is present in the RM/COBOL-85 Compile Command or when Suppress-Numeric-Optimizations=YES is specified on the COMPILE-OPTIONS configuration record, the comparison for equality or inequality between two nonnegative COMP data items or a nonnegative COMP data item and a nonnegative literal will be based on the digit positions only. Identical values with differing representations of a positive sign will compare equal. The RM/COBOL files should still be converted as described above because of problems that can arise from group comparisons and other nonnumeric use of the data.

The RM/COBOL-85 compiler provides a compiler configuration option that allows COMPUTATIONAL data types to use RM/COBOL data files containing these data types. For more information, refer to the description of the COMPUTATIONAL-VERSION keyword of the COMPILER-OPTIONS configuration record in the *Configuration* chapter of the RM/COBOL-85 user's guide.

COMP-1 Data Items

Difference

In **RM/COBOL-85**, the treatment of COMP-1 data items is the same as that of BINARY or COMP-4 data items. The PICTURE character-string is used to determine the number of decimal digits that may be represented in the value of the data item as required for ANSI COBOL 1985. This affects three separate items:

1. The detection of the size error condition is based on whether a nonzero high order decimal digit must be truncated from the result.
2. In a MOVE statement with a COMP-1 data item as a receiving operand, high order decimal digits are truncated from the sending operand, if necessary, as defined by the PICTURE character-string of the COMP-1 data item.

   Note RM/COBOL-8X failed to do this when the sending operand is also binary and has a different PICTURE character-string. This has been corrected in RM/COBOL-85.
3. A literal in a VALUE clause associated with a COMP-1 data item will cause an error diagnostic if its value requires truncation of nonzero decimal digits in order to match it to the PICTURE character-string.

In **RM/COBOL**, the PICTURE character-string for a COMP-1 data item is ignored except in the case of a MOVE statement where the COMP-1 data item is a sending operand and the receiving operand is alphanumeric. The detection of the size error condition and a VALUE literal error is based on the maximum (32767) and minimum (-32768) signed binary values representable in a two's complement 16-bit word.

In a MOVE statement, when the value of the sending operand is 32767 or greater, the COMP-1 data item receives the value 32767; when the value of the sending operand is -32768 or less, the COMP-1 data item receives the value -32768.

Resolution

The -2 Compile Command Option will cause the RM/COBOL-85 compiler and runtime system to ignore the PICTURE character-string for all COMP-1 data items except in the case of VALUE literals. When the -2 Option is specified, a VALUE literal is not truncated as defined by the PICTURE character-string but the compiler will produce a warning diagnostic if the decimal value is larger than allowed by the PICTURE character-string (an error diagnostic will result if it is less than -32768 or greater than +32767). The warning may be ignored since the results are the same as those for RM/COBOL, but may be avoided by correcting either the PICTURE character-string or the VALUE literal. The -2 Option prevents the truncation of higher-order decimal digits in a MOVE statement with a COMP-1 receiving operand, but does, not cause identical results when values greater than +32767 or less than -32768 are moved to a COMP-1 data item; in those cases, high-order bits are truncated rather than the maximum or minimum value being supplied. When the -2 Option is specified, BINARY usage is treated as equivalent to COMP-1 usage since RM/COBOL does not have BINARY usage.

Note We recommend that you replace COMP-1 usage with BINARY usage for better conformance to the COBOL language.

COMP-3 Data Items

Difference

The **RM/COBOL-85** compiler always treats COMP-3 data items without an S in the PICTURE character-string as positive. This is indicated on the allocation map by marking the type as NPP, annotated with PACKED UNSIGNED. The -2 Option directs the RM/COBOL-85 compiler to treat COMP-3 data items without an S in the PICTURE character-string as signed (that is, as if an S had appeared in the PICTURE character-string).

The **RM/COBOL** compiler treats COMP-3 data items without an S in the PICTURE character-string as signed. This is indicated on the allocation map by marking the type as NPS, annotated with PACKED SIGNED.

Resolution

To avoid this difference in treatment of COMP-3 data items whose PICTURE character-string does not contain an S, either edit the source program to include an S in the PICTURE character-strings of all COMP-3 data items, or compile with the -2 Option.

Difference

The **RM/COBOL-85** compiler uses the hexadecimal value C in the rightmost half-character position (that is, the sign position) to indicate a nonnegative value and the hexadecimal value D in the sign position to indicate a negative value for COMP-3 data items with an S coded or assumed in the PICTURE character-string. The hexadecimal value F is placed in the sign position for COMP-3 data items without an S coded or assumed in the PICTURE character-string.

The **RM/COBOL** compiler uses the hexadecimal value F in the sign position to indicate a nonnegative value and the hexadecimal value D in the sign position to indicate a negative value.

This difference causes a problem when comparing for equality or inequality between a data item with an RM/COBOL nonnegative sign and a literal or a data item with an RM/COBOL-85 nonnegative sign. This difference also causes a problem when comparing two groups, one containing data items with the RM/COBOL nonnegative sign and the other containing data items with the RM/COBOL-85 nonnegative sign.

Resolution

Data items using the RM/COBOL sign convention only occur in existing data files written under RM/COBOL. To fix the sign of positive signed COMP-3 data values, copy the file performing an ADD 0 TO statement on every signed COMP-3 field of every record.

When the -2 Option is present in the RM/COBOL-85 Compile Command or when Suppress-Numeric-Optimizations=YES is specified on the COMPILE-OPTIONS configuration record, the comparison for equality or inequality between two nonnegative COMP-3 data items or a nonnegative COMP-3 data item and a nonnegative literal will be based on the digit positions only. Identical values with differing representations of a positive sign will compare equal. The RM/COBOL files should still be converted as described above because of problems that can arise from group comparisons and other nonnumeric use of the data.

RM/COBOL-85 provides a compiler configuration option that allows COMPUTATIONAL-3 data types to use RM/COBOL data files containing these data types. For more information, refer to the description of the COMPUTATIONAL-VERSION keyword of the COMPILER-OPTIONS configuration record in the *Configuration* chapter of the RM/COBOL-85 user's guide.

COMP-6 Data Items

Difference

The **RM/COBOL-85** compiler treats COMP-6 data items with an S in the PICTURE character-string as an error, since the COMP-6 data item cannot support signed data.

The **RM/COBOL** compiler treats COMP-6 data items with an S in the PICTURE character-string as if the S were not present at all.

In RM/COBOL-85, COMP-6 data items with an S cause a PICTURE USAGE CLASH error. But RM/COBOL-85 treats the data item in the same way **RM/COBOL 2.0** does: as if the S were omitted from the PICTURE character-string.

Resolution

This difference in treatment of COMP-6 data items whose PICTURE character-string contains an S does not result in any difference in runtime behavior, even though the compiler has produced an error diagnostic. To avoid this message, edit the source program to remove the S from the PICTURE character-strings of all COMP-6 data items.

INDEX Data Items

Difference

The **RM/COBOL-85** compiler allocates INDEX data items (not to be confused with index-names) as four bytes.

The **RM/COBOL** compiler allocates INDEX data items as two bytes.

If the -2 Option is selected, RM/COBOL-85 allocates INDEX data items as two bytes.

Resolution

This difference can lead to compatibility problems only when the index data items are embedded in the records of an existing data file, or when they are part of a group data item whose size is fixed by other constraints such as a REDEFINES clause. Under these circumstances, the -2 Option can be used to cause the RM/COBOL-85 compiler to allocate the same amount of space to index data items as the RM/COBOL compiler does.

SYNCHRONIZED Clause

There are three incompatibilities with respect to the SYNCHRONIZED clause between RM/COBOL and RM/COBOL-85.

Differences

1. The **RM/COBOL-85** compiler always forces a group length to be even when a SYNCHRONIZED data item is subject to an OCCURS clause; thus, every occurrence of the SYNCHRONIZED data item will be synchronized.

   The **RM/COBOL** compiler does not force a group length to be even; thus, only the odd occurrences of the SYNCHRONIZED data item will be synchronized when the group length is odd.

2. The **RM/COBOL-85** compiler will not synchronize a group RIGHT or LEFT when the last elementary data item of the group is SYNCHRONIZED.

   The **RM/COBOL** compiler attempts to synchronize a group when the last elementary data item is SYNCHRONIZED.

3. The **RM/COBOL-85** compiler allocates a filler byte, if required, before a group containing an initial elementary data item that is SYNCHRONIZED.

   If the first elementary data item of a group is synchronized, the **RM/COBOL** compiler does not guarantee that the containing groups start at the same address as the first elementary data item (in other words, a filler byte exists before the first elementary data item).

Resolution

These differences should not cause conversion problems, except where the SYNCHRONIZED clause is specified within a record description entry of a data file that was created by RN/COBOL and that must now be read by RM/COBOL-85. If this situation arises, edit the source program so that the RM/COBOL-85 compiler allocates the internal fields of the record in the same way as the RM/COBOL compiler.

Implicit EXIT PROGRAM

Difference

The **RM/COBOL-85** compiler produces an implicit EXIT PROGRAM followed by STOP RUN when the Procedure Division terminates without an explicit transfer of control.

The **RM/COBOL** compiler does not produce the implicit EXIT PROGRAM.

The -2 Option suppresses the generation of an implicit EXIT PROGRAM.

Resolution

There is no difference in runtime behavior between the two versions when the program runs as a main program. However, when the program runs as a called program, the version compiled under RM/COBOL-85 returns control to the calling program when control reaches the end of the Procedure Division; under the same circumstance, the version compiled with RM/COBOL terminates the run unit. To make the two behave in the same way, you can either compile the program with the -2 Option or insert a STOP RUN statement at the end of the Procedure Division.

Incorrect Use of Index-Names

Difference

During compilation, **RM/COBOL-85** detects the incorrect use of index-names to index into tables with a different span than the table associated with the index-name. Also, RM/COBOL-85 produces compilation errors for the SET . . . UP/ DOWN BY *index-name* and SET . . . UP BY/DOWN BY/TO ZERO statements.

Since the **RM/COBOL** compiler does not detect this incorrect usage, some RM/COBOL source programs that use index-names as general subscript variables will not compile correctly with RM/COBOL-85. This only includes the subscripting of tables whose lengths do not match that of the table associated with the index-name, and the use of an index-name in the BY phrase of a SET . . . UP/DOWN statement.

Resolution

The source program must be edited to correct the incorrect usage, possibly by substituting an existing general data-name variable in place of the index-name, or by defining a new data item to be used in place of the index-name.

For example, if you had a data structure like this:

```
01 TABLES.
   02 TAB-A.
      03 ELEM-A PIC XX OCCURS 4 INDEXED BY I.
   02 TAB-B.
      03 ELEM-B PIC XXX OCCURS 4.
```

RM/COBOL would allow the following reference:

```
MOVE ELEM-A (I) TO ELEM-B (I).
```

RM/COBOL-85 disallows the reference to ELEM-B, since its span of 3 is not the same as the span of the table associated with I (2). The -2 Option does not resolve this incompatibility.

You could reconcile the conflict by defining J with:

```
03 ELEM-B PIC XXX OCCURS 4 INDEXED BY J.
```

and replacing the MOVE statement with:

```
SET J TO I.
MOVE ELEM-A (I) TO ELEM-B (J).
```

Null Literals

Difference

The **RM/COBOL-85** compiler produces an error message when it encounters a nonnumeric literal with a length of zero.

The **RM/COBOL** compiler produces a warning message when this occurs.

Resolution

To avoid the warning, the source program must be edited to replace the null literal with a valid operand (possibly a space).

Generated END PROGRAM

Difference

The **RM/COBOL-85** compiler does not provide an END PROGRAM statement.

The **RM/COBOL** compiler provides an END PROGRAM statement when the end of the source file is encountered.

Resolution

No conversion action is required.

SELECT OPTIONAL Phrase

Difference

The **RM/COBOL-85** compiler requires that, unless the OPTIONAL phrase of the SELECT clause for a file has been specified, a file must exist to allow an OPEN EXTEND for that file to complete without error.

The **RM/COBOL** compiler will create a file when an OPEN EXTEND is executed for a non-existing file. The OPTIONAL phrase of the SELECT clause is not supported on **RM/COBOL 2.0**.

Programs compiled with the -2 Compile Command Option function the same as RM/COBOL; that is, a file will be created when an OPEN EXTEND is executed for a nonexisting file.

Resolution

The -2 Option can be specified to cause the RM/COBOL-85 system to behave like the RM/COBOL system in this respect.

Compilation Error Checking

Difference

The **RM/COBOL-85** compiler has more thorough compile-time error checking than the **RM/COBOL** compiler and, therefore, previously undiagnosed errors may occur when compiling with RM/COBOL-85. For example, SELECT statements that begin in area A (columns 8 through 11) compile without warning with RM/COBOL, but RM/COBOL-85 flags these lines with a warning.

Resolution

No conversion action is required, except as noted in the specific differences and incompatibilities addressed elsewhere in this section, as any differences not addressed will result in a nonfatal error or warning. To avoid those nonfatal errors or warnings the source program must be edited to comply with the RM/COBOL-85 requirements.

Alphabetic Data Items*

Difference

With the **RM/COBOL-85** compiler, data items described with only the PICTURE symbols A and B are considered alphanumeric edited data items.

With the **RM/COBOL** and **RM/COBOL-8X** compilers, these data items are considered alphabetic data items with editing specified.

Resolution

The RM/COBOL-85 treatment of such data items results in a relaxation of the rules affecting those data items, and should cause no trouble in existing programs. If a problem should occur, either the -2 or -7 Compile Command Option can be entered to direct RM/COBOL-85 to treat these data items as alphabetic with editing specified.

For example, assume the following data definitions:

```
77 S PIC 9999.
77 T PIC AABAA.
```

RM/COBOL disallows this statement:

```
MOVE S TO T
```

since moving a numeric operand to an alphabetic operand is not allowed. RM/COBOL-85 allows such a move, since it treats the receiving operand as alphanumeric edited.

If a problem should occur, either the -2 or -7 Compile Command Option can be entered to direct RM/COBOL-85 to treat these data items as alphabetic with editing specified.

Operational Differences

There are several operational differences which are described in the RM/COBOL-85 user's guide, and which are highlighted here. Note that in this section, RM/COBOL refers to the system conforming to the 1974 standard and that RM/COBOL-85 refers to the system conforming to the 1985 standard.

BLOCK CONTAINS Clause

Difference

The **RM/COBOL-85** runtime system allows you to specify the size of physical data transfers to and from a file, with the BLOCK CONTAINS clause within the file description entry. This block size is a fixed attribute of RM/COBOL-85 indexed files, and the BLOCK CONTAINS clause must be the same in all programs which access that indexed file.

The **RM/COBOL** runtime system treats the BLOCK CONTAINS clause as commentary. Programs with different values in their BLOCK CONTAINS clause can therefore access the same file.

Resolution

When converting files created by RM/COBOL, the actual value for the BLOCK CONTAINS clause in the source program using the file must be provided to the RM/COBOL-85 conversion utility. Also, all source programs that use a file must have the same value in the BLOCK CONTAINS clause for that file. See the Conversion Utilities section beginning on page 33.

RECORD CONTAINS Clause

Difference

The **RM/COBOL-85** runtime system verifies the length of records read against the minimum and maximum lengths specified in the RECORD CONTAINS clause. Furthermore, the minimum and maximum record lengths are considered fixed attributes of RM/COBOL-85 indexed files, and are validated when the file is opened INPUT or I-O. Therefore, the RECORD CONTAINS clause must accurately describe the minimum

and maximum record lengths of a file, and these lengths must be identical in all programs which access an indexed file.

The **RM/COBOL** runtime system does not validate the length of records read, nor does it validate the minimum record length of indexed files. Programs with different RECORD CONTAINS clauses and with record descriptions of different lengths can therefore access the same file.

Resolution

When converting files created by RM/COBOL, the actual value for the RECORD CONTAINS clause in the source program that will access the file must be used to process the file through the RM/COBOL-85 conversion utility (see page 33). Also, all source programs that access a file must have the same values in the RECORD CONTAINS clause for that file.

File Formats

Difference

The external format for relative, indexed and variable-length binary-sequential files has changed. Existing data files of these types must be converted before use in the **RM/COBOL-85** environment.

Resolution

Use the conversion utilities provided, as described beginning on page 33.

File Access Names

Difference

The **RM/COBOL** runtime system appends the file extension .INX to indexed file access names. The **RM/COBOL-85** runtime system does not.

Resolution

When converting indexed files using the conversion utilities provided (see page 33), be sure to specify the same file access names that will be used by your application.

Record Lock Detection

Difference

Under **RM/COBOL-85**, a READ statement for a file in the OPEN INPUT mode will successfully read a record locked by another user.

Under **RM/COBOL**, a READ statement for a file in the OPEN INPUT mode will not successfully read a record locked by another user.

Resolution

No RM/COBOL-85 application should depend on the locked record error for a file in the OPEN INPUT mode since the successful execution of such a READ statement will not lock the record. Any application with this dependency should be changed to use the OPEN I-O mode.

OPEN OUTPUT Statement

Difference

Under **RM/COBOL-85**, an OPEN OUTPUT statement does not default to WITH LOCK.

Under **RM/COBOL**, an OPEN OUTPUT statement defaults to WITH LOCK.

Resolution

For applications that require exclusive use of a file that is opened OUTPUT, edit the source program to add the WITH LOCK phrase to the appropriate OPEN OUTPUT statement.

ACCEPT Statement

Difference

The operation of the **RM/COBOL-85** ACCEPT statement is compatible with **RM/COBOL 2.1**, but not with **RM/COBOL 2.0**, in the case of output conversion on numeric data items when the UPDATE phrase is specified. In general, RM/COBOL 2.0 does not support output conversion. The effects of output conversion are described in the *Procedure Division Verbs* chapter of the *RM/COBOL-85 Language Reference Manual.*

Resolution

No conversion action is required.

Difference

In the **RM/COBOL** runtime system, the Tab key causes an "erase right" function if the UPDATE phrase is specified and the field termination code is changed to that of the Enter key. In **RM/COBOL-85**, this function is provided differently (see the chapter on *RM/COBOL Features* of the RM/COBOL-85 user's guide).

Resolution

When existing programs are moved from the RM/COBOL environment to the RM/COBOL-85 environment, source programs and documentation that provide prompts or "help" functions concerning the use of the Tab key function with ACCEPT with UPDATE may have to be edited to reflect this difference. Alternatively, the RM/COBOL-85 runtime system may be reconfigured to match the RM/COBOL operation (see the chapter on *Configuration* in the RM/COBOL-85 user's guide).

Difference

The **RM/COBOL-85** runtime system has more stringent input conversion rules than the **RM/COBOL** runtime system (see the chapter on *Procedure Division Verbs* of the *RM/COBOL-85 Language Reference Manual).*

Resolution

No conversion action is needed to accommodate this difference between RM/COBOL-85 and RM/COBOL.

Difference

The **RM/COBOL-85** field termination key codes for those keys not in the generic keyboard set may not agree with some of the more common reconfigurations of **RM/COBOL**.

Resolution

Program terminating fields using keys not in the generic keyboard set may have to have the source program edited to recognize the termination codes provided by RM/COBOL-85 and described in the chapter on *RM/COBOL Features* of the RM/COBOL-85 user's guide. For example, if an RM/COBOL product were reconfigured to include the arrow keys as field terminators, this would represent an incompatibility within RM/COBOL-85, since RM/COBOL-85 uses the arrow keys as intrafield editing keys and they would not necessarily cause field termination or return the same termination codes. Alternatively, the RM/COBOL-85 runtime system may be reconfigured to match the RM/COBOL operation (see the chapter on *Configuration* in the RM/COBOL-85 user's guide).

Difference

The **RM/COBOL-85** runtime system has a more extensive set of intrafield editing functions than the **RM/COBOL** runtime system. In particular, the left and right arrow functions (that is, cursor left and cursor right) use different keys.

Resolution

Documentation and any source programs that provide screen prompts or "help" functions may have to be edited to reflect both the new and different intrafield editing capabilities of RM/COBOL-85 as described in the chapter on *RM/COBOL Features* of the RM/COBOL-85 user's guide. Alternatively, the RM/COBOL-85 runtime system may be reconfigured to match the RM/COBOL operation (see the chapter on *Configuration* in the RM/COBOL-85 user's guide).

ACCEPT and DISPLAY with Default Positioning

Difference

When executing an ACCEPT statement followed by an ACCEPT or DISPLAY using the POSITION phrase with a zero value and no LINE phrase (default positioning), the **RM/COBOL-85** runtime system uses the full length of the screen field in determining the next available position, while the **RM/COBOL** runtime system uses the screen position after the last character entered as the next available position.

Resolution

No conversion action is required, although if you are unsatisfied with the appearance of the input screens, the source program must be edited with this functional difference in mind. RM/COBOL-85 will return the cursor position within the screen field (the CURSOR phrase) which, when added to the starting position of the screen field, may be used to provide the starting position for the subsequent ACCEPT or DISPLAY.

An ACCEPT that Extends Beyond the Screen Line Length

Difference

The **RM/COBOL-85** runtime system does not limit an ACCEPT statement screen field to the screen line length, whereas the **RM/COBOL** runtime system truncates an ACCEPT screen field that extends beyond that length.

Resolution

No conversion action is required, unless this characteristic of RM/COBOL was used to truncate an ACCEPT too long for the CRT line

length, in which case the source programs must be edited to include the SIZE phrase, with an appropriate value, in the ACCEPT statement.

ACCEPT of a Numeric Without the CONVERT Phrase

Difference

Executing an ACCEPT statement of a numeric data item without the CONVERT phrase with the **RM/COBOL-85** runtime system causes numeric input conversion to occur, unless the -m Compile Command Option is selected.

Executing an ACCEPT statement of a numeric data item without the CONVERT phrase with the **RM/COBOL** runtime system causes only right justification and zero filling to occur; decimal alignment, rejection of nonnumeric characters, and conversion error reporting does not occur.

Resolution

No conversion action is required.

Interactive Debugger and Error Messages

Difference

The **RM/COBOL-85** runtime system uses line and intraline numbers in Debug commands and messages as well as runtime diagnostic error messages; these numbers are represented in decimal notation. The **RM/COBOL** runtime system uses addresses in hexadecimal notation.

Resolution

No conversion action is required.

C Language Calling Sequence

Difference

The calling sequence presented to C language programs has changed to provide more information about parameters. See the specific Windows and UNIX appendices on alternate methods of preparing non-COBOL Subprograms in the *CodeBridge* manual.

Resolution

The calling sequence and structure within C language subprograms must be altered, compiled and linked to conform to the RM/COBOL-85 requirements, as described in the *CodeBridge* manual.

CALL . . . ON OVERFLOW Condition

Difference

The **RM/COBOL-85** runtime system causes an overflow condition if, for any reason except a recursive call error, the called program cannot be made available.

The **RM/COBOL** runtime system causes an overflow condition only if there is not enough memory for the called program.

Resolution

Documentation and source file editing may be necessary. For example, if the ON OVERFLOW path displays a message:

```
Insufficient memory, choose a smaller program
```

the message might be changed to:

```
Program unavailable, choose a different program
```

Line-Sequential File Blank Stripping

Difference

The **RM/COBOL** runtime system strips trailing blanks from records before writing those records to a line-sequential file if the file has variable-length records, and does not strip trailing blanks if the file has fixed-length records. The **RM/COBOL-85** runtime system uses a different criterion for blank stripping. **RM/COBOL-85** preserves trailing spaces if the device-name in the ASSIGN clause is DISC, DISK or RANDOM, and removes trailing spaces in all other cases, regardless of whether the records of the file are described as fixed or variable length.

Resolution

If the file is described with variable-length records, no conversion action is required. RM/COBOL-85 will preserve trailing spaces when the device-name is RANDOM or equivalent, thus allowing REWRITE statements that were prohibited under RM/COBOL.

If the file is described as having fixed-length records, some conversion action may be required. If a later program intends to issue REWRITE statements on the file, the program that writes the records must specify a device-name of RANDOM or equivalent. If a device-name other than DISC, DISK or RANDOM is specified, RM/COBOL-85 will remove trailing spaces and subsequent REWRITE statements will fail.

Line-Sequential File Blank Padding

Difference

When reading a line-sequential file declared as having variable-length records, the **RM/COBOL** runtime system always pads the record area with spaces to the maximum record length. In order to properly support REWRITE statements, the **RM/COBOL-85** runtime system returns the actual record length and does not pad the record area with spaces when the device-name in the SELECT clause is DISC, DISK or RANDOM.

Resolution

If the program reading the line-sequential file expects the remainder of the record area to be space-filled when reading records from a variable-length file, either change the device-name in the SELECT clause to CARD-READER, CARD-PUNCH, CASSETTE, INPUT, INPUT-OUTPUT or MAGNETIC-TAPE or insert a MOVE SPACES TO record-area statement before the READ statement.

Undefined Arithmetic Operations

Difference

Performing arithmetic operations on nonnumeric data is undefined in COBOL. The effect of doing such an operation is different for RM/COBOL and RM/COBOL-85.

An example of this incompatibility is the comparison of a numeric data item containing LOW-VALUES to ZERO. The **RM/COBOL-85** runtime system may or may not equate these values in different environments whereas the **RM/COBOL** runtime system always finds these values to be not equal.

Resolution

No conversion action is required unless the effects of arithmetic operations on nonnumeric data that are consistent in RM/COBOL are used.

Specifying SUPPRESS-NUMERIC-OPTIMIZATIONS=YES on the COMPILE-OPTIONS configuration record will prevent the compiler from generating optimized code for nonbinary numeric operations.

CLOSE . . . WITH NO REWIND Statement

Difference

The **RM/COBOL-85** CLOSE statement does not cause an automatic page eject on printers: only the WRITE . . .BEFORE/AFTER ADVANCING PAGE statement does.

The **RM/COBOL** runtime system advances to a new page when a file assigned to a printer is closed (unless the WITH NO REWIND phrase suppresses this action).

Resolution

In circumstances where it is necessary to preserve the automatic new page function supplied by the RM/COBOL system, the source program can be edited to insert a WRITE *record-name* FROM SPACES AFTER ADVANCING PAGE statement before the CLOSE statement.

The RM/COBOL-85 runtime system may be configured to provide a page eject at close (see the chapter on *Configuration* in the RM/COBOL-85 user's guide).

Variable-Length Groups as Destinations

Difference

When a variable-length group that contains its own DEPENDING ON item is used as a receiving operand, RM/COBOL-85 and RM/COBOL determine the length of the group by different rules. The **RM/COBOL-85** runtime system always treats the group as if its DEPENDING ON item had its maximum value, ignoring its actual current value; this treatment is in accordance with the latest interpretation of the ANSI COBOL 1985 standard. The **RM/COBOL** runtime system uses the current value of the DEPENDING ON item to determine the length of the group.

Resolution

In situations where the partial filling of the receiving group is critical to the proper operation of the program, the source program must be modified to make it operate the same under RM/COBOL-85. For example, an intermediate receiving group could be defined that has the same internal structure as the original receiving group, and a PERFORM . . . VARYING statement could be used to move the desired portion of the intermediate group to the original group.

CALL . . . USING Statement

Difference

The **RM/COBOL-85** runtime system compares the size of a data item in the calling program to the size of the corresponding linkage data item in the called program. If the size of the linkage data item in the called program is greater than the size of the corresponding data item in the calling program, the run unit is terminated with a data reference error.

The **RM/COBOL** runtime system does not compare the sizes of these data items.

Resolution

Source programs that have this error must be edited so that the sizes of corresponding data items conform to the RM/COBOL-85 requirement.

Runtime Error Checking

Difference

The **RM/COBOL-85** system has more thorough runtime error checking than the **RM/COBOL** system and therefore previously undiagnosed errors may occur when executing programs with RM/COBOL-85. For example, if a data item is defined using the OCCURS . . . DEPENDING phrase and during execution the value of the DEPENDING identifier is outside of the range of the OCCURS clause, RM/COBOL-85 reports this as an error, while RM/COBOL does not detect this error.

Resolution

No conversion action is required, except as noted in the specific differences and incompatibilities addressed elsewhere in this section, as any other execution errors are the result of illegal operations that were previously undiagnosed. To avoid those errors the source program must be edited to comply with the RM/COBOL-85 requirements.

ALPHABETIC Class Condition*

Difference

In the **RM/COBOL-85** runtime system, the ALPHABETIC class condition is true if the data item being tested contains only the characters A through Z, a through z, and the space.

In the **RM/COBOL** and **RM/COBOL-8X** runtime systems, the ALPHABETIC class condition is true if the data item being tested contains only the characters A through Z and the space. Thus, it is false if lowercase letters appear in the data item.

Resolution

In most cases, the RM/COBOL-85 interpretation of the ALPHABETIC class condition is the desired one for modern computing environments. If the RM/COBOL and RM/COBOL-8X interpretation is desired, either the -2 (RM/COBOL compatibility) or -7 (ANSI COBOL 1974 compatibility) Compile Command Option may be specified to cause the compiler to generate code, which results in the RM/COBOL and RM/COBOL-8X (ANSI COBOL 1974) interpretation of the ALPHABETIC class condition. If the program is converted to use ANSI COBOL 1985 features, the ALPHABETIC-UPPER class condition is available to test for only uppercase letters and the space.

PERFORM . . . VARYING Augmentation/Set Order*

Difference

In the **RM/COBOL-85** runtime system, for the PERFORM . . . VARYING statement when TEST BEFORE is specified or implied and one or more AFTER phrases are specified, each time an inner loop completes, the next outer VARYING or AFTER identifier is augmented by its current BY value and then the inner AFTER identifier is set to its current FROM value. Thus, when the FROM identifier depends on the outer VARYING or AFTER identifier or is the same identifier, as when processing the diagonal of a matrix, the resetting of the inner AFTER identifier will include the effect of the augmentation.

In the **RM/COBOL** and **RM/COBOL-8X** runtime systems, for the PERFORM . . . VARYING statement when one or more AFTER phrases are specified, each time an inner loop completes, the inner AFTER identifier is set to its current FROM value and then the next outer VARYING or AFTER identifier is augmented by its current BY value (setting and augmenting reversed from RM/COBOL-85). This sequence of operations is specified by ANSI COBOL 1974.

Resolution

In most cases, the PERFORM . . .VARYING statement with one or more AFTER phrases has no dependency of the FROM or BY values upon the other VARYING or AFTER identifiers of the same statement. Therefore, this difference would have no effect. When a dependency exists, the RM/COBOL-85 behavior of the PERFORM . . .VARYING statement is usually desired. If there is a use for the ANSI COBOL 1974 behavior or it is desired to avoid causing a problem for an existing program, either the -2 (RM/COBOL compatibility) or -7 (ANSI COBOL 1974 compatibility) Compile Command Option may be specified to cause the compiler to generate code which results in the RM/COBOL and RM/COBOL-8X (ANSI COBOL 1974) behavior of the PERFORM . . . VARYING statement.

For example, in RM/COBOL (and in RM/COBOL-8X) the statement:

```
PERFORM PARA VARYING X FROM 1 BY 1 UNTIL X > 3
                  AFTER Y FROM X BY 1 UNTIL Y > 3
```

results in this sequence of values for X and Y:

```
X: 1 1 1 2 2 2 3 3
Y: 1 2 3 1 2 3 2 3
```

The same statement in RM/COBOL-85 results in this sequence:

```
X: 1 1 1 2 2 3
Y: 1 2 3 2 3 3
```

Note This second sequence of values can be used to refer to the elements of an array that are on or above the diagonal of the array.

INSPECT, STRING and UNSTRING Subscripting Evaluation*

Difference

In the **RM/COBOL-85** runtime system, the subscripting for INSPECT, STRING and UNSTRING statements is evaluated only once, as the first operation of the execution of the statement. Thus, if the statement modifies any of the values of the subscripts used in the statement there will be no effect on the results of executing the statement.

In the **RM/COBOL** and **RM/COBOL-8X** runtime systems, the subscripting for the INSPECT, STRING and UNSTRING statements is evaluated each time an identifier is accessed during the execution of the statement. The source item is only accessed once, as the first operation of the execution of the statement, but control and destination identifiers may be accessed more than once or after the operation of the statement has caused the modification of the value of some identifiers. If the modified identifiers, such as those appearing in the TALLYING,

POINTER or COUNT IN phrases, are also used as subscripts in the same statement, it may affect the results of executing the statement.

Resolution

It is likely that programs that are affected by this difference are not properly coded. Programs that depend on the previous behavior of subscripting within these statements must be modified to conform to the new behavior.

Input Redirection with the STOP Literal Statement

Difference

The RM/COBOL-85 runtime system does not allow STOP literal statement responses (Y or N) to be redirected in its default configuration for UNIX. The responses must be entered at the terminal running the program. The RM/COBOL runtime system does allow STOP literal responses to be redirected in its default configuration for UNIX.

Resolution

If a program depends on the input redirection of STOP literal statement responses, these responses must be entered manually at the terminal running the program, or the source program may be edited to allow ACCEPT statements to be substituted for the STOP literal statement and logic added to cause a STOP RUN if the response is N. Another alternative is to use the ERROR-MESSAGE-DESTINATION keyword in the RUN-ATTR configuration record, described in the *Configuration* chapter of the RM/COBOL-85 user's guide.

I-O Status Values*

Difference

In the **RM/COBOL-85** runtime system, the I-O status values returned in the FILE STATUS data item or displayed in I/O error messages are significantly expanded and modified from the values in RM/COBOL and RM/COBOL-85. The new values are the ones required by the ANSI COBOL 1985.

In the **RM/COBOL** and **RM/COBOL-8X** runtime systems, the ANSI COBOL 1974 I-O status values were returned in the FILE STATUS data item and displayed in I/O error messages.

Resolution

The source program can be edited to expect the new I-O status values, which in most cases are more informative. Alternatively, either the -2 (RM/COBOL compatibility) or -7 (ANSI COBOL 1974 compatibility)

Compile Command Option may be specified to cause the compiler to generate code, which results in the RM/COBOL and RM/COBOL-8X (ANSI COBOL 1974) I-O status values being returned in the FILE STATUS data item and displayed in I/O error messages.

Conversion Utilities

This section provides the documentation for the file conversion utilities for RM/COBOL to RM/COBOL-85. You can download these conversion utility programs from the Liant Web site at http://www.liant.com/support/files/.

Use the following utilities to assist you in converting:

- RM/COBOL (74) versions 1 and 2 indexed files to RM/COBOL format (rmifca and rmifcb)
- RM/COBOL (74) version 1.5 relative files to the RM/COBOL format (rmcrl1)
- RM/COBOL (74) version 2 relative files to RM/COBOL format (rmcrl2)
- RM/COBOL (74) version 2 variable-length binary sequential files to RM/COBOL format (rmcseq)

Note In this section RM/COBOL (74) refers to 74 ANSI standard programs, while RM/COBOL refers to the 85 ANSI standard. Beginning with version 6.5, the –85 suffix was removed from the RM/COBOL product name.

Indexed File Conversion Utilities (rmifca and rmifcb)

The **rmifca** and **rmifcb** utilities convert an RM/COBOL (74) version 1 or version 2 indexed file to the RM/COBOL format.

rmifca and **rmifcb** require the following:

1. RM/COBOL compiler and runtime system.
2. Object file **rmifca.cob**.
3. Source file **rmifcb.cbl**.
4. RM/COBOL (74) version 1 or version 2 indexed file to be converted.
5. Available disk space for the creation of the new RM/COBOL indexed file. The new file size typically will be smaller than or equal to the original.

Indexed File Naming Conventions

RM/COBOL (74) version 1 and RM/COBOL maintain data and index information in a single file. No special index file naming conventions are required.

RM/COBOL (74) version 2 supports a dual-file format. In the dual-file format, both the data file and the index file (which contains both header and indexes) have the same name. The data file has the default filename extension, and the index file always has a filename extension of **.inx**. For example, if the data filename is **inxfl.dat**, the file that contains the header and indexes is named **inxfl.inx**. This requires that you make certain you include the **.inx** filename extension when specifying a version 2, dual-file indexed file format.

Using the Utility

Follow these steps to convert RM/COBOL (74) versions 1 and 2 indexed files to RM/COBOL format:

1. Execute **rmifca** using the following command:

   ```
   runcobol rmifca
   ```

   The program will prompt for the information required to convert the file.

2. In response to the first prompt, enter the name of the indexed file to be converted. If you are converting a version 2 indexed file, be sure to include the **.inx** filename extension as part of the filename.

   If you specify a version 2 dual-file indexed file, you will be requested to enter the filename of the data file portion of the indexed file.

3. The next prompts deal with the BLOCK and RECORD sizes. In order to answer correctly, you may need to refer to the RM/COBOL (74) source program in which the file is described.

    a. Describe the BLOCK CONTAINS clause. Enter the number of RECORDS or CHARACTERS in the block, and indicate whether the number you entered applies to RECORDS or CHARACTERS.

    b. Describe the RECORD CONTAINS clause. If the file you are converting is a version 2 indexed file, the maximum record length will be filled in. You need only specify the minimum record size if it is a variable-length record. If the file you are converting is a version 1 indexed file, you must specify both the minimum and maximum record sizes.

    c. Enter the name of the RM/COBOL indexed file to be created.

    Once all the information has been supplied, **rmifca** generates the copy files necessary to describe the conversion process to the actual conversion utility **rmifcb**.

4. Compile and execute **rmifcb**. The program displays what it is going to do and prompts for confirmation to proceed. The program attempts to estimate processing time.

rmifcb.cob can be renamed and saved for use as a standalone conversion program for versions 1 and 2 indexed files. If you are an application developer, this should ease the process of getting your users converted from a version 1 or version 2 system to the current RM/COBOL environment.

The process described above applies only to copying data records from RM/COBOL (74) version 1 or 2 to RM/COBOL indexed files on an "as is" basis. However, since each data record is available during the execution of **rmifcb**, you may wish to customize a copy of **rmifcb** to provide data conversions, such as changing separate signs to combined signs on numeric USAGE display items. Comments in the source file **rmifcb.cbl** suggest where customized code may be placed.

Relative File Conversion Utility: Version 1.5 to RM/COBOL (rmcrl1)

The **rmcrl1** utility converts RM/COBOL (74) version 1.5 relative files to the RM/COBOL format.

Using the Utility

To convert a version 1.5 relative file to the RM/COBOL format, follow these steps:

1. Copy **rmcrl1.cbl** from the installation directory to a working directory.
2. Edit the source program, replacing—where indicated—the record size for the RM/COBOL (74) version 1.5 input and the RM/COBOL output relative files.
3. Enter the RM/COBOL Compile Command (**rmcobol**) to compile the program.
4. Under UNIX, set environment variables to assign file pathnames for the version 1.5 relative file to INPUT, and the RM/COBOL relative file to OUTPUT. For example:

   ```
   INPUT=/dir1/rel1.5 ; export INPUT

   OUTPUT=/dir2/rel85 ; export OUTPUT
   ```

   Under Windows, use synonyms to assign the file pathnames. Setting synonyms is described in the *Installation and System Considerations for Windows* chapter of the RM/COBOL user's guide.

   Note The source RM/COBOL (74) version 1 pathname and destination RM/COBOL pathname must specify different files, that is, the conversion cannot be made in place.
5. Enter the RM/COBOL Runtime Command (**runcobol**) to execute the program.

Relative File Conversion Utility: Version 2 to RM/COBOL (rmcrl2)

There are two source programs for converting version 2 relative files: **rmcrl2l.cbl** and **rmcrl2s.cbl**. These programs can be used to convert RM/COBOL (74) version 2 relative files to RM/COBOL format. **rmcrl2s.cbl** converts files with record sizes less than 512 bytes. **rmcrl2l.cbl** converts files with larger record sizes.

Using the Utility

To convert a version 2 relative file to RM/COBOL format, follow these steps:

1. Copy the appropriate source program from the installation directory to a working directory.
2. Edit the source program, replacing—where indicated—the record size for the RM/COBOL (74) version 2 input and the RM/COBOL output relative files.
3. Enter the RM/COBOL Compile Command (**rmcobol**) to compile the program.
4. Under UNIX, set environment variables to assign the file pathnames for the version 2 relative file to INPUT, and the RM/COBOL relative file to OUTPUT. For example:

   ```
   INPUT=/dir1/rel2.0 ; export INPUT

   OUTPUT=/dir2/rel85 ; export OUTPUT
   ```

   Under Windows, use synonyms to assign the file pathnames. Setting synonyms is described in the *Installation and System Considerations for Windows* chapter of the RM/COBOL user's guide.

   Note The source RM/COBOL (74) version 2 pathname and destination RM/COBOL pathname must specify different files, that is, the conversion cannot be made in place.
5. Enter the RM/COBOL Runtime Command (**runcobol**) to execute the program.

Sequential File Conversion Utility (rmcseq)

Variable-length binary sequential files must be converted from RM/COBOL (74) version 2 format to RM/COBOL format. Line sequential and fixed-length binary sequential files do not require conversion. These files may be used directly by RM/COBOL programs. See the appropriate installation and system considerations chapter for your specific operating system to determine which of the sequential files are variable-length binary sequential files.

Using the Utility

1. Execute the **rmcseq.cob** program using the following command:

   ```
   runcobol rmcseq
   ```

2. Respond to the first prompt by entering an **I** to indicate an Intel (reversed byte) format, or an N (not reversed) to indicate all others. Check your hardware operations manual if you are unsure which response is appropriate.
3. Respond to the second prompt by entering the properly qualified pathname of the RM/COBOL (74) version 2 variable-length binary sequential file to be converted.
4. Respond to the third prompt by entering the pathname of the RM/COBOL variable-length binary sequential file to be produced.
5. The program should then execute, displaying a count of the records being converted. If conversion terminates without error, the following message appears:

   ```
   Convert file complete
   ```

   Other messages indicate an error in conversion, most likely caused by an attempt to convert a file that is not a variable-length binary sequential file.

   Note The source RM/COBOL (74) version 2 pathname and destination RM/COBOL pathname must specify different files, that is, the conversion cannot be made in place.