

# **RMSFILE Reference Guide**

Version 4.00 Level U

Copyright 2003 - 2006

Rosebud Management Systems, LLC

## Overview

The RMSFILE program is a batch utility that provides a set of data and file manipulation utilities. RMSFILE is available beginning with Eden Server version 3.1.

The RMSFILE utility supports the following features:

- Data File concatenation, allowing multiple input data files to be concatenated together to produce one output data file. See the CONCAT command for more information.
- Delete and define processing for all format files including ESDS, RRDS and KSDS type files. See the DELETE and DEFINE commands for more information.
- Record copy functions allowing data records from any file type to be copied to any other file type, including the ability to use FROMKEY and TOKEY values when processing KSDS files. See the CPYREC command for more information.
- File copy functions for entire files. See the CPYFILE command for more information.
- File rename functions. See the RENAME command for more information.

## Using RMSFILE

RMSFILE is supplied as Windows executable, RMSFILE.EXE, and is installed in the Eden Server directory. Note that as RMSFILE requires the use of RMSFH and certain Eden Server internal functions, the use of RMSFILE outside the Eden Server environment is not supported.

If the particular RMSFILE command to be run requires any environment variables (such as SET statements to define data files), then the utility must be run from within a batch job. If the command to be run does not require the use of any environment variables then RMSFILE may be invoked directly from the Eden console.

### Entering RMSFILE commands

RMSFILE accepts input commands from SYSIN, which may be set to either a text data file, the JCL data stream a.k.a. SYSRDR, or if SYSIN is not set input will be prompted for and accepted from the system console.

For example, to have RMSFILE read input commands from a control card file, say 'C:\CTLCARD\CONCAT.txt', the following JCL may be used:

```
.  
.   
SET SYSIN=C:\CTLCARD\CONCAT.txt  
RMSFILE.EXE  
IF ERRORLEVEL 1 ECHO RMSFILE error(s) listed above  
.   
.
```

If RMSFILE commands are coded in-line with the JCL statements, the following JCL may be used:

```
.  
.   
SET SYSIN=SYSRDR  
RMSFILE.EXE  
Rmsfile command line  
/*  
IF ERRORLEVEL 1 ECHO RMSFILE error(s) listed above  
.   
.
```

Note in the above example, the "*Rmsfile command line*" text would be replaced with your specific command line(s).

Note that RMSFILE accepts command in either lower or upper case, however upon input all commands are converted to upper case.

### **RMSFILE output**

RMSFILE will always display its messages and accept command input when it is invoked directly from the Eden Server console. When RMSFILE is run from a batch file, however, it will display output messages on any assigned file for SYSOUT. In the absence of an assignment for SYSOUT RMSFILE will display its output on the Eden Server Console.

### **Specifying file names in commands**

You may use environment variables, i.e., symbolic names, in your RMSFILE SYSIN data. Using environment variables as part of your SYSIN input commands allows you to have variable commands that are resolved at execution time.

For example, assume the following JCL statements preceded execution of RMSFILE:

```
SET DRIVE=C:  
SET DIR=MYDATA_DIRECTORY
```

In such a case input commands to RMSFILE that included the variables %DRIVE% or %DIR% would be expanded as though they had been coded to use the values of C: and MYDATA\_DIRECTORY. For example, the input command:

```
DELETE %DRIVE%\%DIR%\datafile.dat
```

Would be expanded by RMSFILE to read:

```
DELETE C:\MYDATA_DIRECTORY\datafile.dat
```

Note also that all RMSFILE commands allow the file name(s) being operated on to be specified either directly or indirectly. That is, the file name may be specified directly as a fully qualified file name, or indirectly as a DDNAME defined previously in the JCL. See the notes on individual commands for complete information on how to specify a file name.

When using RMSFILE, a special form of the SET statement is also allowed. This special form allows file and record descriptions to be specified as part of the SET statement. This form of the SET is useful when using a CPYREC command that includes the use of the FROMDD or TODD parameters.

The syntax of this form of the SET statement is the same as regular SET syntax, except that a file and record type parameter are allowed.

The definition syntax is as follows:

Existing-set-statement, *file-type*(*min*[,*max*])

Where '*file-type*' is one of the following:

RSQ specifies a record sequential file

LSQ specifies a line sequential file

IDX specifies an indexed file

RRN specifies a relative record file

*Min* and *Max* specify the minimum and maximum record lengths for the file. For fixed length files, the *Max* option may be omitted.

For example, to define a variable length record sequential file with record lengths in the range of 20 to 100 bytes, the following SET could be used:

```
SET Ddname=DSNAME,RSQ(20,100)
```

### **Return Code processing**

Upon completion of RMSFILE processing, JCL may query the return code from RMSFILE by examining the .bat file environment variable 'ERRORLEVEL'. ERRORLEVEL will contain one of three possible values:

- 0 The command was processed in its entirety and no errors or warnings occurred.
- 4 The command did not fail completely, but at least one warning message was produced
- 12 The command encountered a serious error and could not be completed.

Note for return values of 4 and 12, specific warning and or error messages will be produced indicating the source of the problem. See the Messages and Codes section of this document for a list of all RMSFILE messages.

RMSFILE also allows for the use of IF – THEN – ELSE constructs including the ability to check the return code of the most recent RMSFILE command and or the highest return code of all previous RMSFILE commands through the use of the special registers 'LASTCC' and 'MAXCC' respectively. See the IF and SET statements for more information.

## Limitations

- RMSFILE in most cases is unable to support the use of GDG data file names, that is, file names where a GDG generation number is specified as part of the file. In some cases however, RMSFILE can still be used to access data within GDG datasets. See the 'Notes' section of specific RMSFILE commands for information on using GDG's with the command.
- RMSFILE allows file names to be specified either directly or indirectly. To directly specify a file name for a command simply code the fully qualified drive, path and file name and extension. To indirectly specify a file name RMSFILE commands allow the name to be specified as an environment variable or name, which has been previously defined in the JCL with a SET command. See the parameters for specific commands for more information on using indirect file names. Note however that the use of indirect file names will resolve the actual disk file name based on the current setting of the environment variable pointed to by the dd-name variable supplied. RMSFILE will not, however, honor any DISP= parameter that may have been coded as part of the original SET command defining the ddname in question. RMSFILE always treats files it operates on as though it's access was exclusive.

## RMSFILE Commands

### CONCAT command

The CONCAT command concatenates multiple input files into one output file. The command format for CONCAT is as follows:

```
CONCAT environment_name_1 environment_name_2 [FORCE] [LINE] [BULK]
```

Where:

**Environment\_name\_1** is a JCL defined name that specifies a string concatenated data files. The concatenate string may include references to standard files, or GDG files, including the 'all version' GDG specification of (\*) as a generation ID.

**Environment\_name\_2** is a JCL defined name that specifies the output file. The output file may be a standard data file, or a GDG file. If a GDG is used for output, only the (+1) generation ID may be used. Note, CONCAT verifies the output file state prior to starting any copying of data. If the output file exists, the command will not be honored unless the option FORCE parameter is present.

**FORCE** is an optional parameter that will cause CONCAT to automatically delete and redefine the output data file if the output file already exists.

**LINE** is an optional parameter that causes CONCAT to verify that each input file in the concatenate string ends with a line-feed + carriage control string. See the notes below on concatenating line sequential files.

**BULK** is an optional parameter that causes CONCAT to perform it's IO using byte stream disk reads and writes. BULK is a very fast way to copy data, however, use of BULK may negatively affect the output file if the input files contain a Micro Focus file header. BULK should be reserved for use with line sequential files or fixed length record sequential files.

### Notes

CONCAT only supports line and record sequential data files. Note: line sequential files may be any file where records are delimited by x'0D0A' characters, including .txt and most .csv files.

Input files must all be of the same type and specification. Use of mixed input file types, such as including both line and record sequential files in the same concatenation string will produce improper results. The output file is created such that it's file type and specification match that of the first non-empty input file. When determining file type

and specification, CONCAT considers the file type, line or record sequential as well as record type, fixed or variable.

When CONCAT is processing line sequential files, or fixed-length record sequential files, data is bulk-copied. That is to say, data is copied from input to output files without regard to any type of record structure. Use of the LINE parameter will ensure that CONCAT inserts a line-feed, carriage return string if an input file does not contain x'0a0d' as the last two characters of the file. LINE, therefore, ensures that the output file maintains end-of-record delimiting between the last and first records of the concatenated input files.

When CONCAT is processing in BULK mode, either as a result of entering the BULK parameter or as a result of CONCAT determining the files did not contain a Micro Focus file header, processing statistics are presented in terms of the number of bytes copied from the input to the output file.

When CONCAT is not processing in BULK mode, either as a result of omitting the BULK parameter or as a result of CONCAT determining the files did contained a Micro Focus file header, processing statistics are presented in terms of the number of records copied from the input to the output file.

See Appendix A in this document for a list of messages produced by CONCAT.

## CPYFILE command

The CPYFILE command provides the ability to copy the complete content of a file to another file. The command syntax for CPYREC is as follows:

```
CPYREC FROM(file-name) | FROMDD(dd-name) TO(file-name) | TODD(dd-name)
[NOOVERWRITE]
```

Where:

**FROM(file-name)** specifies the name of the file being copied from. The 'file-name' value used should be fully qualified including a drive and path and or UNC name. If the file name string contains any embedded spaces, the name should be enclosed within double quotes. Note, FROM() is mutually exclusive with FROMDD().

**FROMDD(dd-name)** specifies the file to be copied from, as does the FROM() parameter, except FROMDD allows the name of the file to be specified as a dd-name which has been previously defined in the JCL via SET command. Note, FROMDD() is mutually exclusive with FROM().

**TO(file-name)** specifies the name of the file being copied to. The 'file-name' value used should be fully qualified including a drive and path and or UNC name. If the file name string contains any embedded spaces, the name should be enclosed within double quotes. Note, TO() is mutually exclusive with TODD().

**TODD(file-name)** specifies the name of the file being copied to, as does the TO() parameter, except TODD allows the name of the file to be specified as a dd-name which has been previously defined in the JCL via SET command. Note, TODD() is mutually exclusive with TO().

**NOOVERWRITE** is an optional parameter which causes the CPYFILE command to not perform the copy and return an error if the TO file already exists.

### Notes

The use of GDG file names with CPYFILE is not allowed. If you need to copy to or from GDG datasets, use the Windows "COPY" command instead of using RMSFILE if you want to either: **a)** reference the file via it's generic name, i.e., C:\GDGname(-1) or **b)** create a new generation of a GDG, i.e., if you use GDG's as part of your backup methodology.

You may still use CPYFILE to access data stored within GDG files by specifying the fully qualified and expanded name of the GDG dataset as input to CPYFILE. Use the Eden Server LISTGDG command to obtain full and complete disk file names for specific generations of GDG datasets. Note that if you are copying to a GDG, it must be a pre-

existing file. CPYFILE does not communicate with the Eden Server File Manager and as such cannot create new generations of GDG's. As such CPYFILE's use of GDG's will not affect the generation structure of the GDG in question; CPYFILE can only alter the contents of, or read data from a pre-existing GDG file.

If you are copying a Micro Focus indexed file, RMSFILE will not automatically copy the .IDX component of the file, you must include a separate CPYFILE statement for each file's associated .IDX component.

If a CPYFILE and all the associated parameters is longer than 80 characters, each line, except the final line of the command should end with a plus sign (+), for example:

```
CPYFILE FROM(c:\test.dat) TODD(outdd) +  
NOOVERWRITE
```

Keep in mind that if you use environment variables as part of your input command, the possibility exists that the expanded names may increase the length of the text beyond 80 bytes. If your variable names could increase text length beyond 80 characters you should code your statements across multiple lines as shown above to ensure the expanded items do not extend beyond column 80.

## CPYREC command

The CPYREC command provides the ability to copy data records from an input file to an output file. The command syntax for CPYREC is as follows:

```
CPYREC FROM(file-name) | FROMDD(dd-name) TO(file-name) | TODD(dd-name)
FROMKEY(key-string) TOKEY(key-string) COUNT(copy-count) SKIP(skip-count)
PAD(pad-string) OPEN(open-mode)
```

Where:

**FROM(file-name)** specifies the name of the file being copied from. The 'file-name' value used should be fully qualified including a drive and path and or UNC name. If the file name string contains any embedded spaces, the name should be enclosed within double quotes. Note that if the file being copied from does not contain a Micro Focus file header, use of the FROMTYPE parameter is required. Note, FROM() is mutually exclusive with FROMDD().

**FROMDD(dd-name)** specifies the file to be copied from, as does the FROM() parameter, except FROMDD allows the name of the file to be specified as a dd-name which has been previously defined in the JCL via SET command. Note, FROMDD() is mutually exclusive with FROM().

**TO(file-name)** specifies the name of the file being copied to. The 'file-name' value used should be fully qualified including a drive and path and or UNC name. If the file name string contains any embedded spaces, the name should be enclosed within double quotes. Note, TO() is mutually exclusive with TODD().

**TODD(dd-name)** specifies the file to be copied to, as does the TO() parameter, except TODD allows the name of the file to be specified as a dd-name which has been previously defined in the JCL via SET command. Note, TODD() is mutually exclusive with TO().

**FROMKEY(key-string)** may be specified when the file being copied from is an indexed file. The key-string value entered may be a text string or a hex string. When entering simple text strings note that the strings are case sensitive and may not contain spaces. If the key-string is to contain non-displayable, special Hexadecimal or 'spaces', the string must be entered as a hex string, i.e., X'1F2B', etc. The key-string entered, whether in hex or text formats, must be no longer than the physical primary key of the input file. Note that 'generic' type record copies are possible by specifying a key string which is shorter than the physical key.

When a FROMKEY parameter is present, CPYREC will perform a START on the input file using a 'GREATER THAN or EQUAL' phrase. The first record returned from this browse will be the first record copied.

Note use of the FROMKEY parameter with a non-indexed input file will cause an error to be generated by CPYREC. FROMKEY may be used with or without the TOKEY parameter.

**TOKEY(key-string)** may be specified when the file being copied from is an indexed file. The key-string value entered may be a text string or a hex string. When entering simple text strings note that the strings are case sensitive and may not contain spaces. If the key-string is to contain non-displayable, special Hexadecimal or 'spaces', the string must be entered as a hex string, i.e., X'1F2B', etc.

When a TOKEY parameter is present, CPYREC will compare the key of all records that are written to the output file. CPYREC will terminate the operation when a record is attempted to be written to the output file and it's key value is greater than the specified TOKEY value.

Note use of the TOKEY parameter with a non-indexed input file will cause an error to be generated by CPYREC. TOKEY may be used with or without the FROMKEY parameter.

**COUNT(copy-count)** specifies the number of records to be copied. When the specified number of records have been copied, the CPYREC process will terminate. Note the use of COUNT does not guarantee the specified number of records will be copied. For example, reaching end-of-file on the input file will terminate the CPYREC prior to the copy-count value being reached.

**SKIP(skip-count)** specifies the number of records to be skipped prior to writing records to the output file. Note that if the FROMKEY and SKIP parameters are both specified, the FROMKEY parameter is first used to establish a starting position within the input file, followed by skip-count' records then being skipped prior to any records being written to the input file.

**PAD(pad-string)** specifies a text or hexadecimal string that is used to 'pad' an output record that is longer than it's corresponding input record. When entering simple text strings note that the strings are case sensitive and may not contain spaces. If the key-string is to contain non-displayable, special Hexadecimal or 'spaces', the string must be entered as a hex string, i.e., X'1F2B', etc. The pad-string value may be up to 64 characters in length. If no PAD parameter is used, the default value for the padding string is null, i.e., x'00' characters.

Note that padding occurs automatically during CPYREC processing, but occurs only when the input record is shorter than the minimum record length of the output file.

**FROMTYPE(file-type,min[,max])** specifies the input file type and record length specifications. FROMTYPE is required parameter only when the input file type is a line sequential, fixed length record sequential or fixed length relative record file. If any other file type is used as the input file, the FROMTYPE parameter is optional, however if

specified CPYREC will always attempt to compare the actual input file to the FROMTYPE specification. In the event there is mismatch a between the file header information and FROMTYPE specification CPYREC will generate an error message. Values for the 'file-type' sub-parameter are RRN for relative record files, IDX for indexed sequential files, LSQ for line sequential files and RSQ for record sequential files. The 'min' parameter specifies the minimum record length and may be any valid length. If the file is variable length, the ',max' parameter should also be specified. Note that it is possible to specify FROMTYPE information via the JCL SET statement if the FROMDD option is being used. In this case, the special form of the SET statement is allowed; See the previous section 'Specifying file names in commands' for complete information on how to specify FROMTYPE information on the SET.

**TOTYPE(file-type,min[,max])** specifies the output file type and record length specifications. TOTYPE is required parameter only when the output file type is a line sequential, fixed length record sequential or fixed length relative record file. If any other file type is used as the input file, the TOTYPE parameter is optional if the output file is already defined. If specified however, CPYREC will always attempt to compare the actual output file to the TOTYPE specification, if the output file exists. In the event there is mismatch a between the file header information and TOTYPE specification CPYREC will generate an error message. Values for the 'file-type' sub-parameter are RRN for relative record files, IDX for indexed sequential files, LSQ for line sequential files and RSQ for record sequential files. The 'min' parameter specifies the minimum record length and may be any valid length. If the file is variable length, the ',max' parameter should also be specified. Note that when the output file does not exist (and it is not an IDX type file), use of the TOTYPE parameter allows CPYREC to be able to dynamically create the file as part of the CPYREC processing. Note that it is possible to specify TOTYPE information via the JCL SET statement if the TODD option is being used. In this case, the special form of the SET statement is allowed; See the previous section 'Specifying file names in commands' for complete information on how to specify TOTYPE information on the SET.

**OPEN(mode-value)** sets the open mode that CPYREC will open the output file in. The allowed values for 'mode-value' are OUTPUT, IO and EXTEND. These modes correspond to typical open modes for COBOL files. OPEN is an optional parameter, the default for which if not specified is OUTPUT. Note that when using OUTPUT, CPYREC will erase all existing records from the file if the file already exists. In the event IO or EXTEND is specified, the output file must already exist or an error message will be generated.

## Notes

Use of GDG file names with CPYREC is not allowed. If you must copy certain records from a GDG dataset, you must explicitly define the fully qualified and expanded disk file

name of the GDG file. Use the LISTGDG command to obtain the full disk file names of specific generations of a GDG dataset.

You may still use CPYREC to access data stored within GDG files by specifying the fully qualified and expanded name of the GDG dataset as input to CPYREC. Use the Eden Server LISTGDG command to obtain full and complete disk file names for specific generations of GDG datasets. Note that if you are copying to a GDG, it must be a pre-existing file. CPYREC does not communicate with the Eden Server File Manager and as such cannot create new generations of GDG's. As such CPYREC's use of GDG's will not affect the generation structure of the GDG in question; CPYREC can only alter the contents of, or read data from a pre-existing GDG file.

If a CPYREC and all the associated parameters is longer than 80 characters, each line, except the final line of the command should end with a plus sign (+), for example:

```
CPYREC FROM(c:\test.dat) TO(c:\copied_test.dat) FROMTYPE(RSQ,80,120) +  
TOTYPE(RSQ,120) PAD(x'20')
```

Keep in mind that if you use environment variables as part of your input command, the possibility exists that the expanded names may increase the length of the text beyond 80 bytes. If your variable names could increase text length beyond 80 characters you should code your statements across multiple lines as shown above to ensure the expanded items do not extend beyond column 80.

## DEFINE command

The DEFINE command provides the ability to define new disk files, of any type. The command syntax for the DEFINE command is as follows:

```
DEFINE  IDX | RRN | LSQ | RSQ  NAME(file-name)  [MODEL(file-name)]  
PRIKEY(offset,len)  ALTKEY(offset,len[,DUP|NODUP])  RECLen(min[,max])  
[IDXSIZE(size)]
```

Where:

**IDX**, or **RRN**, or **LSQ** or **RSQ** must be the first parameter following the DEFINE command. The string used indicates which type of file is to be defined, as follows: **IDX** indicates an indexed sequential file. Note that use of **IDX** requires that the **PRIKEY** (and optionally the **ALTKEY**) parameter also be defined. **RRN** indicates that a relative record file is to be defined. **LSQ** indicates that a Line sequential file is to be defined. **RSQ** indicates that a Record sequential file is to be defined.

**NAME(file-name)** specifies the fully qualified path, name and extension of the file to be created. Note that the file name may be specified either with a drive and path or a UNC name. If any portion of the name contains an embedded space, the entire name should be enclosed in double quote characters, for example: **NAME("C:\program files\datafile.dat")**.

**MODEL(file-name)** specifies the fully qualified path, name and extension of a file to be used as a model for retrieving all other file definition parameters from. Note that use of a **MODEL** parameter is mutually exclusive with **PRIKEY**, **ALTKEY** and **RECLen** parameters. Note that the file name may be specified either with a drive and path or a UNC name. If any portion of the name contains an embedded space, the entire name should be enclosed in double quote characters, for example: **NAME("C:\program files\datafile.dat")**. The file specified in the **MODEL** parameter must contain a Micro Focus file header. The types of files that may be used as a **MODEL**, based on the type of file being defined are as follows:

| File type being defined | Type of file used as MODEL                    |
|-------------------------|---|
| IDX                     | IDX   |
| RSQ                     | RSQ (only if file is variable length records) |
| LSQ                     | RSQ (only if file is variable length records) |
| RRN                     | RRN (only if file is variable length records) |

**PRIKEY(offset,len)** specifies the key offset (relative to zero) and length for the primary key of an IDX type file. Note that PRIKEY() is allowed only with IDX type files. The 'offset' value should specify the key's position relative to the first byte of the record. I.e., the first position of the record is offset zero. The 'len' value specifies the key length in bytes. For example a key starting in the first position of the record that is twelve bytes long would have a specification of (0,12). Note, only one PRIKEY() parameter is allowed per DEFINE.

**ALTKEY(offset,len[,DUP | NODUP])** specifies the key offset (relative to zero) and length for an alternate key of an IDX type file. Note that ALTKEY() is allowed only with IDX type files. The 'offset' value should specify the key's position relative to the first byte of the record. I.e., the first position of the record is offset zero. The 'len' value specifies the key length in bytes. For example a key starting in the first position of the record that is twelve bytes long would have a specification of (0,12). The DUP and NODUP sub-parameters, which are mutually exclusive, indicate if the alternate key is to allow duplicates. To allow the key to contain duplicates, include the ',DUP' value. NODUP is the default value and will be assumed in no value is coded following the length. Note that up to five ALTKEY() parameters may be coded for a file.

**RECLEN(min[,max])** specifies the record lengths and type of record (i.e., fixed or variable). A 'min' value, defining the minimum record length is always required. A maximum record length is only required if the record format is to be of variable length. The record format for the file is assumed to be fixed if no 'max' value is coded or if the indicated min and max values are the same. Note the maximum record length allowed is 32768.

**IDXSIZE(size)** specifies the index size for an indexed file. This parameter provides the ability to large and very-large indexed files. The values available for use here are the same as those provided for the IDXFORMAT() compiler directive. For complete information on the possible values for this field, and their affect on the file being defined, please refer to the Micro Focus documentation on IDXFORMAT.

## Notes

If a DEFINE and all the associated parameters is longer than 80 characters, each line, except the final line of the command should end with a plus sign (+), for example:

```
DEFINE IDX NAME(c:\test.dat) PRIKEY(0,12) ALTKEY(44,64,DUP) +  
ALTKEY(200,12) RECLEN(4096)
```

Keep in mind that if you use environment variables as part of your input command, the possibility exists that the expanded names may increase the length of the text beyond 80 bytes. If your variable names could increase text length beyond 80 characters you should code your statements across multiple lines as shown above to ensure the expanded items do not extend beyond column 80.



## DELETE command

The DELETE command provides the ability to delete disk files, of any type. The command syntax for the DELETE command is as follows:

```
DELETE file-name | DD(dd-name)
```

Where:

**File-name** is the fully qualified name of the disk file to be deleted. Note the name may be specified with either a drive letter and path, or as a UNC name. If any portion of the name contains an embedded space, the entire name should be enclosed in double quote characters, for example: "C:\program files\datafile.dat". Note that DELETE will delete the specified file name, and if the operation was successful, DELETE will automatically attempt to delete any existing index component of the file. When performing this process, DELETE will either replace the input file name's extension or append an extension of '.IDX' and then attempt to delete any such existing file. Note that no error message will be produced in the event there is not .IDX component deleted, however a confirmation message will be displayed if such a disk entry is deleted. Note use of a file-name is mutually exclusive with the DD() parameter.

**DD(dd-name)** refers to a JCL SET statement which defines the file name. Use of the DD() parameter is functionally equivalent to supplying the file name directly, however it simply allows the name to be defined via the JCL instead of directly via the file-name parameter. Note use of the DD() parameter is mutually exclusive with a file-name.

## IF THEN ELSE statements

RMSFILE supports the use of conditional logic within the input command stream, though the use of the IF, THEN and ELSE statements. The syntax for these statements is as follows:

```
IF (conditional expression)
THEN (RMSFILE command)
ELSE (RMSFILE command)
```

Where:

Conditional expression is a comparison of either MAXCC or LASTCC to a numeric value. Note that the entire comparison string must appear within the confines of an opening and a closing parentheses, and the entire IF must start and end on the same line. That is, IF statements may not be continued across lines by using the standard RMSFILE command continuation method of placing a '+' as the last character on the line.

The comparison may use the following operators:

```
EQ for an 'equal to' comparison
NE for an 'not equal to' comparison
GT for a 'greater than' comparison
GE for a 'greater than or equal to' comparison'
LT for a 'less than' comparison
LE for a 'less than or equal to' comparison'
```

For example to conditionally execute a THEN() statement and it's associated command(s) only when the maximum condition code returned from all previous commands is equal to zero, the following IF might be coded:

```
IF (MAXCC EQ 0)
THEN(RMSFILE command)
ELSE()
```

To conditionally execute an ELSE statement and it's associated command(s) only if the condition code of the most recently completed RMSFILE command was greater than 4, the following IF might be used

```
IF (LASTCC LT 5)
THEN()
ELSE(RMSFILE command)
```

Note that in both the above examples one of the THEN or ELSE statements included no RMSFILE command. This is legal syntax for these statements. Further, note that the

ELSE statement, and specifically its closing parentheses, is interpreted by RMSFILE as and explicit 'END IF'. As such, when coding IF statements that include logical processing only as part of the THEN statement, it is still required that an ELSE be coded. In such a case the ELSE must be coded as it is above in the first example, that is where the ELSE() contains no RMSFILE command.

Within the confines of the THEN or ELSE statements it is possible to include more than one RMSFILE command. In such a case, simply include the additional commands coded on a new line following the previous command as shown in the following example;

```
IF (maxcc ne 0)
THEN (DELETE c:\test.dat
Define RSQ name(c:\x.x) reclen(50,100))
ELSE()
```

## RENAME command

The RENAME command provides the ability to change the name of a disk file. The command syntax for RENAME is as follows:

```
RENAME FROM(file-name) | FROMDD(dd-name) TO(file-name) | TODD(dd-name)
```

Where:

**FROM(file-name)** specifies the current name of the file. The 'file-name' value used should be fully qualified including a drive and path and or UNC name. If the file name string contains any embedded spaces, the name should be enclosed within double quotes. Note that if the file being renamed is an indexed file, the .IDX component will automatically be renamed as well. Note, FROM() is mutually exclusive with FROMDD().

**FROMDD(dd-name)** specifies the current name of the file, as does the FROM() parameter, except FROMDD allows the name of the file to be specified as a dd-name which has been previously defined in the JCL via SET command. Note, FROMDD() is mutually exclusive with FROM().

**TO(file-name)** specifies the new name of the file. The 'file-name' value used should be fully qualified including a drive and path and or UNC name. If the file name string contains any embedded spaces, the name should be enclosed within double quotes. Note, TO() is mutually exclusive with TODD().

**TODD(file-name)** specifies the new name of the file, as does the TO() parameter, except TODD allows the name of the file to be specified as a dd-name which has been previously defined in the JCL via SET command. Note, TODD() is mutually exclusive with TO().

### Notes

The use of GDG file names with RENAME is not allowed.

If a RENAME and all the associated parameters is longer than 80 characters, each line, except the final line of the command should end with a plus sign (+), for example:

```
RENAME FROM(C:\MY_DIR_MY_SUBDIR\test.dat) +  
TODD(outdd)
```

Keep in mind that if you use environment variables as part of your input command, the possibility exists that the expanded names may increase the length of the text beyond 80 bytes. If your variable names could increase text length beyond 80 characters you

should code your statements across multiple lines as shown above to ensure the expanded items do not extend beyond column 80.

## SET statement

The SET statement provides the ability to set the values of the MAXCC and LASTCC special registers used and maintained by RMSFILE. The syntax of the SET statement is as follows:

```
SET MAXCC|LASTCC=numeric-values
```

Where:

**MAXCC** specifies supplied numeric value should be assigned to the MAXCC register. Note that setting MAXCC will affect the overall final return code of RMSFILE itself, and therefore may affect the execution of others steps with in jobs that include RMSFILE. Use SET MAXCC= with caution.

**LASTCC**. specifies supplied numeric value should be assigned to the LASTCC register. Note that setting LASTCC will affect only the condition code of the most recently completed RMSFILE command. Any value assigned by a SET LASTCC= statement will be overwritten by the completion of the next RMSFILE command (i.e., CONCAT, CPYFILE, CPYREC, DEFINE, DELETE or RENAME).

**Numeric-value** may be any position value between 0 and 999,999,999.

For example, to set the LASTCC register to a value of 4, the following statement might be used:

```
SET LASTCC=4
```

Note that use of the SET command on the LASTCC value may have an affect on the MAXCC value. In cases where the value of LASTCC is set to a value higher than the current value of MAXCC, MAXCC's value will be set to that of LASTCC.

## **Appendix A RMSFILE Messages and codes**

RMSFILE messages are arranged by command, where the message text starts with the command name and is followed by a message number and severity level. Message codes are therefore in the form:

Command-message#-severity

Where severity is 'I' for informational, 'W' for a warning and 'E' for an unrecoverable error.

Message codes and descriptions are described on the following pages.

## **RMSFILE general messages**

### **RMSFILE-001-I : Enter RMSFILE commands, /\* to exit.**

Description RMSFILE has been started from the Eden console. As such, commands may only be entered from the console. Enter /\* when done.

Level Informational

Action Enter RMSFILE commands and parameters, or enter /\* to exist RMSFILE.

### **RMSFILE-002-E : Invalid RMSFILE command**

Description An invalid RMSFILE command has been entered.

Level Error

Action Enter a valid RMSFILE command.

### **RMSFILE-003-I : RMSFILE complete**

Description RMSFILE has encountered an end-of-file marker on SYSIN and has terminated.

Level Informational

Action None, restart RMSFILE if more commands need to be run.

### **RMSFILE-004-W : Remaining command input being flushed**

Description The previous command input to RMSFILE has encountered an unrecoverable error. As such, RMSFILE is flushing the remaining command input due to the input being on multiple input cards.

Level Warning.

Action Correct the errors in the previously input command.

### **RMSFILE-005-E : Incomplete IF command**

Description An IF command was encountered, however it did not conform to the RMSFILE syntax rules for IF statements.

Level Error, return code is set to 12

Action Enter a valid RMSFILE IF command. Refer to the IF command notes in the document for syntax help.

### **RMSFILE-006-E : First IF variable is invalid**

Description An IF command was encountered, however the first variable in the comparison was invalid.

Level Error, return code is set to 12

Action Ensure that the variables used in comparisons are either a numeric value, MAXCC or LASTCC only.

### **RMSFILE-007-E : Second IF variable is invalid**

Description An IF command was encountered, however the second variable in the comparison was invalid.

Level Error, return code is set to 12

Action Ensure that the variables used in comparisons are either a numeric value, MAXCC or LASTCC only.

### **RMSFILE-008-C : Expected THEN not found – RMSFILE aborting now**

Description During processing of an IF command, RMSFILE expected to find a THEN phrase at the current position in the input command stream, however no THEN was found.

Level Catastrophic. RMSFILE terminates following this message with an overall return code of 32

Action Ensure that all IF statements follow the required RMSFILE syntax rules; re-code your input command stream and try again.

### **RMSFILE-009-I : Comparison evaluated TRUE**

Description RMSFILE has determined the current IF statement comparison is TRUE.

Level Informational

Action RMSFILE will process the command(s) associated with the following THEN statement.

### **RMSFILE-010-I : Comparison evaluated FALSE**

Description RMSFILE has determined the current IF statement comparison is FALSE.

Level Informational

Action RMSFILE will process the command(s) associated with the following ELSE statement.

### **RMSFILE-011-C :Malformed THEN – RMSFILE aborting now**

Description During processing of an IF command, RMSFILE expected to find at least the beginning portions of a THEN phrase at the current position in the input command stream, however only the THEN phrase, and now the opening parentheses were found.

Level Catastrophic. RMSFILE terminates following this message with an overall return code of 32

Action Ensure that all IF statements follow the required RMSFILE syntax rules; re-code your input command stream and try again.

### **RMSFILE-012-C :Expected THEN not found – RMSFILE aborting now**

Description During processing of an IF command, RMSFILE expected to find a THEN phrase at the current position in the input command stream, however the phrase was not found.

Level Catastrophic. RMSFILE terminates following this message with an overall return code of 32.

Action Ensure that all IF statements follow the required RMSFILE syntax rules; re-code your input command stream and try again.

### **RMSFILE-013-C :Malformed ELSE – RMSFILE aborting now**

**Description** During processing of an IF command, RMSFILE expected to find at least the beginning portions of a THEN phrase at the current position in the input command stream, however only the THEN phrase, and now the opening parentheses were found.

**Level** Catastrophic. RMSFILE terminates following this message with an overall return code of 32.

**Action** Ensure that all IF statements follow the required RMSFILE syntax rules; re-code your input command stream and try again.

## CONCAT messages

### CONCAT-001-I : CONCAT complete. Maximum return code *n*

Description The previously entered CONCAT command has completed. The maximum return code is indicated as '*n*'.

Level Informational

Action None required. If the return code is non-zero, however, the CONCAT did not complete, or a warning message was issued. See the messages issued prior to this message, which indicate the specific situations encountered during CONCAT processing.

### CONCAT-002-I : *environment-name* not set to concatenate string

Description The value assigned to the environment variable for input files is not a concatenation string.

Level Error, return code 12 is set.

Action Change the JCL so that the environment variable corresponding to the input file(s) is a concatenation string. Concatenate strings accepted by the CONCAT command include standard SET dd=dsn +dd=dsn type commands, or a single SET for an 'all-generation-GDG', i.e., SET dd=dsn(\*).

### CONCAT-003-E : No input files processed, all files empty, output file not created

Description CONCAT has finished processing all the input files, however all the files were empty, which forced CONCAT to bypass producing the output file.

Level Error, return code 12 is set.

Action CONCAT can only process input files that are not empty. Resubmit the command after ensuring at least 1 of the input files has at least 1 record.

**CONCAT-004-E : Open error *file-status* on input file *file-name***

Description The indicated input file encountered an open error.

Level Error, return code 12 is set.

Action Research and correct the file open error and resubmit the command.

**CONCAT-005-E : Input file type not supported. Cannot process file *file-name***

Description The indicated file is not a sequential file.

Level Error, return code 12 is set.

Action Ensure that the input concatenation list includes only entries for line sequential or record sequential files. Note that use of the BULK parameter will suppress this error and force the files to be copied regardless of type.

**CONCAT-006-E : Output file may not specify GDG generation (\*)**

Description CONCAT will not accept an 'all-generation-GDG' specification as the output file.

Level Error, return code 12 is set.

Action Change the output file specification and resubmit the command.

**CONCAT-007-E : Output GDG specification error**

Description CONCAT cannot open the requested output file because it is a GDG and the generation value is not set to (+1).

Level Error, return code 12 is set.

Action Ensure that the output file, if it is a GDG, uses generation (+1) and then resubmit the command.

**CONCAT-008-I : Existing output file *file-name* will be over-written, FORCE option in effect.**

Description The FORCE option was entered on the command line and as such the existing output file will be over-written.

Level Informational.

Action None.

**CONCAT-009-E : Existing output file *file-name* cannot be over-written, file may be in use**

Description The FORCE option was entered on the command line, however, the file could not be opened in exclusive mode.

Level Error, return code 12 is set.

Action Determine why the file could not be updated, and then resubmit the command.

**CONCAT-010-E : Output file *file-name* already exists, and FORCE not specified.**

Description The FORCE option was not entered on the command line however the output file already exists.

Level Error, return code 12 is set.

Action Either ensure the output file does not exist when the command is submitted, or use the FORCE parameter to override the error.

**CONCAT-011-I : New output file *file-name* will be created**

Description CONCAT will create the output file, which does not currently exist.

Level Informational.

Action None.

**CONCAT-012-E : Open error *file-status* on output file *file-name***

Description The indicated output file could not be opened.

Level Error, return code 12 is set

Action Research and correct the open error, then resubmit the command.

**CONCAT-013-E : Open error *file-status* on input file *file-name***

Description The indicated input file could not be opened.

Level Error, return code 12 is set

Action Research and correct the open error, then resubmit the command.

**CONCAT-014-E : IO error on output file *file-name*, file status *file-status***

Description While copying data in BULK mode, the indicated output file could not be written to.

Level Error, return code 12 is set

Action Research and correct the IO error, then resubmit the command.

**CONCAT-015-I : Copy complete for *file-name*, *n* bytes copied.**

Description CONCAT has completed BULK mode processing on the indicated file. A total of '*n*' bytes were copied from the input file to the output file.

Level Informational.

Action None, processing continues.

**CONCAT-016-E : IO error on input file *file-name*, file status *file-status***

Description While copying data in BULK mode, the indicated input file could not be read from.

Level Error, return code 12 is set

Action Research and correct the IO error, then resubmit the command.

**CONCAT-017-E : Open error on output file *file-name*, file status *file-status***

Description While copying data in 'record', i.e., non-BULK mode, the indicated output file could not be opened exclusively.

Level Error, return code 12 is set

Action Research and correct the open error, then resubmit the command.

**CONCAT-018-E : IO error on output file *file-name*, file status *file-status***

Description While copying data in 'record', i.e., non-BULK mode, the indicated output file could not be written to.

Level Error, return code 12 is set

Action Research and correct the IO error, then resubmit the command.

**CONCAT-019-I : Copy complete for *file-name*, *n* records copied.**

Description CONCAT has completed 'record', i.e., non-BULK mode processing on the indicated file. A total of '*n*' records were copied from the input file to the output file.

Level Informational.

Action None, processing continues.

**CONCAT-020-E : IO error on input file *file-name*, file status *file-status***

Description While copying data in 'record', i.e., non-BULK mode, the indicated input file could not be read from.

Level Error, return code 12 is set

Action Research and correct the IO error, then resubmit the command.

**CONCAT-021-W : Input file *file-name*, is empty. File is being ignored**

Description CONCAT has encountered an input file that is zero bytes in length.

Level Warning, return code 4 is set

Action None, CONCAT will continue processing with the next input file. Note, if all input files are empty CONCAT will terminate with a return code of 12 and the output file will not be created.

**CONCAT-022-E : Open error *file-status* on input file *file-name***

Description While copying data in record, i.e., non-BULK mode, the indicated input file could not be opened.

Level Error, return code 12 is set

Action Research and correct the Open error, then resubmit the command.

**CONCAT-023-E : Close error on input file *file-name*, file status *file-status***

Description While copying data in either record or BULK mode, the indicated input file could not be closed properly.

Level Error, return code 12 is set

Action Research and correct the close error, then resubmit the command.

**CONCAT-024-E : Close error on output file *file-name*, file status *file-status***

Description While copying data in either record or BULK mode, the indicated output file could not be closed properly.

Level Error, return code 12 is set

Action Research and correct the close error, then resubmit the command.

**CONCAT-025-E : Input file *file-name*, does not exist**

Description While copying data in either record or BULK mode, the indicated input file could not be located.

Level Error, return code 12 is set

Action Research why the file does not exist, or correct the JCL to reference only existing files, then resubmit the command.

## CPYFILE messages

### CPYFILE-001-I : CPYFILE complete. Maximum return code *n*

Description The previously entered CPYFILE command has completed. The maximum return code is indicated as '*n*'.

Level Informational

Action None required. If the return code is non-zero, however, the CPYFILE did not complete, or a warning message was issued. See the messages issued prior to this message, which indicate the specific situations encountered during CPYFILE processing.

### CPYFILE-002-E : Input FROMDD() does not exist

Description The file defined via the FROMDD() statement does not exist, CPYFILE cannot proceed.

Level Error, return code 12 is set.

Action Correct the JCL so that the file to be copied either exists, or determine the reason why the from file cannot be accessed.

### CPYFILE-003-E : Input FROM() does not exist

Description The file defined via the FROM() statement does not exist, CPYFILE cannot proceed.

Level Error, return code 12 is set.

Action Correct the JCL so that the file to be copied either exists, or determine the reason why the from file cannot be accessed.

**CPYFILE-004-E : TO() file exists and NOOVERWRITE was specified CPYFILE not attempted**

Description The file to be copied to already exists, however the NOOVERWRITE statement was included on the CPYFILE command.

Level Error, return code 12 is set.

Action Either remove the NOOVERWRITE statement, delete the file prior to the CPYFILE command, or determine why the file was present when its existence was not expected.

**CPYFILE-005-I : CPYFILE successful**

Description The CPYFILE command completed successfully.

Level Informational.

Action None required.

**CPYREC-006-E : CPYFILE failed, file status *file-status***

Description CPYREC was unable to complete the requested copy operation. The final file status (which is a COBOL file status value) indicates the cause of the problem.

Level Error, return code 12 is set.

Action Research the file status and determine the cause of the failure.

## CPYREC messages

### CPYREC-001-I : CPYREC complete. Maximum return code *n*

Description The previously entered CPYREC command has completed. The maximum return code is indicated as '*n*'.

Level Informational

Action None required. If the return code is non-zero, however, the CPYREC did not complete, or a warning message was issued. See the messages issued prior to this message, which indicate the specific situations encountered during CPYREC processing.

### CPYREC-002-E : Invalid or unrecognized OPEN() parameter

Description The previously entered CPYREC command includes an OPEN parameter that is either malformed, or contains an invalid open mode value.

Level Error, return code set to 12.

Action Correct the input command stream so that the OPEN() parameter specifies one of OUTPUT, IO or EXTEND only, then re-run the command.

### CPYREC-003-E : Input FROM() does not exist

Description The previously entered CPYREC command includes a FROM() file reference as the input file, however the specified file does not exist.

Level Error, return code 12 is set.

Action Verify the file name was properly specified and or determine why the specified file cannot be accessed, then re-run the command.

**CPYREC-004-E : File type mismatch, FROMTYPE(*from-type*) specified, file is *to-type***

Description The previously entered CPYREC specified an FROMTYPE() parameter identifying the input file as shown by '*from-type*', however, CPYREC has determined that the input file is a different file type as indicated by '*to-type*', where either type value may be RSQ, IDX, RRN or LSQ.

Level Error, return code 12 is set.

Action Determine if the FROMTYPE() parameter is correct, or if possible the FROM() parameter is referencing the wrong file name. Re-run the command after correcting whichever situation is discovered.

**CPYREC-005-E : File type mismatch. Expected to find file header based upon input FROMTYPE(*from-type*) parameter**

Description The previously entered CPYREC specified a FROMTYPE() parameter indicating a specific file type for which CPYREC expected to find a Micro Focus file header record at the beginning of the file in question. Upon opening the file, however, CPYREC did not find a file header record.

Level Error, return code 12 is set.

Action Determine if the proper file is being accessed through the FROM() parameter. If the file name is correct, ensure that the FROMTYPE() setting is correct. Note that a file header is expected for the following file types: IDX – fixed and variable length record files, RSQ – variable length record files only and RRN – variable length record files only. All other file types are assumed to have no file header.

### **CPYREC-006-E : Unknown input file type. FROMTYPE() required.**

Description The previously entered CPYREC specified an input file that did not have a file header, and no FROMTYPE() parameter was specified in the CPYREC command. Note that when accessing input files that do not have a Micro Focus file header, the FROMTYPE() parameter is required. Files without headers include: all Line Sequential (LSQ) files as well as fixed length record Relative Record (RRN) and Record Sequential (RSQ) files. All other files are expected to have file header records.

Level Error return code 12 is set.

Action Include a FROMTYPE() parameter on the CPYREC command that accurately describes the input file, then re-run the command.

### **CPYREC-007-E : Open error on FROM(), return code *file-status***

Description The previously entered CPYREC command was unable to open the file specified in the FROM() parameter. The return code shown is either a standard COBOL file status code, or a Micro Focus extended file status code.

Level Error return code 12 is set.

Action Determine the reason for the open error. Consult the Eden Server Messages and Codes document, using either Appendix A for 'standard' file status values, or Appendix B for the Micro Focus extended codes.

### **CPYREC-008-E : Record length discrepancy between FROMTYPE (*type,min,max*) and actual record of *actual-min,actual-max***

Description The previously entered CPYREC command included a specific FROMTYPE() parameter that does not match the actual record lengths of the input file. Note that both the FROMTYPE and actual length(s) are shown.

Level Error, return code 12 is set.

Action Determine if the proper file is being accessed through the FROM() parameter. If the proper file is being accessed, adjust the FROMTYPE() values to match the actual file.

**CPYREC-009-W : FROMKEY() and SKIP() combination may produce undesired Results**

Description The previously entered CPYREC command includes both the FROMKEY() and SKIP() parameters. Typically these parameters are not used in combination with one another.

Level Warning, return code set to 4.

Action No action required, if the specification of both parameters was intentional. Note this message does not cause any processing to be changed or altered, CPYREC will first position to the desired record based upon the FROMKEY() parameter, then the requested number of records (from the SKIP() parameter) will be read and discarded prior to starting record copying.

**CPYREC-010-W : TOKEY() and COUNT() combination may produce undesired results.**

Description The previously entered CPYREC command specified both a TOKEY and a COUNT() parameter. Typically these parameters are not used in combination with one another.

Level Warning, return code set to 4.

Action No action required, if the specification of both parameters was intentional. Note this message does not cause any processing to be changed or altered, CPYREC will copy records until either a record is read that would exceed the TOKEY() specification, or the number of records specified in the COUNT() parameter have been copied. Note that this implies that it is possible CPYREC will stop copying records prior to reaching the limit imposed by the TOKEY() parameter.

**CPYREC-011-E : FROMKEY/TOKEY not allowed with *from-type* files**

Description The previously entered CPYREC command specified a FROMKEY() and or TOKEY() parameter(s), however these parameters are not allowed with the type of file specified in the FROMTYPE() parameter.

Level Warning, return code set to 12.

Action Determine if the FROMTYPE() parameter incorrectly specified a file type other than IDX, or if the FROMKEY/TOKEY parameters were incorrectly included in the CPYREC command.

**CPYREC-012-W : Existing TOFILE() will be reset prior to CPYREC due to OPEN(OUTPUT)**

Description The previously entered CPYREC command specified an open mode of OUTPUT for the output TOFILE, and that file existed prior to CPYREC starting its processing.

Level Warning, return code set to 4.

Action No action required, CPYREC will proceed. Note, to suppress this warning message, include a DELETE command for the output file prior to the CPYREC command.

**CPYREC-013-I : Existing TOFILE will be updated according to OPEN()**

Description The previously entered CPYREC command specified either an OPEN(IO) or OPEN(EXTEND) parameter. CPYREC will update the output file according to the specific open mode selected.

Level Informational.

Action None required. Note, an OPEN(IO) parameter will cause CPYREC to update any records (on IDX type output files) where there addition of the record being copied would cause a duplicate key situation. An OPEN(EXTEND) parameter, under the same situation, would cause an error. Be sure to code your OPEN() parameter accordingly.

**CPYREC-014-W : TOKEY() and COUNT() combination may produce undesired results.**

Description The previously entered CPYREC command specified both a TOKEY and a COUNT() parameter. Typically these parameters are not used in combination with one another.

Level Warning, return code set to 4.

Action No action required, if the specification of both parameters was intentional. Note this message does not cause any processing to be changed or altered, CPYREC will copy records until either a record is read that would exceed the TOKEY() specification, or the number of records specified in the COUNT() parameter have been copied. Note that this implies that it is possible CPYREC will stop copying records prior to reaching the limit imposed by the TOKEY() parameter.

**CPYREC-015-E : Specified OPEN() requires TOFILE() to already exist, file not found**

Description The previously entered CPYREC command specified an OPEN() parameter that included either IO or EXTEND, both of which require that the file being copied to already exist, but the file being copied to does not exist or cannot be accessed.

Level Error, return code set to 12.

Action Verify that the proper file is specified in the TOFILE() parameter, and or change the OPEN() parameter to OPEN(OUTPUT), then re-run the command.

**CPYREC-016-E : File type mismatch, TOTYPE(*to-type*) specified, file is *actual-type***

Description The previously entered CPYREC command specified a TOTYPE() parameter that does not match the actual file's type.

Level Error, return code set to 12.

Action Verify the file name specified in the TOFILE() parameter is the proper file and that it matches the TOTYPE() parameter, then re-run the command.

**CPYREC-017-E : File type mismatch. Expected to find file header based upon input TOTYPE(*from-type*) parameter**

Description The previously entered CPYREC specified a TOTYPE() parameter indicating a specific file type for which CPYREC expected to find a Micro Focus file header record at the beginning of the file in question. Upon opening the file, however, CPYREC did not find a file header record.

Level Error, return code 12 is set.

Action Determine if the proper file is being accessed through the TO() parameter. If the file name is correct, ensure that the TOTYPE() setting is correct. Note that a file header is expected for the following file types: IDX – fixed and variable length record files, RSQ – variable length record files only and RRN – variable length record files only. All other file types are assumed to have no file header.

### **CPYREC-018-E : Unknown input file type. TOTYPE() required.**

Description The previously entered CPYREC specified an input file that did not have a file header, and no TOTYPE() parameter was specified in the CPYREC command. Note that when accessing input files that do not have a Micro Focus file header, the TOTYPE() parameter is required. Files without headers include: all Line Sequential (LSQ) files as well as fixed length record Relative Record (RRN) and Record Sequential (RSQ) files. All other files are expected to have file header records.

Level Error return code 12 is set.

Action Include a TOTYPE() parameter on the CPYREC command that accurately describes the input file, then re-run the command.

### **CPYREC-019-E : Open error in TO(), return code *file-status***

Description The previously entered CPYREC command was unable to open the file specified in the TO() parameter. The return code shown is either a standard COBOL file status code, or a Micro Focus extended file status code.

Level Informational

Action Determine the reason for the open error. Consult the Eden Server Messages and Codes document, using either Appendix A for 'standard' file status values, or Appendix B for the Micro Focus extended codes.

### **CPYREC-020-E : Record length discrepancy between TOTYPE (*type,min,max*) and actual record of *actual-min,actual-max***

Description The previously entered CPYREC command included a specific TOTYPE() parameter that does not match the actual record lengths of the input file. Note that both the TOTYPE and actual length(s) are shown.

Level Error, return code 12 is set.

Action Determine if the proper file is being accessed through the TO() parameter. If the proper file is being accessed, adjust the TOTYPE() values to match the actual file.

### **CPYREC-021-E : Cannot auto-define IDX type TO() when FROM() not IDX type**

Description The previously entered CPYREC command included a specified a TOTYPE(IDX) parameter and a FROM() file that was not an IDX. The TO() file doesn't exist on disk and as such CPYREC is unable to auto-define the TO() file.

Level Error, return code 12 is set.

Action Use the DEFINE command to create the TO() file prior to using CPYREC, or allow CPYREC to create the file by specifying a FROMTYPE() for a file type other than IDX. Note, when copying to a non-existent file, CPYREC can only auto-define an IDX type file if the FROM() file was also an IDX. In such a case the TO() file will be created with the same attributes as the FROM() file.

### **CPYREC-022-W : Length of FROMKEY value greater than physical key, value truncated**

Description The previously entered CPYREC command included a FROMKEY() parameter, however the length of the supplied key in this parameter is longer than the actual physical key on the FROM() file. CPYREC has truncated the value in the FROMKEY() parameter to match the length of the actual key.

Level Warning, return code 4 is set.

Action Verify that FROMKEY values do not specify a key value that is longer than the physical key of the file.

### **CPYREC-023-W : Length of TOKEY value greater than physical key, value truncated**

Description The previously entered CPYREC command included a TOKEY() parameter, however the length of the supplied key in this parameter is longer than the actual physical key on the TO() file. CPYREC has truncated the value in the TOKEY() parameter to match the length of the actual key.

Level Warning, return code 4 is set.

Action Verify that TOKEY values do not specify a key value that is longer than the physical key of the file.

**CPYREC-024-E : TOTYPE(RRN) may not use OPEN(IO) unless FROM() is also an RRN type file**

Description The previously entered CPYREC command included an OPEN(IO) parameter for an output RRN type file. This type of processing is not allowed unless the FROM() file is also an RRN file.

Level Error, return code 12 is set.

Action Either change the open mode to OUTPUT or change the TOTYPE to another file type, then re-run the command.

**CPYREC-025-I : TOTYPE(RRN) opened in EXTEND mode, first record will be RRN # *rrn-number***

Description The previously entered CPYREC command will be copying records to an output RRN file which has been opened in EXTEND mode. The first RRN number written to the output file will be as indicated.

Level Informational

Action None required. CPYREC will proceed.

**CPYREC-026-I : Records skipped : *count***

Description The previously entered CPYREC command has finished processing. The number of records skipped (as a result of a SKIP() parameter) is indicated.

Level Informational.

Action None required.

**CPYREC-027-I : Records copied : *count***

Description The previously entered CPYREC command has finished processing. The number of records copied is indicated.

Level Informational.

Action None required.

**CPYREC-028-I : Records updated : *count***

Description The previously entered CPYREC command has finished processing. The number of pre-existing records updated on the TO() file is indicated.

Level Informational.

Action None required. Note, this message is only displayed when an OPEN(IO) parameter was included in the CPYREC command.

**CPYREC-029-E : Invalid or unrecognized COUNT() parameter**

Description The previously entered CPYREC command included a COUNT() parameter that was either malformed, or included an invalid value.

Level Error, return code set to 12.

Action Ensure the COUNT() parameter conforms to CPYREC syntax, also that a value in the range of 1 through 9,999,999 only is included.

**CPYREC-030-E : Invalid or unrecognized SKIP() parameter**

Description The previously entered CPYREC command included a SKIP() parameter that was either malformed, or included an invalid value.

Level Error, return code set to 12.

Action Ensure the SKIP() parameter conforms to CPYREC syntax, also that a value in the range of 1 through 9,999,999 only is included.

**CPYREC-031-E : Invalid or unrecognized FROMTYPE() parameter**

Description The previously entered CPYREC command included a FROMTYPE() parameter that was either malformed, or included invalid value(s).

Level Error, return code set to 12.

Action Ensure the FROMTYPE() parameter conforms to CPYREC syntax.

### **CPYREC-032-E : Invalid or unrecognized TOTYPE() parameter**

Description The previously entered CPYREC command included a TOTYPE() parameter that was either malformed, or included invalid value(s).

Level Error, return code set to 12.

Action Ensure the TOTYPE() parameter conforms to CPYREC syntax.

### **CPYREC-033-E : Invalid type in FROMTYPE() parameter**

Description The previously entered CPYREC command included a FROMTYPE() parameter that did not specify a valid file type..

Level Error, return code set to 12.

Action Ensure the FROMTYPE() parameter conforms to CPYREC syntax and includes only a type of IDX, RSQ, LSQ or RRN.

### **CPYREC-034-E : Invalid type in TOTYPE() parameter**

Description The previously entered CPYREC command included a TOTYPE() parameter that did not specify a valid file type..

Level Error, return code set to 12.

Action Ensure the TOTYPE() parameter conforms to CPYREC syntax and includes only a type of IDX, RSQ, LSQ or RRN.

### **CPYREC-035-E : Invalid or unrecognized FROM) parameter**

Description The previously entered CPYREC command included a FROM() parameter that was malformed.

Level Error, return code set to 12.

Action Ensure the FROM() parameter conforms to CPYREC syntax. If the name specified includes an embedded space, ensure the name is enclosed by double quote characters.

### **CPYREC-036-E : Invalid or unrecognized TO() parameter**

Description The previously entered CPYREC command included a TO() parameter that was malformed.

Level Error, return code set to 12.

Action Ensure the TO() parameter conforms to CPYREC syntax. If the name specified includes an embedded space, ensure the name is enclosed by double quote characters.

### **CPYREC-037-E : Invalid or unrecognized FROMKEY() parameter**

Description The previously entered CPYREC command included a FROMKEY() parameter that was malformed.

Level Error, return code set to 12.

Action Ensure the FROMKEY() parameter conforms to CPYREC syntax. If the value specifies a hex string, ensure the string contains full hex characters and that it is entered with X' ' notation.

### **CPYREC-038-E : Invalid or unrecognized TOKEY() parameter**

Description The previously entered CPYREC command included a TOKEY() parameter that was malformed.

Level Error, return code set to 12.

Action Ensure the TOKEY() parameter conforms to CPYREC syntax. If the value specifies a hex string, ensure the string contains full hex characters and that it is entered with X' ' notation.

### **CPYREC-039-E : Invalid or unrecognized PAD() parameter**

Description The previously entered CPYREC command included a PAD() parameter that was malformed.

Level Error, return code set to 12.

Action Ensure the PAD() parameter conforms to CPYREC syntax. If the value specifies a hex string, ensure the string contains full hex characters and that it is entered with X' ' notation.

### **CPYREC-040-E : Invalid hex value in FROMKEY() parameter**

Description The previously entered CPYREC command included a FROMKEY() parameter that contained an invalid hex string.

Level Error, return code set to 12.

Action Ensure the FROMKEY() value contains only hexadecimal characters, that is, characters in the range of x'00' through x'FF'.

### **CPYREC-041-E : Invalid hex value in TOKEY() parameter**

Description The previously entered CPYREC command included a TOKEY() parameter that contained an invalid hex string.

Level Error, return code set to 12.

Action Ensure the TOKEY() value contains only hexadecimal characters, that is, characters in the range of x'00' through x'FF'.

### **CPYREC-042-E : Invalid hex value in PAD() parameter**

Description The previously entered CPYREC command included a PAD() parameter that contained an invalid hex string.

Level Error, return code set to 12.

Action Ensure the PAD() value contains only hexadecimal characters, that is, characters in the range of x'00' through x'FF'.

### **CPYREC-043-E : Invalid or unrecognized PAD() parameter**

Description The previously entered CPYREC command included a PAD() parameter that contained invalid syntax or an unrecognizable value string.

Level Error, return code set to 12.

Action Ensure the PAD() parameter is properly formed.

### **CPYREC-044-E : Write error on TO() in *open-mode* mode, return code *file-status***

Description An IO error was encountered while writing records to the file specified in the TO() parameter. Note the open mode (which may be I/O, OUTPUT or EXTEND) and the file status values.

Level Error, return code set to 12.

Action Determine the reason the indicated file status was produced. Most likely the reason has to do with either loading a new file (OUTPUT) with records that are out of sequence, or adding records to an existing file (EXTEND) and duplicates are encountered.

### **CPYREC-045-I : TXT input file type assumed to be FROMTYPE(LSQ)**

Description A CPYREC command found a FROM file that included an extension of .txt, additionally no FROMTYPE() parameter was found. CPYREC is assuming that the from file is a Line Sequential file.

Level Warning, return code set to 4.

Action None required unless the 4 return code is undesired. If the file is in fact not a Line Sequential file, add the FROMTYPE parameter to the command

### **CPYREC-046-W : TOTYPE() values assumed from FROMTYPE() values**

Description A CPYREC command for a FROMTYPE(RSQ) or FROMTYPE(LSQ) did not include a TOTYPE() parameter and the file type being copied to cannot be determined.

Level Warning, return code set to 4.

Action None required unless the 4 return code is undesired. If the file is in fact not a Line Sequential file, add the FROMTYPE parameter to the command

## DEFINE messages

### **DEFINE-001-I : DEFINE complete. Maximum return code *n***

Description The previously entered DEFINE command has completed. The maximum return code is indicated as '*n*'.

Level Informational

Action None required. If the return code is non-zero, however, the DEFINE did not complete, or a warning message was issued. See the messages issued prior to this message, which indicate the specific situations encountered during DEFINE processing.

### **DEFINE-002-E : Invalid file type, must be LSQ, RSQ, RRN or IDX**

Description The DEFINE command does not indicate one of the 4 valid file types.

Level Error, return code set to 12

Action Correct the DEFINE command and re-run the command.

### **DEFINE-003-E : Key information only allowed with IDX type file**

Description The DEFINE command includes a PRIKEY or ALTKEY parameter which is allowed only when the file type being defined is IDX.

Level Error, return code set to 12

Action Correct the DEFINE command to either define an IDX file, or remove the PRIKEY and or ALTKEY parameters, then re-run the command.

### **DEFINE-004-E : NAME() parameter missing**

Description The DEFINE command did not include a valid NAME() parameter, or if present, the NAME() parameter did not include a valid file name within the parentheses.

Level Error, return code set to 12

Action Correct the DEFINE command then re-run the command.

### **DEFINE-005-E : RECLLEN() parameter missing or incorrectly specified**

Description The DEFINE command did not include a valid RECLLEN parameter, or if present, the minimum and or maximum record lengths were zero..

Level Error, return code set to 12

Action Correct the DEFINE command then re-run the command.

### **DEFINE-006-E : Primary key missing or has zero length**

Description The DEFINE command for an IDX type file did not include a PRIKEY() statement that referenced a valid, non-zero-length primary key.

Level Error, return code set to 12

Action Correct the DEFINE command then re-run the command.

### **DEFINE-007-E : Primary key extends beyond minimum record length**

Description The DEFINE command for an IDX file specified a PRIKEY parameter that would cause the end of the primary key to extend beyond the minimum record length

Level Error, return code set to 12

Action Correct the DEFINE command so that the primary key is contained completely within the boundaries of the minimum record length.

### **DEFINE-008-E : Alternate key *n* extends beyond minimum record length**

Description The DEFINE command for an IDX type file specified an ALTKEY parameter that would cause the end of the key to extend beyond the minimum record length

Level Error, return code set to 12

Action Correct the DEFINE command so that the primary key is contained completely within the boundaries of the minimum record length. Note, the key number refers to the 'n'th ALTKEY parameter found in the input command.

#### **DEFINE-009-E : Create error *return-code***

Description The DEFINE command for an IDX type file has failed. The return code shown is a standard COBOL file status.

Level Error, return code set to 12

Action Determine the cause of the failure and correct it prior to re-running the command.

#### **DEFINE-010-E : Unknown data following key length**

Description A PRIKEY or ALTKEY parameter was incorrectly formed.

Level Error, return code set to 12

Action Correct the PRI/ALT KEY parameter to conform to proper RMSFILE syntax, then re-run the command.

#### **DEFINE-011-E : DUP setting in KEY definition not allowed on PRIKEY**

Description The DEFINE command for an IDX type file specified a DUP parameter within the PRIKEY parameter. This is not allowed.

Level Error, return code set to 12

Action Correct the DEFINE command so that the PRIKEY parameter specifies only an offset and a length, then re-run the command.

#### **DEFINE-012-E : Key length must be greater than zero bytes**

Description The DEFINE command for an IDX type file specified an ALTKEY or PRIKEY parameter that specified a zero length key.

Level Error, return code set to 12

Action Correct the DEFINE command so that the all KEY statements include a key length of at least 1 byte.

### **DEFINE-013-E : Too many alternate keys defined**

Description The DEFINE command for an IDX type file specified 6 or more ALTKEY parameters.

Level Error, return code set to 12

Action Correct the DEFINE command so that it includes not more than 5 ALTKEY statements.

### **DEFINE-014-E : *string* is not a valid KEY definition**

Description The DEFINE command for an IDX type file specified an invalid ALTKEY or PRIKEY parameter.

Level Error, return code set to 12

Action Correct the DEFINE command so that the ALTKEY and or PRIKEY parameter conforms to standard RMSFILE syntax.

### **DEFINE-015-E : Minimum record length must not be greater than maximum**

Description The DEFINE command specified a minimum record length which was greater than the maximum record length. This is not allowed.

Level Error, return code set to 12

Action Correct the DEFINE command so that the RECLLEN parameter include a minimum record length which is less than, or at most, equal to the maximum record length.

### **DEFINE-016-E : *string* is not a valid RECLLEN definition**

Description The DEFINE command specified an invalid or incomplete RECLLEN() parameter

Level Error, return code set to 12

Action Correct the DEFINE command so that the primary key is contained completely within the boundaries of the minimum record length. Note, the key number refers to the 'n'th ALTKEY parameter found in the input command.

**DEFINE-017-E : *string* is not a valid NAME definition**

Description The DEFINE command included an invalid or incomplete NAME() parameter.

Level Error, return code set to 12

Action Correct the DEFINE command so that the NAME() parameter references a valid, fully qualified file name including the file extension.

**DEFINE-018-E : File already exists, cannot define duplicate file**

Description The DEFINE command referenced a file, via the NAME() parameter that already exists.

Level Error, return code set to 12

Action Correct the DEFINE command so that it references a non-existent file, or include a DELETE command prior to the DEFINE command.

**DEFINE-019-E : MODEL file does not exist, cannot retrieve required information**

Description The DEFINE command referenced a file, via the MODEL() parameter that doesn't exist.

Level Error, return code set to 12

Action Correct the DEFINE command so that it either references an existing file via the MODEL parameter, or include any required PRIKEY, ALTKEY and or RECLLEN parameters.

**DEFINE-020-E : MODEL file does not contain a file header, file may not be used as a MODEL() file**

Description The DEFINE command referenced a file, via the MODEL() parameter, that exists, but does not contain a Micro Focus file header from which RMSFILE can retrieve file definition information.

Level Error, return code set to 12

Action Correct the MODEL parameter to point to a file with a file header (i.e., an indexed file, a variable length record sequential file or a variable length relative record file only), or add any required RECLEN, PRIKEY and or ALTKEY parameters as needed.

**DEFINE-021-E : Unable to retrieve file information from MODEL file, return code *file-status***

Description The DEFINE command referenced a file, via the MODEL() parameter, that exists, however RMSFILE was unable to retrieve the file information from the file.

Level Error, return code set to 12

Action Determine the reason RMSFILE was unable to access the file based on the file status code provided, then re-run the command.

**DEFINE-022-E : Incompatible file types, DEFINE *define-file-type* may not use MODEL type of *model-file-type***

Description The DEFINE command referenced a file, via the MODEL() parameter, that is not of an organization that is compatible with the file being defined.

Level Error, return code set to 12

Action Use only like type files for MODEL and DEFINE types. That is, if an IDX file is being defined, use an IDX type file as the MODEL file. See the description of the MODEL parameter for the DEFINE command for complete information on allowed file types for MODEL.

## DELETE messages

### **DELETE-001-I : DELETE complete. Maximum return code *n***

Description The previously entered DELETE command has completed. The maximum return code is indicated as '*n*'.

Level Informational

Action None required. If the return code is non-zero, however, the DELETE did not complete, or a warning message was issued. See the messages issued prior to this message, which indicate the specific situations encountered during DELETE processing.

### **DELETE-002-I : Disk file deleted.**

Description The previously entered DELETE command has successfully processed. DELETE will continue if the file being deleted was an Indexed Sequential file, otherwise processing is complete.

Level Informational

Action None required.

### **DELETE-003-E : Error deleting file, return code *n***

Description The previously entered DELETE command encountered an error. The return code displayed is a COBOL file status. See the Eden Server Messages and Codes manual for a description of the return code.

Level Error, return code 12 is set

Action Review the return code description in the Eden Server Messages and Codes manual for help in determining why the file could not be deleted.

#### **DELETE-004-I : Index component deleted.**

Description The previously entered DELETE command referenced an Indexed Sequential file. As such, the DELETE command has also deleted the “.IDX” index component of the file.

Level Informational

Action None required.

#### **DELETE-005-E : Error deleting index component, return code *n***

Description The previously entered DELETE command referenced an Indexed Sequential file. During delete processing for the “.IDX” index component of the file, an error occurred. The return code displayed is a COBOL file status. See the Eden Server Messages and Codes manual for a description of the return code.

Level Error, return code 12 is set

Action Review the return code description in the Eden Server Messages and Codes manual for help in determining why the index component could not be deleted. Note, errors may be encountered when attempting to access or re-create the Indexed Sequential file until the index component is also deleted. The index component may either be deleted manually or by entering an RMSFILE DELETE command that specifically references the index component by name.

#### **DELETE-006-W : File does not exist, nothing to delete**

Description The previously entered DELETE command referenced a disk file name that is not present.

Level Warning, return code 8 is set.

Action Verify the proper file name was specified.

## RENAME messages

### **RENAME-001-I : RENAME complete. Maximum return code *n***

Description The previously entered RENAME command has completed. The maximum return code is indicated as '*n*'.

Level Informational

Action None required. If the return code is non-zero, however, the RENAME did not complete, or a warning message was issued. See the messages issued prior to this message, which indicate the specific situations encountered during RENAME processing.

### **RENAME-002-E : Input FROMDD() does not exist**

Description The previously entered RENAME referenced a file via the FROMDD parameter, that does not exist. The RENAME command will not continue.

Level Error, return code 12 is set.

Action Determine the reason for the named file not being present and proceed accordingly.

### **RENAME-003-E : Input FROM() does not exist**

Description The previously entered RENAME referenced a file via the FROM parameter, that does not exist. The RENAME command will not continue.

Level Error, return code 12 is set.

Action Determine the reason for the named file not being present and proceed accordingly.

### **RENAME-004-E : A file already exists with the requested TO() name**

Description The previously entered RENAME command referenced a 'rename-to' name that already exists. The RENAME command will not continue.

Level Error, return code 12 is set.

Action Determine the cause of another file already being present with the desired name and proceed accordingly.

### **RENAME-005-I : RENAME successful, include .IDX component**

Description The previously entered RENAME command has completed without error. Additionally, RENAME determined that the base file being renamed as a Micro Focus indexed file. As such, RENAME also renamed the .IDX component of the file.

Level Informational

Action None required. The .IDX component was renamed to the same file name as the base component, except of course it retains its .IDX file extension.

### **RENAME-006-E : Error renaming .IDX component, return code *file-status***

Description The previously entered RENAME command successfully renamed the base file name, however as RENAME was able to determine that the base file was a Micro Focus indexed file, a RENAME of the .IDX component was also automatically attempted. This automatic rename of the .IDX component failed.

Level Error, return code 12 is set.

Action Determine the reason the RENAME failed based upon the file status value supplied. Also, be aware that the original base file being renamed was in fact renamed. The base file name was not subsequently re-named back to its original name following the .IDX component rename failure. This means your indexed file will be considered 'corrupt' (due to a 'missing' .IDX file). You should either manually change the base file name back to its original name or, manually rename the .IDX component. Note, if this error was caused by a missing .IDX component, use the REBUILD utility supplied with Micro Focus Net Express and rebuild the .IDX component, then re-run the rename operation if it is still needed.