Rocket M204 Fast/Unload

Reference

Version 4.4

November 2013
FUN-0704-RM-02

Notices

Edition

Publication date: November 2013
Book number: FUN-0704-RM-02
Product version: Version 4.4

Copyright

Trademarks

Examples

This information might contain examples of data and reports. The examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

License agreement

Note: This product may contain encryption technology. Many countries prohibit or restrict the use, import, or export of encryption technologies, and current use, import, and export regulations should be followed when exporting this product.

Contact information

Website: www.rocketsoftware.com

Rocket Global Headquarters
77 4th Avenue
Waltham, MA 02451–1468
USA
Tel: +1 781 577 4321
Fax: +1 617 630 7100

Contacting Global Technical Support

If you have current support and maintenance agreements with Rocket Software, you can access the Rocket Customer Portal and report a problem, download an update, or read answers to FAQs. The Rocket Customer Portal is the primary method of obtaining support.

To log in to the Rocket Customer Portal or to request a Rocket Customer Portal account, go to www.rocketsoftware.com/support

Alternatively, you can contact Global Technical Support by email or telephone:

Email: support@rocketsoftware.com
Telephone: 1 855 577 4323 (Toll-free US) or +1 781 577 4323 (International)

Contents

Summary of Changes

This section describes significant changes to the documentation. In most cases these changes correspond to enhancements made to the underlying product.

Fast/Unload Version 4.4

The following are features introduced in version 4.4:

- Support for the FILEORG X'80' feature introduced in version V6R3 of *Model 204*.
- Support for unloading an ad-hoc group of files: the OPEN command in a "batch" *Fast/Unload* job ("OPEN datafile" on page 67) now allows the specification of multiple filenames. Formerly. you were required to use the *Fast/Unload User Language Interface* to unload a group.
- The #FILENAME special variable is available. It obtains the name of the file currently being unloaded. See "Special variables" on page 28.

Fast/Unload Version 4.3

The following are features introduced in version 4.3:

- As described in "BLOB/CLOB processing considerations" on page 165, the principal new feature in version 4.3 of *Fast/Unload* is the ability to operate on BLOB and CLOB (collectively called "Lob") fields, which are introduced with V6R1 of *Model 204*. The following features support this:
  - FUEL %variables may contain strings longer than 255 bytes.
  - #functions may both accept arguments and produce results in excess of 255 bytes.
  - These operations are supported on/with Lob fields:
    - The CHANGE and ADD[C] statements
    - UAI (and so UNLOAD[C] statements)
    - Using Lob field values where strings longer than 255 bytes may be used
    - The NEW field statement (by ading WITH CLOB or WITH BLOB)
  - An FSTATS statistic for Table E page usage for each Lob field.

- The handling of floating point values in *Fast/Unload* has been extensively reviewed in version 4.3. Although it is not expected to affect anyone's current use of *Fast/Unload*, many small changes have been made which can produce different results, and it is strongly recommended that all customers upgrade to version 4.3 to obtain the revised, correct float handling now in place.

  The principal problem is that the results of a FUEL program can be incorrect, for many cases, if a FLOAT LEN 4 field is explicitly referenced in the program. For instance, a FLOAT LEN 4 field may not work with comparisons (the IF statement), with arithmetic, as a #function argument, with the PUT statement, or as an item in the UAI SORT statement. (In these last two cases, the statement will work correctly when `4` is specified for the PUT or SORT item length.)

  Assigning a FLOAT LEN 4 field to a %variable, then using that %variable, can produce the above errors — and possibly others, as well (particularly if FLOAT LEN 16 fields are explicitly referenced in a FUEL program).

  There are other odd cases in which float handling is performed incorrectly.

  "Floating Point Arithmetic and Numeric Conversion" on page 235 contains a complete specification of the handling of floating point values in *Fast/Unload*.

- In conjunction with the floating point project, a new #function is added: #FLOAT8 ("#FLOAT8" on page 127) accepts a numeric argument and returns the value of the argument as an 8-byte floating point value. If the argument is a 4-byte floating point value, then the conversion is done by appending binary zeroes; otherwise, it is done by the normal FUEL conversion to an 8-byte floating point value.

Although not related to new features, the following change has been made to the documentation:

- In addition to field name references, the SORT FIELDS statement ("Using SORT FIELDS" on page 208) allows references to %variables or (some) special variables.

Fast/Unload Version 4.2

The following are features introduced in version 4.2:

- UAI unloading of procedures and procedure aliases

  - The feature is invoked by the new UAI statement option `PROCS` (see "UAI statement options" on page 89), or by default, that is, by specifying neither `PROCS` nor the new UAI statement option `NOPROCS`.

  - Statistics about the file's procedures are available as part of FSTATS processing (see "Description of procedure statistics" on page 56). Generating these statistics enables an optimization of the procedure dictionary reloading.

Sirius Mods Version 6.7

The following are *Fast/Unload* related features introduced in *Sirius Mods* version 6.7:

- FastUnload and FastUnloadTask methods for the *Fast/Unload User Language Interface* ("Fast/Unload User Language Interface" on page 199).
- New system parameters FUNPARM ("FUNPARM" on page 204) and FUNMAXT ("FUNMAXT" on page 204).

Fast/Unload Version 4.1

The following are features introduced in version 4.1:

- New or changed program parameters:
  - FNvmask (see "FNvmask=X"yz"" on page 9)
  - LIBuff (see "LIBuff=n" on page 12)
  - FUNOUT, SORTOUT, SORTOUTD (see "FUnout" on page 11, "SORTOut | SORTOUTD" on page 17, and "Changing the default sort parameter" on page 292)
- New #function:
  - #ABDUMP (see "#CONCAT: Concatenate strings" on page 103)
- New "preprocessor" feature:
  - #IF/#ELSEIF/#ELSE/#ENDIF conditional compilation (see "#IF" on page 37)
- Support for multiple-output streams (see "Output Streams" on page 20), which includes these FUEL additions:
  - TO *destination* option added to multiple statements
  - OUT TO *destination* statement
  - NOUNLOAD statement

Fast/Unload Version 4.0

Following are new features introduced in version 4.0:

- New ADDC statement
- New DELETEC statement

- New FSTATS directive
- New LEAVE statement
- New REPEAT statement
- New UNLOAD[C] field statement
- New or changed program parameters:
  - ABenderr
  - List and NOList (and customization zap, allowing NOLIST as default)
  - FSTATS=MINMAX or FSTATS=AVGTOT (and customization zap, allowing MINMAX as default)
- New #functions:
  - #CONCAT_TRUNC
  - #DEBLANK
  - #DELWORD
  - #FIND
  - #LEFT
  - #LOWCASE
  - #NUM2STR
  - #ONEOF
  - #PAD
  - #PADR
  - #REVERSE
  - #RIGHT
  - #STRIP
  - #TRANSLATE
  - #UPCASE
  - #WORD
  - #WORDS
- New #OUTLEN and #OUTPOS special variables.
- Additional information produced by FSTATS.
- Additional job statistics.
- MISSING value treated as zero in numeric contexts.
- " date format token, numeric date format separators.
- BM, BD, BH date format tokens.
- Strict date matching in “string” #DATExxx functions.

Following are features which were also released as zaps to one or more versions prior to version 4.0:

- UAI SORT support for 2-digit years.
- Customize SORT program name.
- Use of FUEL outside FOR EACH RECORD.

Fast/Unload Version 3.1

- SPANSIZE window width introduced.
- CENTSPAN default changed from -75 to -50.
- ZYY date format token.
- Interpretation of I and * date format tokens.
- APPDATE clock value passed to *Fast/Unload* from User Language Interface.
- Support for *Sir2000 Field Migration Facility*.
  - SIRFIELD definitions unloaded by UAI.
  - FUEL supports ALIAS names
  - FUEL honors REFERENCE WARN and REFERENCE CANCEL

Fast/Unload Version 3.0

Major rewrite of this manual, coinciding with major enhancements to *Fast/Unload*.

CHAPTER 1 Introduction

Fast/Unload is a utility whose primary function is to quickly unload data from a *Model 204* data file to one or more sequential data sets. The *Fast/Unload* utility consists of several functional units. One unit is a data extraction facility which reads data from a *Model 204* data file. Another unit is a compiler which converts a special data language describing the output format to machine language. The generated machine language actually performs the output function. A third unit is a reporting facility which provides a job log and reports any special conditions or errors that might have occurred during a run.

Fast/Unload can either be invoked directly as a standalone load module or from a User Language program. When invoked as a standalone load module, *Fast/Unload* enqueues the data file in share mode unless explicitly requested otherwise. If the enqueue fails, the *Fast/Unload* terminates with an error condition code. It is thus the file manager's job to ensure that when *Fast/Unload* is run as a standalone load module, *Model 204* does not have the file to be unloaded locked in exclusive mode.

The *Fast/Unload User Language Interface*, purchased as a separate *Fast/Unload* option, allows one to invoke *Fast/Unload* using the FastUnload or FastUnloadTask method of the Recordset class or the $Funload function in a User Language program. With this approach, an application builds a set of records to be unloaded, using standard User Language statements; for example, you can reduce unload time by restricting the set of records using indexed *Model 204* fields. The *Fast/Unload* load module runs in a subtask (or PST under CMS) of *Model 204*. The *Fast/Unload User Language Interface* also provides the ability to unload data from a *Model 204* group.

With the *Fast/Unload User Language Interface*, the processing can be performed either **synchronously** or **asynchronously**.

A system manager can cancel or examine *Fast/Unload* requests, and each user can perform the same functions on asynchronous requests that he or she initiated.

See “Fast/Unload User Language Interface” on page 199 for more information about the *Fast/Unload User Language Interface*.

CHAPTER 2 Invoking Fast/Unload as a Standalone MVS Program

Fast/Unload is simply invoked via the EXEC JCL card. The program name of *Fast/Unload* as distributed is FUNLOAD. The following DDNAMEs are used by *Fast/Unload*:

- FUNIN — This DD contains the statements used to describe the unload. FUNIN must be composed of fixed length, 80 byte records.
- FUNPRINT — This DD will be used as a log file and for reporting errors. This DD will also be referred to as the **report data set**.
- *destination* — A DD is required for each of the sequential data sets (which must be unique) declared as an output stream to which records are to be unloaded. FUNOUT is the default output stream for FUEL programs written for versions prior to 4.1.

A DD statement is also required for the input *Model 204* data file. The DD statement for the *Model 204* file **must** match the internal name of the data file. In addition, if a *Model 204* data file is made up of multiple physical files, DD cards must be provided for all physical files making up the logical file.

It is also recommended that you allocate either a SYSUDUMP or SYSMDUMP file to the *Fast/Unload* job step.

The following is an example of JCL that runs *Fast/Unload* in an MVS environment.

```
//FUNLOAD  JOB (0),CLASS=C,MSGCLASS=A,NOTIFY=HOMER
//FUNLOAD  EXEC PGM=FUNLOAD,REGION=1024K
//STEPLIB  DD  DSN=SIRIUS.LOAD,DISP=SHR
//SYSMDUMP DD  DSN=HOMER.DUMP,DISP=SHR
//SIRXREFD DD  DSN=ULSPF.V404.SIRXREFD,DISP=SHR
//FUNOUT   DD  UNIT=TAPE,VOL=SER=DUMP1,
//             LABEL=(1,SL),DISP=(NEW,PASS),
//         DSN=SIRXREFD.OUTPUT,DCB=BLKSIZE=30000
//FUNPRINT DD  SYSOUT=*
//FUNIN    DD  *
OPEN SIRXREFD
FOR EACH RECORD
  PUT '*'
  OUTPUT
  PAI
END FOR
//
```

CHAPTER 3 Invoking Fast/Unload as a Standalone CMS Program

Fast/Unload must be invoked by the *Model 204* CMS interface because it uses the CMS interface's EXCP and BSAM simulation. The program name of *Fast/Unload* as distributed is FUNLOAD. A FILEDEF must be provided for any DD to be used by *Fast/Unload*. Note that because *Fast/Unload* uses the *Model 204* CMS interface, any file can be on an OS format minidisk. The following DDNAMEs are used by *Fast/Unload*:

- FUNIN - This DD contains the FUEL used to describe the unload. FUNIN must be composed of fixed length, 80 byte records.
- FUNPRINT - This DD will be used as a log file and for reporting errors. This DD will also be referred to as the **report data set**.
- *destination* - A DD is required for each of the sequential data sets (which must be unique) declared as an output stream to which records are to be unloaded. FUNOUT is the default output stream for FUEL programs written for versions prior to 4.1.

A FILEDEF statement is also required for the input *Model 204* data file. The FILEDEF statement for the *Model 204* file **must** match the internal name of the data file. In addition, if a *Model 204* data file is made up of multiple physical files, FILEDEF commands must be provided for all physical files making up the logical file.

The following is an example of an EXEC that runs *Fast/Unload* in a CMS environment. Note that while REXX is used here, the EXEC could be written in EXEC or EXEC2.

```
/* Exec to run Fast/Unload */
Address command;
'FILEDEF * CLEAR';
'FILEDEF FUNIN DISK FUN FUNLOAD A';
'FILEDEF FUNPRINT DISK FUN LISTING A',
                  '(RECFM VB LRECL 137 BLOCK 4096';
'FILEDEF FUNOUT TAP1 (LRECL 10000 BLOCK 30000 RECFM VB';
'FILEDEF SIRXREFD I DSN ULSPF V404 SIRXREFD';
'M204CMS FUNLOAD';
Exit rc;
```

Note that in the above example the *Fast/Unload* Extraction Language program is in file FUN FUNLOAD on a CMS format disk and the *Model 204* data file SIRXREFD is on an OS format minidisk.

CHAPTER 4 Parameters

Some basic parameters are provided to control the operation of *Fast/Unload*. These parameters must be provided either as a PARM on the EXEC card in MVS, for example:

```
//FUNLOAD  EXEC PGM=FUNLOAD,REGION=4096K,
//              PARM='NEBUFF=8 SEBUFF=1 SBBUFF=3 SEQ'
```

or as options on the M204CMS command in CMS, for example:

```
'M204CMS FUNLOAD ( NEBUFF 8 SEBUFF 1 SBBUFF 3 SEQ %'
```

Under MVS, parameters are specified either by a parameter name alone or a parameter name followed by an equals sign (=) followed by the parameter value. Under CMS, parameters are specified either by a parameter name followed by a space and a percent sign (%) or a parameter name followed by a space and a parameter value. In general, one does not have to specify the entire parameter name for *Fast/Unload* to recognize it. A given parameter can only be set once.

This chapter describes the parameter settings which are available for *Fast/Unload*. Note that the minimum required part of the parameter name is specified in upper case while the rest of the parameter is specified in lower case.

4.1 ABenderr=rc

This parameter specifies the minimum *Fast/Unload* return code which triggers an ABEND at the end of the run. Your JCL could include, for example:

```
// EXEC PGM=FUNLOAD,PARM=(ABENDERR=8)
//FUNOUT DD DISP=(NEW,CATLG,DELETE),...
```

so that the FUNOUT dataset is not cataloged when a severe error is encountered.

The default value of this parameter is zero, which means that *Fast/Unload* will not trigger an ABEND due to the return code.

You can customize the default (see “Setting default ABENDERR” on page 295), but note that the default using the User Language Interface is always 0.

This parameter is new in *Fast/Unload* version 4.0.

4.2 ALlmsg

This parameter only has meaning when *Fast/Unload* is invoked via the User Language Interface. This parameter indicates that you want to see all messages that would ordinarily go to the report data set when *Fast/Unload* is invoked as a standalone program. When invoked via the *Fast/Unload User Language Interface*, *Fast/Unload* attempts to minimize message traffic by suppressing certain informational messages. If you want to see all *Fast/Unload* report data, specify the ALLMSG option.

For example, the *Fast/Unload* input program when invoked using the User Language Interface is not ordinarily echoed on the report. That is to say, when invoked via the *Fast/Unload User Language Interface*, the NOLIST parameter is the default unless ALLMSG is specified. If you want to see the input program, specify either the LIST or the ALLMSG parameter.

4.3 Asynch

This parameter only has meaning when *Fast/Unload* is invoked via the *Fast/Unload User Language Interface*. This indicates that you want for $Funload to return as soon as an unload request has been accepted, rather than upon completion of the unload. When using the ASYNCH parameter, you cannot unload a found set that is locked in exclusive mode (with a FIND AND RESERVE) and you cannot have any data returned to a list. The ASYNCH parameter is assumed when data is being unloaded to the $FUNIMG and $FUNSSTR functions.

4.4 Dbcs={IBM|FUJITSU|HITACHI|NONE}

This indicates that your database file has fields that contain DBCS data. DBCS must be followed by the type of DBCS environment under which your database file was created. Any of the following types are valid: IBM, FUJITSU, HITACHI, or NONE. The default value for DBCS is NONE. See “Customization of Defaults” on page 291 if you want to set a DBCS default other than NONE.

4.5 Every=n

This indicates that you want *Fast/Unload* to unload every Nth record in the database. This option is useful for sampling records in a database. For example, to process every other input record, set EVERY to 2. EVERY processing counts existing records, not *Model 204* record numbers. You can use SKIPREC to set the starting record for EVERY. For example, you can unload every tenth record starting with the 100th record by setting SKIPREC to 99 and setting EVERY to 10. *Fast/Unload* would process the 100th record, the 110th record, etc. The default for this parameter is 0 which means that all records are processed.

4.6 FNvmask=X"yz"

FNVMASK specifies which of the 8 characters of the name of the file being unloaded may differ from the file name stored on the disk pages of the *Model 204* file. It must be specified as 5 characters of the form:

X"yz" where ***yz*** are two hexadecimal digits representing an 8-bit mask. Each 1-bit in the mask corresponds to a character position in the file name which may be different than the file name stored on the disk pages of the *Model 204* file.

In the following example, the file 'BFILE' has the name 'AFILE' on its disk pages, and so the first character (X"80") must be masked as different:

```
//CREATE   EXEC PGM=BATCH204
//AFILE1   DD DISP=SHR,DSN=DATA.AFILE1
//AFILE2   DD DISP=SHR,DSN=DATA.AFILE2
//CCAIN    DD *
 ...
CREATE AFILE FROM AFILE1, AFILE2
 ...
//COPY1    EXEC PGM=IEBCOPY
//SYSUT1   DD DISP=SHR,DSN=DATA.AFILE1
//SYSUT2   DD DISP=SHR,DSN=DATA.BFILE1
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
//COPY2    EXEC PGM=IEBCOPY
//SYSUT1   DD DISP=SHR,DSN=DATA.AFILE2
//SYSUT2   DD DISP=SHR,DSN=DATA.BFILE2
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
//UNLOAD   EXEC PGM=FUNLOAD,PARM='FNVMASK=X"80"'
//BFILE1   DD DISP=SHR,DSN=DATA.BFILE1
//BFILE2   DD DISP=SHR,DSN=DATA.BFILE2
//FUNIN DD *
OPEN BFILE
 ...
```

If *Fast/Unload* is invoked via the *Fast/Unload User Language Interface*, the value of the FNVMASK parameter in *Model 204* is automatically communicated, and this is the default value of FNVMASK for the unload. It may also be passed in the fifth argument of $Funload or the *Parameters* parameter to the FastUnload or FastUnloadTask methods, but if so, its value must be the same as the *Model 204* FNVMASK parameter. If the *Fast/Unload User Language Interface* is used to unload a file with FNVMASK set to a non-zero value, you must use version 6.1 or later of the *Sirius Mods*. The *Model 204* FNVMASK parameter is part of the *Sirius Performance Enhancements V3*; that product, at least through its release in version 6.1, is not generally available.

This parameter is new in *Fast/Unload* version 4.1.

4.7 Frecord=n

This sets the *Model 204* record number at which unloading is to begin. FRECORD must be followed by a positive decimal integer that indicates the desired starting record number. This can be useful when debugging a FUEL program or when doing tuning runs for *Fast/Unload* and one believes that the records at the start of the data file are not representative of the rest of the data file. The default for this value is 0 which means that *Fast/Unload* starts unloading at record 0 of the input file. Note that if the indicated record number does not exist, unloading simply begins at the first existing record after the indicated starting record number.

The FRECORD parameter only applies to the first file, if a group unload is being performed.

4.8 FStats[=AVGTOT|MINMAX]

This will gather field, Table, and procedure statistics, and it will check file integrity during the run. If this option is selected, the *Fast/Unload* report will contain a list of all defined fields in the database file, with field definition information and statistics about occurrences of the fields. It will also perform various integrity checks, and provide statistics about Table B and the file's procedures.

The FSTATS directive can be used instead of the FSTATS parameter. Also, the quantity of statistics reported for each field can be controlled by the FSTATS directive, using the AVGTOT or MINMAX option. PARM='FSTATS=AVGTOT' or PARM='FSTATS=MINMAX' provides the same processing as the corresponding FSTATS directives; the FSTATS directive over-rides any specification of the FSTATS parameter. See “FSTATS [AVGTOT | MINMAX]” on page 50 for a description of the FSTATS directive, and for an explanation of the statistics displayed on the *Fast/Unload* report listing.

The default processing for FSTATS with neither AVGTOT nor MINMAX is MINMAX, although this default can be changed with a customization zap (see “Setting default FSTATS processing” on page 295).

FSTATS is not valid if the Field Statistics Option is not linked with your Fast Unload load module.

4.9 FUnout

This indicates that you would like *Fast/Unload* to perform the I/O to the output sequential data set(s) even if going through a sort due to SORT statement(s) and/or UAI statement(s) with the SORT option. In most cases, when sorting data on one or more output streams, the sort package is responsible for performing output to that output stream's sequential data set. This parameter forces sorted output to be passed back to *Fast/Unload* via an E35 exit which then performs the I/O.

You might want to use the FUNOUT option because:

- You want to have sorted data go to an OS format minidisk under CMS.
- You want to use *Fast/Unload*'s defaults for output dataset format, rather than your sort package's defaults.
- Your sort package sometimes truncates output records.

Note that use of the FUNOUT option will probably increase CPU overhead and maybe I/O overhead. You can customize *Fast/Unload* so that FUNOUT is the default; see "Changing the default sort parameter" on page 292.

The FUNOUT option is mutually exclusive with the SORTOUT and SORTOUTD options (see "SORTOut | SORTOUTD" on page 17).

A line showing FUNOUT = ON in the *Fast/Unload* report data set indicates that FUNOUT is in effect. SORTOUT and SORTOUTD will indicate OFF.

4.10 Harderr= {ABEND|CANCEL|SKIP|SKIP0|IGNORE|IGNORE0}

This sets the action to be performed when a hard error is encountered. A hard error is a missing record or extension record or an unknown field in the *Model 204* data file. These can result from running against a data file or record set that is not protected from change with a record set lock when using the *Fast/Unload User Language Interface* or a database file enqueue when running *Fast/Unload* as a standalone load module. This type of error can also occur if running *Fast/Unload* against a broken file (other broken file errors, for example, inconsistencies in Table D, always cause *Fast/Unload* to end with a diagnostic dump).

The valid values of HARDERR are:

- ABEND - Which means stop the *Fast/Unload* with a job step ABEND.
- CANCEL - Which means stop the *Fast/Unload* with a minimum job step completion code of 4.
- SKIP - Which means do not unload any more from the current record for a UAI or PAI, or skip the rest of the record for other forms of unload; the minimum job step completion code is 4. In order to perform the UAI and PAI operations as quickly as

possible, some of the record may have been unloaded when a hard error is detected.

- SKIP0 - Which means skip (as above), but do not change the minimum job step completion code.
- IGNORE - Which means ignore the error. This option is identical to SKIP except in the case of a missing extension record. In this case HARDERR=IGNORE means that *Fast/Unload* should simply act as if the record did not contain an extension record pointer. The minimum job step completion code is 4.
- IGNORE0 - Which means ignore (as above), but do not change the minimum job step completion code.

The default value for this parameter is CANCEL. In all cases, hard errors are always reported.

4.11 IOAPP=pp

For Fujitsu/AE systems, this sets the two character name of the *Model 204* DCB appendage. *Fast/Unload* uses the *Model 204* DCB appendage when running under Fujitsu/AE systems to specify storage areas to be page fixed for EXCP I/O. The value of this parameter should be the same as the *Model 204* "EXCPVR" parameter. IOAPP should be specified for batch mode operation only. When invoked via the *Fast/Unload User Language Interface*, this parameter is ignored and the *Model 204* "EXCPVR" parameter is used. IOAPP is ignored for Hitachi and IBM systems.

4.12 LIBuff=n

This specifies the size of the work area used to hold a normalized FUEL statement (“normalized“ means reducing to a single blank all multiple blanks separating statement tokens, and putting together physical line continuations).

LIBUFF must be followed by a positive decimal integer that indicates the desired size. It is specified in bytes; the default is 7000, which should be more than enough for almost all FUEL programs. If you receive a FUNL error message indicating the line is too long, you can increase this parameter.

This parameter is new in *Fast/Unload* version 4.1.

4.13 List

This indicates that the *Fast/Unload* input program lines should be printed on the *Fast/Unload* report data set. This parameter is the default for *Fast/Unload*, unless it has been customized (see “Setting NOLIST as default” on page 294). The inverse of this parameter is NOList.

When *Fast/Unload* is invoked via the *Fast/Unload User Language Interface*, if the ALLMSG parameter is not specified, the NOLIST parameter is default.

The program listing contains the hexadecimal program offset corresponding to the input line, the nesting level (when it changes) with an indication of an increase (+) or decrease (-) in the level or that a block is introduced which has the same nesting level (=), the statement number, and the program source statement.

This parameter is new in *Fast/Unload* version 4.0.

4.14 Maxrec=n

This sets the maximum number of input records that will be processed. MAXREC must be followed by a positive decimal integer that indicates the desired maximum. This can be useful when debugging a FUEL program or when doing tuning runs for *Fast/Unload*. When the maximum number of records has been processed the *Fast/Unload* terminates as it ordinarily would when all records have be processed. The default for this parameter is 0 which means that all records to the end of the data file are unloaded.

4.15 Nbbuff=n

This specifies the number of base record buffers. NBBUFF must be followed by a decimal integer. This integer specifies the number of base record buffers. The size of these buffers is determined by the track size of the input device and the value specified for SBBUFF. Specifying a value of 1 for NBBUFF prevents any read ahead of base record buffers. Specifying a value greater than 2 would not be likely to provide much of a performance gain and could introduce significant delays in retrieving extension records. In fact, if extension records are expected to be numerous and physically distant from the base records, a single base record buffer might provide better performance than multiple base record buffers. The default value for NBBUFF is 2 meaning that *Fast/Unload* will always have a current base record buffer and will read ahead into the other buffer.

4.16 NEbuff=n

This specifies the number of extension record buffers. NEBUFF must be followed by a decimal integer that specifies the number of extension record buffers. The size of these buffers is determined by the track size of the input device and the value specified for SEBUFF.

Specifying a value greater than 1 for NEBUFF allows the extension records buffers to act as a first-in/first-out buffer pool. If you expect many physically scattered extension records, or if the *Fast/Unload* "Wait for extension buffer" statistic has a high value, you may want to use a high value for NEBUFF. This would be especially desirable if you have a large amount of real memory on your CPU.

The default value for NEBUFF is 2 when invoked via the *Fast/Unload User Language Interface*, and it is 40 when invoked as a standalone load module. This prevents *Fast/Unload* from wasting unnecessary storage when running in an ONLINE address space while providing a good sized extension buffer pool when running in its own address space. Note that extreme cases of extension record scattering will create uncorrectable performance problems for *Fast/Unload* as well as for *Model 204*.

4.17 NObuff=n

This specifies the number of output record buffers. NOBUFF must be followed by a decimal integer that specifies the number of output record buffers. The size of these buffers is determined by the DD or FILEDEF statements for the output data set(s). If no block size is specified in the data definitions for the output data set(s), *Fast/Unload* uses the largest possible block size given other data set characteristics. If output is going directly to a sort package, the size of the output buffers is always either the largest possible block size less than or equal to 4096, or the record length if output records can be more than 4096 bytes long.

If the *Fast/Unload* "Wait for output buffer" statistic has a high value, you might want to use a high value for NOBUFF. This would be especially desirable if you have a large amount of real memory on your CPU. The default value for NOBUFF is 2 meaning that *Fast/Unload* will be able to write one buffer while it is filling the other, thus providing overlap of output I/O with CPU processing. If running under MVS and using a relatively small output block size, specifying a larger value for NOBUFF would enable *Fast/Unload* to take advantage of MVS chained scheduling of I/O which would probably provide a significant speed increase.

The maximum value for NOBUFF is 99.

4.18 NOEnq

This indicates that *Fast/Unload* does not attempt to use standard *Model 204* enqueueing on the input data file. By default, *Fast/Unload* attempts to obtain a share lock on the input data file, and it will terminate if it is unable to obtain the share lock.

The NOENQ parameter may be useful if you want to run against a *Model 204* data file that has been opened by an ONLINE in exclusive mode but which you know is not being updated, if you want to run against a *Model 204* data file that is update protected by a security package, or if absolute data consistency is not critical (for creating certain reports, say, as opposed to reorganizing a file).

This parameter has no meaning when *Fast/Unload* is invoked via the *Fast/Unload User Language Interface*.

4.19 NOList

This indicates that the *Fast/Unload* input program lines should not be printed on the *Fast/Unload* report data set. The inverse of this parameter is List.

When *Fast/Unload* is invoked via the *Fast/Unload User Language Interface*, if the ALLMSG parameter is not specified, the NOLIST parameter is default. NOLIST can also be customized as the default for your installation; see "Setting NOLIST as default" on page 294.

A `LIST = OFF` line in the *Fast/Unload* report data set indicates that NOLIST is in effect.

This parameter is new in *Fast/Unload* version 4.0.

4.20 NOTify

This indicates that you want to be notified with a warning when the unload is complete. This parameter only has meaning when *Fast/Unload* is invoked via the *Fast/Unload User Language Interface*.

4.21 Orecerr={CANCEL|SKIP|IGNORE}

This sets the action to be performed when an output record error is encountered. This currently only refers to an output record exceeding the output record size. The valid values ORECERR are:

- CANCEL — Stop the *Fast/Unload*.

- SKIP — Skip the current record.
- IGNORE — Ignore the error and simply truncate the record.

The default value for this parameter is 'CANCEL'. For any value of ORECERR, output record errors are reported.

4.22 Sbbuff=n

This specifies the size of base record buffers in tracks. SBBUFF must be followed by a decimal integer. This integer specifies the size of base record buffers in tracks. Base record buffers are used to read in the non-extension record parts of *Model 204* data files. Extension records which happen to be contained in a current base record buffer will also be read from the base record buffer.

This value must be evenly divided into the number of tracks per cylinder on the input device. For example, if the input device is a 3380, this value could be either 1, 3, 5 or 15. If the input data set allocation is not in cylinder units and the input device does not support DEFINE EXTENT, the only valid value for this parameter is 1. (Prior to version 4.0 of *Fast/Unload*, SBBUFF greater than 1 requires data set allocation in cylinder units, regardless of the support for DEFINE EXTENT.) Most modern DASDs support DEFINE EXTENT.

The default value for SBBUFF is 1, meaning base record buffers are read in units of one track. If *Fast/Unload* statistics indicate a high "Base buffer wait time" value, one might be able to achieve better performance using a higher value for SBBUFF.

4.23 SEbuff=n

This specifies the size of the extension record buffers. SEBUFF must be followed by a decimal integer. This integer specifies the size of extension record buffers in tracks. When an extension record cannot be found in a base record buffer, it must be synchronously read into an extension record buffer.

This value must be evenly divisible into the number of tracks per cylinder on the input device. For example, if the input device is a 3380, this value could be either 1, 3, 5 or 15. If the input data set allocation is not in cylinder units and the input device does not support DEFINE EXTENT, the only valid value for this parameter is 1. (Prior to version 4.0 of *Fast/Unload*, SEBUFF greater than 1 requires data set allocation in cylinder units, regardless of the support for DEFINE EXTENT.) Most modern DASDs support DEFINE EXTENT.

The default for SEBUFF is 1, meaning extension records are read in units of one track. In general there is probably no advantage to setting this value to anything other than 1 and in fact a large SEBUFF would probably produce degraded performance.

4.24 SEQ

This parameter indicates that the input program (FUNIN in batch mode) has sequence numbers in columns 73 through 80. If this parameter is specified, all data after column 72 in the input program is ignored. By default, *Fast/Unload* reads all columns in the input program.

4.25 SKiprec=n

This sets the number of input records that will be skipped before the first record is processed. The FUEL program or UAI statement is not processed for skipped records. This statement is useful when splitting an unload into multiple pieces. For example, if a database file has approximately 2 million records and you want to split the unload into 2 pieces you could do one unload with MAXREC set to 1000000 then a second unload with SKIPREC set to 1000000. This parameter is different from FRECORD because FRECORD uses *Model 204* record numbers while SKIPREC does not count unused record numbers. When using SKIPREC in conjunction with FRECORD, FRECORD is used to set a starting record number and then SKIPREC records are skipped from that record number. Records skipped for SKIPREC are not counted as processed records against MAXREC. The default for this parameter is 0 which means that no records are skipped.

4.26 SORTOut | SORTOUTD

In contrast to the FUNOUT parameter, both these parameters tell the external sort package to write the sorted records directly to the output sequential data set(s) after the sort. As of version 4.1, SORTOUTD is the default for *Fast/Unload*, unless it has been customized (see “Changing the default sort parameter” on page 292).

For FUEL programs with one output stream, not explicitly declared (which includes all programs written prior to version 4.1), you can specify SORTOUT to maintain the pre-version 4.1 behavior: UAI SORT output is sent to the DD named FUNOUT, while non-UAI output is sent to the sort's SORTOUT DD. Alternatively, SORTOUTD provides a more consistent handling of such legacy programs: all sorted output, whether UAI or non-UAI, goes to FUNOUT, written by the sort program.

For programs with multiple outputs, if SORTOUT or SORTOUTD is specified or implied, *Fast/Unload* tells the sort program the user-supplied *destination* for each sorted output stream, and the sort program does its own output.

SORTOUT, SORTOUTD, and FUNOUT are mutually exclusive. In the *Fast/Unload* report dataset, only one of these will be ON. See “FUnout” on page 11.

4.27 SOrtp={31|24}

This parameter indicates the type of parameter list to be used to pass data to an external sort package. SORTP must be followed by either 24 or 31. If 24 is specified, the "old-fashioned" 24-bit parameter list is used. If 31 is specified, the 31-bit extended parameter list is used. The default for this parameter as shipped is 31, but you can modify this setting if your site requires. For more details about this parameter see "Sort Parameter List" on page 291.

4.28 UPArm="string"

This specifies a string which can be accessed in the FUEL program as a special variable (**#UPARM**). (See "Entities" on page 26 for a discussion of special variables in FUEL.) If the string contains any blanks, it can be preceded by and followed by one double quotation mark, for example:

```
// EXEC PGM=FUNLOAD,PARM='UPARM="Good job"'
```

The double quotation marks will not be part of the value of #UPARM. Under MVS, the maximum length of the string, not including the quotation marks, is 100. Under CMS, the string may not contain blanks, and the length of the string is restricted to 8.

4.29 Upper

This indicates that you want the report data set to be written using upper case characters only. This option should be used if you are using terminals or printers which do not correctly handle mixed case output. By default, *Fast/Unload* produces mixed case output.

CHAPTER 5 Fast/Unload Extraction Language

A user can specify the format and contents of data to be unloaded by *Fast/Unload*. This is done with the *Fast/Unload* Extraction Language (FUEL). The FUEL compiler is a separately priced *Fast/Unload* feature. If the FUEL compiler is not purchased, the only fast Unload statement available to a user is UAI. This chapter specifies the syntax and semantics of FUEL.

When *Fast/Unload* is invoked via the *Fast/Unload User Language Interface*, the FUEL statements are entered through $functions documented in http://m204wiki.rocketsoftware.com/index.php/List_of_$functions. When *Fast/Unload* is invoked as a standalone load module, the FUEL statements are provided in the FUNIN data stream. When *Fast/Unload* is invoked via the *Fast/Unload User Language Interface*, compiler errors are displayed in a specified sequential data set or returned to a $list. When *Fast/Unload* is invoked as a standalone load module, compiler errors are displayed on the FUNPRINT DD.

5.1 Program Structure

A FUEL program must have exactly one FOR EACH RECORD loop (except if it has just one UAI statement, in which case the FOR EACH RECORD loop is optional). Certain FUEL statements may occur inside the FOR EACH RECORD LOOP; these are called **executable** statements. Statements that may not occur inside the FOR EACH RECORD LOOP are called **directives**.

All executable statements except UNLOAD may also occur either before or after the FOR EACH RECORD loop. Directives must occur before any executable statement, in any order, excepting OPEN, which must be the first statement in the program.

The executable statements before the FOR EACH RECORD loop are performed once, at the start of the *Fast/Unload* job. The statements inside the FOR EACH RECORD loop are performed for each input record. The statements after the FOR EACH RECORD loop are performed after the end of the input file (the last input file, if a group unload is being performed).

As mentioned, a program with exactly one declared UNLOAD ALL INFORMATION (UAI) output may be coded without any accompanying FOR EACH RECORD loop. A single UAI directive coded without a FOR EACH RECORD loop simply implies that every record in the database is unloaded to the implicit FUNOUT destination. If any other declaring directive is present (that is, more than one UAI, no UAI, or any OUT TO), a FOR EACH RECORD loop is required.

See “UNLOAD ALL INFORMATION or UAI” on page 88 for more information about UAI, and see “UNLOAD[C] [field [(occur | *)]]” on page 83 for more information about using UAI with a FOR EACH RECORD LOOP.

This structure of a FUEL program can be summarized as follows:

```
OPEN filename
  -- Optional with the Fast/Unload User Language Interface
  -- Otherwise required
Other directives
  -- Optional
Statements executed once at start of job
  -- Optional (UNLOAD not allowed)

FOR EACH RECORD
   Statements executed once per input record
END FOR
  -- Optional if UAI specified

Statements executed once at end of job
  -- Optional (UNLOAD not allowed)
```

5.1.1 Output Streams

FUEL uses the concept of **output streams**, where a stream is the data that is output to a particular destination, typically a data set. Legacy FUEL programs (that is, those written for versions prior to 4.1) have one output stream, and that stream's **destination** (FUNOUT) is implicitly declared. As of version 4.1, a FUEL program may have multiple output streams, and their destinations are explicitly declared using one or both of the following:

- An OUT TO directive (see “OUT TO destination” on page 68)
- The `TO destination` option on a UAI directive (see “UNLOAD ALL INFORMATION or UAI” on page 88).

Once declared, the stream name is used:

- In the `TO destination` clause on subsequent output-generating statements.
- As the `TO destination` qualifier that associates the SORT control statement to a particular stream.
- As the parenthesized qualifier on certain special variables.

The output stream controlling statements are grouped below. For an example of a multiple output program, see “Sample Program with Multiple Outputs” on page 22.

OUT TO *destination* [DEF[AULT]]
UAI [TO *destination* [DEF[AULT]]] ...

SORT [TO *destination*] ...
[TO *destination*] PUT ...
[TO *destination*] OUTPUT
[TO *destination*] PAI
[TO *destination*] UNLOAD[C] [...]
[TO *destination*] NOUNLOAD [...]

... #RECOUT[(*destination*)] ...
... #OUTLEN[(*destination*)] ...
... #OUTPOS[(*destination*)] ...

Programming notes:

- For a legacy FUEL program, a programmer may not mix UAI/UNLOAD features with PUT/OUTPUT features. With a multiple-output program, these features may not be mixed *within* any single output stream, but a FUEL program may include PUT/OUTPUT stream(s) (which include PAI output) and UAI/UNLOAD stream(s).

- When compiling output-generating statements in the executable portion of a FUEL program with more than one output stream, the compiler has to insert code to see if the output stream has to change, and if it does, has to insert code to swap internal information relevant to the switching streams. It will minimize such checking and context switching if you group statements for each output stream when possible. For example:

```
PUT TO A FIELDA
PUT TO A FIELDB
PUT TO B SOMEFIELD
PUT TO B SOMEOTHERFIELD
```

  The statement sequence above is more efficient than the sequence below (although the difference is marginal except in large unloads with many streams).

```
PUT TO A FIELDA
PUT TO B SOMEFIELD
PUT TO A FIELDB
PUT TO B SOMEOTHERFIELD
```

5.1.2 Output Records

For each declared output stream (and in legacy code, in the implicitly declared single FUNOUT output stream), FUEL uses the concept of an **output record**. At any given point in a FOR EACH RECORD loop, there is exactly one output record for each output stream. The output record is considered empty at the start of each execution of the FOR EACH RECORD loop and after the execution of an OUTPUT or PAI statement.

Data is added to the current output record as follows:

- For a PUT/OUTPUT stream, data is added with PUT statements. A **cursor** (a position relative to the start of the output record) is initially set to point to the first character in the output record. Data may be placed into the output record at either the current cursor position, a position relative to the current cursor position, or an absolute position in the output record. The cursor is always positioned at the character after the last character added by a PUT statement.

  The OUTPUT statement marks the end of the output record and adds it to the output stream. If a PAI statement follows OUTPUT, the PAI places each *Model 204* record value into a separate output record in the stream.

- For a UAI/UNLOAD stream, data is added to the current output record with UNLOAD[C] field [(occurrence)] statements or with an UNLOAD statement (which, with no field specified, unloads all hitherto not unloaded fields, and is called a "blanket" UNLOAD).

As of version 4.1, except for legacy FUEL programs, these statements that write data to an output stream (PUT, OUTPUT, PAI, UNLOAD[C]), as well as those special variables that refer to aspects of an output stream (#RECOUT, #OUTPOS, #OUTLEN), must be qualified to indicate which of the declared output streams they affect. The qualifiers are the "TO destination" prefix on these statements (see “TO [destination | *]” on page 83) and, for the special variables, the destination in parentheses. Statements without such qualification (called "naked" output statements) can be included if they apply to an output stream declared to be the default stream (one PUT/OUTPUT stream and one UAI/UNLOAD stream may be declared to be the default for their type of output).

5.1.3 Sample Program with Multiple Outputs

The program below splits an employee database into a full unload copy and two separate user-defined sorted datasets. One of the sorted datasets contains data on employees older than fifty, and the other contains the same information for employees fifty and younger.

```
OPEN EMPFILE

/* Output stream for age-discrimination */
OUT TO OLDONES
SORT TO OLDONES -
        FIELDS=(AGE,A,LASTNAME,A,FIRSTNAME,A)
SORT TO OLDONES RECORD TYPE=F,LENGTH=(500)

/* Output stream for callow youths */
OUT TO KEEPERS DEFAULT
SORT TO KEEPERS -
        FIELDS=(AGE,A,LASTNAME,A,FIRSTNAME,A)
SORT TO KEEPERS RECORD TYPE=F,LENGTH=(500)

/* Output stream for full unload */
UAI TO UNLOADED DEFAULT

FOR EACH RECORD
  IF AGE > 50
    /* Put info on OLDONES stream */
    TO OLDONES PUT LASTNAME
    TO OLDONES PUT FIRSTNAME
    TO OLDONES PUT AGE
    TO OLDONES PUT EMPNUM
    TO OLDONES PUT SALARY
    TO OLDONES PUT HIREDATE
    TO OLDONES PUT DATA(*)
    TO OLDONES OUTPUT
  ELSE
    /* Put info on KEEPERS (default) stream */
    PUT LASTNAME
    PUT FIRSTNAME
    PUT AGE
    PUT EMPNUM
    PUT SALARY
    PUT HIREDATE
    PUT DATA(*)
    OUTPUT
  END IF

  /* Unload to UNLOADED (default) stream */
  UNLOAD

END FOR
```

5.1.4 Outside the FOR EACH RECORD loop

You may code additional FUEL statements before the FOR EACH RECORD loop and/or after the loop. The statements before the FOR EACH RECORD loop are executed once at the start of the job; the statements after the loop are executed once at the end of the job. These statements have, in effect, an empty input record.

The following example shows the use of outside-the-loop statements to report the maximum value of a field in the file:

```
OPEN DATA
FOR EACH RECORD
   IF %MAX < FIELDA THEN
      %MAX = FIELDA
   END IF
END FOR
REPORT 'Highest value of FIELDA:' AND %MAX
```

Here is an example of statements that create a "trailer" record:

```
OPEN INFIL
FOR EACH RECORD
   PUT '01'    /* Output record type data
   PUT ...
     ...
   OUTPUT
END FOR     /* End of FOR EACH RECORD
PUT '02'    /* Output record type trailer
PUT ...
  ...
OUTPUT
```

Notes:

- All statements allowed within a FOR EACH RECORD loop, except UNLOAD, are allowed outside a FOR EACH RECORD loop.
- Fields from the input file can be referenced outside a FOR EACH RECORD loop.
- Outside a FOR EACH RECORD loop, all fields have zero occurrences, unless an ADD statement (see “ADD[C] field = expr” on page 38) is executed.

5.2 Input Program (FUNIN) Conventions

FUEL statements are terminated by the end of a logical record. A logical record consists of one or more physical records. If a physical record's last non-blank character is a hyphen (-), the following physical record is considered part of the same logical record.

Comments can be placed into a FUEL program in two ways.

- Any logical line whose first non-blank character is an asterisk (*) is considered a comment.
- Any part of a logical line that occurs after a slash-asterisk character combination (/*) is considered a comment.

Note that comments can be continued the same way that any other physical line can be continued. In the statement

```
PUT FIELD1 AS FLOAT(4)     /* This is a comment
```

the phrase 'This is a comment' is a comment. The whole line

```
*  This is a comment line
```

is a comment. In the following example

```
PUT '*'       /*  Put out an asterisk -
PUT FIELD1
```

the phrase 'PUT FIELD1' would be considered part of the comment because the first physical line ends with a continuation character. Finally,

```
*   This comment goes on              -
                  and on              -
                   and on
```

is another example of a continued comment line. Since the hyphen is not the last non-blank character on the physical record, the following example is not a continuation:

```
REPORT '*' AND -  /* Put out an asterisk
   #RECORD
```

The above example will cause a syntax error.

Fast/Unload will read either all of each input record, or only columns 1-72, depending on the **SEQ** parameter. See the discussion of the **SEQ** parameter in “Parameters” on page 7.

FUEL programs are compiled into efficient object code for use by the actual data unload step. Any FUEL program which contains syntax errors will generate compilation errors and will not be executed. In the syntax descriptions that follow, optional keywords or clauses are enclosed in square brackets ([ ]).

5.3 Program Elements

The basic program element which denotes a value in FUEL is called an **entity**. In addition, the **assignment statement** allows you to assign a value to a %variable, which is one form of entity. The value that can be assigned to a %variable can be an entity, an **expression**, or a **#function call**.

This section explains these concepts.

5.3.1 Entities

Many FUEL statements operate on entities. The entity types are described in the following subsections.

5.3.1.1 Field name with occurrences

This type of entity describes data in the input *Model 204* data file. A field name can be followed by an optional occurrence number enclosed in parentheses. If an occurrence number is not specified, the FUEL compiler assumes that you are referring to the first occurrence of the field.

The field name may be for any type of *Model 204* field (including BLOB or CLOB, as of *Fast/Unload* 4.3).

The occurrence number can be either an integer count, a loop control variable, a %variable, or the number (#) symbol:

- If the occurrence number is a loop control variable, the occurrence number is derived from the current value of the loop control variable.
- If the occurrence number is a %variable, it must contain a numeric value greater than or equal to 1 (fractional values are ignored). If the %variable is non-numeric, is less than 1, or is greater than the maximum fixed value, *Fast/Unload* is cancelled.
- The number symbol indicates a reference to the occurrence count rather than to a particular occurrence. The PUT statement also allows an asterisk (*), which indicates all occurrences.
- If a field name is specified without an occurrence number, the occurrence number defaults to 1.

If a field occurrence is referenced that is greater than the number of occurrences of that field in the record, that occurrence has the MISSING value, and is treated as the null, or zero-length, string in contexts needing a string value, or as zero in contexts needing a numeric value. For example, this means that a statement such as

```
%TOTAL = %TOTAL + SALARY
```

can be placed in your FUEL program **without** requiring a test such as

```
IF SALARY EXISTS
   %TOTAL = %TOTAL + SALARY
END IF
```

(Treating the MISSING value as zero in numeric contexts is new in version 4.0 of *Fast/Unload*; prior to that, a MISSING value in numeric context would terminate the job.)

If a field name appears in a context where it might be interpreted as part of a *Fast/Unload* statement, the ambiguity can be removed by placing a single quote (') around the ambiguous part of the field name. For example, the field name YEARS AT RESIDENCE could be coded `YEARS 'AT' RESIDENCE` in a PUT statement.

5.3.1.2 Constants

There are three kinds of constants: string constants, fixed constants, and floating point constants.

A string constant must be enclosed in single quotes ('). If you want the single quote to appear in a string constant, you must double it. For example, if you want to refer to a string constant containing "That's show business", you must code it as `'That''s show business'`.

A fixed constant can begin with an optional sign and can contain only decimal digits. It must be within the range -2**31 + 1 (-2,147,483,647) to 2**31 - 1 (2,147,483,647); otherwise it is treated as a floating point constant. `23`, `-18`, and `1678` are examples of valid fixed constants.

A floating point constant must contain a single decimal point and/or the letter `E`, or it must be outside the range -2**31 + 1 (-2,147,483,647) to 2**31 - 1 (2,147,483,647). It can begin with an optional sign, and can end with the letter `E` (to indicate a power of 10 multiplier) followed by a fixed constant. Other than the leading sign, the decimal point, the `E`, and the sign of the power of 10 multiplier, it must otherwise contain only decimal digits.

The following are examples of floating point constants:

```
1E10
77.19
3.14159
-2.718
1.3E2
12345678901
1.E-4
```

A floating point constant, expressed in base 10 in a FUEL program, is converted to IBM 360 floating point representation; see “Floating Point Arithmetic and Numeric Conversion” on page 235 for a discussion of this.

5.3.1.3 Loop control variables

These variables are represented by a single letter of the alphabet (thus allowing at most 26 of them), and they are given values by the FOR/FROM/TO statement. References to loop control variables outside their corresponding FOR loop produces unpredictable results.

5.3.1.4 Special variables

These variables represent internal values maintained by *Fast/Unload*. The special variables are:

#ERROR Indicates the result of a PUT operation. Success sets #ERROR to zero, a missing value sets it to 1, and a conversion or truncation error sets it to 2.

#FILENAME The name of the file currently being unloaded.

#GRPMEM The current file number within the *Model 204* group (1 for the first file, etc.). #GRPMEM is simply 1 if a group is not being unloaded.

#GRPSIZ The number of files in the group being unloaded. #GRPSIZ is simply 1 if a group is not being unloaded.

#OUTLEN[(destination)]

Reflects the length of the current output record in a non-UAI output stream.

If there is exactly one output stream in your FUEL program (that is, in a program written prior to *Fast/Unload* version 4.1, or in a program with exactly one explicitly declared output stream), you may use the unqualified form of #OUTLEN. Otherwise, you must indicate which output stream's record length is meant by specifying the stream destination in parentheses:

```
#OUTLEN(destination)
```

In FUEL programs with more than one output stream, the destination qualifier may be omitted if you want to refer to the default OUT TO stream, that is, the destination that is declared in an OUT TO statement with the DEFAULT or DEF attribute (see “OUT TO destination” on page 68).

Notes:

- When used in a PUT statement, #OUTLEN(*destination*) is the length of the output record on the *destination* output stream prior to the PUT statement. The *destination* value need **not** be the same stream as the PUT statement.
- #OUTLEN may not be used in the REPORT statement (you may, of course, assign #OUTLEN to a %variable, and use that %variable in a REPORT statement).
- #OUTLEN is not valid for a UAI format output stream.

This special variable is valid in *Fast/Unload* version 4.0 and above.

#OUTPOS[(destination)]

Reflects the output position that will be used by the next PUT statement on the explicit or implied destination, if it does not contain the AT clause. Unless a previous AT clause has specified something less than the current position, the value of #OUTPOS will be the value of #OUTLEN plus one, on the particular stream.

If there is exactly one such output stream in your FUEL program (that is, in a program written prior to *Fast/Unload* version 4.1, or in a program with exactly one explicit OUT TO *destination* declaration), you may use #OUTPOS without a stream qualifier. Otherwise, you must indicate which output stream's position is meant by specifying the stream destination in parentheses:

```
#OUTPOS(destination)
```

In FUEL programs with more than one output stream, the destination qualifier may be omitted if you want to refer to the default OUT TO stream, that is, the destination that is declared in an OUT TO statement with the DEFAULT or DEF attribute (see "OUT TO destination" on page 68).

Notes:

- When used in a PUT statement, #OUTPOS(*destination*) is the position in the output record on the *destination* output stream prior to the PUT statement. The *destination* value need **not** be the same stream as the PUT statement.
- #OUTPOS may not be used in the REPORT statement (you may, of course, assign #OUTPOS to a %variable, and use that %variable in a REPORT statement).
- #OUTPOS is not valid for a UAI format output stream.

This special variable is valid in *Fast/Unload* version 4.0 and above.

#RECIN

The current input record number in the *Model 204* data file. In FUEL placed before the FOR EACH RECORD loop, #RECIN is -400000000 (-4E8). In FUEL placed after the FOR EACH RECORD loop, #RECIN is -300000000 (-3E8).

Note that the record number is not the same as number of input records, because of deleted records, "skipped" record numbers due to the values of the BRECPPG and BRESERVE *Model 204* parameters, and so on.

For example, if you are unloading a file's records to two output streams, RECS1 and RECS2, and you want to limit the first stream to the first 50,000 records. Using a loop with #RECIN < 50000 to route the records will **not** succeed in most cases because of the likely missing record numbers. Instead, code like the following is what you need:

```
OPEN MYFILE
UAI TO RECS1 OINDEX
UAI TO RECS2 OINDEX
%RECBLOCK = 50000
FOR EACH RECORD
    IF %RECBLOCK GT 0.0 THEN
       %RECBLOCK = %RECBLOCK - 1
       TO RECS1 UNLOAD
     ELSE
       TO RECS2 UNLOAD
    END IF
END FOR
```

Using 0.0 above is not necessary, but it forces a float type comparison, which is the type of value in %RECBLOCK after the subtraction. Thus it avoids converting the current value to a fixed value on each loop, which happens if you use %RECBLOCK GT 0.

#RECOUT[(destination)]

The current output record number. For a non-UAI unload, it is equal to the total number of OUTPUT statements executed, plus the number of records written by the PAI statement on the *destination* output stream. For a UAI unload, it is the position in the output file of the last record written for the last UNLOAD statement on the *destination* output stream.

The number of #RECOUT records depends on your output LRECL value and on the size of the unloaded records. A single *Model 204* input record might consume more than one output record.

If there is exactly one output stream in your FUEL program (that is, in a program written prior to *Fast/Unload* version 4.1, or in a program with exactly one explicitly declared output stream), you may use the unqualified form of #RECOUT. Otherwise, you must indicate which output stream is meant by specifying the stream destination in parentheses:

```
#RECOUT(destination)
```

Note that, since #RECOUT is valid for both UAI and non-UAI type streams, you must still provide the destination qualifier for #RECOUT in cases where both types of stream output are declared, even if some output stream is declared to be the default. For more information about declaring default streams, see “UNLOAD ALL INFORMATION or UAI” on page 88 and “OUT TO destination” on page 68.

#UPARM The value of the UPARM parameter.

Examples:

- You are processing record number 5672 from your *Model 204* data file (not using a group) and you have currently output 1783 records: #RECIN would be 5672, #RECOUT would be 1783, and #GRPSIZ and #GRPMEM would be 1.

- You are processing record number 143 from your the second *Model 204* data file in a group of 5 files, and you have currently output 81223 records: #RECIN would be 143, #RECOUT would be 81223, #GRPSIZ would be 5, and #GRPMEM would be 2.

Note that both #RECIN and #RECOUT start counting at zero, while #GRPMEM starts counting at 1.

5.3.1.5 %Variables

This type of entity can be used in all contexts in which a field occurrence can be used. A %variable can also be used as a field occurrence number or as either of the limits on a **FOR** ... **FROM** ... **TO** statement.

A %variable consists of a percent sign (%) followed by any combination of the following characters:

- The percent sign (%).
- The underscore (_).
- The uppercase letters (A-Z).
- The decimal digits (Ø-9).

There are no %variable declarations in FUEL. A %variable can hold any of the basic FUEL types (string, fixed, and float). The initial value of any %variable is "MISSING" (same value as a field that doesn't occur in the record). Just like a field occurrence, the type and existence of a %variable can be tested using the **MISSING**, **EXISTS**, **IS FIXED**, or **IS FLOAT** phrases on the **IF** statement.

If a %variable has the MISSING value, it is treated as the null, or zero-length, string in contexts needing a string value, or as zero in contexts needing a numeric value. For example, this means that a statement such as

```
%COUNT = %COUNT + 1
```

can be placed in your FUEL program **without** requiring an initialization statement such as

```
%COUNT = Ø
```

(Treating the MISSING value as zero in numeric contexts is new in version 4.0 of *Fast/Unload*; prior to that, a MISSING value in numeric context would terminate the job.)

Every %variable is left unchanged until your program resets it. Therefore, %variables can be used for counters, totals, maximums, etc., and they can be used on subsequent unloaded records and/or at the end of the unload of a file or at the end of the unload job.

The value of a %variable is changed either by placing it before the equals sign (=) on the assignment statement or by placing it as an output argument to a #function.

As of *Fast/Unload* version 4.3, the value of a %variable may be a string longer than 255 bytes. Using such a %variable, however, is limited to the contexts described in "Permitted use of long string values" on page 167.

This is an example, somewhat contrived, of the use of a %variable:

```
* Minimum value of repeating string field:
FOR EACH RECORD
   %MIN = FLD
   FOR I FROM 2 TO FLD(#)
      IF FLD(I) < %MIN THEN
         %MIN = FLD(I)
      END IF
   END FOR
   PUT %MIN
   OUTPUT
END FOR
```

5.3.2 Expressions

The right hand side of the assignment statement and the ADD and CHANGE statements consists of an expression which denotes a value to be stored in a %variable or a field. This element of a FUEL program, the **expression**, has the following syntax:

```
   #function ( [[-]entity] ,...)
|  entity
|  [-] entity {+ | - | * | /  [-] entity} ...
```

The first form of expression is a #function call; the value of the expression is the result returned by the #function. See "#Function calls" on page 33 for an explanation of the #function call.

The second form of expression is simply an entity; the value of the expression is the value of the entity. See "Entities" on page 26 for a discussion of entities.

The third form of expression is an arithmetic calculation; the value of the expression is the floating point value obtained from the indicated entities and operands, using the rules of arithmetic. All entities in the expression must contain a numeric value. If any does

not, the FUEL program will end. If an overflow or underflow error occurs, the FUEL program will end, with an error message indicating the type of error and the line number being executed.

Parentheses are not allowed within an arithmetic expression in this version of *Fast/Unload*. An arithmetic expression is evaluated from left to right, with multiplication and division having the same precedence, which is greater than that of addition and subtraction; addition and subtraction have the same precedence.

A %variable that contains a string longer than 255 bytes is not allowed in an arithmetic expression.

Note that the compiler will combine constants in most cases where possible; if an underflow or overflow occurs while combining constants, the program will immediately end during compilation, with a generic message that *Fast/Unload* has abended. The last line printed will contain the cause of the error.

See "Floating Point Arithmetic and Numeric Conversion" on page 235 for a discussion of the algorithms involved in floating point arithmetic calculations.

5.3.3 #Function calls

A #function call is an expression that executes a certain algorithm, determined by the #function name, using values specified by a list of arguments, and returns a single value, called the #function result. The syntax of a #function call is:

```
funcname ( [[-] entity] [,...] )
```

The *funcname* consists of a number sign (#) followed by any combination of the following characters:

- The number sign (#).
- The underscore (_).
- The uppercase letters (A-Z).
- The decimal digits (0-9).

The arguments to a #function are specified by position; each one can be omitted if the argument is optional. If an argument is supplied it is one of the forms of FUEL entities.

The algorithm is specified for each individual #function. A set of standard #functions is included with *Fast/Unload*; they are described in "Standard #Functions" on page 99. In addition, your site can use any customer-written #functions you have written; see the FUNCTIONS statement and "Customer-written Assembler #Function Packages" on page 213.

Each #function has a different form, in terms of the number and type of arguments. The maximum number of commas that can be specified on a #function call is 30 (maximum

number of arguments is 31). The types of the arguments, and some other items important in specifying how to use a #function, are as follows:

required argument

If an argument to a #function is required, then you must code an entity for it in the call. If you do not code an entity, a compilation error message will be issued and the unload will not be performed.

optional argument

If an argument to a #function is optional, you need not code an entity for it in the call. If you provide any arguments following it, then a comma must be used to indicate the missing argument; for example:

```
%P = #VERPOS(%S, %C, , 3)
```

The third argument, which is optional, has not been coded above.

omitted argument

A particular #function may treat an omitted argument differently than a supplied argument which has the MISSING value or which has the null string value.

input argument

If an argument to a #function is an input argument, you can specify any form of entity for the argument. The #function can not change the value of an input argument.

argument with MISSING value

If an argument has the MISSING value (a %variable or a field), the #function algorithm can detect this, although, except as noted, for all arguments of the standard #functions this is the same as passing the null string for a string argument, or as passing zero for a numeric argument.

Since the numeric use of MISSING is zero, the MISSING value is not allowed for an argument which may not be zero. Also, the MISSING value is not allowed for some other #function arguments, such as the CENTSPAN argument and the first argument in the #Nx2DATE family.

#function result

The value of an expression that is a #function call is called the #function result. See the evaluation process below for an explanation of how this is set.

output argument

In addition to the result, a #function can set additional values. This is provided by output arguments. If you supply an output argument, you must supply a %variable (leading minus sign is not allowed); if you supply an argument which is not a %variable, a compilation error message is issued and the unload will not execute.

See the evaluation process below for an explanation of how the value of an output argument is modified.

The process of evaluation of a #function call uses 'call by copy/result' semantics, that is:

1. The value of each argument entity is copied to a distinct intermediate variable which the #function can access (this is true of both input and output arguments in the current version, but in terms of the details of #function evaluation, what is important is that output arguments are copied).

2. The value of the result is set to MISSING.

3. During the execution of the #function, the argument values (input or output) can be accessed and output arguments can be modified, in the intermediate variables. The result can be modified or accessed by the #function; this is done directly in the %variable which the #function result is assigned to.

4. After termination of the #function, the value of each intermediate variable corresponding to an output argument is copied to the %variable coded as the output argument.

Since #function calls are pervasive in advanced FUEL programs, there are many examples of their use throughout this manual. Here is an example which converts a field stored as IBM packed decimal into a numeric format usable by User Language:

```
UAI
FOR EACH RECORD
   %V = Ø
   %X = #C2X(SALARY)
   %L = #LEN(%X)
   %L = %L - 1
   FOR I FROM 1 TO %L
      %C = #SUBSTR(%X, I, 1)
      %T = #INDEX('Ø123456789', %C)
      %T = %T - 1
      %V = %V * 1Ø + %T
   END FOR
   CHANGE SALARY = %V
   UNLOAD
END FOR
```

5.3.4 Assignment statement

The assignment statement is the only FUEL statement that does not begin with a keyword. The syntax of the assignment statement is:

```
%variable = expr
```

This statement changes the value of *%variable* to be the value of the **expression** *expr*. See “Expressions” on page 32 for the forms of legal FUEL expressions.

If the value of *expr* is the MISSING value that becomes the value of *%variable*. A missing field occurrence, an un-assigned %variable, and various #function results are different sources of the MISSING value.

Since the assignment statement is pervasive in advanced FUEL programs, there are many examples of its use throughout this manual. Here is a simple example:

```
FOR EACH RECORD
   IF ALERT NE 1 THEN
      SKIP
   END IF
   IF REC.TYPE = 'MGR' THEN
      %TITLE = 'Vice President '
      %SUFF = #CONCAT( '(', DEPT, ' Department', ')' )
   ELSE
      %TITLE = ''
      %SUFF = #CONCAT('hired on ', HIREDT)
   END IF
   REPORT 'Alert' AND %TITLE AND NAME AND %SUFF
END FOR
```

5.4 #ELSE

The #ELSE statement begins an always true section within a #IF block (see “#IF” on page 37 for a full explanation of #IF blocks). A #IF block can have zero or one #ELSE section.

5.5 #ELSEIF

The #ELSEIF statement has the form

```
#ELSEIF fieldname { DEFINED | UNDEFINED }
```

An #IF block may have zero or more #ELSEIF sections. If the #IF block has not had a prior true #IF or #ELSEIF section, then the next #ELSEIF condition is tested to see if the dependent lines should be compiled. See “#IF” on page 37 for a full explanation of #IF blocks.

5.6 #END IF

The #END IF statement ends a #IF block (see "#IF" for a full explanation of #IF blocks).

5.7 #IF

The #IF statement begins an #IF block and has the form

```
#IF fieldname { DEFINED | UNDEFINED }
```

A #IF block may have zero or more #ELSEIF sections (see "#ELSEIF" on page 36) and zero or one #ELSE section (see "#ELSE" on page 36), and it is ended by a #END IF statement (see "#END IF").

The logic of #IF blocks is just like that of IF blocks, except that they affect which FUEL program statements are compiled, rather than the program flow at execution time.

A #IF or #ELSEIF section is true if

- the specified *fieldname* is defined and the DEFINED condition is specified
- the specified *fieldname* is *not* defined and the UNDEFINED condition is specified

fieldname may be any type of *Model 204* field (including BLOB or CLOB, as of *Fast/Unload* 4.3).

The FUEL statements within the first true section are compiled; all others are skipped. If no true sections have been found when an #ELSE section is encountered, the #ELSE section is true by definition. Once a true section has been processed, all FUEL statements are skipped until the #END IF is encountered.

#IF blocks may not be nested.

The FUEL program below can be used as is with input files that have a telephone number field named `TELNO`, with other input files in which the telephone number is in a field named `TELEPHONE`, and with still other files that have no telephone number field at all.

```
UAI
 FOR EACH RECORD
 #IF TELNO DEFINED
  %T = TELNO
 #ELSEIF TELEPHONE DEFINED
  %T = TELEPHONE
 #ELSE
  %T='XXX-XXX-XXXX'
 #END IF
  %L = #LEN(%T) - 3
  IF %L > Ø THEN
   ADD TEL_KEY = #SUBSTR(%T, %L)
  ELSE IF %L > -3 THEN
   REPORT 'Bad TELNO' AND %T AND 'record' AND #RECIN
  END IF
 UNLOAD
END FOR
```

5.8 ADD[C] field = expr

This statement adds an occurrence of the field *field* to the current record. The value of the added occurrence is the value of the **expression** *expr*. See "Expressions" on page 32 for the forms of legal FUEL expressions.

For the ADD statement, *expr* may not have the MISSING value. The ADDC statement allows the MISSING value, and in that case no occurrence is added for the field. Otherwise the statements are the same.

The added occurrence can be referenced as an entity, and it will be output by the UNLOAD or PAI statements.

The following example produces a value for an INVISIBLE KEY field by using the last 4 digits of the telephone number:

```
UAI
FOR EACH RECORD
   %L = #LEN(TELNO) - 3
   IF %L > Ø THEN
      ADD TEL_KEY = #SUBSTR(TELNO, %L)
   ELSE IF %L > -3 THEN
      REPORT 'Bad TELNO' AND TELNO AND 'record' AND #RECIN
   END IF
   UNLOAD
END FOR
```

Notes:

- If you are using ADD with a UAI type of unload, be sure to code the UNLOAD statement.
- If any ADD[C] *field* statements are in the program, then no ordered index information is unloaded for *field*.
- Field and date statistics are generated using the values of field occurrences before any ADD statements are executed.
- For PAI, any ADDed occurrences are placed at the end of the output for the record, in the order in which the ADD statements were executed.
- The order of output for the normal UNLOAD statement is the same as for PAI, unless the field in question is the HASH field or the first SORT field on the UAI statement. In that case, the occurrence designated will be output first in the record, whether or not it is an ADDed occurrence.
- The definition of the field is ignored by the ADD statement: the behaviour for any ADDed occurrence is as if the field were defined as FLOAT LENGTH 8 if a float value is assigned, and defined as STRING otherwise.
- As described in "Permitted use of long string values" on page 167, as of *Fast/Unload* 4.3, the *field* in an ADD[C] statement may be a BLOB or CLOB; the *expr* may be a %variable that contains a string longer than 255 bytes.
- The check for `%L > 0` above is necessary if there is any chance of it being false, since #SUBSTR requires that the second argument (`%L`) be a number greater than or equal to one.

The ADDC field statement is new in *Fast/Unload* version 4.0.

5.9 ADDC field = expr

The ADDC statement is the same as the ADD statement, except that ADDC allows the assigned ***expr*** to be the MISSING value.

See "ADD[C] field = expr" on page 38.

The ADDC field statement is new in *Fast/Unload* version 4.0.

5.10 CANCEL [ccode]

This statement terminates the unload. The cancel statement can be followed by an optional fixed constant which indicates an optional condition code to be returned for the step under MVS or return code for the FUNLOAD command under CMS. For example, the statement

```
CANCEL 22
```

would terminate the unload with a condition code (or return code) of 22.

The CANCEL statement would typically be found inside a conditional clause, whose truth indicates a severe error. For a UAI OINDEX or INVISIBLE type of unload, this statement will prevent the output of an indexing data. For a PUT type of unload, any data that has been 'PUT' into the current output record is lost, unless an OUTPUT statement preceded the CANCEL statement. For example, in the program

```
OPEN BIGFILE
FOR EACH RECORD
  PUT KEY.FIELD AS STRING(7)
  IF #ERROR NE Ø THEN
    REPORT 'SEVERE ERROR IN RECORD' AND #RECIN
    CANCEL
  END IF
  OUTPUT
END FOR
```

if there is an error placing KEY.FIELD into the output record; for example if it's missing, then the unload is terminated.

5.11 CHANGE field [(occurrence)] = expr

This statement changes the value of the designated *occurrence* of the field *field* in the current record. The value of the changed occurrence is the value of the **expression** *expr*. See “Expressions” on page 32 for the forms of legal FUEL expressions.

Expr may not have the MISSING value.

Occurrence defaults to 1. As with any occurrence number, the value of *occurrence* must be numeric and greater than or equal to 1 (fractional values are ignored). If this is not true, *Fast/Unload* is cancelled.

The occurrence of the field must exist.

The changed occurrence can be referenced as an entity, and it will be output by the UNLOAD or PAI statements.

The following example removes leading blanks from a field:

```
UAI
FOR EACH RECORD
   %P = #VERPOS(ADDRESS, ' ')
   IF %P > 1 THEN
      CHANGE ADDRESS = #SUBSTR(ADDRESS, %P)
   END IF
   UNLOAD
END FOR
```

Notes:

- If you are using CHANGE with a UAI type of unload, be sure to code the UNLOAD statement.
- If a field is modified by CHANGE, then no ordered index information is unloaded for that field.
- Field and date statistics are generated using the values of field occurrences before any CHANGE statements are executed.
- The CHANGE statement does not affect the order in which fields are output for the UNLOAD or PAI statements.
- The definition of the field is ignored by the CHANGE statement: the behaviour for any CHANGEd occurrence is as if the field were defined as FLOAT LENGTH 8 if a float value is assigned, and defined as STRING otherwise.
- As described in “Permitted use of long string values” on page 167, as of *Fast/Unload* 4.3, the *field* in an CHANGE statement may be a BLOB or CLOB; the *expr* may be a %variable that contains a string longer than 255 bytes.

5.12 CHECK condition ... CANCEL | WARN | ALLOW

This statement allows you to specify the conditions to be checked before unloading the file(s), so that you do not inadvertently unload a file that may need some corrective action. You can enter this statement multiple times; it must occur before any FOR EACH RECORD statement. You can use this statement to override the conditions checked by default.

The following CHECK statement ensures that the unloaded file does not have any procedures nor any INVISIBLE non-ORDERED field definitions by canceling the unload if any are detected:

```
OPEN BIGFILE
CHECK PROCS INVIS CANCEL
UAI OINDEX
```

The *condition* list in the CHECK statement can contain one or more of the following keywords:

DUPDT Checks if the file is in Deferred Update Mode.

BROKE-LOGIC
Checks if the file is Logically Inconsistent.

BROKE-PHYS
Checks if the file is Physically Inconsistent.

INVIS Checks if the file has any INVISIBLE fields defined that would not be unloaded:

- INVISIBLE non-ORDERED fields (*Fast/Unload* cannot access these).
- INVISIBLE ORDERED fields without the presence of a UAI OINDEX or a UAI INV statement.

Note: If your INVISIBLE fields can be derived, you can create the values in the unload (UAI, PAI, or other) by using the ADD statement in FUEL (see "ADD[C] field = expr" on page 38). This approach should be distinctly faster than adding the values with User Language, although both approaches require building the index, usually with a sort and the *Model 204* Z command.

PROCS For UAI output streams only, checks whether the file contains any procedures (including aliases). For versions prior to 4.2, *Fast/Unload* cannot unload them, and they are lost during a file re-org if you do not otherwise unload them (using perhaps the *Model 204* COPY PROC command, or DISPLAY PROC to a USE dataset) before re-creating the file.

As of *Fast/Unload* 4.2, the UAI statement unloads procedures and aliases by default, or if you explicitly specify the PROCS option of UAI (see "UAI statement options" on page 89).

A CHECK PROCS statement is ignored if the unload contains no UAI statements.

Insert the CHECK PROCS statement before the (first) UAI statement.

The BROKE-LOGIC, DUPDT, and BROKE-PHYS conditions are values represented in the FISTAT file status parameter (http://m204wiki.rocketsoftware.com/index.php/FISTAT_parameter).

The checks performed on *Fast/Unload* depend upon the defaults in effect ("CHECK statement defaults" on page 43) and upon the type of unload performed. Conditions specified in any CHECK statements override the defaults for those conditions, if any.

These messages in your FUNPRINT report inform about the unload checking:

- FUNL0133 shows the conditions checked as a result of the defaults, the type of unload, and any CHECK statements.
- FUNL0131 reports ("Check failed") any non-ALLOWed conditions (CANCEL or WARN) that are found to exist in the file.
- FUNL0132 suggests possible responses to the conditions reported in FUNL0131.

5.12.1 CHECK statement actions

The following actions may be taken if a condition is found to exist in the file to be unloaded. One and only one action must be specified on each CHECK statement.

CANCEL The *Fast/Unload* run will be cancelled before unloading any records.

WARN *Fast/Unload* will proceed with the unload, but at termination will have a completion code of 4 or greater.

ALLOW *Fast/Unload* will proceed with the unload, without affecting the completion code.

5.12.2 CHECK statement defaults

If no CHECK statement is included in your FUEL program, the following default checks are still in effect as long as the NOENQ parameter is **not** specified. If NOENQ ("NOEnq" on page 15) is specified, no CHECK conditions are in effect.

- Whether the file has been initialized.
  Fast/Unload always checks for this, and it cannot be overridden.
- `CHECK BROKE-PHYS BROKE-LOGIC CANCEL`,
  if neither UAI OINDEX nor UAI INV is specified.
- `CHECK BROKE-PHYS BROKE-LOGIC DUPDT CANCEL`,
  if UAI OINDEX or UAI INV is specified.
- `CHECK PROCS ALLOW`,
  for UAI statements only.

For information about customizing the CHECK defaults for your site, see "Default CHECK conditions and actions" on page 293.

5.13 DATESTAT [SUMMARY | DETAIL]

This statement causes *Fast/Unload* to analyze the file for fields which contain date values, and to provide information about those fields.

The analysis is done in two phases; a sampling phase of 1000 records evenly distributed in the file, and an analysis in the second phase of selected fields. A field is examined in the second phase if in the first it is determined to have date values or is not found in the sampled records. The second phase is performed during the normal unload pass of the file. For a detailed description of the analysis of date fields, see "DATESTAT Analysis" on page 189.

The reporting of information about date fields is done at the end of the processing of the file; field statistics (if FSTATS processing is performed) are reported before date information. You can choose a brief report with 1 to 3 lines per date field, or a comprehensive report with 1 page per date field. The brief report is the default, or it can be specified with the SUMMARY option of DATESTAT. The comprehensive report can be specified with the DETAIL option of DATESTAT. In both cases, the report shows each field's most common date format, and, if the field contains any 2-digit ("YY") years, an estimate of a 100-year span containing all the 2-digit year dates. In addition, an indication is given of the level of quality of the data stored in the field. Date values which may conform to multiple formats (for example, MM/DD/YY vs. DD/MM/YY) are accounted for.

For a detailed description of the DATESTAT reports, see "DATESTAT Reporting" on page 190.

The following *Fast/Unload* program will generate date statistics without creating a FUNOUT file:

```
OPEN filename
DATESTAT type
FOR EACH RECORD
END FOR
```

The field values reported by DATESTAT are the values before any changes by the ADD, CHANGE, or DELETE statements.

5.14 DELETE[C] field [(occurrence)]

This statement deletes the designated *occurrence* of the field *field* from the current record.

Occurrence defaults to 1. As with any occurrence number, the value of *occurrence* must be numeric and greater than or equal to 1 (fractional values are ignored). If this is not true, *Fast/Unload* is cancelled.

For the DELETE statement, the occurrence of the field must exist. For the DELETEC statement, the occurrence need not exist, and in that case no occurrence is deleted. Otherwise the statements are the same.

The following example allows you to redefine a field as AT-MOST-ONE by moving multiple occurrences to a new field:

```
NEW SEC_ADDR
UAI
FOR EACH RECORD
   FOR I FROM 2 TO ADDRESS(#)
      /* Must do ADD before DELETE
      ADD SEC_ADDR = ADDRESS(2)
      /* Make ADDRESS(2)=ADDRESS(3), etc.
      DELETE ADDRESS(2)
   END FOR
   UNLOAD
END FOR
```

Notes:

- The DELETE statement causes the field occurrences to be "shifted down by one". That is, after DELETEing the *I*th occurrence of a field, the value of the *J*th occurrence becomes the value of the former *J*+1st occurrence, for all *J* greater than or equal to *I*.

  Therefore, the order of ADD and DELETE in the example above is necessary,

- If you are using DELETE with a UAI type of unload, be sure to code the UNLOAD statement.

- If a field is modified by DELETE, then no ordered index information is unloaded for that field.

- Field and date statistics are generated using the values of field occurrences before any DELETE statements are executed.

- For PAI and UNLOAD, any DELETEd occurrences are simply removed from the output.

Note that if you are using UAI to unload a file and you want to simply remove all occurrences of a field, and to remove the field definition from the file, the best way to accomplish this is not by using the DELETE statement in FUEL. You should do a "normal" UAI, unloading the entire file, and when you reload the file, use the following statement:

```
LAI NOFDEF DELFIELD
```

This will require you to insert field definitions for all fields in the file after the INITIALIZE command; omitting the DEFINE FIELD for the fields you want to delete will accomplish your objective.

5.15 DELETEC field[(occurrence)]

The DELETEC statement is the same as the DELETE statement, except that DELETEC allows you to specify a field and occurrence which may be missing on the current record. In that case, nothing is deleted for that statement.

See “DELETE[C] field [(occurrence)]” on page 44.

The DELETEC field statement is new in *Fast/Unload* version 4.0.

5.16 ELSE

This statement marks the beginning of statements to be executed when the all the previous associated IF and ELSEIF clauses are false. For example, in the program

```
OPEN BIGFILE
FOR EACH RECORD
  IF +FIELD1 > FIELD2
    PUT FIELD1 AS FLOAT(4)
  ELSE
    PUT FIELD2 AS FLOAT(4)
  END IF
  OUTPUT
END FOR
```

the greater of the first occurrence of FIELD1 and FIELD2 would be placed into the output record.

5.17 ELSEIF cond [THEN]

This statement is executed if all previous associated IF and ELSEIF clauses have proved to be false; if *cond* is true, then the group of statements following the ELSEIF statement, up to the matching ELSE or END IF statement, are executed.

See “IF cond [THEN]” on page 57 for a description of *cond*. For example, in the program

```
OPEN BIGFILE
FOR EACH RECORD
  IF FIELD1 < 'D'
    PUT '1'
  ELSEIF FIELD1 < 'P'
    PUT '2'
  ELSE
    PUT '3'
  END IF
  OUTPUT
END FOR
```

a '2' is placed in the output record if FIELD1 is not less than 'D' but is less than 'P'.

5.18 END FOR

This statement terminates a FOR EACH RECORD or FOR/FROM/TO clause. The program

```
OPEN BIGFILE
FOR EACH RECORD
  PUT WELL.NUMBER AS FIXED(4)
  FOR I FROM 1 TO 100
    PUT DEPTH(I) AS FIXED(4)
    PUT MEASURE(I) AS FLOAT(4)
  END FOR
  OUTPUT
END FOR
```

demonstrates both uses of the END FOR statement.

5.19 END IF

This statement terminates an IF clause and all subsequent ELSEIF and ELSE clauses.

See “IF cond [THEN]” on page 57 for an example of END IF.

5.20 END REPEAT

This statement terminates a REPEAT clause.

See “REPEAT” on page 78 for an example of END REPEAT.

The END REPEAT statement is new in *Fast/Unload* version 4.0.

5.21 END SELECT

This statement marks the end of a SELECT clause and the last WHEN or OTHERWISE sub-clause. For example, in the following program ID would be placed in the output record regardless of the value of REC.TYPE, because the 'PUT ID' statement occurs after the END SELECT statement:

```
OPEN BIGFILE
FOR EACH RECORD
  SELECT REC.TYPE
  WHEN 1
    PUT 'PHYSICIAN' AT 1
  WHEN 2
    PUT 'PATIENT' AT 1
  END SELECT
  PUT ID AT 10 AS STRING(9)
  OUTPUT
END FOR
```

In this example, if REC.TYPE is equal to 3, columns 1 through 9 would be left blank in the output record.

5.22 FOR v FROM begin TO end

This statement marks the beginning of a clause that is executed once for each value of loop control variable **v** as specified by the range **begin TO end**. **Begin** can either be a fixed constant, greater than 0, or a %variable, which must contain a numeric value greater than or equal to 1 (fractional values are ignored), and in the range of the fixed values. **End** can either be a fixed non-negative constant, the count of a field occurrence, for example, ADDRESS(#), or a %variable, which must contain a numeric value greater than or equal to 0 (fractional values are ignored), and in the range of the fixed values. If both **begin** and **end** are constants, **begin** must be less than or equal to **end**.

The **end** value is evaluated once, before the first iteration of the loop. Therefore, the following example will be performed for two iterations:

```
%V = 2
FOR I FROM 1 TO %V
   %V = %V + 1
END FOR
```

Some valid FOR v statements are

- FOR A FROM 1 TO 10
- FOR I FROM 1 TO CHILD(#)
- FOR I FROM 2 TO %NUM
- FOR I FROM %V1 TO CHILD(#)

This statement must always be paired with an END FOR statement. The loop control variable can be referred to inside the loop either as an entity or as an occurrence number for a field. Note that a loop control variable can only be a single alphabetic character. Nested FOR loops cannot use the same loop control variable. Non-nested FOR loops can use the same loop control variable.

A LEAVE FOR statement may be placed within a FOR loop; executing the LEAVE FOR will terminate the innermost FOR loop containing the LEAVE FOR. See “LEAVE clause_type” on page 61 for a description of LEAVE and for an example of the use of LEAVE FOR.

Within a FOR loop, if a field name is the same as the loop control variable, you must use quotes to refer to the field, for example **PUT 'F'**. The program

```
OPEN BIGFILE
FOR EACH RECORD
  PUT FIELD1 AT 1 AS STRING(10)
  FOR I FROM 1 TO FIELD2(#)
    PUT FIELD2(I) AS STRING(10)
    PUT FIELD3(I) AS STING(10)
  END FOR
  OUTPUT
END FOR
```

is an example of the use of the FOR statement.

5.23 FOR EACH RECORD

This statement must occur exactly once in the FUEL program and must be associated with exactly one END FOR statement. Statements inside the FOR EACH RECORD/END FOR bracket are executed once for each input record.

5.24 FSTATS [AVGTOT | MINMAX]

The FSTATS directive will gather field, Table B, and procedure statistics and check file integrity during the run. If this option is selected, the *Fast/Unload* report will contain a list of all defined fields in the database file, with field definition information and statistics about occurrences of the fields. It will also perform various integrity checks, and provide statistics about Table B and the file's procedures.

Each field is displayed with the first 50 bytes of its name, and the following information is available:

- the field's storage type (STRING, FLOAT, CODED, etc., along with an INVISIBLE indicator)
- all non-default field definition information, if there is any other than storage type
- maximum and minimum for the field's occurrences and lengths
- average and total for the field's occurrences and lengths
- total Table E pages used, if a BLOB or CLOB field
- counts of records missing the field, if there are any

To restrict the FSTATS field display to only contain the storage type and the minimum and maximum of occurrences and lengths, you can use the FSTATS MINMAX directive; FSTATS AVGTOT requests the more complete field information.

If you specify the FSTATS directive in your FUEL program without a qualifying MINMAX or AVGTOT, the type of processing is determined by the FSTATS program parameter; if there is no FSTATS=MINMAX nor FSTATS=AVGTOT program parameter, then the default processing for the FSTATS directive is AVGTOT. You can change this default to be MINMAX by a customization zap (see “Setting default FSTATS processing” on page 295).

In addition to listing summary information about fields and Table B, FSTATS processing will cause checking of some possible inconsistencies of field definition information, and a thorough check of the integrity of the Table B records on the file. The following checks are made for Table B consistency:

1. "Trailing" non-null preallocated fields.

2. Invalid field types (neither float, string, nor binary/coded).

3. Last field of record on a page longer than space allocated to record (this will be done whether doing FSTATs or not).

4. Unknown coded value for CODED field.

5. Coded value stored for non-CODED field.

6. Binary value stored for non-BINARY field.

7. Float value stored for non-FLOAT field.

8. Field value indexed but field not an indexed type, or vice-versa.

9. Preallocated field stored beyond preallocated field block.

If you specify **FSTATS**, the following *Fast/Unload* program will generate field statistics without creating a FUNOUT file:

```
OPEN filename
FOR EACH RECORD
END FOR
```

The field values reported by FSTATS are the values before any changes by the ADD, CHANGE, or DELETE statements.

The FSTATS program parameter can be used instead of the FSTATS directive, although the parameter does not allow you to specify AVGTOT or MINMAX. See "FStats[=AVGTOT|MINMAX]" on page 10.

FSTATS is not valid if the Field Statistics Option is not linked with your Fast Unload load module.

This directive is new in *Fast/Unload* version 4.0, as is the additional information that is provided with FSTATS AVGTOT.

5.24.1 Description of Table B statistics

In addition to the field information, FSTATS processing produces information about Table B utilization of the file. These are produced with any type of FSTATS processing. The Table B information is as follows:

Table B pages in use and Base records in file
These are taken from the file parameters.

Base records processed

This is the same value that is shown the FUNL0054 message. It is the number of input records processed, which is based on the size of the file (or found set, when using the *Fast/Unload User Language Interface*), up until processing stops, which may be due to the FUEL CANCEL statement. In "normal" processing (that is, no CANCEL statement, and not using the *Fast/Unload User Language Interface*), this should be the same as the number shown for Base records in file.

Base records processed without extensions

This is the number of base records processed which do not have any extension record.

Extension records in file

This is taken from the file parameters.

Extension records processed

This is the total number of extension records of the base records processed.

Average extensions per extended record

This is the average number of extension records among the base records processed that have extensions, that is:

```
Extension records processed
   divided by
(Base records processed -
Base records processed without extensions)
```

Minimum record length

This is the length of the smallest record processed.

Maximum record length

This is the length of the largest record processed.

Total length of records

This is the total length of all records processed.

Average record length

This is the average of the lengths of all records processed. It is calculated as:

```
Total length of records
   divided by
Base records processed
```

Standard deviation of record length

This is the deviation from the average of the lengths of all records processed.

Note that the **record length** statistics are based on the “internal” lengths of the base record and all its extensions. It includes all space used by the record in Table B, except for any extension record pointers, pointers to records on the page, and spill pointers. This approach is used because the general purpose of the record length statistics is to provide information about the Table B space that is **required** to hold the data, specifically to allow you to size a file so that it will achieve an optimum layout after a reorganization. After a reorganization, you can control the number of extensions in the file to depend only on the record length, but before a reorganization, the number of extensions, and the extra Table B space used for each extension's “overhead”, is dependent on the random availability of Table B space on the pages being updated.

The FSTATS Table B information is only available with version 4.0 and later of *Fast/Unload*.

5.24.2 Description of field statistics

The field by field listing contains one line for the first 50 bytes of each field's name, and the following information is also available:

- the field's storage type (NEW, or STRING, FLOAT, CODED, etc., along with an INVISIBLE indicator)
- all non-default field definition information, if there is any other than storage type
- maximum and minimum for the field's occurrences and lengths
- average and total for the field's occurrences and lengths
- counts of records missing the field, if there are any

All of this information is produced with FSTATS AVGTOT processing; for FSTATS MINMAX processing, only the storage type is presented, or, for INVISIBLE fields, as much as will fit on the single line of field information.

For the most part, field definition information is not displayed for those parts of the definition that use the *Model 204* field definition defaults. Information that is default (for example, UPDATE IN PLACE) is not presented. The field definition information is displayed using the follow abbreviations:

NEW A field introduced with the NEW directive. No other information is provided for the field in the field statistics report.

BINARY A BINARY field.

STRING A non-BINARY, non-FLOAT, non-DBCS field.

FLOAT*n* A FLOAT field, of length *n*

CODED A CODED MANY-VALUED field.

CODFEW A CODED FEW-VALUED field.

PURE DBCS
A STRING DBCS field.

MIXED DBCS
A STRING MIXED DBCS field.

INVISIBLE
An INVISIBLE field. Of course, no lengths, counts, averages, etc., are displayed for INVISIBLE fields. Also, most of the field definition for INVISIBLE fields is printed on the first line of the field statistics report, so it is printed even if FSTATS MINMAX processing is in effect. A special informational label is printed, **(Derived for NR)**, if the field on the statistics display is a field that is automatically defined by *Model 204* to contain index information for a NUMERIC RANGE field.

NR A NUMERIC RANGE field.

KEY A KEY (Table C indexed) field.

OCC*nn***/PAD=X***pp***/LEN***jj*
A fixed OCCURS (preallocated) field. *Nn* indicates the number of occurrences, *pp* is the hexadecimal representation of the PAD character, and *jj* is the length.

OCCONE/PAD=X*pp***/LEN***jj*
A fixed OCCURS (preallocated) field which is also defined to be AT-MOST-ONE. *Pp* is the hexadecimal representation of the PAD character, and *jj* is the length.

ONE An AT-MOST-ONE field.

FRV An FRV (“FOR-EACH-VALUE”) field which is also either CODED or MANY-VALUED.

FRVFEW An FRV (“FOR-EACH-VALUE”) field which is FEW-VALUED and not CODED.

NDEF A NON-DEFERRABLE field.

UPEND An UPDATE-AT-END, non-INVISIBLE field.

UNQ A UNIQUE field.

LVL*sec* A secured field, with security level *sec*.

ORDCH LRSV=*lr* **NRSV=***nr* **SPLT=***sp* **IMM=***im*

An ORDERED CHARACTER field, where:

- *lr* is the value of LRESERVE
- *nr* is the value of NRESERVE
- *sp* is the value of SPLITPCT
- *im* is the value of IMMED

In addition to the field definition information, statistics are presented concerning the occurrences of the field. The following statistics are presented about each field:

Minimum and maximum occurrences

The minimum and maximum number of occurrences of the field on a single record. This information is produced with any FSTATS processing.

Minimum and maximum length

The minimum and maximum length of any occurrence of the field. This information is produced with any FSTATS processing. Note that the calculation of field length:

- for non-preallocated fields, does not include the two-byte field code nor (for string fields) the length byte
- is the “string” length of any CODED values and is the stored length of FLOAT or BINARY values

Records with zero occurrences

The count of records which contain zero occurrences of the field is displayed, if there are any such records. This information is produced only with FSTATS AVGTOT processing.

Total and average occurrences

The total number of occurrences of the field is displayed, along with the average occurrences per record, among the records that have at least on occurrence of the field. This information is produced only with FSTATS AVGTOT processing.

Total and average length

The total length of all occurrences of the field is displayed, along with the average length. The average length is simply the total length divided by the number of occurrences of the field. Note that the calculation of field length:

- for non-preallocated fields, does not include the two-byte field code nor (for string fields) the length byte
- is the “string” length of any CODED values and is the stored length of FLOAT or BINARY values

This information is produced only with FSTATS AVGTOT processing.

The additional information that is provided with FSTATS AVGTOT is only available starting with *Fast/Unload* version 4.0.

5.24.3 Description of procedure statistics

In addition to the field and Table B information, FSTATS processing produces information about the file's procedures, largely from the procedure dictionary.

The statistics are produced with any type of FSTATS processing, although those that pertain to procedure text quantity are produced only if the following is true: at least one UAI output stream is specified explicitly or implicitly to unload procedures (that is, UAI PROCS is specified, or UAI NOPROCS is **not** specified).

If FSTATS is not specified, the statistics are still produced if at least one UAI stream is specified explicitly or implicitly to unload procedures.

The procedure information is as follows:

Chunks in PD
Blocks of contiguous pages that comprise the procedure dictionary.

Pages per chunk
Number of pages in each procedure dictionary chunk.

Total pages in PD
Total number of pages in the unloaded procedure dictionary.

Hash cells per page
Number of entries (for individual procedure or alias information) per procedure dictionary page.

Total number of cells
Total number of cells in the unloaded procedure dictionary.

Total number of procs
Total number of procedure cells in the unloaded procedure dictionary.

Total number of aliases
Total number of alias cells in the unloaded procedure dictionary.

Average length of proc/alias names
Average length of the procedure and alias names in the procedure dictionary.

Total text pages
Number of Table D pages used to store the text of procedures.

Average proc length in bytes
Average length in bytes of the text of a procedure.

Average proc length in pages
Average length in pages of the text of a procedure.

5.25 FUNCTIONS [IN *|DDname] member member ...

This statement is used to inform *Fast/Unload* where customer-written assembler language #functions are located. Each *member* is a #function package in the load module library referenced on the *DDname* DD statement. If the **IN** is missing or **IN *** is specified, the STEPLIB and JOBLIB are searched.

IN *DDname* is not allowed under CMS; **IN *** can be specified; it is the default and means to examine all accessed minidisks for files named *member* **TEXT**.

A #function package is a module which contains a set of #functions. When a #function call occurs in a FUEL program, the standard set of FUEL #functions is searched first; if the #function name is not in the standard set, customer-written #function packages are searched in the order specified in all FUNCTIONS statements, first-come first-searched. The total number of #function packages may not exceed 10.

The **FUNCTIONS** statement can be repeated several times, as desired, as long as the number of package names (*member*) totalled from all **FUNCTIONS** statements does not exceed 10.

See “Customer-written Assembler #Function Packages” on page 213 for information about creating a #function package.

5.26 IF cond [THEN]

This statement indicates that all the statements between the IF statement and the corresponding END IF, ELSEIF or ELSE statement are to be executed if cond is true. **cond** can be one or more comparisons joined by logical operators.

A comparison consists of two *Fast/Unload* entities separated by a comparison operator. The comparison operator can be one of the following:

'<' or 'LT'
'>' or 'GT'
'=' or 'EQ'
'¬=' or 'NE'
'>=', '=>', or 'GE'
'<=', '=<', or 'LE'

If the comparison contains a constant, the type of the comparison will be made on the basis of the constant type. For example, a comparison with a fixed constant will result in a fixed comparison.

When comparing two non-constant entities, for example a field and a %variable, the comparison is always a string comparison unless forced to a floating point comparison by preceding one of the entities with a plus sign (+) or to a fixed comparison by preceding one of the entities with a dollar sign ($).

For example, the following is a string comparison:

```
FIELD1 > %VAR
```

The following is a floating point comparison:

```
+FIELD1 > FIELD2
```

The following is a fixed comparison:

```
FIELD1 > $%VAR
```

Any entity that cannot be converted to the required comparison type is assumed to be zero for the purposes of a numeric comparison, or to be the null string (zero length string) for the purposes of a string comparison. If a comparison type is forced and one of the entities is a constant, it must be of the same type as the forced type of the comparison.

Therefore, the following comparison is **illegal**:

```
+FIELD1 > 0
```

The following comparison is legal:

```
+FIELD1 > 0.0
```

In the following FUEL fragment example:

```
%X = 1
%Y = '1.0'
IF %X EQ %Y THEN
PUT 'string EQ'
ELSEIF +%X EQ %Y THEN
PUT 'float EQ'
END IF
OUTPUT
```

The result is:

```
float EQ
```

Note: The %variable in a comparison may not contain a string longer than 255 bytes (except for use with EXISTS and MISSING phrases, introduced below, which do not reference the value of a %variable). Similarly, BLOB and CLOB fields (of any length) may only be used in comparisons that use EXISTS or MISSING.

5.26.1 Using EXISTS, MISSING, IS FIXED, or IS FLOAT

A comparison can also be an entity name followed by the word `EXISTS` or `MISSING`. These comparisons test for the existence of the entity. This is useful for performing statements based on the existence of an occurrence of a field, or on whether a value has yet to be assigned to a %variable, or on whether the result of a #function assigned to a %variable was the MISSING value.

If occurrence 5 of field FIELD1 exists in the current record, the following is true:

```
FIELD1(5) EXISTS
```

But the following is **false**:

```
FIELD1(5) MISSING
```

An entity followed by the phrase `IS FIXED` or `IS FLOAT` tests for the possibility of converting a value to fixed point or floating point. For example, if the field FIELD1 contains the value `12.5`:

```
FIELD1 IS FLOAT
FIELD1 IS FIXED
```

Both the above are true, since `12.5` can be converted to both a floating point value and a fixed point value (albeit with truncation). If the field FIELD1 contains `9999999999`, the following is true:

```
FIELD1 IS FLOAT
```

But the following is **false**, since the value `9999999999` cannot be represented as a 4-byte binary integer:

```
FIELD1 IS FIXED
```

Note: Prior to *Fast/Unload* version 4.3, you can combine the following in a single test:

- IS FIXED and IS FLOAT type checking
- Forcing of the type in a comparison using a plus sign (`+`) or a dollar sign (`$`), as described on the previous page

However, most of these combinations cause the result to be independent of the value of a field occurrence or %variable. Since it is believed that such FUEL code is more likely

to be a coding error than intended to express a desired result, these constructs are illegal in FUEL in 4.3 and later versions.

5.26.2 Using AND and OR

All comparisons can be joined with AND and OR clauses. The words AND and OR can be used alternately with the ampersand (&) and vertical bar (|) symbols, respectively. *Fast/Unload* does the comparisons from left to right with AND having the same precedence as OR, unless comparisons are grouped with parentheses.

For example, this comparison is true if FIELD1='9', the second occurrence of FIELD2 did not exist, and FIELD3(2)='5':

```
FIELD1 > 12 AND FIELD2(2) EXISTS OR FIELD3(2) < 10
```

But the following is **false** for the same values:

```
FIELD1 > 12 AND ( FIELD2(2) EXISTS OR FIELD3(2) < 10 )
```

Note that this is different than many programming languages, which use AND precedence greater than OR. Continuing with the same values as above, the following statement is **false**:

```
FIELD2(2) MISSING OR FIELD3(2) < 10 AND FIELD1 > 12
```

But the following statement is true:

```
FIELD2(2) MISSING OR (FIELD3(2) < 10 AND FIELD1 > 12)
```

Only the comparisons required to determine the truth of the IF statement are performed. It is thus more efficient to place the more likely of two comparisons first in an OR clause, and to place the less likely first in an AND clause. The following program demonstrates the IF statement:

```
OPEN BIGFILE
FOR EACH RECORD
  IF KEY.FIELD MISSING
    REPORT 'MISSING KEY IN RECORD' AND #RECIN
    SKIP
  END IF
  IF (#RECIN < 5000) OR (KEY.FIELD>'2000000' -
              & KEY.FIELD<'3000000')
    PUT KEY.FIELD AS STRING(7)
    PUT STUFF(*) AS STRING(10)
    OUTPUT
  END IF
END FOR
```

5.27 LEAVE clause_type

This statement “breaks out of” the closest enclosing body of FUEL code as indicated by *clause_type*. *Clause_type* can be any of the following:

FOR LEAVE FOR must be within a FOR v FROM loop; the remaining statements of the loop are skipped and the next FUEL statement executed is the one after the END FOR closest enclosing that loop.

Note that LEAVE FOR cannot be used to terminate a FOR EACH RECORD clause; the SKIP or CANCEL statements can be used for that.

REPEAT LEAVE REPEAT must be within a REPEAT loop; the remaining statements of the loop are skipped and the next FUEL statement executed is the one after the END REPEAT closest enclosing that loop.

SELECT LEAVE SELECT must be within a WHEN or OTHERWISE clause; the remaining statements of the loop are skipped and the next FUEL statement executed is the one after the END SELECT closest enclosing the WHEN or OTHERWISE.

Examples for LEAVE FOR and LEAVE SELECT are shown in the following sections; since LEAVE REPEAT is used in most REPEAT loops, an example for it is shown in “REPEAT” on page 78.

The LEAVE statement is new in *Fast/Unload* version 4.0.

5.27.1 LEAVE FOR example

In this example, LEAVE FOR is used to bypass field occurrences which are in ascending date order, after a cutoff date.

Here is a PAI of a *Model 204* record:

```
NAME = DAVE
TITLE = CANNERY ROW
DUE = 36387
TITLE = SWEET THURSDAY
DUE = 36389
TITLE = THE RED PONY
DUE = 36391
```

When processing the above record, the following FUEL program:

```
OPEN DMEWORK
%D = #DATE('YYYY/MM/DD')
PUT 'Fines on books at $0.40/day as of '
PUT %D
OUTPUT
%D = #DATE2ND(%D, 'YYYY/MM/DD')
%TOTAL = 0

FOR EACH RECORD
%FINE = 0
FOR I FROM 1 TO DUE(#)
   IF DUE(I) >= %D
      LEAVE FOR
   END IF
   %DUE = #ND2DATE(DUE(I), 'MM/DD/YY')
   %LATE = %D - DUE(I)
   %FINE = %FINE +  %LATE * .40
   PUT %DUE
   PUT ' ('
   PUT %LATE
   PUT ' days late): '
   PUT TITLE(I)
   OUTPUT
END FOR
IF %FINE > 0 THEN
   PUT 'Fine: $'
   PUT %FINE AS DECIMAL(6,2)
   PUT ' owed by '
   PUT NAME
   OUTPUT
   %TOTAL = %TOTAL + %FINE
END IF
END FOR

PUT 'The library has receivables of: $'
PUT %TOTAL AS DECIMAL(7,2)
OUTPUT
```

produces the following output:

```
Fines on books at $ 0.40/day as of 1999/08/21
08/17/99 (4 days late): CANNERY ROW
08/19/99 (2 days late): SWEET THURSDAY
Fine: $  2.40 owed by DAVE
The library has receivables of: $   2.40
```

Remember that LEAVE FOR cannot be used to terminate a FOR EACH RECORD clause; the SKIP or CANCEL statements can be used for that.

5.27.2 LEAVE SELECT example

The following example uses LEAVE SELECT:

```
FOR I FROM 1 TO 3
   %X = Ø
   PUT 'SELECT '
   SELECT I
   WHEN 1
      FOR J FROM 1 TO 3
         %X = %X + 1
         PUT %X
         PUT '/'
         IF J = 3
            LEAVE SELECT
         END IF
      END FOR
   WHEN 2
      FOR J FROM 1 TO 3
         %X = %X + 1
         PUT %X
         PUT '/'
         IF J = 2
            LEAVE SELECT
         END IF
      END FOR
   OTHERWISE
      FOR J FROM 1 TO 3
         %X = %X + 1
         PUT %X
         PUT '/'
         IF J = 1
            LEAVE SELECT
         END IF
      END FOR
    END SELECT
    OUTPUT
END FOR
```

The above FUEL fragment produces the following output:

```
SELECT 1/2/3/
SELECT 1/2/
SELECT 1/
```

5.28 MSGCTL [FUNL]n ABDUMP

This statement allows Rocket Software to obtain diagnostic information for certain problems.

When the message numbered *n* is issued, the *Fast/Unload* program will abend and create a diagnostic dump. If Rocket Software requests you to use the MSGCTL statement, be sure to have the appropriate setup (for example, SYSMDUMP in an MVS batch FUNLOAD job) to capture the dump.

The number *n* must be greater than or equal to zero and less than or equal to the largest *Fast/Unload* message number; it can be padded on the left with zeroes. The keyword 'FUNL' is optional, but if present there must not be any space between the letters “FUNL” and the message number.

5.29 NEW fieldname [WITH BLOB | CLOB]

This statement defines a new field name. To create occurrences of the field in the current record, use the ADD statement. The new field name can be referenced just as any other field in the file, and any ADDed occurrences will be produced in the UAI or PAI statements.

For example, the following program creates a new field in the file which contains the current date and time:

```
OPEN DATAFILE
NEW DT_MOD
FOR EACH RECORD
   ADD DT_MOD = #DATE('CYYDDDHHMISSXX')
   PAI
END FOR
```

The NEW statement must occur after the OPEN statement, before the FOR EACH RECORD statement, and before the UAI statement (if one is present).

You must use a WITH BLOB or WITH CLOB clause (introduced in *Fast/Unload* version 4.3) to define a BLOB or CLOB field. This is primarily useful for a UAI type unload, allowing you to create values in the new field that are loaded by LAI as Lob occurrences.

Notes:

- Prior to *Fast/Unload* version 4.3, the new field you define has the *Model 204* default field attributes (FRV, KEY, CODED, UPDATE AT END). As of version 4.3, the default attributes are NFRV, NKEY, NCOD, UPDATE IN PLACE.

  If you don't want the default definition, you can issue a *Model 204* DEFINE FIELD command before the FLOD program, so the field will be loaded with the attributes you specify. Or, you can issue a DEFINE FIELD before you unload the file, which would circumvent the need for a NEW directive.

- If you are using NEW (and ADD) with a UAI type of unload, be sure to code the UNLOAD statement.

- If you are using NEW with a UAI OINDEX unload, the new field will not have an ordered index in the unload output, so it will go through the normal multi-step processing to build an index if you do a Fast/Reload.

5.30 NOUNLOAD [field [(occurrence | *)]]

The NOUNLOAD statement limits the UNLOAD statement ("UNLOAD[C] [field [(occur | *)]]" on page 83): it prevents subsequent unloading (by UNLOAD or UNLOADC statements) of some or all fields to some or all destination output streams.

The NOUNLOAD statement must be coded inside a FOR EACH RECORD loop. It is only valid for an output stream declared with a `UAI TO destination` directive, which is described in "UAI statement options" on page 89.

NOUNLOAD, optionally preceded by a "TO *destination*" clause ("TO [destination | *]" on page 83), has two forms:

- `[TO destination] NOUNLOAD`
  This "blanket" NOUNLOAD marks all field occurrences in the current record so that any subsequent UNLOAD or UNLOADC to the TO clause (or implied default) destination(s) is an error.

- `[TO destination] NOUNLOAD field [(occurrence|*)]`
  This "NOUNLOAD field" form means that from this point on in processing the current record, the specified field occurrence(s) may not be unloaded (with UNLOAD or UNLOADC) to the TO clause (or implied) destination(s), nor may they be unloaded by a subsequent "blanket" UNLOAD.

  Occurrence defaults to the first occurrence of *field*; an asterisk (*) specifies all occurrences of the field in the current record that have not been unloaded.

Note: Unlike the UNLOAD statement, if the specified (or implied) field occurrence(s) are missing in the current record, it is **not** an error.

NOUNLOAD applies to the destination output stream specified in its TO clause prefix (or to the implied output stream, if there is no TO clause). The TO clause may be omitted if there is exactly one output stream, or if the output is to go to the stream declared with the DEFault attribute on an OUT TO directive (see "OUT TO destination" on page 68).

Examples

```
TO DESTA NOUNLOAD COMMENTS(*)
```

If you issue the NOUNLOAD statement above, a subsequent UNLOAD statement like the following is caught as an error (FUNL0154):

```
TO DESTA UNLOAD COMMENTS(2)
```

And no occurrences of COMMENTS are unloaded by the subsequent blanket UNLOAD:

```
TO DESTA UNLOAD
```

The following statements put field FOO on destination DESTA and on no other destination:

```
TO DESTA UNLOAD FOO   /* first UNLOAD of FOO
TO * NOUNLOAD FOO
```

This statement sequence puts FOO on three output streams, but on no more thereafter:

```
TO DESTA UNLOAD FOO
TO DESTB UNLOAD FOO
TO DESTC UNLOAD FOO
TO * NOUNLOAD FOO
```

The following statements prevent a field from appearing on any of the unloaded output streams. To get the same result without using NOUNLOAD, you would have to forgo the blanket unloads and explicitly unload all the other fields:

```
TO * NOUNLOAD SEX   /* SEX not previously unloaded
IF SEX = 'M'
   TO MALES UNLOAD
ELSE
   TO FEMALES UNLOAD
END IF
```

5.31 OPEN datafile

The OPEN statement indicates the internal name of each *Model 204* data file from which data is to be extracted.

Prior to version 4.4, OPEN specifies the only file from which data may be extracted, and the syntax of the OPEN statement is `OPEN filename`, where *filename* is the internal name of the *Model 204* data file.

The internal name of the data file is also used as the DDNAME of the first physical file which makes up the entire logical *Model 204* data file.

As of version 4.4, FUEL programs may specify a group in the OPEN statement, and the OPEN statement is either of these forms:

OPEN FILE filename
: This form indicates that a single file is to be opened, and its internal name is *filename*.

OPEN filename1 [, filename2] ...
: This form, if there is more than one filename in the comma-separated list, indicates that a group of files is to be opened, with DD names *filename1*, *filename2*, and so on.

You can also use the *Fast/Unload User Language Interface* to unload a group; prior to version 4.4, using the *Fast/Unload User Language Interface* was the only way to unload a group.

The OPEN statement **must** be the first statement in any FUEL program.

The following program is an example of the use of the OPEN statement:

```
OPEN SIMPSONS
FOR EACH RECORD
  PUT '*'
  OUTPUT
  PAI
END FOR
```

5.32 OTHERWISE

This statement marks the beginning of a clause that indicates the actions to be performed when a field, or a particular occurrence of a field, specified on the currently active SELECT statement did not match any of the values indicated on WHEN statements.

An OTHERWISE clause is terminated by an END SELECT statement.

The following program demonstrates a use of the OTHERWISE statement:

```
OPEN BIGFILE
FOR EACH RECORD
  SELECT SEX
  WHEN 'MALE'
    PUT 'M'
  WHEN 'FEMALE'
    PUT 'F'
  OTHERWISE
    PUT 'U'
  END SELECT
  OUTPUT
END FOR
```

5.33 OUT TO destination

In versions prior to 4.1, a FUEL program either has a UAI directive or it does not. The presence of this directive determines whether the single permitted output stream is written to with UNLOAD[C] statements or with PUT/OUTPUT/PAI statements. As of version 4.1, which allows multiple output streams, a FUEL program can contain multiple UAI directives as well as multiple directives declaring non-UAI streams.

To declare a non-UAI stream, you provide an `OUT TO destination` directive for each such stream. The *destination* becomes the name of the stream, and it is used by stream-specific output statements (see "TO [destination | *]" on page 83) and special variables (#RECOUT, #OUTLEN, #OUTPOS) to designate their particular stream.

The format of the OUT TO directive is:

```
OUT TO destination [DEFault]
```

where:

- *destination* must be unique across all OUT TO and UAI TO destinations. Each destination requires a dataset definition (JCL statement or FILEDEF), and no two file names in these definitions may refer to the same underlying dataset.

- DEF or DEFAULT designates a stream as the default stream for naked output statements (those not qualified by the "TO destination" prefix). At most one OUT TO directive may be designated the default stream.

  A legacy program, that is, one with no OUT TO directives and that does not use the TO parameter on a UAI directive, has one default stream whose destination is FUNOUT. That stream is a UAI stream if the program has a UAI directive, and it is a non-UAI stream otherwise.

Note: If any directive explicitly declares a destination, then all must. A program cannot have both an `OUT TO destination` directive and a UAI directive that has no TO clause, for example.

OUT TO directives are valid as of version 4.1.

5.34 OUTPUT [FILTER loadmod]

This statement is used to place the current output record into an output data set. The output data set is either:

- On the stream indicated by *destination*, if the OUTPUT statement has a "TO *destination*" prefix ("TO [destination | *]" on page 83)
- On the implied output stream, if there is no "TO *destination*" prefix

The prefix may be omitted if there is exactly one output stream, or if the output is to go to the stream declared with the DEFault attribute on the OUT TO directive (see "OUT TO destination" on page 68).

If no data has been placed into the output record for a stream, the OUTPUT statement is a no-op.

Note: The END of the FOR EACH RECORD loop discards the current OUTPUT record for all output streams. If no OUTPUT statement is specified, any data placed in the output record with PUT statements will be lost.

The OUTPUT statement also has a FILTER option for passing the output record data through a user-written output filter. For more more information about this parameter, see "Using User Exits or Filters" on page 229.

The OUTPUT statement is only valid on an output stream for a non-UAI destination.

The following example is a program that would create two output records for each input record, writing them on the output stream DOH:

```
OPEN SIMPSONS
OUT TO DOH
FOR EACH RECORD
  FOR I FROM 1 TO 10
    TO DOH PUT HOMER(I) AS STRING(20)
  END FOR
  TO DOH OUTPUT
  FOR I FROM 1 TO 10
    TO DOH PUT MARGE(I) AS STRING(20)
  END FOR
  TO DOH OUTPUT
END FOR
```

5.35 PRINT ALL INFORMATION or PAI

The PRINT ALL INFORMATION or PAI statement provides an output format identical to *Model 204*'s like-named statements. That is, each value in a record is placed into a separate output record as a *fieldname = value* pair.

The *fieldname = value* output records go to the output data set on either:

- The stream indicated by *destination*, if the PAI statement has a "TO *destination*" prefix ("TO [destination | *]" on page 83)
- The implied output stream, if there is no "TO *destination*" prefix

The prefix may be omitted if there is exactly one output stream, or if the output is to go to the stream declared with the DEFault attribute on the OUT TO directive (see "OUT TO destination" on page 68).

Note: If any PUT statements precede a PAI statement, make sure they are followed by an OUTPUT statement. If they are not, the first **fieldname = value** pair will be concatenated to the partial output record.

The following program is an example of the use of the PAI statement:

```
OPEN PERSONEL
OUT TO DUMPIT
FOR EACH RECORD
  TO DUMPIT PUT '* '
  TO DUMPIT PUT #RECIN
  TO DUMPIT OUTPUT
  TO DUMPIT PAI
END FOR
```

The PAI statement is only valid on an output stream for a non-UAI destination.

Be careful when coding your selection criteria for PAI statements. After a PAI statement, you may not issue another PAI statement for the same record on the same output stream. If your FUEL program attempts a PAI for the same record on the same output stream, the unload will be terminated.

5.36 PUT

This statement is used to place data into the output record for either:

- The stream indicated by *destination*, if the PUT statement has a "TO *destination*" prefix ("TO [destination | *]" on page 83)
- The implied output stream, if there is no "TO *destination*" prefix

The prefix may be omitted if there is exactly one output stream, or if the output is to go to the stream declared with the DEFault attribute on the OUT TO directive (see "OUT TO destination" on page 68).

The format of the PUT statement is:

```
[TO destination] PUT info AT loc AS format MISSING mvalue  -
                                           ERROR evalue
```

where

destination
must have been declared as an output stream in an OUT TO directive (see "OUT TO destination" on page 68).

info can be one of the following:

- An entity. This results in the value of the entity being placed in the output buffer with the indicated format.
- Any valid *Model 204* fieldname followed by the asterisk (*) symbol in parentheses. This results in each occurrence of the specified field being placed in the output buffer with the indicated format, one after the other.

  Note: As specified in "Permitted use of long string values" on page 167 and "Permitted use of Lobs" on page 168, *info* may not be a %variable that contains a value longer than 255 bytes, or be a BLOB or CLOB field.

loc can be either an absolute position in the output record or a position relative to the current output cursor for the PUT statement's output stream. For example, **AT 25** indicates that data is to be placed at the 25th byte in the output record, offset 24 from the start of the record. On the other hand, **AT +5** indicates that data is to be placed 5 bytes after the end of the last PUT statement's data. If the **AT loc** clause is omitted, data is placed into the output record at the current position of the output cursor.

format can be one of the following:

- FIXED(n1,n2) — This places a binary integer of length **n1** bytes into the output record. **n1** can have any value from 1 to 4. **n2** specifies the power of 10 by which the number is to be multiplied before placing it in the output record. **N2** is not a required parameter. For example, if the input field contains a 12.55, the output field would contain F'12' if the output format is FIXED(4), and contain F'1255' if the output format is FIXED(4,2). The default missing value for a FIXED format field is -1. If the input field cannot be converted to fixed format the error value is placed into the output record.

Note that a negative number cannot be converted to a fixed format of length 1; therefore, since the MISSING value must be convertible to the output format, a format of FIXED(1) will generally (except for an AT-MOST-ONE field with a DEFAULT-VALUE) require an explicit MISSING clause.

- FLOAT(n1) — This places a floating point value of length **n1** bytes into the output record. **N1** can be 4 to produce a short floating point, 8 to produce a long floating point or 16 to produce an extended floating point value. The floating point value stored into the output record is always normalized. The default missing value for a FLOAT format field is -1. If the input field cannot be converted to float format, the error value is placed into the output record.

- PACKED(n1,n2) — This places a packed decimal integer of length **n1** bytes into the output record. **n1** can have any value from 1 to 16. **n2** specifies the power of 10 by which the number is to be multiplied before placing it in the output record. **N2** is not a required parameter. For example, if the input field contains a 12.75, the output field would contain a X'00012C' if the output format is PACKED(3) and a X'01275C' if the output format is PACKED(3,2). The default missing value for a PACKED field is -1. If the input field cannot be converted to packed format the error value is placed into the output record.

- STRING(n1,adjust,pad,start) — This places the contents of the input field into the output record without any conversion. This would typically be used for string fields. **n1** indicates the length of the output string. **adjust** can be either the letter R or the letter L enclosed in quotes and indicates whether the result string is right or left adjusted in the output string. This is not a required parameter, and it is assumed to be 'L'. **Pad** is a single EBCDIC character enclosed in quotes or a single hexadecimal character expressed as X'nn'. This character is used as the pad character if the input field is shorter than the output string. This is not a required parameter, and it is assumed to be X'40' (blank). **start** indicates the characters number at which output is to start. This provides a way of unloading substrings.

  If the input field is longer than the output string, the input field is truncated. If the input field is shorter than the output string it is padded with the pad character. Padding occurs on the right if the field is left justified and on the left if it is right justified. The default missing value for a string field is the field totally filled with the pad character.

- DECIMAL(n1,n2,n3) — This places the contents of the input field as a string of decimal characters (EBCDIC) into the output buffer. **n1** is the total length of the output field. **n2** is the number of digits to be placed to the right of the decimal point. **n2** is not a required parameter, and it is assumed to be zero. If **n2** is zero, a decimal point is not placed into the

output string. **n3** specifies the number of digits to be placed in an exponential format exponent. The default for this argument is 0, which indicates that exponential format is not to be used.

For example, the value 12.75 would be output as “ 12” if the output format is DECIMAL(4), “ 12.750” if the output format is DECIMAL(7,3), and “ 1.27E+001” if the output format is DECIMAL(12,2,3).

If it is impossible to convert the input field to decimal format, the output field is set to the error value. The default missing value for decimal format is -1.

Note that, as with all conversion of fractional values to fixed width output formats in the PUT statement, any low order fraction digits are dropped without rounding. To create a numeric string which uses rounding for dropped low order digits, see “#NUM2STR: Convert number to string with decimal point” on page 137.

- ZONED(n1,n2) — This places a signed zoned decimal integer of length **n1** bytes into the output record. **n1** can have any value from 1 to 32. **n2** specifies the power of 10 by which the number is to be multiplied before placing it in the output record. **N2** is not a required parameter. For example, if the input field contains a 12.75, the output field would contain a '1B' (X'F1C2') if the output format is ZONED(2), and contain a '127E' (X'F1F2F7C5') if the output format is ZONED(4,2). The sign is contained in the zone field of the last byte in the output. The sign is represented in "IBM preferred" format; a value of 21 is represented as X'F2C1', and a value of -21 appears as X'F2D1'. Note that the last byte will not be displayed as a decimal digit. The default missing value for a ZONED field is -1. If the input field cannot be converted to zoned format, the error value is placed into the output record.

The following input field type to output format mappings are possible:

- FLOAT to FIXED. The floating point value is converted to a fixed binary integer. If the conversion results in an overflow, the error value is set.
- FLOAT to PACKED. The floating point value is converted to a fixed packed decimal. If the conversion results in an overflow, the error value is set.
- FLOAT to FLOAT. The floating point value is converted to the correct length and normalized. If the input floating point field does not contain a valid floating point number, the error value is set.
- FLOAT to STRING. The floating point value is converted to a STRING value in a way compatible with *Model 204* (it will not contain any “E” power of 10 multiplier, and, like any conversion from a floating point

representation, it will use the algorithms specified in “Floating Point Arithmetic and Numeric Conversion” on page 235).

- FLOAT to DECIMAL. The floating point value is converted to decimal format. If the conversion results in an overflow, the error value is set.
- FLOAT to ZONED. The floating point value is converted to zoned decimal format. If the conversion results in an overflow, the error value is set.
- BINARY to FIXED. The binary value is converted to the correct length and possibly adjusted for binary fractional places. If the output field is not large enough to hold the resulting value, the output field is set to the error value.
- BINARY to PACKED. The binary value is converted to the correct length and possibly adjusted for packed decimal fractional places. If the output field is not big enough to hold the resulting value, the output field is set to the error value.
- BINARY to FLOAT. The binary value is converted to a floating point number of appropriate length.
- BINARY to DECIMAL. The binary value is converted to decimal format. If there is not enough space to place the binary number into the output string the output field is set to the error value.
- BINARY to STRING. The binary value is converted to a STRING value in a way compatible with *Model 204*.
- BINARY to DECIMAL. The binary value is converted to decimal format. If there is not enough space to place the binary number into the output string the output field is set to the error value.
- BINARY to ZONED. The binary value is converted to zoned decimal format. If there is not enough space to place the binary number into the output string the output field is set to the error value.
- STRING to FIXED. An attempt is made to convert the string value into a fixed binary number. If this is not possible, the output field is set to the error value.
- STRING to PACKED. An attempt is made to convert the string value into a packed decimal number. If this is not possible, the output field is set to the error value.
- STRING to FLOAT. An attempt is made to convert the string value into a floating point number. If this is not possible, the output field is set to the error value.

- STRING to STRING. No conversion is done. The string is moved byte for byte to the output record.

- STRING to DECIMAL. An attempt is made to convert the string value into a decimal number. If this is not possible, the output field is set to the error value.

- STRING to ZONED. An attempt is made to convert the string value into a zoned decimal number. If this is not possible, the output field is set to the error value.

Note that coded fields are treated as string fields where the contents are considered to be the uncoded value of the field contents. Note also that the only possible conversion error when going to STRING format is if the length of the output field is not large enough to hold the result, that is if the conversion would result in truncation.

See "Floating Point Arithmetic and Numeric Conversion" on page 235 for a discussion of the algorithms involved in converting from or to a numeric value.

mvalue can be either a decimal or string constant (in quotes) which is placed in the output record if the input value does not exist. In addition, **mvalue** can be an action keyword that indicates an action to be performed when a missing value is encountered. The constants can always be followed by the word **REPORT** or **NOREPORT**. This would indicate whether the missing value should be reported on the *Fast/Unload* report data set. The default for this value is determined by the output format. Basically, the default is always -1 unless the output format is STRING, in which case the default is to fill the output field with blanks. The non-string default can be customized to be 0 at your site; see "Default for MISSING clause on PUT statement" on page 292. The default is also NOREPORT, that is not to report a missing value on the report data set, unless an action keyword is specified. In this case, the default is to report missing values for the field in the PUT statement. Valid action keywords are:

- SKIP — This means that the entire input record is discarded. Note that if output records had been created with an OUTPUT statement before a missing value causes a SKIP, the output records would remain in the output data set. A partial output record that has been created before the SKIP would not go to the output data set.

- CANCEL — This means the entire *Fast/Unload* job is terminated. Use this value if a missing field occurrence indicates a severe logic error in your data file structure.

All action keywords for **MISSING** result in the action being reported in the report data set, unless NOREPORT is specified as part of **mvalue**.

evalue can be either a decimal or string constant (in quotes) which is placed in the output field if the input field cannot be correctly converted to the output format. The default is always REPORT, that is any conversion error is reported on the report data set.

In addition, **evalue** can be an action keyword that indicates an action to be performed when a unconvertible value is encountered. The constants can always be followed by the word **REPORT**. This would indicate that the unconvertible value should be reported on the *Fast/Unload* report data set. The default for this value is always the same as the missing value except when the output format is STRING. In this case, the default is TRUNCATE. The default can be customized to be CANCEL at your site; see "Default for ERROR clause on PUT statement" on page 292.

Valid action keywords are:

- TRUNCATE or TRUNC — This action keyword is only valid if the output format is STRING. If this is the case and the input record has a string longer than the output format indicates, then the input string would be truncated on the right if the input format indicates left justification and on the left for right justification. This action would not be reported on the report data set unless the REPORT keyword is specified.

- SKIP — This means that the entire input record is discarded. Note that if output records had been created with an OUTPUT statement before a missing value causes a SKIP, the output records would remain in the output data set. A partial output record that has been created before the SKIP would not go to the output data set.

- CANCEL — This means the entire *Fast/Unload* job is terminated. Use this value if a field occurrence conversion error indicates a severe error in your data file structure.

All action keywords for **ERROR** result in the action being reported in the report data set.

The following program demonstrates several types of PUT statements. The output stream destination name is OUTSTRM. The PUT and OUTPUT statements in the program do not need any TO OUTSTRM prefixes, because OUTSTRM was declared to be the default for non-UAI output streams.

```
OPEN BIGFILE
OUT TO OUTSTRM DEFAULT
FOR EACH RECORD
  PUT '*'
  PUT #RECIN   AT 5  AS FIXED(4)
  PUT REC.TYPE AT 9  AS STRING(1) MISSING REPORT ERROR TRUNC
  PUT USERID   AT 1Ø AS STRING(1Ø) ERROR TRUNC NOREPORT
  PUT CHARGE   AT 2Ø AS FIXED(4,2) MISSING -999 ERROR -99
  PUT BALANCE  AT 24 AS PACKED(8,2) MISSING -999 ERROR -99
  PUT CPUTIME  AT 32 AS DECIMAL(12,5) MISSING -9999 ERROR -1
  PUT WEIGHT   AT 44 AS FLOAT(8)
  %AVAIL = Ø
  IF LIMIT IS FLOAT AND BALANCE IS FLOAT THEN
     %AVAIL = LIMIT - BALANCE
  END IF
  PUT %AVAIL   AT 52 AS PACKED(8,2)
  OUTPUT
END FOR
```

In the following program

```
OPEN BIGFILE
FOR EACH RECORD
  PUT USERID   AS STRING(5, ,'*', 3)
  OUTPUT
END FOR
```

If USERID had a value of 'SIMPSON', 'MPSON' would be output. If USERID had a value of 'MARGE', 'RGE**' would be output. If USERID had a value of 'SCRATCHY', 'RATCH' would be output.

The following program can be used to add the value of a derived INVISIBLE KEY field to a PAI output:

```
%TOT = 0      /* Initialize
OPEN BIGFOOT
FOR EACH RECORD
   PAI
   %I = FIELD1(#)
   IF %I < FIELD2(#) THEN
      %I = FIELD2(#)
   END IF
   FOR I = 1 TO %I
      %V = #CONCAT(FIELD1, '.', FIELD2)
      PUT 'INVIS_KEY ='
      PUT %V
      OUTPUT
      %TOT = %TOT + 1
   END FOR
END FOR

REPORT 'Number of INVISIBLE KEY values created:' AND %TOT
```

However, note that to accomplish the same thing, you could use an ADD statement in place of the assignment to %V, delete the PUT and OUTPUT statements, and move the PAI to after the first END FOR.

The PUT statement is not valid for a UAI format unload.

5.37 REPEAT

This statement marks the beginning of a clause that is executed repeatedly until the body of the clause is explicitly terminated, usually by a LEAVE REPEAT statement.

This statement must always be paired with an END REPEAT statement, and, to avoid a never-ending loop, the loop should contain a statement such as LEAVE REPEAT to terminate the loop when some condition occurs. CANCEL and SKIP can also be used to terminate the loop, of course, but they also bypass statements outside the loop.

Here is an example of a REPEAT loop that is used to extract items from a delimited string:

```
%X = 'A/MAN/A/PLAN/A/CANAL/PANAMA'
%I = 1
%L = #LEN(%X)
%L = %L + 1
REPEAT
   %W = #INDEX(%X, '/', %I)
   IF %W = Ø
      %W = %L   /* Not found, go past end
   END IF
   %T = %W - %I
   %S = #SUBSTR(%X, %I, %T)
   PUT %S
   OUTPUT
   IF %W = %L   /* Last item?
      LEAVE REPEAT
   END IF
   %I = %W + 1  /* New scan position
END REPEAT
```

The above FUEL fragment produces the following output:

```
A
MAN
A
PLAN
A
CANAL
PANAMA
```

The REPEAT statement is new in *Fast/Unload* version 4.0.

5.38 REPORT entity [AND | WITH entity] ...

This statement causes information to be reported on the report data set. The data consists of *Fast/Unload* entities separated by the words 'AND' or 'WITH'. As in User Language, a WITH separator indicates that no space is to be placed between the entities on output while an AND separator indicates that a single space is to be placed

between entities. For example, in the program

```
OPEN BIGFILE
FOR EACH RECORD
  PUT KEY.FIELD AS STRING(7)
  FOR I FROM 1 TO 10
    %TEST = FIELD1(I)
    PUT %TEST  AS STRING(5)
    PUT FIELD2(I) AS STRING(5)
    IF (%TEST EXISTS) AND (FIELD2(I) MISSING)
      REPORT 'FIELD1(' WITH I WITH ')  =' AND -
             %TEST WITH -
             '. UNMATCHED IN RECORD' AND #RECIN
    END IF
  END FOR
  OUTPUT
END FOR
```

if in record 22, the third occurrence of FIELD1 was '123' but there was no third occurrence of FIELD2

```
FIELD1(3) = 123. UNMATCHED IN RECORD 22
```

would appear in the report data set.

Any numeric entity will be converted to a string representation (without any “E” power of 10 multiplier); see also “Floating Point Arithmetic and Numeric Conversion” on page 235 for a discussion of the algorithms involved in converting from a numeric value.

Note: The #OUTLEN and #OUTPOS special variables may not be used in the REPORT statement.

5.39 SELECT entity

This statement marks the beginning of a clause which indicates actions to be taken based on the value of an entity. The SELECT statement must be matched with an END SELECT statement, one or more WHEN statements and an optional OTHERWISE statement. You may not place any statements after the SELECT statement and the

WHEN statement which follows it. For example, in the program

```
OPEN BIGFILE
FOR EACH RECORD
  SELECT REC.TYPE
  WHEN 'COUNTRY'
    PUT 'COUNTRY'
    PUT PRESIDENT AS STRING(30)
  WHEN 'STATE'
    PUT 'STATE  '
    PUT GOVERNOR AS STRING(30)
  WHEN 'CITY'
    PUT 'CITY   '
    PUT MAYOR AS STRING(30)
  OTHERWISE
    REPORT 'INVALID REC.TYPE =' AND REC.TYPE AND -
          'IN RECORD' AND #RECIN
  END SELECT
  OUTPUT
END FOR
```

REC.TYPE is used as a trigger to determine what type of information is to be placed into the output record.

To test that a field or %variable contains one of several values, you should use SELECT; it is better than both of the following alternatives:

- an IF statement with multiple **OR** clauses
- an IF statement with the #ONEOF or #FIND function

SELECT is easier to code and runs more efficiently; it is the best way to implement a "ONEOF" test. Another typical use of SELECT is shown here:

```
SELECT AREACODE
WHEN 617, 508, 413, 781, 978
   %STATE = 'MA'
WHEN 407, 813, 305, 352, 954, 561
   %STATE = 'FL'
WHEN 307
   %STATE = 'WY'
END SELECT
```

Even with a single WHEN statement SELECT is preferred, rather than an IF statement:

```
SELECT RECTYPE
WHEN 'MAST', 'DETL', 'HIST'
   UNLOAD
END SELECT
```

5.40 SKIP

This statement skips to the next iteration of FUEL code. Before the FOR EACH RECORD loop, SKIP specifies that the rest of the code up to the FOR EACH RECORD loop is skipped, and the code in the FOR EACH RECORD loop, if any, is executed for the first record. Within the FOR EACH RECORD loop, this statement skips the rest of the current iteration of loop and the FOR EACH RECORD loop is resumed for the next record. After the FOR EACH RECORD loop, SKIP specifies that the rest of the code in the FUEL program is skipped.

The SKIP statement would typically be found inside a conditional clause. Any data that has been 'PUT' into the current output record is lost, unless an OUTPUT statement preceded the SKIP statement. For example, in the program

```
OPEN BIGFILE
FOR EACH RECORD
  PUT KEY.FIELD AS STRING(7)
  IF #RECIN EQ 5000 THEN
    SKIP
  END IF
  OUTPUT
END FOR
```

no data would be output for the input *Model 204* data file's record number 5000.

5.41 SORT [TO destination]

For a PUT/OUTPUT stream, SORT directives indicate that the stream data is to be passed to an external SORT routine en route to the output data file. For such output streams, the SORT directives provide the control input to the SORT routine.

The SORT clause of the UAI statement indicates that the unloaded data are to be sorted and provides the control input for the SORT routine. For such output streams, the only valid independent SORT directives are SORT OPTION and SORT PGM.

Except in legacy (prior to *Fast/Unload* version 4.1) code, if even a single output stream is declared in an OUT TO or UAI directive, every SORT directive must have the `TO destination` qualifier. The `destination` used on a SORT TO directive must be declared in an OUT TO or UAI directive. SORT, OUT TO, and UAI directives can occur in any order.

For more information about the SORT statement, see “Using an External Sort Package” on page 207.

5.42 SORT PGM sortprogramname

This directive, which is allowed at most once in a FUEL program, is used to override the default sort routine to be used if any output streams are to be sorted. The *sortprogramname* program is used to sort all sorted output streams in place of the default sort routine.

5.43 TO [destination | *]

The qualifying clause `TO destination` is used as a prefix with PUT, OUTPUT, PAI, and PRINT ALL INFORMATION statements, and with UNLOAD, UNLOADC, and NOUNLOAD statements to indicate the output stream to which the statement applies. This clause is used as a suffix with the SORT statement for the same purpose. For *Fast/Unload* programs prior to version 4.1, this clause does not exist, and all output goes to the single FUNOUT stream.

A *destination* stream must be declared in a preceding OUT TO or UAI TO directive (see “OUT TO destination” on page 68, and see “UNLOAD ALL INFORMATION or UAI” on page 88). The PUT, OUTPUT, PAI, and PRINT ALL INFORMATION operations require an OUT TO directive; UNLOAD[C] requires a UAI TO directive.

To apply one of the statements above to all the appropriately declared output streams, you specify:

```
TO *
```

If a *destination* is declared with the DEF or DEFAULT attribute on an OUT TO directive, "naked" PUT and OUTPUT statements (that is, PUT and OUTPUT statements without any `TO destination` prefix) will apply to the default stream. And similarly, for a stream declared with the DEF or DEFAULT attribute on a UAI TO directive, naked UNLOAD[C] statements will apply to the default stream.

5.44 UNLOAD[C] [field [(occur | *)]]

The UNLOAD statement unloads a record, or one or all occurrences of a field in a record, in the UAI format, to either:

- The output stream indicated by *destination*, if the UNLOAD statement has a “TO *destination*” prefix (“TO [destination | *]”)
- The implied output stream, if there is no “TO *destination*” prefix

The UNLOAD statement must be coded inside a FOR EACH RECORD loop, and it is only valid for an output stream declared with a `UAI TO destination` directive (see “UNLOAD ALL INFORMATION or UAI” on page 88).

The TO clause prefix may be omitted if there is exactly one output stream, or if the output is to go to the stream declared with the DEF or DEFAULT attribute on a UAI TO directive.

Unloads created with the UNLOAD statement can be used as input for the LAI statement in *Fast/Reload*, which is described in the ***Rocket Model 204 Fast/Reload Reference Manual***.

The UNLOAD statement has two forms: one that unloads all fields in the record and one that unloads specified fields. These forms are described in the sections that follow.

For information about a statement that lets you selectively stop or prevent subsequent unloading of some or all fields to some or all destinations, see "NOUNLOAD [field [(occurrence | *)]]" on page 65.

5.44.1 UNLOAD (all fields)

An UNLOAD statement with no field specification is called a **normal** or sometimes a **blanket** unload. This UNLOAD statement unloads *the rest of* the input record, that is, all field occurrences not yet specifically unloaded. If not preceded by any UNLOAD statements with field specifications, the normal UNLOAD statement simply unloads all of the *Model 204* input record.

The UNLOAD statement allows you to select which database records to include in a UAI unload: if no UNLOAD statement is executed in the FOR EACH RECORD body for a particular record, that record is not included in the UAI.

In the following normal UNLOAD example, any record with a selected city is unloaded using the UAI format:

```
OPEN BIGFILE
UAI TO UNLOAD DEF
FOR EACH RECORD
  SELECT CITY
  WHEN 'KALAMAZOO', 'ASHTABULA'
     UNLOAD
  END SELECT
END FOR
```

Note: You must be careful when coding your selection criteria for UNLOAD statements: If your FUEL program attempts any kind of UNLOAD for a record after a normal UNLOAD for the same record, the unload will be terminated.

5.44.2 UNLOAD[C] (specified fields)

An UNLOAD statement with a field specification is called an **UNLOAD field** statement. It controls the order in which fields are placed in the output dataset for the output stream

destination (and hence the order they will be reloaded in the *Fast/Reload* LAI). It can also be used to unload only some of the fields of a record: if you omit the "normal UNLOAD" statement for a record, only the data in explicit UNLOAD field statements is unloaded.

The syntax for an UNLOAD field statement is:

```
[TO destination] UNLOAD[C] field [(occurrence|*)]
```

where:

- *TO destination* is as described earlier and in "TO [destination | *]" on page 83.
- *field* is required, and it may be any type of *Model 204* field (including BLOB and CLOB, as of *Fast/Unload* 4.3).
- A field *occurrence* number is optional (it defaults to the first occurrence of *field*), or it may instead be an asterisk (*), meaning all occurrences of the field in the current record that have not been unloaded.
- Specifying UNLOADC instead of UNLOAD allows processing to continue in the event an occurrence of a specified field is missing from a record. UNLOADC is otherwise the same as UNLOAD.

The UNLOAD field statement is valid as of *Fast/Unload* version 4.0.

5.44.2.1 Using UNLOAD[C] field

General references in the following usage notes to "UNLOAD field" apply equally to UNLOADC field unless otherwise noted.

- The UNLOAD field statement prohibits unloading the same field occurrence more than once for each UAI-declared output stream. An attempt to do so will end the unload.
- You can use the UNLOAD field(*) statement to unload **all** occurrences of a field, even if that field sometimes does not exist on a record. However, UNLOAD field with a specific occurrence requires that the field and occurrence specified in the statement exist on the record being unloaded. If you want to unload a single occurrence of a field even though that occurrence may not exist on some records, use UNLOADC, which allows the occurrence to be missing without stopping the unload.

  If you use UNLOADC, or if "UNLOAD field(*)" is used and the field has no occurrences, nothing is unloaded for that statement, except in the following case: If it is the first UNLOAD[C] field statement for the record, the initial data for the record is unloaded. This initial data includes an empty *Model 204* record, and it may

include an implicitly unloaded field, if one is due to UAI SORT or HASH (see "UAI SORT or HASH and field unload order" on page 95).

- Since the UAI HASH statement implicitly unloads a field, you are not allowed to use UNLOAD field for the HASH field occurrence.

  In many circumstances, the UAI SORT statement also implicitly unloads the first sort field (see "UAI SORT or HASH and field unload order" on page 95). In such a case, you may not use UNLOAD field for the field occurrence specified as the first SORT item. You may, however, use UNLOAD field(*) to unload all occurrences of the HASH or SORT field.

  If a field is implicitly unloaded by UAI SORT or HASH, an UNLOAD field statement that refers to that field with a %variable or loop control variable as the occurrence is a compilation error, unless you have specified AS FIRST in the UAI statement.

- The following section contains examples that perform a **partial unload**: that is, an UNLOAD field statement is executed for a record, and a normal UNLOAD statement is not, potentially leaving some fields that are not unloaded.

  Note: If you specify OINDEX or INVISIBLE on the UAI statement, any record that you partially unload causes the run to be cancelled. Every unloaded record must be complete for the ordered index to be valid.

5.44.2.2 Examples

In the following example, the UNLOAD field statement is used to place the first occurrence of certain fields at the beginning of the record. Making this type of change to the physical representation can provide better performance, if these fields are heavily referenced in *Model 204* applications.

```
OPEN BIGFILE
UAI OINDEX  /* No partials, OINDEX OK
FOR EACH RECORD
 UNLOAD NAME
 UNLOAD ADDRESS
 UNLOAD     /* Rest of fields
END FOR
```

The following example shows how the UNLOAD field statement can be used without a subsequent normal UNLOAD to unload only certain fields in some records. This differs from an approach using the DELETE statement, because it deletes all fields not referenced; using DELETE, you need to know the names of the fields you want to remove. Either approach will obtain very high performance, but note that this approach may not be used with the UAI OINDEX or UAI INVISIBLE statement.

```
OPEN BIGFILE
UAI
%DATE = #DATEND
%KREC = %DATE - 62  /* Cutoff for obsolete records
%KFLD = %DATE - 366 /* Cutoff for obsolete fields
FOR EACH RECORD
 IF EXPIRE_DATE < %KREC  /* Unload partial?
   UNLOAD NAME          /* Yes
   UNLOAD ADDRESS
   FOR I FROM 1 TO PAYMENT(#)
     IF PAYMENT_DATE(I) < %KFLD
        LEAVE FOR  /* Order: descending date
     END IF
     UNLOAD PAYMENT_DATE(I)
     UNLOAD PAYMENT(I)
   END FOR
 ELSE      /* Unload all fields
   UNLOAD
 END IF
END FOR
```

The following example unloads only certain fields and achieves some field reordering, in this case making the largest payment the first in the record. Note again that a partial unload may not be used with the UAI OINDEX or UAI INVISIBLE statement. Also note that this example uses UNLOADC, signifying that some records might not contain an occurrence of PAYMENT.

```
OPEN BIGFILE
UAI
FOR EACH RECORD
 IF REC_TYPE = 'PMT'   /* Records of interest?
   UNLOAD NAME        /* Yes
   UNLOAD ADDRESS
   %MAX = 1
   %PMT = PAYMENT(1)
   FOR I FROM 2 TO PAYMENT(#)
     IF PAYMENT(I) > + %PMT   /* +: numeric
        %MAX = I
        %PMT = PAYMENT(I)
     END IF
   END FOR
   UNLOADC PAYMENT_DATE(%MAX)  /* Max first
   UNLOAD PAYMENT(%MAX)
   UNLOADC PAYMENT_DATE(*)     /* Unload rest
   UNLOAD PAYMENT(*)
 END IF
END FOR
```

5.45 UNLOAD ALL INFORMATION or UAI

The UNLOAD ALL INFORMATION statement provides a method of quickly unloading all records in the database. UAI unloads can be used as input to *Fast/Reload* LAI. For information about *Fast/Reload* LAI, see the ***Fast/Reload Reference Manual***. Together, UAI and LAI provide a way to quickly reorganize a *Model 204* database file or *Model 204* group of files.

The destination datasets for UAI output data have these requirements:

- The record format (RECFM) must be VB (variable blocked).
- The minimum record length (LRECL) for a UAI unload is 271 plus an increment for a SORT or HASH key length (if UAI SORT or UAI HASH) or for procedure unloading (if UAI PROCS), as follows:
  - If you are using UAI SORT, the minimum record length must be increased by the number of sort fields plus the sum of the LENGTHs for all the sort fields. For example, if you are using two sort keys with LENGTHs of 255 and 30, the minimum record length is 271 + 2 + 255 + 30 or 558.

- If you are using UAI HASH, the minimum record length must be increased by 4.

- Whether you are using SORT or HASH, if the file you are unloading contains procedures that you are unloading (UAI PROCS is explicit or implied) the minimum LRECL size is at least 297.

  If UAI NOPROCS is specified for an output stream, or if the file to be unloaded contains no procedure dictionary pages, the minimum LRECL size is not incremented for procedure handling. The “no increment if no procedure dictionary pages” rule does **not** apply to a database that has procedure dictionary pages but no procedures (that is, procedures were once there but now are deleted and the file has not yet been reorganized).

The UAI statement must immediately follow the OPEN statement in the FUEL program. The UAI statement is not permitted inside a FOR EACH RECORD loop.

For information about doing selective UAI unloads, see the UNLOAD statement (“UNLOAD[C] [field [(occur | *)]]” on page 83).

5.45.1 UAI statement options

The format of the UAI statement is:

```
UAI [ TO destination [ DEFault ] ]
    [ PROCS | NOPROCS ]
    [ SORT item1 [ AND item2 [ AND ... ] ] ]
    [ HASH fieldname [AS FIRST] | %var ]
    [ BSIZE  bsize  ]
    [ MAXREC maxrec ]
    [ OINDEX ]
    [ INV | INVISIBLE ]
    [ MAXRECO maxreco ]
```

where

TO destination [DEFault]

Declares *destination* to be the name of an output stream to be used for UAI format output.

A *destination* you declare in a UAI statement is used in a TO clause prefix to UAI output-generating statements (UNLOAD and UNLOADC) to indicate the output stream that is being written to. In such statements, a "TO destination" prefix is optional if a "UAI TO destination" statement with the DEF or DEFAULT attribute is declared.

To declare multiple destinations in the case where you are using multiple output streams, specify a separate UAI statement for each destination. Each such destination must be unique across all UAI TO (and OUT TO)

destinations. Each destination requires a dataset definition (JCL statement or FILEDEF), and no two file names in these definitions may refer to the same underlying dataset.

Prior to *Fast/Unload* version 4.1, and thereafter for unloads that have a single, UAI, output, the implicitly declared output stream is FUNOUT, and no destination stream needs to be specified.

Note: If any of multiple output streams is declared in an OUT TO or UAI directive, all output streams must be declared.

PROCS or NOPROCS

Dictates whether procedures (and procedure aliases) in the file are to be unloaded. If you specify PROCS, any procedures or procedure aliases in the file are unloaded; if NOPROCS, they are not unloaded.

Note: PROCS is the default: If you specify neither PROCS nor NOPROCS for all output streams, procedures (and procedure aliases) **are** unloaded.

Procedure statistics are generated in the report data set, if at least one UAI output stream is specified explicitly or implicitly to unload procedures (that is, you specify UAI PROCS, or you do **not** specify UAI NOPROCS). These statistics are described in "Description of procedure statistics" on page 56.

The PROCS/NOPROCS option is available as of *Fast/Unload* 4.2.

SORT item1 [AND item2]...

Orders the output stream file so that records are grouped together for LAI and, if the file is a SORT file when reloaded, **item1** can provide each record's sort key. (See "UAI SORT or HASH and field unload order" on page 95 for certain cases which render the first item unusable to be used as a reloaded soerted file's sort key.)

SORT and HASH are mutually exclusive.

SORT may not be specified with a *Model 204* group, and INVISIBLE, BLOB, and CLOB fields may not be sorted.

item1, item2, ... specify how you want the UAI records sorted; they have the following format:

```
fieldname | %var
[ STRING [TRUNC] | FLOAT | PACKED | FIXED | ZONED ]
[LENGTH length]  [ASCEND | DESCEND]
[MISSING mvalue] [FORMAT fmt]
[AS FIRST | AS PLACED]  (only first item, if field)
(old programs may have MAXLEN, a synonym for LENGTH)
```

You can specify as many as ten different sort keys. Each sort key specification must be separated from the next with the keyword AND.

- About sort key data types:

  You can specify a sort key data type of string, floating point, packed decimal, fixed binary, or zoned decimal. The item is converted to the sort key data type if necessary. If it is converted from or to a numeric type, the conversion algorithms specified in "Floating Point Arithmetic and Numeric Conversion" on page 235 are used. If it is converted from a floating point representation to STRING, there is no "E" power of 10 in the converted value.

  Additional data type details:

  - The default data type is STRING. The default LENGTH for STRING is 255.
  - If you specify FLOAT, you can specify a LENGTH of 4, 8, or 16. The default length for FLOAT is 8.
  - If you specify PACKED, the LENGTH parameter can be a value between 1 and 16, inclusive. The default length for PACKED is 16.
  - If you specify FIXED, you can specify a LENGTH of 1, 2, 3, or 4. The default length for FIXED is 4.
  - If you specify ZONED, the LENGTH parameter can be a value between 1 and 32, inclusive. The default length for ZONED is 32.

  The output data type can effect the order of the fields in the output; see "UAI SORT or HASH and field unload order" on page 95.

- TRUNC indicates that you do not care if the sort key is truncated for sort purposes. If you do not specify TRUNC and an item's value must be truncated, the unload is terminated.

  TRUNC is only valid if the output data type is STRING.

  TRUNC can effect the order of the fields in the output; see "UAI SORT or HASH and field unload order" on page 95.

- AS FIRST and AS PLACED, valid as of *Fast/Unload* version 4.0, affect the order of field unloading. They can only be specified on the first item. See "UAI SORT or HASH and field unload order" on page 95.

  AS FIRST may only be specified if the item is a field and the output data type is STRING; AS FIRST may not be specified if TRUNC is specified (see "UAI SORT or HASH and field unload order" on page 95).

- ASCEND and DESCEND indicate for any sort key whether the lowest key value should appear first (ASCEND) or last (DESCEND) in the sorted output file.

- The MISSING keyword lets you provide a value for the sort key when the field is missing from the database record or the %variable has the MISSING value.

  mvalue must be a constant convertible to the sort key data type. For example, you must not specify the missing value "NOTTHERE" when the output data type is FLOAT, since the character string cannot be converted to floating point.

  The default value for MISSING is the null string for a STRING item and is -1 for other (i.e., numeric) items.

  Note that any sequence of characters can be used as a string MISSING value; that is, it need not be enclosed in quotes. For example, the following is valid:

  ```
  UAI SORT FOO LENGTH 20 MISSING GOOD NIGHT IRENE
  ```

  In the above statement, the missing value is the following 16 characters:

  ```
  GOOD NIGHT IRENE
  ```

  However, the following is invalid, assuming LOW is not a defined field:

  ```
  UAI SORT FOO LENGTH 20 MISSING HIGH AND LOW
  ```

  This is because the keyword AND terminates the missing value.

- LENGTH indicates the amount of space to be reserved for the sort key. The maximum value for **length** is 255. No field value will ever be longer than 255.

  Note: MAXLEN is an alias for LENGTH.

- The FORMAT option indicates that the “format” operand in the SORT FIELDS statement should use the indicated **fmt** value, rather than the format that corresponds to the type of the item. For example, the following statement:

  ```
  UAI SORT EXPIRE_DATE LENGTH 6 FORMAT Y2T
  ```

  will create a SORT FIELDS statement such as:

  ```
  SORT FIELDS=(...,position,6,Y2T,A)
  ```

where *position* is the position in the UAI record of the EXPIRE_DATE field. *Y2T* is the sort format supported by DFSORT to sort character or zoned date fields with leading two-digit years.

You can use the SORT OPTION statement along with the FORMAT keyword to establish the 100 year window for date sorting. For example, using DFSORT, the following statement:

```
SORT OPTION Y2PAST=1970
```

indicates to DFSORT that two-digit year sort formats occur in the years 1970 through 2069. These two-digit year sorting features are available in DFSORT Release 14 with PTF UQ22534, or DFSORT Release 13 with PTFs UN90139, UQ05520 and UQ22533 (according to http://www.storage.ibm.com/software/sort/srtmy2p.htm).

In addition to the SORT OPTION statement, you can also use the SORT PGM statement with UAI SORT; see "Using an External Sort Package" on page 207.

HASH fieldname [AS FIRST] | %var

Orders the FUNOUT output file so that if the file is a HASH file when reloaded, the LAI operation is efficient and each record's hash key is provided.

HASH is mutually exclusive with SORT. HASH may not be specified with a *Model 204* group.

fieldname | %var can be a %variable, or it can be the name of any field in the database file, optionally followed by 1 as a field occurrence constant, for example: ID(1). The default occurrence is 1, and only the first occurrence can be used.

If **fieldname** is specified, it will be first in the output of the UAI. This is in case it is needed as the record hash key when the file is reloaded by LAI. You may specify **AS FIRST** after the HASH fieldname, in case you want to unload *other* occurrences of the field using the UNLOAD field statement with a %variable or loop control variable (see "UAI SORT or HASH and field unload order" on page 95). AS FIRST does not change any processing, it simply allows you to bypass the compiler error checking on such an UNLOAD field statement.

BSIZE Indicates the Table B size of the file that will be loaded with this UAI data. This is only necessary if the Table B size of the target file differs from the Table B size of the unloaded file. This option is ignored if the HASH option is not also specified.

MAXREC When you request a UAI unload, each *Model 204* table B record is written as one or more variable length UAI records. If you are using the SORT option, the best performance occurs when you set the LRECL of the FUNOUT DD so that it contains the average-length average *Model 204* table B record. There is one unusual case in which you might want to have the table B UAI records shorter than LRECL; you can use MAXREC to limit the size of the UAI output records. (This would be only if a larger LRECL for the OINDEX records would be demonstrably better than the optimal SORT LRECL for table B records.)

If specified, the value of MAXREC must be at least 271 plus the length of any SORT or HASH key field plus the number of SORT fields. If MAXREC is larger than the LRECL of FUNOUT, it is ignored (and LRECL is used as the limit of records sent to the sort).

OINDEX Indicates that you want *Fast/Unload* to unload Ordered Index data. This allows you to avoid resorting Ordered Index data on the reload and can thus improve reload performance. The OINDEX option will always result in the unloading of INVISIBLE Ordered Index data, thus the use of the OINDEX option obviates the need to use the INVISIBLE option.

Do not use the OINDEX option if the Ordered Index may be updated during the unload, which is possible if you use the NOENQ parameter or are running against an unenqueued found set or list when using the *Fast/Unload User Language Interface*.

Any field that is modified by an ADD, CHANGE, or DELETE statement will not have its Ordered Index values unloaded.

Note that when you specify OINDEX, any record that you partially unload (by use of an `UNLOAD field` statement without a normal UNLOAD statement) causes the run to be cancelled (see "Using UNLOAD[C] field" on page 85). Every unloaded record must be complete for the Ordered Index to be valid.

INV or INVISIBLE
indicates that you want *Fast/Unload* to unload Ordered Index data for INVISIBLE fields. This allows you to preserve INVISIBLE Ordered Index data over a reorg. Specifying both OINDEX and INVISIBLE is identical to simply specifying OINDEX.

Note that an INVISIBLE field is not unloaded if it does not have the ORDERED attribute.

Do not use the INVISIBLE option if the Ordered Index may be updated during the unload (due to the NOENQ parameter, or if using the *Fast/Unload User Language Interface*).

Any field that is modified by an ADD, CHANGE, or DELETE statement will not have its Ordered Index values unloaded.

Note that when you specify INVISIBLE, any record that you partially unload (by use of an `UNLOAD field` statement without a normal UNLOAD statement) causes the run to be cancelled (see "Using UNLOAD[C] field" on page 85). Every unloaded record must be complete for the Ordered Index to be valid.

MAXRECO

Is the maximum record length you want to use for Ordered Index data records. Ordered index data records never require sorting. Specify this value if you are unloading Ordered Index data and the output file has no explicit LRECL.

For a discussion of the performance implications of reloading OINDEX or INVISIBLE data see the ***Fast/Reload Reference Manual***.

5.45.2 UAI SORT or HASH and field unload order

If any sort item except the first is a field, it will occur twice in the output records, once as part of the sort key and a second time as part of the data (only that second instance will be reloaded by LAI). If the first sort item is a field and you specify TRUNC or AS PLACED for it, or specify an output data type other than STRING, it too will occur as both a sort key and as part of the output record. This ensures that you will never lose data because of sort key truncation.

Except in these special cases (TRUNC or AS PLACED or non-STRING), if **item1** (the first one of the SORT list) is a field, it will be first in the output of the UAI. This is in case it is needed as the sort key when the file is reloaded by LAI. This implicit unload is also performed if the UAI HASH statement is specified; the HASH field is implicitly unloaded as the first in the output of the UAI.

In any of the above special cases (TRUNC or AS PLACED or non-STRING), or if the first item is a %variable, the UAI output **is not suitable** for loading into a sorted file.

The implicit unloading must be considered if the UNLOAD field statement is used; if you are controlling the order in which fields are unloaded, you may need to be aware of the field implicitly unloaded first. *Fast/Unload* will not let you unload a field occurrence twice, so in most cases if you use an UNLOAD field statement which ***might be*** the same as the implicitly unloaded field, a compilation error occurs. If you want to use UNLOAD field with the same field name as the implicitly unloaded field, and UNLOAD field refers to that field with a %variable or loop control variable as the occurrence, you can avoid the compilation error by specifying **AS FIRST** in the UAI statement.

See also "Using Fast/Unload with the Sir2000 Field Migration Facility" on page 233, which explains that a field RELATED to the implicitly unloaded HASH or SORT field is also implicitly unloaded.

5.46 WHEN value(s)

This statement marks the beginning of a clause which indicates the actions to be taken when the entity specified on the currently active SELECT statement matches **value**. Value can specify a single value, a range of values, or multiple values and ranges of values separated by commas. A single value is indicated by a string, fixed point or floating point constant. For example, 'BART', 22, and -17.76 are valid single values. A range of values is indicated by two constants separated by a range indicator.

Valid range indicators are:

- An inclusive range
-- An inclusive range
>> An exclusive range
-> A range inclusive of the first value and exclusive of the last
>- A range exclusive of the first value and inclusive of the last

For example:

Range	*Means*
10-99	10 and 99 are considered in the range.
10>>99	Neither 10 nor 99 is considered in the range.
10->99	10 is considered in the range, and 99 is not.
10>-99	10 is not considered in the range, and 99 is.

The instructions executed in a WHEN clause are terminated by another WHEN statement, an OTHERWISE statement, or the END SELECT statement. Only one WHEN or OTHERWISE clause in a SELECT clause will be executed.

The program below demonstrates several forms of the WHEN statement:

```
OPEN BIGFILE
FOR EACH RECORD
  SELECT FINAL.GRADE
  WHEN 'INCOMPLETE'
    PUT -1 AS FIXED(2)
  WHEN 'WITHDREW'
    PUT -2 AS FIXED(2)
  WHEN 90-100, 'A'
    PUT 4 AS FIXED(2)
  WHEN 80->90, 'B'
    PUT 3 AS FIXED(2)
  WHEN 70->80, 'C'
    PUT 2 AS FIXED(2)
  WHEN 60->70, 'D'
    PUT 1 AS FIXED(2)
  OTHERWISE
    PUT 0 AS FIXED(2)
  END SELECT
  OUTPUT
END FOR
```

The values in a WHEN statement can have different types, for example, string, fixed point, or floating point. The comparison is made on the basis of the constant type. If an entity cannot be converted to fixed or floating point for the purposes of comparison, it is simply treated as zero.

CHAPTER 6 Standard #Functions

This chapter lists the #functions provided with *Fast/Unload*. #Functions can be used as the expression in an assignment statement, as described in “Assignment statement” on page 35. Each #function is presented here with a brief phrase denoting its use, a short explanation, the form and types of its arguments and the result (or "output value"), one or more examples, any error conditions, and special notes.

Unless otherwise indicated, all arguments shown are required and are input arguments, and any arguments with the `MISSING` value are treated as either null strings (for string arguments) or zeroes (for numeric arguments). See “#Function calls” on page 33 for an explanation of required/optional, output, and other types of arguments.

In some instances the behavior of a #function will be compared to the behavior of a corresponding $function in *Model 204* User Language. For documentation of the User Language $functions, see http://m204wiki.rocketsoftware.com/index.php/M204wiki_main_page#.24Functions.

FUEL also contains program entities called “special variables” that might be initially mistaken for #functions. Similar to #functions in appearance and syntax, special variables are described in “Special variables” on page 28.

Unless otherwise indicated, all input strings of the standard #functions are treated as non-DBCS, and all output strings have type of non-DBCS.

In addition to the reference material in this chapter, the index contains a major heading labelled **#Function prototypes**. Under this heading are minor headings containing the form of the #functions, for your convenient reference.

6.1 Run-time errors during standard #function calls

The default approach for handling errors in calls to standard #functions is strict:

- *Fast/Unload* treats most errors which the programmer can reasonably avoid as serious and terminates the run, if you have not provided for checking of the error.

- At termination, *Fast/Unload* will issue a message describing the nature of the error, the input program line number being executed and, if the error occurred during the execution of a #function, a dump of the current value of each #function argument.

Fast/Unload provides an optional output "return code" argument for some #functions. You can test the value set in this argument for errors in the value of the other arguments. Various non-zero values for the return code argument indicate error conditions; unless otherwise indicated, when one of these is set, the #function result is set to the MISSING value. If you omit the return code argument and there is an error in the value of an argument, the program is cancelled.

For example, since 'ONE WEEK' isn't numeric, the following fragment:

```
%NEW_DT = #DATECHG('MM/DD/YY', 'Ø1/Ø1/96', 'ONE WEEK',, -
          %TST)
IF %TST NE Ø THEN
   REPORT 'Error incrementing date:' AND %TST
END IF
```

will set %NEW_DT to the MISSING value and will produce the following line on the FUNPRINT dataset:

```
Error incrementing date: 12
```

while the following fragment:

```
 %NEW_DT = #DATECHG('MM/DD/YY', 'Ø1/Ø1/96', 'ONE WEEK')
```

will cause *Fast/Unload* to immediately terminate, issuing an error message that a value is non-numeric, as shown in the following sample from FUNPRINT:

```
FUNLØØ53 Unload started.
FUNLØ116 Value is non-numeric or is out of range in line 2Ø3;
  Fast/Unload cancelled.
FUNLØ127 Fast/Unload cancelled unloading input record number Ø in file
  PROCFILE.
FUNLØ13Ø Fast/Unload cancelled during execution of line 2Ø3.
FUNLØ128 3 argument positions passed to #function; values :
FUNLØ129 Ø1=MM/DD/YY.
FUNLØ129 Ø2=Ø1/Ø1/96.
FUNLØ129 Ø3=ONE WEEK.
```

Each #function description in this chapter has a figure showing the error conditions for the #function and, if there is a "return code" argument, the non-zero values for it. Notice that, in general, there are no run-time errors for missing required arguments or too many arguments; this is because such checking is done at compile time.

6.2 #ABDUMP: End Fast/Unload with ABEND and dump

The #ABDUMP function accepts a numeric argument, and causes the *Fast/Unload* job to terminate with user abend code equal to that argument, producing a dump.

```
%junk = #ABDUMP(ccode)
```

where

ccode Condition code for user ABEND; optional, defaults to 0.

%junk This #function does not return, but the FUEL syntax requires a place to store the #function result.

For example, the following fragment abends the job when the input record number is 12345:

```
IF #RECIN EQ 12345
  %FOO = #ABDUMP(99)
END IF
```

Notes

- The purpose of this #function is to assist in problem diagnosis; Technical Support will direct you if ever it needs to be used.

This #function is new in *Fast/Unload* version 4.1.

6.3 #CONCAT: Concatenate strings

The #CONCAT function accepts two or more arguments and returns a string value that is the concatenation of all of its arguments.

```
%out = #CONCAT(stra, strb ...)
```

where

stra First input string (required).

strb Second input string (required).

... Additional input strings (optional). For versions of *Fast/Unload* prior to 4.3, total length of all input strings may not exceed 255.

%out Concatenation of inputs.

For example, the following fragment prints the line `Bugs Bunny` on the FUNPRINT dataset:

```
%RABBIT = #CONCAT('Bugs', ' ', 'Bunny')
REPORT %RABBIT
```

The *Fast/Unload* run is cancelled with a return code of 8 when:

- Total length of input strings exceeds 255 and *Fast/Unload* version is less than 4.3.
- Argument missing (see “Notes” below).

Notes

- If the *n*th argument is present, where *n* is greater than 1, then the *n-1*st argument must be also be present. That is, the following will result in a run-time error, because the third argument is absent:

  ```
  %X = #CONCAT(%A, %B, , %D)
  ```

- See “#CONCAT_TRUNC: Concatenate strings, allowing truncation” on page 104 if your version of *Fast/Unload* is earlier than 4.3 and if you cannot ensure that the total length of the #CONCAT input strings is less than 256.

6.4 #CONCAT_TRUNC: Concatenate strings, allowing truncation

The #CONCAT_TRUNC function accepts an optional output argument and two or more string arguments and returns a string value that is the concatenation of the string arguments, to a maximum of 255 bytes. It allows the total length of the string arguments to exceed 255, and returns the total length of those arguments as the absolute value of the output argument, setting it to a negative value if the length exceeds 255.

```
%out = #CONCAT_TRUNC(%lenrc, stra, strb ...)
```

where

%lenrc Optional %variable, set to total length of *stra*, *strb*, ..., or to the negative of that length, if it exceeds 255.

stra First input string (required).

strb Second input string (required).

... Additional input strings (optional).

%out Concatenation of *stra*, *strb*, ..., or the first 255 bytes of that concatenation, if it exceeds 255.

For example, the following fragment

```
%FOO = #CONCAT_TRUNC(%LEN, 'Hello,', -
   ' ', 'World!')
REPORT %FOO AND 'length is' AND %LEN
```

prints the following line on the FUNPRINT dataset:

```
Hello, World! length is 13
```

The *Fast/Unload* run is cancelled with a return code of 8 when:

- Argument missing (see “Notes” below).

Notes

- If the *n*th argument is present, where *n* is greater than 2, then the *n-1*st argument must be also be present. That is, the following will result in a run-time error, because the fourth argument is absent:

```
%X = #CONCAT_TRUNC(%L, %A, %B, , %D)
```

- See “#CONCAT: Concatenate strings” on page 103 if you want *Fast/Unload* to terminate if the total length of the input strings is greater than 255.

This #function is new in *Fast/Unload* version 4.0.

6.5 #C2X: Convert character string to hex representation

The #C2X function expects one required argument and returns a string containing the hexadecimal representation of the value in the argument string. Each byte position of the input string is converted to two characters in the result string, using only the characters **0**-**9** and **A**-**F**.

```
%hex = #C2X(str)
```

where

str String to convert. Must be 127 bytes or less.

%hex String containing hex digits.

For example, the code fragment

```
%JUNK = #C2X('ABabc')
REPORT 'X''' WITH %JUNK WITH ''''
```

would produce **X'C1C2818283'**, while

```
%JUNK = #C2X(' 1 ')
REPORT 'X''' WITH %JUNK WITH ''''
```

would produce **X'40F140'**.

The *Fast/Unload* run is cancelled with a return code of 8 when:

- Length of *str* exceeds 127.

Notes

- The inverse of this #function is #X2C.
- The User Language $C2X function (http://m204wiki.rocketsoftware.com/index.php/$C2X) allows a maximum input length of 126, and an error causes the null string to be returned.
- The User Language $IHEXA function (http://m204wiki.rocketsoftware.com/index.php/$IHexA) ignores any input characters after the first 127.

6.6 #DATE: Current date and/or time

The #DATE function accepts an optional datetime format argument and an optional output return code argument and returns the current date and time in the specified (or defaulted) format.

```
%dat = #DATE(fmt, %rc)
```

where

fmt Optional format for returned date, default is **YYYY-MM-DD**.

%rc Optional output return code variable.

%dat Datetime string with indicated format.

For example, the following fragment stores the current date in the last occurrence of the REORG_DATE field:

```
ADD REORG_DATE = #DATE()
```

Non-zero values of *%rc*, or terminating conditions, are shown in the following figure (see the discussion in “Run-time errors during standard #function calls” on page 100).

Errors: if *%rc* present, set to corresponding number and set *%dat* to MISSING; if absent, cancel *Fast/Unload* with return code 8.

4 *fmt* is not a valid datetime format.

Notes

- The only difference between #TIME and #DATE is the default value for *fmt*.
- The default format returned by the User Language $DATE and $SIR_DATE functions is “YY-MM-DD”; the default format returned by #DATE is “YYYY-MM-DD”.
- “Datetime Formats” on page 172 explains valid datetime formats.

6.7 #DATECHG: Add some days to datetime

The #DATECHG function adds a specified number of days to an input datetime, returning the incremented datetime. It requires a datetime format argument, a datetime value, and a signed number of days. It accepts an optional CENTSPAN value for interpreting datetimes with two digit years. The incremented datetime is returned in the same format as the input datetime. An optional output return code argument allows the FUEL program to intercept error conditions.

```
%odat = #DATECHG(fmt, dat, n, span, %rc)
```

where

fmt Format of *dat* and *%odat*.

dat Datetime string.

n Number of days to add to *dat*.

span Optional **CENTSPAN** value.

%rc Return code variable (optional, output).

%odat Set to *dat* plus *n* days.

For example, the following fragment prints the date one week after the run date on the FUNPRINT dataset:

```
%X = #DATE('DAY Month, YYYY')
%X = #DATECHG('DAY Month, YYYY', %X, 7)
REPORT %X
```

Non-zero values of *%rc*, or terminating conditions, are shown in the following figure (see the discussion in "Run-time errors during standard #function calls" on page 100).

Errors: if *%rc* present, set to corresponding number and set *%odat* to MISSING; if absent, cancel *Fast/Unload* with return code 8.

```
4   fmt is not a valid datetime format.
8   dat does not match fmt or result date out of range.
12  n is not a number or span is invalid.
```

Notes

- "Datetime Formats" on page 172 explains valid datetime formats and valid dates.
- "CENTSPAN" on page 178 explains **CENTSPAN** arguments.

- The User Language $DATECHG function (http://m204wiki.rocketsoftware.com/index.php/$DATECHG) does not provide a call-level argument for interpreting two-digit years, the thread-level CENTSPLT/DEFCENT parameters are used instead.

- The User Language $SIR_DATECHG function (http://m204wiki.rocketsoftware.com/index.php/$Sir_Datechg) provides a CENTSPAN argument in the same fashion as #DATECHG.

6.8 #DATECHK: Check if datetime matches format

The #DATECHK function verifies that a specific datetime value is valid for a given datetime format. It requires a datetime format argument and a datetime value. It accepts an optional CENTSPAN value for interpreting datetimes with two digit years. An optional output return code argument allows the FUEL program to intercept error conditions. The #DATECHK function returns 1 if all arguments are valid and consistent, else 0 if all arguments are valid except for the date.

```
%tst = #DATECHK(fmt, dat, span, %rc)
```

where

fmt Datetime format string for *dat*.

dat Datetime string to be validated against *fmt*.

span Optional **CENTSPAN** value.

%rc Return code variable (optional, output).

%tst Set to 1 if *dat* matches *fmt*, 0 otherwise.

For example, the following fragment prints the string `Bad` on the FUNPRINT dataset:

```
%X = #DATECHK('DAY Month, YYYY', '30 February, 1997')
IF %X = 1 THEN
   REPORT 'Good'
ELSE
   REPORT 'Bad'
END IF
```

Non-zero values of *%rc*, or terminating conditions, are shown in the following figure (see the discussion in "Run-time errors during standard #function calls" on page 100).

Errors: if *%rc* present, set to corresponding number and set *%tst* to MISSING; if absent, cancel *Fast/Unload* with return code 8.

```
4   Fmt is not a valid datetime format.
12  span is an invalid CENTSPAN value.
```

Notes

- "Datetime Formats" on page 172 explains valid datetime formats and valid dates.
- "CENTSPAN" on page 178 explains **CENTSPAN** arguments.

- The CCA User Language $DATECHK function does not provide a call-level argument for interpreting two-digit years, the thread-level CENTSPLT/DEFCENT parameters are used instead.

- The Sirius User Language $SIR_DATECHK function provides a CENTSPAN argument in the same fashion as #DATECHK.

6.9 #DATECNV: Convert datetime to different format

The #DATECNV function converts a datetime value from one datetime format to another datetime format. It requires an input datetime value, a corresponding input datetime format string, and an output datetime format string. It accepts an optional CENTSPAN value for interpreting datetimes with two digit years. An optional output return code argument allows the FUEL program to intercept error conditions. The #DATECNV function returns the input datetime value in the format specified by the output datetime format string.

```
%odat = #DATECNV(infmt, outfmt, dat, span, %rc)
```

where

infmt Datetime format string for *dat*.

outfmt Datetime format string for *odat*.

dat Input datetime string.

span Optional **CENTSPAN** value.

%rc Return code variable (optional, output).

%odat Set to the value of *dat*, converted to output format.

For example, this prints the string 19970101 on the FUNPRINT dataset:

```
%X = #DATECNV('YYMMDD', 'YYYYMMDD', '970101', 1950)
REPORT %X
```

Non-zero values of *%rc*, or terminating conditions, are shown in the following figure (see the discussion in "Run-time errors during standard #function calls" on page 100).

Errors: if *%rc* present, set to corresponding number and set *%odat* to MISSING; if absent, cancel *Fast/Unload* with return code 8.

```
4   infmt or outfmt is not a valid datetime format.
8   dat does not match infmt.
12  span is an invalid CENTSPAN value.
16  Converted datetime value out of range for %outfmt.
```

Notes

- "Datetime Formats" on page 172 explains valid datetime formats and valid dates.
- "CENTSPAN" on page 178 explains **CENTSPAN** arguments.

6.10 #DATEDIF: Difference between two dates

The #DATEDIF function subtracts a second date from a first date and returns the difference in days, ignoring time portions for both dates. It requires a first datetime format string, a first datetime value, and a second datetime value. It accepts an optional second datetime format string and an optional CENTSPAN value for interpreting datetimes with two digit years. An optional output return code argument allows the FUEL program to intercept error conditions.

```
%dif = #DATEDIF(fmta, data, fmtb, datb, -
                span, %rc)
```

where

fmta Datetime format string for *data*.

data First datetime string.

fmtb Optional second datetime format string for *datb*. Default is to use *fmta*.

datb Second datetime string.

span Optional **CENTSPAN** value.

%rc Return code variable (optional, output).

%dif Set to the number of days obtained by subtracting the second date from the first date, ignoring any time components of both dates.

For example, the following fragment prints the string `7 days` on the FUNPRINT dataset:

```
%X = #DATEDIF('YYMMDD', '970308', , '970301')
REPORT %X AND 'days'
```

Non-zero values of *%rc*, or terminating conditions, are shown in the following figure (see the discussion in "Run-time errors during standard #function calls" on page 100).

Errors: if *%rc* present, set to corresponding number and set *%dif* to MISSING; if absent, cancel *Fast/Unload* with return code 8.

```
4   fmta or fmtb is not a valid datetime format.
8   data does not match fmta, or datb does not match fmtb.
12  span is an invalid CENTSPAN value.
```

Notes

- Time is ignored in the subtraction.

- In the example above, even though the input dates have 2-digit years, there is no need for a *span* argument, since the dates are in the same year.

- As in the CCA User Language $DATEDIF function, the single **span** argument is used for both dates; if it is necessary to get the difference between two dates which both have 2-digit years and are in different 100-year windows, you must first use #DATECNV to convert one of them to some 4-digit year format.

- “Datetime Formats” on page 172 explains valid datetime formats and valid dates.

- “CENTSPAN” on page 178 explains **CENTSPAN** arguments.

6.11 #DATEFMT: Validate datetime format string

The #DATEFMT function is used to validate a datetime format string. It requires a datetime format string. It returns a value of 1 if the datetime format string is valid, else it returns a value of 0.

```
%tst = #DATEFMT(fmt)
```

where

fmt Datetime format string to be validated.

%tst Set to 1 if *fmt* is a valid format string, else set to 0.

For example, the following fragment prints the string `Good` on the FUNPRINT dataset:

```
%X = #DATEFMT('CYYDDDHHMISSXXX')
IF %X = 1 THEN
   REPORT 'Good'
ELSE
   REPORT 'Bad'
END IF
```

This #function has no terminating conditions.

Notes

- “Datetime Formats” on page 172 explains valid datetime formats.

6.12 #DATE2N: Convert datetime string to number of seconds*300

The #DATE2N function converts a datetime value in a string format into a numeric form that is the corresponding number of seconds*300 since January 1, 1900. It requires a datetime value string and a corresponding datetime format string. It accepts an optional CENTSPAN value for interpreting datetimes with two digit years. An optional output return code argument allows the FUEL program to intercept error conditions.

```
%num = #DATE2N(dat, fmt, span, %rc)
```

where

dat Datetime value string.

fmt Datetime format string for *dat*.

span Optional **CENTSPAN** value.

%rc Return code variable (optional, output).

%num Set to the number of seconds*300 since 1 Jan 1900 12:00 AM that corresponds to the datetime value of *dat*.

For example, the following fragment prints the value `BEFORE` on the FUNPRINT dataset:

```
IF #DATE2N('121494', 'MMDDYY') < -
           #DATE2N('040195', 'MMDDYY') THEN
  REPORT 'BEFORE'
END IF
```

Non-zero values of *%rc*, or terminating conditions, are shown in the following figure (see the discussion in "Run-time errors during standard #function calls" on page 100).

Errors: if *%rc* present, set to corresponding number and set *%num* to MISSING; if absent, cancel *Fast/Unload* with return code 8.

```
4   fmt is not a valid datetime format.
8   dat does not match fmt or result date out of range.
12  span is an invalid CENTSPAN value.
```

Notes

- Values returned by #DATE2N will often exceed the range that can be represented in a 4-byte integer, so you should probably avoid storing the value in a BINARY or FLOAT4 field.

- Dates prior to 1 January 1900 will return a negative number.
- The inverse of this #function is #N2DATE.
- “Datetime Formats” on page 172 explains valid datetime formats and valid dates.
- “CENTSPAN” on page 178 explains **CENTSPAN** arguments.

6.13 #DATE2ND: Convert datetime string to number of days

The #DATE2ND function converts a datetime value in a string format into a numeric form that is the corresponding number of days since January 1, 1900. It requires a datetime value string and a corresponding datetime format string. It accepts an optional CENTSPAN value for interpreting datetimes with two digit years. An optional output return code argument allows the FUEL program to intercept error conditions.

```
%num = #DATE2ND(dat, fmt, span, %rc)
```

where

dat Datetime value string.

fmt Datetime format string for *dat*.

span Optional **CENTSPAN** value.

%rc Return code variable (optional, output).

%num Set to the number of days since 1 Jan 1900 12:00 AM that corresponds to the datetime value of *dat*.

For example, the following fragment prints the value `BEFORE` on the FUNPRINT dataset:

```
IF #DATE2ND('121494', 'MMDDYY') < -
           #DATE2ND('040195', 'MMDDYY') THEN
  REPORT 'BEFORE'
END IF
```

Non-zero values of *%rc*, or terminating conditions, are shown in the following figure (see the discussion in "Run-time errors during standard #function calls" on page 100).

Errors: if *%rc* present, set to corresponding number and set *%num* to MISSING; if absent, cancel *Fast/Unload* with return code 8.

```
4   fmt is not a valid datetime format.
8   dat does not match fmt or result date out of range.
12  span is an invalid CENTSPAN value.
```

Notes

- Dates prior to 1 January 1900 will return a negative number.
- The inverse of this #function is #ND2DATE.

- “Datetime Formats” on page 172 explains valid datetime formats and valid dates.
- “CENTSPAN” on page 178 explains **CENTSPAN** arguments.

6.14 #DATE2NM: Convert datetime string to number of milliseconds

The #DATE2NM function converts a datetime value in a string format into a numeric form that is the corresponding number of milliseconds since January 1, 1900. It requires a datetime value string and a corresponding datetime format string. It accepts an optional CENTSPAN value for interpreting datetimes with two digit years. An optional output return code argument allows the FUEL program to intercept error conditions.

```
%num = #DATE2NM(dat, fmt, span, %rc)
```

where

dat Datetime value string.

fmt Datetime format string for *dat*.

span Optional **CENTSPAN** value.

%rc Return code variable (optional, output).

%num Set to the number of milliseconds since 1 Jan 1900 12:00 AM that corresponds to the datetime value of *dat*.

For example, the following fragment prints the value 'BEFORE' on the FUNPRINT dataset:

```
IF #DATE2NM('121494', 'MMDDYY') < -
           #DATE2NM('040195', 'MMDDYY') THEN
  REPORT 'BEFORE'
END IF
```

Non-zero values of *%rc*, or terminating conditions, are shown in the following figure (see the discussion in "Run-time errors during standard #function calls" on page 100).

Errors: if *%rc* present, set to corresponding number and set *%num* to MISSING; if absent, cancel *Fast/Unload* with return code 8.

```
4   fmt is not a valid datetime format.
8   dat does not match fmt or result date out of range.
12  span is an invalid CENTSPAN value.
```

Notes

- Values returned by #DATE2NM will often exceed the range that can be represented in a 4-byte integer, so you should probably avoid storing the value in a BINARY or FLOAT4 field.

- Dates prior to 1 January 1900 will return a negative number.
- The inverse of this #function is #NM2DATE.
- “Datetime Formats” on page 172 explains valid datetime formats and valid dates.
- “CENTSPAN” on page 178 explains **CENTSPAN** arguments.

6.15 #DATE2NS: Convert datetime string to number of seconds

The #DATE2NS function converts a datetime value in a string format into a numeric form that is the corresponding number of seconds since January 1, 1900. It requires a datetime value string and a corresponding datetime format string. It accepts an optional CENTSPAN value for interpreting datetimes with two digit years. An optional output return code argument allows the FUEL program to intercept error conditions.

```
%num = #DATE2NS(dat, fmt, span, %rc)
```

where

dat Datetime value string.

fmt Datetime format string for *dat*.

span Optional **CENTSPAN** value.

%rc Return code variable (optional, output).

%num Set to the number of seconds since 1 Jan 1900 12:00 AM that corresponds to the datetime value of *dat*.

For example, the following fragment prints the value 'BEFORE' on the FUNPRINT dataset:

```
IF #DATE2NS('121494', 'MMDDYY') < -
           #DATE2NS('040195', 'MMDDYY') THEN
  REPORT 'BEFORE'
END IF
```

Non-zero values of *%rc*, or terminating conditions, are shown in the following figure (see the discussion in "Run-time errors during standard #function calls" on page 100).

Errors: if *%rc* present, set to corresponding number and set *%num* to MISSING; if absent, cancel *Fast/Unload* with return code 8.

```
4   fmt is not a valid datetime format.
8   dat does not match fmt or result date out of range.
12  span is an invalid CENTSPAN value.
```

Notes

- Values returned by #DATE2NS will often exceed the range that can be represented in a 4-byte integer, so you should probably avoid storing the value in a BINARY or FLOAT4 field.

- Dates prior to 1 January 1900 will return a negative number.
- The inverse of this #function is #NS2DATE.
- “Datetime Formats” on page 172 explains valid datetime formats and valid dates.
- “CENTSPAN” on page 178 explains **CENTSPAN** arguments.

6.16 #DEBLANK: Remove leading and trailing blanks from substring

The #DEBLANK function extracts a substring (starting at pos for length len) and returns it, with leading and trailing blanks removed. It requires a string to be deblanked. It accepts an optional starting position within the string to be deblanked, and an optional length.

```
%out = #DEBLANK(str, pos, len)
```

where

str String to be deblanked; required.

pos Optional position within *str* to start deblanking. Must be numeric greater than or equal to 1, default is 1.

len Optional length, starting at *pos* of string to deblank. Must be numeric greater than or equal to 0, default is the remainder of the input string.

%out Set to the substring within *str*, starting at *pos* for length *len*; leading and trailing blanks are then removed from this substring. Intermediate blanks within the substring are not affected.

Examples:

```
#DEBLANK(' ABC ')          returns  'ABC'
#DEBLANK(' ABC ', 1, 2)    returns  'A'
#DEBLANK(' A  BC ', 2, 4) returns  'A  B'
```

The *Fast/Unload* run is cancelled with a return code of 8 when:

- *pos* is not numeric, or less than 1.
- *len* is not numeric, or less than 0.

Notes

- This #function is similar to the #STRIP function ("#STRIP: Remove leading and/or trailing copies of pad character" on page 152).

This #function is new in *Fast/Unload* version 4.0.

6.17 #DELWORD: Remove blank-delimited words from string

The #DELWORD function removes one or more blank-delimited words from a string. It requires an input string and a word number within the string. It accepts an optional number of words to delete.

```
%out = #DELWORD(str, word, count)
```

where

str String containing words to be deleted; required.

word Number of first word within *str* to remove; required. Must be numeric greater than or equal to 1.

count Optional count of words to remove. Must be numeric greater than or equal to 0, default is the remainder of words in the input string.

%out Set to *str* with the substring removed which starts with the first character of word number *word* and ends with the last blank character before word number *word+count*.

Examples:

```
#DELWORD('A B C', 1)      returns  ''
#DELWORD('A B C', 2)      returns  'A '
#DELWORD(' A B  C ', 3)   returns  ' A B  '
#DELWORD('A B C', 4)      returns  'A B C'
#DELWORD('  ', 1)         returns  '  '
#DELWORD('A B C', 1, 1)   returns  'B C'
#DELWORD('A B C', 1, 4)   returns  ''
#DELWORD('A B C', 2, 2)   returns  'A '
```

The *Fast/Unload* run is cancelled with a return code of 8 when:

- *word* is not numeric, or less than 1.
- *count* is not numeric, or less than 0.

This #function is new in *Fast/Unload* version 4.0.

6.18 #FIND: Word position of one word sequence within another

The #FIND function determines the word position, within a string of blank-delimited words, of one or more blank-delimited words. It requires an input string to search within and an input sequence of words to search for.

```
%pos = #FIND(haystack, words)
```

where

haystack String containing words to be searched within; required.

words String containing one or more blank-delimited words to find; required.

%pos Set to the word position, within *haystack*, of the first word of a sequence of words that matches the sequence in *words*. If *words* is not found, *%pos* is set to 0.

Examples:

```
#FIND('A B C', 'B')          returns  2
#FIND('A B C', 'D')          returns  0
#FIND('A B C', 'A B')        returns  1
#FIND('     ', ' ')          returns  0
#FIND('A B', 'A B C')        returns  0
#FIND('B B B A', 'B B A')    returns  2
#FIND('A A B', ' A   B ')    returns  2
#FIND(' A  A   B  ', 'A B')  returns  2
```

This #function has no cancelling conditions.

Notes

- Multiple blanks in both input arguments are ignored.
- This #function is similar to the #ONEOF function ("#ONEOF: See if string is in delimited list of strings" on page 142). Note, however, that both #FIND and #ONEOF are inferior to the SELECT statement, when the only purpose is to test whether an entity has one of several values (see "SELECT entity" on page 80).

This #function is new in *Fast/Unload* version 4.0.

6.19 #FLOAT8: Get 8-byte float, padding 4-byte input with 0

The #FLOAT8 function accepts a numeric argument, and it returns the value of the argument as an 8-byte floating point value. If the argument is a 4-byte floating point value, then the conversion is done by appending binary zeroes; otherwise, it is done by the normal FUEL conversion to an 8-byte floating point value.

```
%out = #FLOAT8(in)
```

where

%out Set to 8-byte floating point value of input argument.

in Numeric input value.

Notes

- See “Floating Point Arithmetic and Numeric Conversion” on page 235 for a specification of conversions to floating point values.
- Numeric operations in FUEL and in User Language are based on **decimal**, not **binary**, interpretation of floating point values, so this #function is seldom used.

  However, #FLOAT8 may be useful in unusual situations, in particular to perform a file reorganization that expands a FLOAT LEN 4 field to a FLOAT LEN 8 field, using the “raw” floating point conversion (such as can be done in a structured file reorganization using the `X'0080'` mode bit in FLOD).

  For example, if field FLT is defined in the input as FLOAT LEN 4, and you want to convert it to a FLOAT LEN 8 or FLOAT LEN 16 in a UAI/LAI file reorganization in such a way that the new field's values consist of the old ones with binary zeros added, you can use the following:

```
OPEN PRODFILE
UAI
FOR EACH RECORD
  FOR I FROM 1 TO FLT(#)
    CHANGE FLT(I) = #FLOAT8(FLT(I))
  END FOR
  UNLOAD
END FOR
```

  In this example, if the input value of FLT is 411028F6, which is the closest 4-byte floating point value to the decimal value 1.01, it is converted on output to 411028F600000000, which User Language will display as 1.01000022888184 (demonstrating that #FLOAT8 is only to be used in special circumstances).

A “normal” UAI/LAI conversion of 411028F6 to a FLOAT LEN 8 field would be to the hexadecimal value 411028F5C28F5C28, which User Language will display as 1.01.

This #function is new in *Fast/Unload* version 4.3.

6.20 #INDEX: Position of second string within first

The #INDEX function locates the first occurrence, if any, of a search string (needle) within a searched string (haystack). It requires a string to be searched and accepts an optional string to locate. It accepts an optional starting position within the string to be searched. #INDEX returns either the starting position within the searched string of the first occurrence of the string to be located (starting at the specified position), or 0.

```
%opos = #INDEX(haystack, needle, pos)
```

where

haystack String to be searched; required.

needle String to be located, may be omitted.

pos Optional starting position within *haystack* for search. Must be numeric greater than or equal to 1, default is 1.

%opos Set to starting position of first occurrence of *needle* within *haystack*, with search starting at *pos*, else 0 if not found or if either string length is zero.

Examples:

```
#INDEX('123xy', '23')     returns  2
#INDEX('123xy', '45')     returns  0
#INDEX('12312', '12', 2)  returns  4
#INDEX('123xy', '23', 3)  returns  0
```

The *Fast/Unload* run is cancelled with a return code of 8 when:

- *pos* is not numeric, or less than 1.

Notes

- If *needle* or *haystack* is the null string, 0 is returned.

6.21 #LEFT: Initial substring, followed by pad characters to specified length

The #LEFT function returns a padded initial substring from a source string. It expects a source string argument and a numeric length of the output string. It accepts an optional pad character. #LEFT returns the string that begins at the first position of the source string and is of the specified output length. If the output length is less than or equal to the length of the source string, the first *length* characters of the source are returned. Otherwise, the source string is padded to the output length by following it with sufficient copies of the pad character.

```
%out = #LEFT(str, len, pad)
```

where

str Source string.

len Length of the output string. Must be numeric >= 0.

pad Optional pad character. Default is blank. If supplied, must be a string of length 1.

%out If *len* is greater then #LEN(*str*):

- all of *str*, followed by *len* - #LEN(*str*) copies of the pad character

Otherwise:

- substring from *str* starting at position 1 with length *len*

For example:

```
#LEFT('ABC', 1)       returns  'A'
#LEFT('ABC', 3)       returns  'ABC'
#LEFT('ABC', 4)       returns  'ABC '
#LEFT(4.6, 4, Ø)      returns  '4.6Ø'
```

The *Fast/Unload* run is cancelled with a return code of 8 when:

- *Len* not numeric, or less than 0.
- *Pad* supplied, and length not 1.

Notes

- This is the same as the #PADR function, except for the order of the arguments (see "#PADR: Initial substring, followed by pad characters to specified length" on page 146).

This #function is new in *Fast/Unload* version 4.0.

6.22 #LEN: Length of string

The #LEN function returns the length of a string, in number of bytes. It expects a required string argument and returns a numeric result.

```
%len = #LEN(str)
```

where

str String whose length you want to know. For *Fast/Unload* version 4.3 and later. *str* may be longer than 255 bytes.

%len Set to length of *str*.

For example, the following fragment prints the string `Length: 13` on the FUNPRINT dataset:

```
%X = #LEN('Hello, world!')
REPORT 'Length:' AND %X
```

This #function has no cancelling conditions.

6.23 #LOWCASE: Change uppercase letters of string to lowercase

The #LOWCASE function returns a copy of the input string, with all uppercase EBCDIC letters changed to the corresponding lowercase letters. It expects a source string argument.

```
%out = #LOWCASE(str)
```

where

str Source string, required.

%out Copy of *str*, with all uppercase EBCDIC letters changed to their lowercase EBCDIC equivalents.

For example:

```
#LOWCASE('?abc')    returns  '?abc'
#LOWCASE('?ABC')    returns  '?abc'
#LOWCASE('')        returns  ''
```

This #function has no cancelling conditions.

This #function is new in *Fast/Unload* version 4.0.

6.24 #ND2DATE: Convert number of days to datetime string

The #ND2DATE function converts a numeric datetime value expressed as the number of days since January 1, 1900 into a datetime string value according to a specified datetime format string. It requires a datetime numeric value and a datetime format string. An optional output return code argument allows the FUEL program to intercept error conditions.

```
%dat = #ND2DATE(datn, fmt, %rc)
```

where

datn Numeric datetime value expressed as the number of days since 1 Jan 1900 12:00 AM. This argument may not have the MISSING value.

fmt Datetime format string to use for creating *%dat*.

%rc Return code (optional, output)

%dat Set to datetime string value, in format specified by *fmt*, corresponding to *datn*.

For example, the following fragment prints the string `Ø7/31/84` on the FUNPRINT dataset:

```
%X = #DATE2ND('840730123Ø', 'YYMMDDHHMI')
%X = %X + 1  /* Add 1 day
%X = #ND2DATE(%X, 'MM/DD/YY')
REPORT %X
```

Non-zero values of *%rc*, or terminating conditions, are shown in the following figure (see the discussion in "Run-time errors during standard #function calls" on page 100).

Errors: if *%rc* present, set to corresponding number and set *%dat* to MISSING; if absent, cancel *Fast/Unload* with return code 8.

```
4   fmt is not a valid datetime format.
8   datn is out of range for fmt.
```

Notes

- The inverse of this #function is #DATE2ND.
- "Datetime Formats" on page 172 explains valid datetime formats and valid dates.

6.25 #NM2DATE: Convert number of milliseconds to datetime string

The #NM2DATE function converts a numeric datetime value expressed as the number of milliseconds since January 1, 1900 into a datetime string value according to a specified datetime format string. It requires a datetime numeric value and a datetime format string. An optional output return code argument allows the FUEL program to intercept error conditions.

```
%dat = #NM2DATE(datn, fmt, %rc)
```

where

datn Numeric datetime value expressed as the number of milliseconds since 1 Jan 1900 12:00 AM. This argument may not have the MISSING value.

fmt Datetime format string to use for creating *%dat*.

%rc Return code (optional, output)

%dat Set to datetime string value, in format specified by *fmt*, corresponding to *datn*.

For example, the following fragment prints the string `07/31/84` on the FUNPRINT dataset:

```
%X = #DATE2NM('8407301230', 'YYMMDDHHMI')
%X = %X + 1000 * 60 * 60 * 15   /* Add 15 hours
%X = #NM2DATE(%X, 'MM/DD/YY')
REPORT %X
```

Non-zero values of *%rc*, or terminating conditions, are shown in the following figure (see the discussion in "Run-time errors during standard #function calls" on page 100).

Errors: if *%rc* present, set to corresponding number and set *%dat* to MISSING; if absent, cancel *Fast/Unload* with return code 8.

4 *fmt* is not a valid datetime format.
8 *datn* is out of range for *fmt*.

Notes

- The inverse of this #function is #DATE2NM.
- "Datetime Formats" on page 172 explains valid datetime formats and valid dates.

6.26 #NS2DATE: Convert number of seconds to datetime string

The #NS2DATE function converts a numeric datetime value expressed as the number of seconds since January 1, 1900 into a datetime string value according to a specified datetime format string. It requires a datetime numeric value and a datetime format string. An optional output return code argument allows the FUEL program to intercept error conditions.

```
%dat = #NS2DATE(datn, fmt, %rc)
```

where

datn Numeric datetime value expressed as the number of seconds since 1 Jan 1900 12:00 AM. This argument may not have the MISSING value.

fmt Datetime format string to use for creating *%dat*.

%rc Return code (optional, output)

%dat Set to datetime string value, in format specified by *fmt*, corresponding to *datn*.

For example, the following fragment prints the string `Ø7/31/84` on the FUNPRINT dataset:

```
%X = #DATE2NS('8407301230', 'YYMMDDHHMI')
%X = %X + 60 * 60 * 15   /* Add 15 hours
%X = #NS2DATE(%X, 'MM/DD/YY')
REPORT %X
```

Non-zero values of *%rc*, or terminating conditions, are shown in the following figure (see the discussion in "Run-time errors during standard #function calls" on page 100).

Errors: if *%rc* present, set to corresponding number and set *%dat* to MISSING; if absent, cancel *Fast/Unload* with return code 8.

4	*fmt* is not a valid datetime format.
8	*datn* is out of range for *fmt*.

Notes

- The inverse of this #function is #DATE2NS.
- "Datetime Formats" on page 172 explains valid datetime formats and valid dates.

6.27 #NUM2STR: Convert number to string with decimal point

The #NUM2STR function converts a number to a string with an integer part, followed by an optional decimal point and decimal fraction digits, with control over the number of integer and fraction digits, padding, and rounding. It requires a numeric value. Optional arguments specify the width of the resulting integer digits area, the width of the resulting fraction digits area, rounding vs. truncation and fixed width vs. variable/minimal width integer area, and the leading pad character for the integer digits area.

The final optional argument is an output argument whose absolute value is set to the number of integer characters (after leading zeroes are stripped) in the number, plus 1 if the number is negative. Specifying this argument allows the *Fast/Unload* program to continue if the first argument cannot be represented by the result, either as a result of truncation of the high-order integer digits, or because the first argument can not be converted to a number.

```
%str = #NUM2STR(num, intw, fracw, opt, pad, %intlen)
```

where

num Input number to be converted.

intw Number of characters used to express the integer portion and leading minus sign, if any, of *num*. Optional, must be 0 or more if specified. If omitted, only the characters needed are used to represent the integer part: a leading minus sign, if *num* is negative, is followed by the integer portion with leading zeroes stripped (except if the integer portion is zero, a single digit 0 is used).

If the fourth argument (*opt*) contains the letter 'V', *intw* is the minimal width used for the integer part, and the number of characters needed is used if that exceeds *intw*.

fracw Number of digits used to express the fraction portion of *num*. Optional, must be 0 or more if specified. If this argument is omitted:

- only the characters needed are used to represent the fraction part
- if the fraction part is 0, no decimal point occurs in the result
- if the fraction part is not 0, trailing zeroes are removed

%opt One or two characters, optional, with one choice from either of the following two pairs:

- R (round) or T (trunc)

  R Round up the final digit of the result, if the most significant discarded fraction digit is 5 or more. This is the default.

T Final digit of result unaffected by any discarded fraction digits.

This character has no meaning if argument three (*fracw*) is omitted, since in that case no fraction digits are discarded.

- F (fixed width) or V (variable width)

F The integer portion of the result is fixed width; that is, as many characters are used for the integer (and leading minus sign) as the value of argument two (*intw*). This is the default.

V The integer portion of the result is variable/minimal width. That is, if *w* characters are needed for the integer (and leading minus sign) with leading zeroes removed, then *w* characters are used for the integer if *w*>*intw*; *intw* characters are used otherwise.

This character has no meaning if argument two (*intw*) is omitted, since in that case exactly the characters needed for the integer are used.

pad The leading pad character used to fill the integer portion to the width specified by argument two (*intw*). Optional, must be one character, defaults to blank. If the pad character is blank, a leading minus sign **follows** any blank pad characters; otherwise a leading minus sign **precedes** any pad characters.

%intlen Set to the value *len*, where the absolute value of *len* is the number of characters needed for the integer part of the first argument (*num*), with any leading zeroes stripped (or one zero if the integer part is zero), including one additional character if *num* is negative.

If *%intlen* is returned with a negative value, that indicates truncation of the integer part of *num*; this occurs if all of the following conditions hold:

- Argument two (*intw*) is specified.
- The absolute value of *len* is greater than *intw*.
- *Intw*>0, or *num*<0, or *num*>=1.
- Argument four (*opt*) does not contain the letter 'V'.

If *num* cannot be converted to a number, then *len* is returned as zero, and the result (value of *%str*) of #NUM2STR is the MISSING value.

Otherwise, *len* is positive.

As these rules indicate, *%intlen*, if supplied, allows the *Fast/Unload* program to continue when the first argument is non-numeric or the integer part is truncated in the result. If either of these conditions occur and *%intlen* is omitted, the *Fast/Unload* program is cancelled.

Examples:

```
%N_HUMAN = 6 * 1000 * 1000 * 1000
#NUM2STR(%N_HUMAN) -> "6000000000"

%PI = 3.14159265
#NUM2STR(%PI) -> "3.14159265"
#NUM2STR(%PI, , 3) -> "3.142"
#NUM2STR(%PI, , 3, 'T') -> "3.141"

%PCT7_5 = 7.5/100
#NUM2STR(%PCT7_5) -> "0.075"
#NUM2STR(%PCT7_5, , 4) -> "0.0750"
#NUM2STR(%PCT7_5, 0, , , '*', %W) -> ".075"
             with    ...               %W = 1
#NUM2STR(%PCT7_5, 2, , , '*', %W) -> "*0.075"
             with    ...               %W = 1

%NEGP7_5 = - %PCT7_5
#NUM2STR(%NEGP7_5) -> "-0.075"
#NUM2STR(%NEGP7_5, 1, , , '*') -> "-.075"
#NUM2STR(%NEGP7_5, 2, , , '*') -> "-0.075"
#NUM2STR(%NEGP7_5, 3, , , '*') -> "-*0.075"
#NUM2STR(%NEGP7_5, 3, , , ' ') -> " -0.075"

%TBIL = 6.025
%VISA = 18.5
#NUM2STR(%TBIL, 1, 3, , '0') -> "6.025"
#NUM2STR(%VISA, 1, 3, , '0', %W) -> "8.500"
             with    ...     %W = -2
#NUM2STR(%VISA, 1, 3, 'V', '0', %W) -> "18.500"
             with    ...        %W = 2

%BAD = 'PIZZA'
#NUM2STR(%BAD, , , , , %W) -> MISSING value
        with   ...     %W = 0
```

Non-zero values of *%intlen*, or terminating conditions, are shown in the following figure (see the discussion in “Run-time errors during standard #function calls” on page 100).

Errors: if *%intlen* present, set to corresponding number; if absent, cancel *Fast/Unload* with return code 8.

```
-N  N is the length required for the integer part of num,
    and n>intw and opt does not contain the letter "V".
Ø   Num can not be converted to a numeric value (%str is
    set to MISSING).
**  Negative or non-numeric value for intw or fracw (this
    error always cancels Fast/Unload, regardless of the
    presence of %intlen).
**  Invalid character in opt (this error always cancels
    Fast/Unload, regardless of the presence of %intlen).
**  Length of pad not 1 (this error always cancels
    Fast/Unload, regardless of the presence of %intlen).
```

Note: If you are creating a string to place in the *Fast/Unload* output file, you can also use the PUT with the AS STRING or AS DECIMAL clauses (see "PUT" on page 70).

One difference between #NUM2STR and PUT AS STRING or PUT AS DECIMAL is that all conversion of fractional values to fixed width output formats in the PUT statement causes low order fraction digits to be dropped without rounding, but #NUM2STR offers rounding of dropped low order digits.

This #function is new in *Fast/Unload* version 4.0.

6.28 #N2DATE: Convert number of seconds*300 to datetime string

The #N2DATE function converts a numeric datetime value expressed as the number of seconds*300 since January 1, 1900 into a datetime string value according to a specified datetime format string. It requires a datetime numeric value and a datetime format string. An optional output return code argument allows the FUEL program to intercept error conditions.

```
%dat = #N2DATE(datn, fmt, %rc)
```

where

datn Numeric datetime value expressed as 300 times the number of seconds since 1 Jan 1900 12:00 AM. This argument may not have the MISSING value.

fmt Datetime format string to use for creating *%dat*.

%rc Return code (optional, output)

%dat Set to datetime string value, in format specified by *fmt*, corresponding to *datn*.

For example, the following fragment prints the string `07/31/84` on the FUNPRINT dataset:

```
%X = #DATE2N('8407301230', 'YYMMDDHHMI')
%X = %X + 300 * 60 * 60 * 15   /* Add 15 hours
%X = #N2DATE(%X, 'MM/DD/YY')
REPORT %X
```

Non-zero values of *%rc*, or terminating conditions, are shown in the following figure (see the discussion in "Run-time errors during standard #function calls" on page 100).

Errors: if *%rc* present, set to corresponding number and set *%dat* to MISSING; if absent, cancel *Fast/Unload* with return code 8.

4 *fmt* is not a valid datetime format.
8 *datn* is out of range for *fmt*.

Notes

- The inverse of this #function is #DATE2N.
- "Datetime Formats" on page 172 explains valid datetime formats and valid dates.

6.29 #ONEOF: See if string is in delimited list of strings

The #ONEOF function determines whether a string is found in a delimited list of strings. It requires an input string to search for and a delimited list of strings to search within. It accepts an optional delimiter character, which is used to separate the strings in the list.

```
%test = #ONEOF(str, list, delim)
```

where

str String to find; required.

list String containing words to be searched within; required.

delim Character used to separate strings in *list*; optional. Default is semi-colon (;).

%test Set to 1 if *str* is one of the strings in *list*, delimited by *delim*. Otherwise, *%test* is set to 0.

Examples:

```
#ONEOF('HOW', 'HOW NOW', ' ')  returns  1
#ONEOF('NOW', 'HOW;NOW')       returns  1
#ONEOF('ABC', 'ABC')           returns  1
#ONEOF('', 'HOW;;NOW')         returns  1
#ONEOF('', ';HOW;NOW')         returns  1
#ONEOF('', 'HOW;NOW;')         returns  1
#ONEOF('', 'HOW;NOW')          returns  0
#ONEOF('NO', 'HOW;NOW')        returns  0
#ONEOF('HOW;NOW', 'HOW;NOW')   returns  0
#ONEOF('', '')                 returns  0
```

The *Fast/Unload* run is cancelled with a return code of 8 when:

- *Delim* supplied, and length not 1.

Notes

- If argument 1 (*str*) contains *delim*, 0 is returned.
- If argument 2 (*list*) is the null string, 0 is returned.
- If argument 1 (*str*) is the null string, 1 is returned if, and only if, there is a leading, trailing, or two adjacent copies of, *delim*.

- This #function is similar to the #FIND function (“#FIND: Word position of one word sequence within another” on page 126). Note, however, that both #FIND and #ONEOF are inferior to the SELECT statement, when the only purpose is to test whether an entity has one of several values (see “SELECT entity” on page 80).

This #function is new in *Fast/Unload* version 4.0.

6.30 #PAD: Final substring, preceded by pad characters to specified length

The #PAD function returns a padded final substring from a source string. It expects a source string argument and a numeric length of the output string. It accepts an optional pad character. #PAD returns the string that ends at the last position of the source string and is of the specified output length. If the output length is less than or equal to the length of the source string, the last *length* characters of the source are returned. Otherwise, the source string is padded to the output length by preceding it with sufficient copies of the pad character.

```
%out = #PAD(str, pad, len)
```

where

str Source string.

pad Optional pad character. Default is blank. If supplied, must be a string of length 1.

len Length of the output string. Must be numeric >= 0.

%out If *len* is greater then #LEN(*str*):

- *len* - #LEN(*str*) copies of the pad character, followed by all of *str*

Otherwise:

- substring from *str* starting at position #LEN(*str*) - *len+1* with length *len*

For example:

```
#PAD('ABC', , Ø)        returns  ''
#PAD('ABC', , 1)        returns  'C'
#PAD('ABC', , 3)        returns  'ABC'
#PAD('ABC', , 4)        returns  ' ABC'
#PAD(456, Ø, 4)         returns  'Ø456'
```

The *Fast/Unload* run is cancelled with a return code of 8 when:

- *Len* not numeric, or less than 0.
- *Pad* supplied, and length not 1.

Notes

- This is the same as the #RIGHT function, except for the order of the arguments ("#RIGHT: Final substring, preceded by pad characters to specified length" on page 149).

This #function is new in *Fast/Unload* version 4.0.

6.31 #PADR: Initial substring, followed by pad characters to specified length

The #PADR function returns a padded initial substring from a source string. It expects a source string argument and a numeric length of the output string. It accepts an optional pad character. #PADR returns the string that begins at the first position of the source string and is of the specified output length. If the output length is less than or equal to the length of the source string, the first *length* characters of the source are returned. Otherwise, the source string is padded to the output length by following it with sufficient copies of the pad character.

```
%out = #PADR(str, pad, len)
```

where

str Source string.

pad Optional pad character. Default is blank. If supplied, must be a string of length 1.

len Length of the output string. Must be numeric >= 0.

%out If *len* is greater then #LEN(*str*):

- all of *str*, followed by *len* - #LEN(*str*) copies of the pad character

Otherwise:

- substring from *str* starting at position 1 with length *len*

For example:

```
#PADR('ABC', , 1)      returns  'A'
#PADR('ABC', , 3)      returns  'ABC'
#PADR('ABC', , 4)      returns  'ABC '
#PADR(4.6, 0, 4)       returns  '4.60'
```

The *Fast/Unload* run is cancelled with a return code of 8 when:

- *Len* not numeric, or less than 0.
- *Pad* supplied, and length not 1.

Notes

- This is the same as the #LEFT function, except for the order of the arguments (“#LEFT: Initial substring, followed by pad characters to specified length” on page 130).

This #function is new in *Fast/Unload* version 4.0.

6.32 #REVERSE: Get reverse of string

The #REVERSE function returns a copy of the input string, with the order of the bytes reversed. It expects a source string argument.

```
%out = #REVERSE(str)
```

where

str Source string; required.

%out Copy of *str*, with all the last input byte first, followed by the next to last input byte, etc.

For example:

```
#REVERSE('abc')    returns  'cba'
```

This #function has no cancelling conditions.

This #function is new in *Fast/Unload* version 4.0.

6.33 #RIGHT: Final substring, preceded by pad characters to specified length

The #RIGHT function returns a padded final substring from a source string. It expects a source string argument and a numeric length of the output string. It accepts an optional pad character. #RIGHT returns the string that ends at the last position of the source string and is of the specified output length. If the output length is less than or equal to the length of the source string, the last *length* characters of the source are returned. Otherwise, the source string is padded to the output length by preceding it with sufficient copies of the pad character.

```
%out = #RIGHT(str, len, pad)
```

where

str Source string.

len Length of the output string.

pad Optional pad character. Default is blank. If supplied, must be a string of length 1. Must be numeric >= 0.

%out If *len* is greater then #LEN(*str*):

- *len* - #LEN(*str*) copies of the pad character, followed by all of *str*

Otherwise:

- substring from *str* starting at position #LEN(*str*) - *len+1* with length *len*

For example:

```
#RIGHT('ABC', 0)        returns  ''
#RIGHT('ABC', 1)        returns  'C'
#RIGHT('ABC', 3)        returns  'ABC'
#RIGHT('ABC', 4)        returns  ' ABC'
#RIGHT(456, 4, 0)       returns  '0456'
```

The *Fast/Unload* run is cancelled with a return code of 8 when:

- *Len* not numeric, or less than 0.
- *Pad* supplied, and length not 1.

Notes

- This is the same as the #PAD function, except for the order of the arguments (see "#PAD: Final substring, preceded by pad characters to specified length" on page 144).

This #function is new in *Fast/Unload* version 4.0.

6.34 #SNDX: Create SOUNDEX code for string

The #SNDX function calculates a **SOUNDEX** code for a string, producing the same result as the *Model 204* $SNDX function. It expects a single argument and returns the **SOUNDEX** code for the string value of the argument.

```
%out = #SNDX(str)
```

where

str String, presumed to contain a name.

%out Set to the **SOUNDEX** code corresponding to *str*.

For example, the following fragment will build values for an INVISIBLE KEY field which is the $SNDX value of the field NAME:

```
UAI OINDEX
FOR EACH RECORD
   FOR I FROM 1 TO NAME(#)
      ADD NAME_SNDX = #SNDX(NAME(I))
   END FOR
   UNLOAD
END FOR
```

This #function has no cancelling conditions.

6.35 #STRIP: Remove leading and/or trailing copies of pad character

The #STRIP function removes leading, trailing, or both, copies of a pad character from a string. It requires a string to be stripped. It accepts an optional specification of which characters to strip (Leading, Trailing, or Both). It accepts an optional pad character argument, specifying the character to be stripped.

```
%out = #STRIP(str, B|L|T, pad)
```

where

str String to be stripped; required.

B|L|T Optional indicator of the type of strip:

- 'B...' (any string beginning with uppercase B): to strip **B**oth leading and trailing pad characters.
- 'L...' (any string beginning with uppercase L): to strip **L**eading pad characters only.
- 'T...' (any string beginning with uppercase T): to strip **T**railing pad characters only.

Defaults to **B**.

%out Set to a copy of *str*, with leading, trailing, or both, as specified, copies of *pad* removed.

Examples:

```
#STRIP('  ABC  ')          returns  'ABC'
#STRIP('  ABC  ', 'L')     returns  'ABC  '
#STRIP('  ABC  ', 'T')     returns  '  ABC'
#STRIP('000123', 'L', '0') returns  '123'
```

The *Fast/Unload* run is cancelled with a return code of 8 when:

- *B|L|T* does not begin with either uppercase B, uppercase L, or uppercase T.
- *Pad* specified and length is not 1.

Notes

- This #function is similar to the #DEBLANK function ("#DEBLANK: Remove leading and trailing blanks from substring" on page 124).

This #function is new in *Fast/Unload* version 4.0.

6.36 #SUBSTR: Substring

The #SUBSTR function returns a substring from within a source string. It expects a source string argument and a numeric byte position within the source string. It accepts an optional numeric maximum length of the output substring. #SUBSTR returns the string that begins at the indicated position of the source string and ends either at the end of the source string or when the supplied maximum length has been reached.

```
%out = #SUBSTR(str, pos, len)
```

where

str Source string. For *Fast/Unload* version 4.3 and later. *str* may be longer than 255 bytes.

pos Beginning position within the source string for the desired substring. Must be numeric >= 1.

len Optional maximum length of substring; default is 255. Must be numeric >= 0.

%out Substring from *str* starting at position *pos* with length that is the minimum of *len* and #LEN(*str*) +1 - *pos*.

For example, you might want to get the "right hand half" of a string (it will be to the right of the middle character if the string length is odd):

```
%LEN = #LEN(%STR)
%RIGHT = %LEN / 2 + 1   /* See note below
%RIGHT = #SUBSTR(%STR, %RIGHT)
```

The *Fast/Unload* run is cancelled with a return code of 8 when:

- *Pos* not numeric, or less than 1.
- *Len* not numeric, or less than 0.

Notes

- You can omit the + 1 in the "right hand half" example above if you want to include the middle character, but you must check that the string is not shorter than 2 bytes. If it is shorter, omitting the + 1 will cause your program to terminate, because *pos* must be 1 or more.
- The maximum length of *%out* is #LEN(*str*) - *pos* + 1.

- To take an initial or final substring, you can also use the #functions described in “#LEFT: Initial substring, followed by pad characters to specified length” on page 130 or “#RIGHT: Final substring, preceded by pad characters to specified length” on page 149.

6.37 #TIME: Current time and/or date

The #TIME function returns the current date and time in a datetime string. It accepts an optional datetime format string, with a default that returns just time information. An optional output return code argument allows the FUEL program to intercept error conditions.

```
%tim = #TIME(fmt, %rc)
```

where

fmt Optional datetime format string for *%tim*. Defaults to **HH:MI:SS**.

%rc Return code (optional, output).

%tim Set to datetime string with current date and time, using *fmt*.

For example, the following fragment stores the current time in the last occurrence of the REORG_TIME field:

```
ADD REORG_TIME = #TIME()
```

Non-zero values of *%rc*, or terminating conditions, are shown in the following figure (see the discussion in "Run-time errors during standard #function calls" on page 100).

Errors: if *%rc* present, set to corresponding number and set *%tim* to MISSING; if absent, cancel *Fast/Unload* with return code 8.

4 *fmt* is not a valid datetime format.

Notes

- The only difference between #TIME and #DATE is the default value for *fmt*.
- "Datetime Formats" on page 172 explains valid datetime formats.

6.38 #TRANSLATE: Change characters of string using from/to pairings

The #TRANSLATE function returns a copy of the input string, with all characters which are contained in the input table translated to the corresponding characters in the output table. It expects a source string argument, and at least one of 3 optional string arguments. It accepts optional output table, input table, and pad character (used to extend the output table if it is shorter than the input table).

```
%out = #TRANSLATE(str, tbl_out, tbl_in, pad)
```

where

str Source string; required.

tbl_out String of "to" characters (optional). Default is null string. Trailing pad characters are added to this string, if needed, so that its length is equal to the length of *tbl_in*.

tbl_in String of "from" characters, optional. Defaults to 256 characters consisting of all byte values, in order, that is, X'00010203...FCFDFEFF'.

pad Pad character for *tbl_out* (optional, must be length 1). Default blank.

%out Copy of *str*, with all characters which are contained in the input table translated to the corresponding characters in the output table.

For example:

```
#TRANSLATE('ab', 'x', 'b')    returns  .ax.
#TRANSLATE('ab', , '', 'z')   returns  .ab.
#TRANSLATE('ab', , , 'z')     returns  .zz.
#TRANSLATE('ab', 'AB', , 'z') returns  .zz.
#TRANSLATE('ab', '', , 'z')   returns  .zz.
```

The *Fast/Unload* run is cancelled with a return code of 8 when:

- Only *str* argument supplied.
- *Pad* supplied, and length not 1.

Notes

- As a consequence of the default rules, if neither *tbl_out* nor *tbl_in* are supplied, *%out* is set to as many copies of *pad* as #LEN(*str*).

- If a character occurs more than once in *tbl_in*, all instances of it after the first are ignored.

- As noted, the default *tbl_in* is a **256** byte string; it is not possible to specify such a value in *Fast/Unload*. If you want to specify a translation for all 256 characters, you can do the following:

```
%TBL_OUT = ...  /* A 255 byte value
%TBL_FF  = ...  /* Value that X'FF' translates to
%OUT = #TRANSLATE(%STR, %TBL_OUT, , %TBL_FF)
```

This #function is new in *Fast/Unload* version 4.0.

6.39 #UPCASE: Change lowercase letters of string to uppercase

The #UPCASE function returns a copy of the input string, with all lowercase EBCDIC letters changed to the corresponding uppercase letters. It expects a source string argument.

```
%out = #UPCASE(str)
```

where

str Source string; required.

%out Copy of *str*, with all lowercase EBCDIC letters changed to their uppercase EBCDIC equivalents.

For example:

```
#UPCASE('?abc')    returns  '?ABC'
#UPCASE('?ABC')    returns  '?ABC'
#UPCASE('')        returns  ''
```

This #function has no cancelling conditions.

This #function is new in *Fast/Unload* version 4.0.

6.40 #VERPOS: Position in string of character not in or in list

The #VERPOS function scans a search string from an optional starting position and then finds either the first character that is **N**ot in a list of target characters, or the first character that **M**atches a character in a list of target characters. It expects a string argument that is searched and a string comprising the target characters. It accepts an optional argument indicating the type of search to perform and an optional position at which to start the search. #VERPOS returns the starting position of the identified character, or 0 as follows:

- For the **N**ot matched (default) search, returns the position in the search string of the first character not in the target string, or 0 if all characters in the search string are in the target string.
- For the **M**atched search, returns the position in the search string of the first character which is in the target string, or 0 if no characters in the search string are in the target string.

```
%opos = #VERPOS(search, target, NorM, pos)
```

where

search String to be searched.

target String containing set of target characters.

NorM Optional indicator of the type of search being formed:

- 'N...' (any string beginning with uppercase N): to find position of first character of *search* not present in *target*.
- 'M...' (any string beginning with uppercase M): to find position of first character of *search* present in *target*.

Defaults to **N**.

pos Optional position within *search* to begin scanning. Must be a positive numeric value, default is 1.

%opos Set to position of first character in *search* that is not in ('N') or is in ('M') *target*, at or after the position identified by *pos*.

Examples:

```
#VERPOS('SIRIUS', 'ETANOISHRDLU')  returns    0
#VERPOS('HOMER', 'MRH')            returns    2
#VERPOS('HOMER', 'MRH', 'M')       returns    1
#VERPOS('PLATO', 'AEIOU', , 3)     returns    4
#VERPOS('FRED', '', 'N', 2)        returns    2
#VERPOS('BETTY', '', , 'M', 3)     returns    0
#VERPOS('WILMA', 'AEIOU', 'M', 3)  returns    5
```

The *Fast/Unload* run is cancelled with a return code of 8 when:

- *NorM* does not begin with either uppercase N or uppercase M.
- *Pos* not numeric, or less than 1.

Notes

- If *search* is null, #VERPOS returns 0, as long as the other arguments are not in error. If *pos* is greater than #LEN(*search*), #VERPOS returns 0.
- If *search* is non-null and *target* is null, #VERPOS returns 0 if *NorM* starts with 'M', otherwise #VERPOS returns *pos*.
- The function which performs this service is usually called "verify", but #VERPOS is used to distinguish it from CCA's $VERIFY function:
  - $VERIFY has only a "Boolean" (0 or 1) return; #VERPOS returns a position or 0 to indicate search "failed".
  - $VERIFY only has a "nomatch" type of search.
  - $VERIFY does not have a *pos* argument.
  - $VERIFY returns 0 if there is a character in *search* which is not in *target*; #VERPOS (Nomatch) returns 0 if there is *no* such character.
  - $VERIFY returns "some character in *search* is not in *target*" (0) if *search* is null and *target* is not null; #VERPOS returns "all characters in *search* are in *target*" if *search* is null, whether or not *target* is null.

6.41 #WORD: Return nth blank-delimited word from string

The #WORD function returns the designated blank-delimited word within a string. It requires an input string and a word number within the string.

```
%out = #WORD(str, word)
```

where

str
String containing blank-delimited words; required.

word Number of word within *str* to return; required. Must be numeric greater than or equal to 1.

%out Set to the *word*th blank-delimited word within *str*. If *word* is greater than the number of words in *str*, the null string is returned.

Examples:

```
#WORD('A B C', 1)       returns  'A'
#WORD('A B C', 2)       returns  'B'
#WORD('A B C', 3)       returns  'C'
#WORD('A B C', 4)       returns  ''
#WORD('     ', 1)       returns  ''
```

The *Fast/Unload* run is cancelled with a return code of 8 when:

- *word* is not numeric, or less than 1.

Notes

- The User Language $WORD function also has an optional delimiter character argument, and its "*word*" argument is argument number 3.

This #function is new in *Fast/Unload* version 4.0.

6.42 #WORDS: Count number of blank-delimited word in string

The #WORDS function returns the number of blank-delimited word in a string. It requires an input string.

```
%count = #WORDS(str)
```

where

str String containing blank-delimited words; required.

%count Set to number of word within *str*.

Examples:

```
#WORDS('A B C')      returns  3
#WORDS('ABC')        returns  1
#WORDS('   ')        returns  Ø
```

This #function has no cancelling conditions.

Notes

- The User Language $WORDS function also has an optional delimiter character argument.

This #function is new in *Fast/Unload* version 4.0.

6.43 #X2C: Convert hex representation to character string

The #X2C function converts a string of hexadecimal characters (the digits **0**-**9** and the letters **A**-**F**) to the corresponding byte string. Each two characters in the input string become one character in the result string, with a leading **0** added to input strings of odd length.

```
%str = #X2C(hex)
```

where

hex String to convert; contains only the characters 0-9 and A-F.

%str Set to the string of bytes represented by *hex*.

For example, in

```
%JUNK = #X2C('F1F2F3')
```

%JUNK would be set to the string **123** (EBCDIC X'F1F2F3'), and in

```
%JUNK = #X2C('1Ø2')
```

%JUNK would be set to EBCDIC X'0102' which is a non-displayable string.

The *Fast/Unload* run is cancelled with a return code of 8 when:

- Non-hexadecimal digit character in *hex*.

Notes

- The inverse of this #function is #C2X.
- If the number of characters of *hex* is odd, a leading character 0 is added for the conversion. The User Language $X2C function (http://m204wiki.rocketsoftware.com/index.php/$X2C) requires an even number of characters in *hex*.
- The $X2C function returns the null string if *hex* has an error.
- The User Language $IHEXA function (http://m204wiki.rocketsoftware.com/index.php/$IHexA) converts any non-hex characters in *hex* to 0.

CHAPTER 7 BLOB/CLOB processing considerations

As of version 4.3, *Fast/Unload* includes the ability to operate on BLOB and CLOB (collectively called “Lob”) fields, which were introduced with V6R1 of *Model 204*.

The following *Fast/Unload* 4.3 features support these fields:

- NEW field declaration statements may have a WITH CLOB or WITH BLOB designation (for example, for creating a Lob field by concatenating “old” non-Lob fields). See “NEW statement option for Lobs”.
- #functions may both accept arguments and produce results in excess of 255 bytes. See:
  - “#CONCAT supports long string arguments and result” on page 166
  - “#LEN supports a long string argument” on page 166
  - “#SUBSTR supports a long string argument and result” on page 166
- The contexts where FUEL %variables may contain strings longer than 255 bytes, the statements, #functions, and directives that allow string values longer than 255 bytes, and the contexts where Lob fields may be used are specified (see “Contexts for long strings and Lobs” on page 167).
- A job statistic reports Table E page usage for each Lob field (“Lob statistics” on page 169).

The above features are discussed in the rest of this chapter, at the end of which is a pair of examples that use these features (“Lob field examples” on page 169).

7.1 Statement and #function modifications

The NEW statement and three #functions are changed to define Lob fields and work with strings longer than 255 bytes.

7.1.1 NEW statement option for Lobs

As of version 4.3, the NEW statement (“NEW fieldname [WITH BLOB | CLOB]” on page 64) lets you specify that the new field you are defining is either a BLOB or CLOB field. This is primarily useful for a UAI type unload, allowing you to create values in the new field that are loaded by LAI as Lob occurrences.

The syntax is:

```
NEW fieldname WITH BLOB | CLOB
```

Note: Version 4.3 also introduces a change to the default attributes that are assigned to fields defined with NEW. As of 4.3, the default attributes are NFRV, NKEY, NCOD, UPDATE IN PLACE (formerly, they were FRV, KEY, CODED, UPDATE AT END).

See "Creating a NEW Lob field" on page 169 for an example that uses a Lob option.

7.1.2 #CONCAT supports long string arguments and result

The arguments of #CONCAT ("#CONCAT: Concatenate strings" on page 103) may now be string values that exceed 255 bytes in length (as the contents of %variables or Lob fields).

The result of #CONCAT may now be a string longer than 255 bytes.

There is no compatibility issue with previous use of the #CONCAT function: the maximum length of an argument was 255 bytes, and if the concatenation of the arguments exceeded 255 bytes, the FUEL program was terminated.

See "Creating a NEW Lob field" on page 169 for an example that uses this #function with a long string value.

7.1.3 #LEN supports a long string argument

The first argument of #LEN ("#LEN: Length of string" on page 132) may now be a string value that exceeds 255 bytes in length (as the content of a %variable or Lob field).

There is no compatibility issue with previous use of the #LEN function, because the maximum length of an argument was 255 bytes.

See "Structured unload of Lob field" on page 170 for an example that uses this #function with a long string value.

7.1.4 #SUBSTR supports a long string argument and result

The first argument of #SUBSTR ("#SUBSTR: Substring" on page 154) may now be a string value that exceeds 255 bytes in length (as the content of a %variable or Lob field).

The result of #SUBSTR may now be a string longer than 255 bytes.

There is no compatibility issue with previous use of the #SUBSTR function, because the maximum length of an argument was 255 bytes.

See “Structured unload of Lob field” on page 170 for an example that uses this #function with a long string value.

7.2 Contexts for long strings and Lobs

The version 4.3 *Fast/Unload* accommodations for Lob fields include allowing %variables to contain strings longer than 255 bytes and specifying the contexts that allow such strings and Lob fields.

7.2.1 %Variables containing strings longer than 255

The value of a %variable may be a string longer than 255 bytes. This can arise as the result of:

%v1 = %v2	assignment from another %variable which contains a string longer than 255 bytes
%v = fld	assignment from a Lob field
%v = #SUBSTR(...)	assignment from a substring of a string value longer than 255 bytes
%v = #CONCAT(...)	assignment from the concatenation of strings, whose lengths total more than 255 bytes

“Permitted use of long string values” specifies the contexts in which a %variable may be used if it contains a string longer than 255 bytes.

7.2.2 Permitted use of long string values

A long string value may be used in the following contexts:

- as an argument of #CONCAT
- as the argument of #LEN
- as the first argument of #SUBSTR
- as the right hand side of an assignment to a %variable
- as the right hand side of a CHANGE or ADD[C] statement, when the field on the left hand side is a Lob

If the **value** of a %variable is used in any other context, and it is a string longer than 255 bytes, the FUEL program is terminated. For example, the following program creates one line of output, because the PUT statement does not allow a %variable containing a string longer than 255:

```
OPEN MYFILE
%X = #LEFT('ABC', 150, 'Z')
PUT %X      /* Length is 150
OUTPUT
%X = #CONCAT(%X, %X)   /* Length is 300
PUT %X      /* FUEL program will be cancelled here
OUTPUT
FOR EACH RECORD        /* Make it a legal FUEL program
END FOR
```

Other examples of contexts prohibiting a %variable containing a string longer than 255 bytes include arithmetic expressions, comparisons in the IF statement, and more.

Note that, since the EXISTS and MISSING clauses of the IF and ELSEIF statements do not reference the value of a %variable, you may use them to test a %variable even if it contains a string longer than 255 bytes. That is, the following statement is acceptable in all cases:

```
IF %S MISSING THEN   /* OK even if #LEN(%S) > 255
```

7.2.3 Permitted use of Lobs

The value of a Lob field may only be used in the contexts discussed above that allow a string longer than 255 bytes, even if the actual length of the Lob field occurrence does not exceed 255. Use of a Lob field in an invalid context causes the compilation of the FUEL program to fail; it never begins execution.

There are four contexts in which any field, Lob or not, may be referenced:

- The UNLOAD(C) statement
- The EXISTS and MISSING clauses of an IF/ELSEIF statement
- The #IF/#ELSEIF directives
- Preceding the number sign (#) “qualifier,” which specifies the number of occurrences of the field

  For example, the following statement is valid for any type of field:

  ```
  FOR I FROM 1 TO BLOB(#)   /* OK for any field
  ```

7.3 Lob statistics

If you display field statistics in the *Fast/Unload* job statistics, the total number of pages used in Table E is shown for each Lob field on the second line of the field's display.

Note that the length statistics given for a Lob field, just like other fields, is based on the field occurrence values: in this case, the number of bytes in Table E **used** by each field occurrence value (that is, unused bytes in Table E pages are not included in the length statistics).

The **Table B** usage for a Lob field is:

- 27 bytes for a non-preallocated Lob field occurrence (in addition to the overhead, as usual, for a count byte and field code)
- 28 bytes for a preallocated Lob field occurrence

For more information about the field statistics, see “FSTATS [AVGTOT | MINMAX]” on page 50.

7.4 Lob field examples

7.4.1 Creating a NEW Lob field

The following example unloads file PRODFILE such that, when it is reloaded, all occurrences of field COMMENT are combined into a single Lob field named ALLCOMMENTS:

```
OPEN PRODFILE
UAI OINDEX
NEW ALLCOMMENTS WITH BLOB
FOR EACH RECORD
IF COMMENT EXISTS THEN
   %X = '' /* Initialize BLOB value
   FOR I FROM 1 TO COMMENT(#)
      %X = #CONCAT(%X, COMMENT)
      DELETE COMMENT
   END FOR
   ADD ALLCOMMENTS = %X
END IF
UNLOAD
END FOR
```

Note that the first occurrence of COMMENT is used in each iteration of the FOR I loop; when it is deleted at the tail of the loop, the occurrence after it becomes the first occurrence on the next iteration.

7.4.2 Structured unload of Lob field

The following example unloads file PRODFILE, creating one output record for each 255 bytes (the maximum for a PUT statement) of the Lob field named ALLCOMMENTS:

```
OPEN PRODFILE
FOR EACH RECORD
PUT '* '
PUT CUSTOMER_ID
OUTPUT
IF ALLCOMMENTS EXISTS THEN
   %COM = ALLCOMMENTS
   %LENGTH = #LEN(ALLCOMMENTS)
   %I = 1
   %LIM = %LENGTH - 254
   REPEAT
      IF +%I >= %LIM THEN
      LEAVE REPEAT
      END IF
      %X = #SUBSTR(%COM, %I, 255)
      PUT %X
      OUTPUT
      %I = %I + 255
   END REPEAT
      %X = #SUBSTR(%COM, %I)
      PUT %X
      OUTPUT
END IF
END FOR
```

Important notes:

- The **plus sign** (+) in `IF +%I >= %LIM` is very important — otherwise a string comparison will be done, which is not correct. For example, if the length is 1,000,000, the first 255 bytes would be unloaded and the final PUT will fail, because then the length of `%X` would be 1,000,000-254.

- The above approach is *vastly superior* to an approach that uses something like `%COM = $SUBSTR(#COM, 256)` to repeatedly remove the first 255 bytes, because that would involve unnecessary copying on the order of the square of the number of bytes in each field.

CHAPTER 8 Datetime Processing Considerations

This chapter presents date processing issues, including usage of *Fast/Unload* past the year 1999, an explanation of its processing of dates, and any rules and restrictions you must follow to achieve correct results using date values with *Fast/Unload*.

Fast/Unload uses dates in the following ways:

- To examine the CPU clock (as returned by the STCK hardware instruction) to determine the current date, in case *Fast/Unload* is under a rental or trial agreement
- As arguments to various #functions, and returned values from them

Please note that in addition to the above date processing performed by *Fast/Unload*, it also unloads *Model 204* files and allows manipulation of other values which might contain two-digit year date values. The customer must ensure that any application using that data has an algorithm or rule for unambiguously determining the correct century for the values.

For example, the UAI statement with the SORT clause allows you to sort by a *Model 204* field; if you are sorting by a two-digit year date field, you need to supply information to enable the sort program to determine the century. You can do this using the FORMAT keyword in the UAI SORT items, as described in "UNLOAD ALL INFORMATION or UAI" on page 88.

For headers on pages or rows that occur on printed pages or displayed screens, Sirius Software products generally use a full four-digit year format, although they may display dates with two-digit years in circumstances where the proper century can be inferred from the context.

You must examine all uses of date values in your applications to ensure that each of your applications produces correct results. Furthermore, both the operating system and *Model 204* must correctly process and transmit dates beyond 1999 in order for *Fast/Unload* to operate properly.

Most Sirius date processing involves the use of datetime #functions. Occasionally, we refer to the "#DATExxx" functions; this is meant to also include #TIME and the #Nxxx2DATE functions.

In operational terms, there are two classes of datetime #functions:

1. #Functions using a numeric value to represent a datetime, where 0 represents 12:00 AM, 1 January 1900; for example, #DATE2NM and #NM2DATE (number of milliseconds since the start of 1900).

These #functions perform **non-strict** matching of date strings to date formats; for example, a leading blank is allowed for the HH token.

2. Other #functions that only manipulate strings and associated datetime formats; for example, #DATECHG (add number of days to given date).

   These #functions perform **strict** matching of date strings to date formats; for example, a leading blank is **not** allowed for the HH token. These #functions generally produce the same results as CCA $DATExxx functions, with additional enhancements.

See "Strict and non-strict format matching" on page 179 for a discussion of strict and non-strict format matching, including a technique for accomplishing strict date checking using the non-strict #functions.

Notes:

- All #DATExxx functions that can have argument errors (that is, all #functions except #DATEFMT) accept an optional "return code" argument. If an argument error occurs and the return code argument is absent, *Fast/Unload* terminates; if the return code argment is present, an error will set the return code to a non-zero number and the result of the #function is the MISSING value.

  The User Language $DATExxx and $SIR_DATExxx functions take a different approach to error handling; each uses a special return value (or class of values) to indicate an argument error.

- The default format for #DATE is "YYYY-MM-DD"; the default for $DATE and $SIR_DATE is "YY-MM-DD".

The rest of this chapter contains a discussion of datetime formats, valid datetime strings, processing of two-digit year values, and datetime error handling. It also contains example datetime formats and corresponding example datetime strings. Finally, there is a list of benefits of Sirius datetime processing.

8.1 Datetime Formats

The representation of a date is determined by a *datetime format*. This value is a character string, composed of the concatenation of tokens (for example, "YYYY" for a four-digit year, and "MI" for minutes) and separator characters (for example, "/" in "MM/DD/YY" for two-digit month, day, and year separated by slashes).

These *datetime format* strings are used in many products in addition to *Fast/Unload*. The products using datetime format strings are:

- *Fast/Unload*

- *Janus Open Client*
- *Janus Open Server*
- *Janus Specialty Data Store*
- *Janus Web Server*
- *SirDBA*
- *Sirius Functions*
- *Sir2000 Field Migration Facility*
- *Sir2000 User Language Tools*

The rules for these *datetime format* strings are consistent throughout all these products, though certain uses of these strings might impose extra restrictions. For example, a leading blank is allowed for the HH, DD, and MM parts of a date argument using a non-strict date #function, such as #DATE2NS, but is not allowed for the strict date #functions.

There are certain rules applied to determine if a format is valid. The basic rules are:

1. If a format string contains a numeric datetime token (that is "ND", "NM", or "NS"), then the format string must consist of only one token. Numeric datetime tokens are only supported in format strings for the *Sir2000 Field Migration Facility*.

2. You must specify at least one time, weekday, or date token.

3. Except for "weekday", you can't specify redundant information. More specifically this means

   - Except for "I", no token can be specified twice.
   - At most one year format (contains Y) can be specified.
   - At most one month format (contains MON, Mon, or MM) can be specified.
   - At most one day format (DD or Day) can be specified.
   - At most one weekday format (WKD, Wkd, WKDAY, or Wkday) can be specified.
   - If AM is specified, then PM can not be specified.
   - At most one fractions-of-a-second format (contains X) can be specified.
   - If DDD is specified, then neither a day nor month format can be.

4. If ZYY is specified in a format string, no other token that denotes a variable-length value may be used.

5. If a format string contains other tokens that denote variable length values, then an * token may only appear as the last character of the format string.

6. The DAY token may not be immediately followed by another token whose value may be numeric, regardless of whether the following token repsents a variable length value. Thus, DAY may not be followed by *, I, YY, YYYY, CYY, MM, HH, MI, SS, X, XX, or XXX; DAY may not be followed by a decimal digit separator, and DAY may not be followed by a quote followed by a decimal digit.

7. When a pair of format strings are used for transforming date values, for example for #DATECNV or processing of updates to SIRFIELD RELATEd fields, additional rules apply to the pattern matching tokens:

   - If one of the format strings includes one or more "I" tokens, then the other format string must contain the same number of "I" tokens. Note that the placement of "I" tokens within the format strings is not restricted. The "I" tokens are processed left to right, with each character from the input string that corresponds to the nth "I" token in the input format being copied unchanged to the character position in the output string that corresponds to the nth "I" token in the output format.

   - If one of the format strings contains an asterisk (*) token, then the other format string must also contain an asterisk token. All of the characters from the input string that correspond to the asterisk token in the input format, if any, are copied unaltered to the output string, begining in the position that corresponds to the asterisk token in the output format.

   SIRFIELD is part of the *Sir2000 Field Migration Facility*.

8. The maximum length of a format string is 100 characters.

Note: A common mistake is to use "MM" for minutes; it should be "MI".

The valid tokens in a date format are shown in the following list. In general, the output format rule for a token is shown. For some of the #functions, the input format rule for a token is the same as the output format rule; this is the definition of "strict date format matching." However, non-strict #functions sometimes allow a string to match a token on input that would not be produced by that token on output.

All of the tokens that match alphabetic strings (for example, "MON") match any case for non-strict matching. All other tokens that have differing strict and non-strict matching rules are listed under "Special date format rules" in the index at the back of the manual, and usage notes for them are contained in “Datetime and format examples” on page 180. Each input datetime format argument in the description of a #function specificies whether the use of the format observes strict or non-strict format matching. See “Strict and non-strict format matching” on page 179.

NM numeric datetime value containing the number of milliseconds (1/1000 of a second) since January 1, 1900 at 12:00 AM. (This token is allowed only in the *Sir2000 Field Migration Facility*.)

NS	numeric datetime value containing the number seconds since January 1, 1900 at 12:00 AM. (This token is allowed only in the *Sir2000 Field Migration Facility*.)
ND	numeric date value containing the number of days since January 1, 1900. (This token is allowed only in the *Sir2000 Field Migration Facility*.)
*	Ignore entire variable-length substring matching pattern, if any, when only retrieving a date value. Substitute with null string when only creating a date value. When copying date values, copy entire variable-length substring matching pattern, if any, from input value to location identified by * token in output string. See “Datetime and format examples” on page 180.
I	Ignore corresponding input character when only retrieving a date value. Store a blank in corresponding output character when only creating a date value. When copying date values, copy each character matching an I token from from the input value to location in the output string identified by the corresping I token in the output format. See “Datetime and format examples” on page 180.
"	Following character is "quoted", that is, it acts as a separator character. See “Datetime and format examples” on page 180.
YYYY	Four-digit year
YY	Two-digit year
CYY	Year minus 1900 (three digits, including any leading zero). See “Datetime and format examples” on page 180.
ZYY	Year minus 1900, two-digit or three-digit year number, excluding any leading zero (variable length data). Non-strict #functions allow a three-digit number with leading zero on input, but any number less than 100 always produces a two-digit number on output. See “Datetime and format examples” on page 180.
MONTH	Full-month name (uppercase variable length). Non-strict #functions allow any mixture of uppercase and lowercase on input, but all uppercase is always produced on output.
Month	Full-month name (mixed-case variable length). Non-strict #functions allow any mixture of uppercase and lowercase on input, but an initial uppercase letter followed by all lowercase is always produced on output.
MON	Three-character month abbreviation (uppercase). Non-strict #functions allow any mixture of upper and lowercase on input, but all uppercase is always produced on output.
Mon	Three-character month abbreviation (mixed case). Non-strict #functions allow any mixture of upper and lower case on input, but initial upper case letter followed by all lowercase is always produced on output.
MM	Two-digit month number. Non-strict #functions allow a two-character number with leading blank on input, but two decimal digits are always produced on output. See “Datetime and format examples” on page 180.
BM	Two-character month number; if less than 10, first character is blank. Non-strict #functions allow a two-digit number with leading zero on input, but any number less than 10 always produces a blank followed by a decimal digit on output. See “Datetime and format examples” on page 180.
DDD	Three-digit Julian day number

DD Two-digit day number. Non-strict #functions allow a two-character number with leading blank on input, but two decimal digits are always produced on output. See "Datetime and format examples" on page 180.

BD Two-character day number; if less than 10, first character is blank. Non-strict #functions allow a two-digit number with leading zero on input, but any number less than 10 always produces a blank followed by a decimal digit on output. See "Datetime and format examples" on page 180.

DAY One-digit or two-digit day number (variable length data). Non-strict #functions allow a two-digit number with leading zero on input, but any number less than 10 always produces a one-digit number on output. See "Datetime and format examples" on page 180.

WKDAY Full day-of-week name (uppercase variable length). Non-strict #functions allow any mixture of uppercase and lowercase on input, but all uppercase is always produced on output.

Wkday Full day-of-week name (mixed-case variable length). Non-strict #functions allow any mixture of uppercase and lowercase on input, but initial upper case letter followed by all lowercase is always produced on output.

WKD Three-character day-of-week abbreviation (uppercase). Non-strict #functions allow any mixture of uppercase and lowercase on input, but all uppercase is always produced on output.

Wkd Three-character day-of-week abbreviation (mixed case). Non-strict #functions allow any mixture of uppercase and lowercase on input, but initial upper case letter followed by all lowercase is always produced on output.

HH Two-digit hour number. Non-strict #functions allow a two-character number with leading blank on input, but two decimal digits are always produced on output. See "Datetime and format examples" on page 180.

BH Two-character hour number; if less than 10, first character is blank. Non-strict #functions allow a two-digit number with leading zero on input, but any number less than 10 always produces a blank followed by a decimal digit on output. See "Datetime and format examples" on page 180.

MI Two-digit minute number

SS Two-digit second number

X Tenths of a second

XX Hundredths of a second

XXX Thousandths of a second (milliseconds)

AM AM/PM indicator

PM AM/PM indicator

The valid separators in a date format are:

blank (" ")
apostrophe ("'")
slash ("/")
colon (":")
hyphen ("-")
back slash ("\")
period (".")
comma (",")

underscore ("_")
left parenthesis ("(")
right parenthesis (")")
plus ("+")
vertical bar ("|")
equals ("=")
ampersand ("&")
at sign ("@")
sharp ("#")
the decimal digits ("0" - "9").

In addition, any character may be a separator character if preceeded by the quoting character (").

See "Datetime and format examples" on page 180 for examples which include use of various separator characters.

8.2 Valid Datetimes

For a datetime string to be valid it must meet the following criteria:

- Its length must be less than 128 characters.
- It must be compatible with its corresponding format string.
- It must represent a valid date and/or time. For example, at most 23:59:59.999 for a time, 01-12 for a month, 01-31 or less (depending on the month) for a day, February 29 is only valid in leap years (only centuries divisible by 4 are leap years: 2000 is but neither 1800, 1900, nor 2100 are).

  Note: Weekdays are not checked for consistency against the date; for example, both Saturday, 02/15/97 and Friday, 02/15/97 are valid.
- It must be within the date range allowed for the corresponding format. A datetime string used with a CYY or ZYY format can only represent dates from 1900 to 2899, inclusive. A datetime string used with a YY format can only represent dates in a range of 100 or less years, as determined by CENTSPAN and SPANSIZE. The valid range of dates for all other formats is from 1 January 1753 thru 31 December 9999.

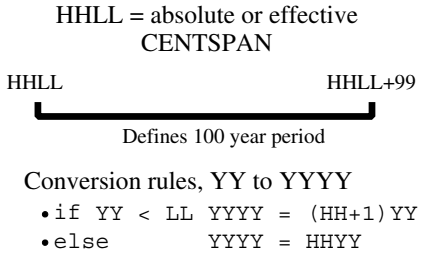
8.3 Processing Dates With Two-Digit Year Values

A date field with only two digits for the year value is capable of representing a range of up to one hundred years. When we compare a pair of two-digit year values we are accustomed to thinking of the century as fixed, so that all dates are either "19xx" or "20xx". However, a date field with two-digit year values can actually represent dates from two different centuries, provided that the *range* of dates does not exceed 100 years.

8.3.1 CENTSPAN

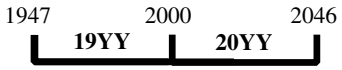
CENTSPAN provides a mechanism for unambiguously converting dates with two-digit year values into dates with four-digit year values. The CENTSPAN mechanism allows two-digit year values to span two centuries without confusion. CENTSPAN identifies the four-digit year value that is the *start* of a range of years represented by the two-digit year values.

CENTSPAN may be specified as an *absolute* unsigned four digit value between 1753 and 9999, or it may be specified as a *relative* signed value between -99 and +99, inclusive. A relative CENTSPAN value is dynamically converted to an *effective* absolute value before it is used to perform a YY to YYYY conversion. The effective CENTSPAN value is formed by adding the relative CENTSPAN to the current four-digit year value at the time the relative value is converted.



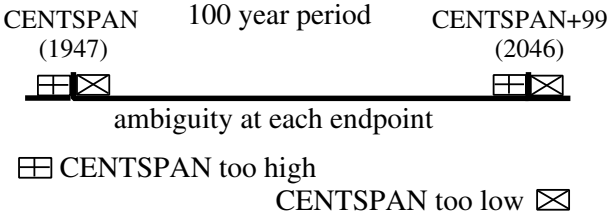
Example:

```
CENTSPAN = -50
current date = 1997
effective CENTSPAN = 1947
```



A simple algorithm is used to convert a two-digit year value (YY) to a four-digit year value, using a four-digit absolute or effective CENTSPAN value (HHLL). If the two-digit year value is less than the low-order two digits of the CENTSPAN value, then the resulting century is one greater than the high-order two digits of the CENTSPAN value. Otherwise the resulting century is the same as the high-order two digits of the CENTSPAN value.

Using all one hundred available years for mapping two-digit year values can cause significant confusion and result in data integrity errors: dates just above and just below the 100-year window are mapped to the other end of the window. From the previous example, the date "47" will be intepreted as 1947, when it could have conceivably been 2047. Similarly, the date "46" will be intepreted as 2046, when it might have been 1946.



If CENTSPAN is set to a value that is too high, dates that are just prior to CENTSPAN will appear to occur 100 years hence. If CENTSPAN is set to a value that is too low, dates that fall just after CENTSPAN+99 will appear to have occured 100 years earlier. A full one-hundred year window also can not detect attempts to represent more than one hundred years of values with a two-digit year.

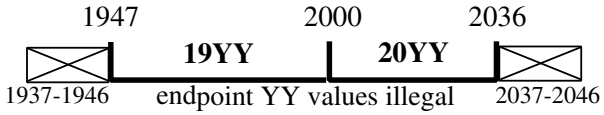
8.3.2 SPANSIZE

There is a method to protect from the ambiguities that can occur at each end of the 100-year window defined by CENTSPAN. SPANSIZE is used to restrict the size of the window used for mapping two-digit year values. The effect is to create two *guard bands*, one just below the date window and one just above. An attempt to represent a date value that lands in a guard band produces an error.

Each guard band contains CENTSPAN-SPANSIZE years, hence a SPANSIZE of 100 removes the protection. SPANSIZE is a value which you can customize in your load module; see "CENTSPAN and SPANSIZE" on page 294. If you do not customize it, the value of SPANSIZE is 90, which provides protection for two ten year windows: one below the CENTSPAN setting and one starting at CENTSPAN+90. Note that in *Fast/Unload* version 3.0, SPANSIZE is 100 (and it can not be customized). From our previous example:

Example:

```
CENTSPAN = -50
SPANSIZE = 90
current date = 1997
```



An attempt to represent the values "37" through "46" will be rejected. This protects the range 1937 through 1946 as well as the range 2037 through 2046. Note that an intended value of 2047, expressed as "47" will be accepted and interpreted as 1947. In general a smaller SPANSIZE provides the highest assurance of correct mappings. However, any setting of SPANSIZE less than 100 will probably detect the case where a range greater than one hundred years is being used.

8.4 Strict and non-strict format matching

As mentioned in "Datetime Formats" on page 172, for some of the #functions, the input format rule for a token is the same as the output format rule; this is the definition of "strict date format matching". However, non-strict #functions sometimes allow a string to match a token on input that would not be produced by that token on output. The types of strict matching are as follows:

Alpha tokens For alphabetic tokens (for example, **Month**), a strict match requires the input value to be the correct case. For example, the "MON" token is strictly matched by "JAN" but not by "Jan", and the reverse is true for the "Mon" token. For non-strict matching, the alphabetic tokens are matched by any combination of uppercase and lowercase input.

HH, MM, DD For these tokens, a strict match requires a leading zero for values less than 10. For non-strict matching, a value less than 10 can also be represented by a leading blank followed by a single numeric digit.

BH, BM, BD For these tokens, a strict match requires a leading blank for values less than 10. For non-strict matching, a value less than 10 can also be represented by a leading zero followed by a numeric digit.

DAY For this token, a strict match requires a single digit for values less than 10. For non-strict matching, a value less than 10 can also be represented by a leading zero followed by a numeric digit.

ZYY For this token, a strict match requires two digits for values less than 100. For non-strict matching, a value less than 100 can also be represented by a leading zero followed by a two numeric digits.

If you want to check a datetime string using strict rules, you can use the following technique with the non-strict date #functions:

```
IF <date> EQ '' OR <date> NE #NM2DATE(-
   #DATE2NM(<date>, <fmt>), -
   <fmt>) THEN
   <error handling>
END IF
```

8.5 Datetime and format examples

There is an extensive set of format tokens, as shown in “Datetime Formats” on page 172. These tokens and the various separator characters can be combined in almost limitless possibility, giving rise to an extremely large set of datetime formats. This section provides examples of some common datetime formats, and also tries to explain the use of some of the format tokens which might not be obvious. It also has examples for formats which have usage with the *Fast/Unload* which differs from their usage with other Sirius products. These are noted in the examples and are indexed at the back of this manual under the heading “Special date format rules”. Each example format is explained and also presented with some matching datetimes; again, bear in mind that these tokens can be combined in very many ways and only a very few are shown here. It is assumed that these examples are invoked sometime between the years 1998-2040, as the basis for relative CENTSPAN calculations.

YYMMDD This is the common 6-digit date format which supports sort order if all dates are within a single century. The following FUEL fragment

```
%X = #DATE2ND('960229', 'YYMMDD')
IF %X > -9.E12 THEN
   REPORT 'OK'
END IF
```

prints the value “OK”.

YYYYMMDD

This is the common 8-digit date format which supports sort order with dates in 2 centuries. The following FUEL fragment

```
%N = #DATE2ND('921212', 'YYMMDD')
%N = #ND2DATE(%N, 'YYYYMMDD')
REPORT %N
```

prints the value 19921212.

MM/DD/YY

This is the U.S. 6-digit date format for display. The following FUEL fragment

```
%X = #DATE2ND('12/14/94', 'MM/DD/YY')
IF %X > -9.E12 THEN
   REPORT 'OK'
END IF
```

prints the value “OK”.

Notes:

- With non-strict format matching, such as #DATE2ND, the leading zero corresponding to an MM token may be given as a blank, thus allowing “ 7/15/98”. With strict matching, however, such leading blank is not allowed for MM; a leading blank month value with a strict #function requires the BM token. If the data contains leading zeroes in some month instances and leading blanks in others, you must use a non-strict #function. The BM token is available starting with version 3.2 of *Fast/Unload*.

DD.MM.YY

This is a European 6-digit date format for display. The following FUEL fragment

```
%X = #DATE2ND('14.12.94', 'DD.MM.YY')
IF %X > -9.E12 THEN
   REPORT 'OK'
END IF
```

prints the value “OK”.

Notes:

- With non-strict format matching, such as #DATE2ND, the leading zero corresponding to a DD token may be given as a blank, thus allowing “ 1.01.00”. With strict matching, however, such leading blank is not

allowed for DD; a leading blank day value with a strict #function requires the BD token. If the data contains leading zero days in some instances and leading blanks in others, you must use a non-strict #function. The BD token is available starting with version 3.2 of *Fast/Unload*.

Wkday, DAY Month YYYY "A"T HH:MI

This is a format which could be used for report headers. The following FUEL fragment

```
%N = #DATE -
  ('Wkday, DAY Month YYYY "A"T HH:MI')
REPORT %N
```

prints a value like “Friday, 7 February 1998 AT 21:33”.

Notes:

- If an input format contains AM or PM, then the time (HH:MI) must be between 00:01 and 12:00 and must be accompanied by either AM or PM.
- If an input format contains DAY (e.g., “DAY MON YY”) with non-strict format matching, such as #DATE2ND, the string matching it may have a leading zero, thus allowing “06 MAY 98”. With strict matching #functions however, such leading zero is not allowed for DAY; a single digit must be supplied for days 1 through 9.
- If an input format contains HH with non-strict format matching, such as #DATE2ND, the string matching it may have a leading blank, thus allowing “ 8:30”. With strict matching, however, such leading blank is not allowed for HH; a leading blank hour value with a strict #function requires the BH token. If the data contains leading zero hours in some instances and leading blanks in others, you must use a non-strict #function. The BH token is available starting with version 3.2 of *Fast/Unload*.

YYIIII

This is a format which could be used for data which contains a 2-digit year prefixing other information, such as a sequence number. The following FUEL fragment

```
%D = #DATE2ND('92ABCD', 'YYIIII')
%D = %D + 10*365.25 + .8
%N = #ND2DATE(%D, 'YY')
REPORT %N
```

prints the value “02”.

Note:

- When a pair of format strings are used for transforming date values, e.g. for #DATECNV or processing of updates to SIRFIELD RELATEd fields, both formats must have the same number of I tokens.

  SIRFIELD is part of the *Sir2000 Field Migration Facility*.

YY*

This is a format which could be used for data which contains a 2-digit year prefixing other information, such as a sequence number, when the other information is variable length. The following FUEL fragment

```
%X = #DATE2ND('92', 'YY*')
IF %X > -9.E12 THEN
   REPORT 'OK'
END IF
%X = #DATE2ND('1992ABC', 'YYYY*')
IF %X > -9.E12 THEN
   REPORT 'OK'
END IF
```

prints the values “OK” and “OK”.

Notes:

- At most one occurrence of the * token may appear in a datetime format.
- When a pair of format strings are used for transforming date values, e.g. for #DATECNV or processing of updates to SIRFIELD RELATEd fields, then if a * token appears in one of the formats, a * must also appear in the other format.

  SIRFIELD is part of the *Sir2000 Field Migration Facility*.

CYYDDD

This is a compact 6-digit date format with explicit century information, from 1900 through and including 2899. The following FUEL fragment

```
%X = #DATE2ND('Ø97Ø31', 'CYYDDD')
IF %X > -9.E12 THEN
   REPORT 'OK'
END IF
```

prints the value “OK”.

ZYYMMDD

This is a compact 6- or 7-digit date format with explicit century information, from 1900 through and including 2899, that can often be used with “old” YYMMDD date values in the 1900's. The following FUEL fragment

```
* Check 1 Dec, 1997:
%X = #DATE2ND('971201', 'ZYYMMDD')
IF %X > -9.E12 THEN
   REPORT 'OK'
END IF
* Check 1 Dec, 2000:
%X = #DATE2ND('1001201', 'ZYYMMDD')
IF %X > -9.E12 THEN
   REPORT 'OK'
END IF
```

prints the values “OK” and “OK”.

Notes:

- With non-strict format matching (such as #DATE2ND), a three digit number with a leading zero may correspond to a ZYY token, thus allowing “0971201”. With strict matching, however, a 3 digit value with leading zero is not allowed for ZYY; a 3-digit value less than 100 with a strict #function requires the CYY token. If the data contains values less than 100 as 3 digits in some instances and as 2 digits in others, you must use a non-strict #function.

YY0000 Decimal digits can be used as separator characters. The following FUEL fragment

```
%N = #DATE2ND('92000', 'YY000')
%N = #ND2DATE(%N, 'YYYY"N"A')
REPORT %N
```

prints the value “1992NA”.

Notes:

- Numeric separators, unlike alphabetic separators, do not need to be preceeded by a quote character (").
- Numeric separators are available starting with version 3.2 of *Fast/Unload*.

8.6 Datetime Error Handling

Due to an invalid argument value to a datetime #function, any of the following errors can occur:

- invalid datetime format specification
- datetime string not matching format

- datetime out of range for the format
- invalid CENTSPAN value
- datetime out of range for CENTSPAN/SPANSIZE combination

One way to detect these errors is to check for the appropriate error return value:

1. # Functions using a numeric value to represent a datetime, and #TIME and #DATE, have error return values of -9.E12 or a null string for numeric or string result #functions, respectively.

2. # Functions (other than #TIME and #DATE) that only manipulate strings and associated datetime formats have error return values of a variable number of asterisks (or, in the case of #DATEDIF the value 99,999,999).

Most of the standard #DATExxx functions have an optional output "return code" argument (see “Run-time errors during standard #function calls” on page 100). If you specify, for example, an invalid CENTSPAN argument and you specify the return code argument, you can test the return code for CENTSPAN errors. If you specify an invalid CENTSPAN argument and you do not specify the return code argument, the *Fast/Unload* run terminates with an error message indicating the type of error and the line number being executed; the argument values are dumped as well.

8.7 #DATExxx Functions CENTSPAN Argument

Many of the #DATExxx functions accept an optional argument containing a CENTSPAN value to be used for the call. The default value of any CENTSPAN argument is -50. You can customize the default value of CENTSPAN in your load module; see “CENTSPAN and SPANSIZE” on page 294. Note that in version 3.0 of *Fast/Unload*, the default CENTSPAN argument can not be customized. The default value should be adequate in most cases; if you have carefully determined it should be different in some application, code the value on the relevant #function invocations.

For a different approach, see the description of the CENTSPLT and DEFCENT parameters (for example, **http://m204wiki.rocketsoftware.com/index.php/CENTSPLT_parameter**) and $function arguments.

Note that the CENTSPAN argument may not be specified as an entity whose value is MISSING. For most #function numeric arguments, the MISSING value is allowed if a value of zero for the argument is allowed. Zero is allowed for CENTSPAN, but since it is an unusual CENTSPAN value, the MISSING value may not be supplied.

8.8 Benefits of Sirius datetime processing

Following is a list of benefits offered by Sirius datetime processing. To provide concrete comparisons, there are some references to the standard User Language date $functions.

SPANSIZE

The SPANSIZE processing creates a very strong barrier to detecting otherwise un-noticed 2-digit year processing errors. This is unique to Sirius datetime processing.

Relative CENTSPAN

The relative CENTSPAN specification (for example, "-50") allows you to maintain a flexible "rolling" window for 2-digit year processing.

Default CENTSPAN

One significant advantage of a relative CENTSPAN is that it allows the default (**-50**) of a reasonable value without parameter changes in all batch and online jobs.

Format tokens

There is a very large set of tokens in the Sirius datetime formats. For example, there are 4 different tokens representing the day of the week, and time of day can be represented. Standard User Language date formats do not have any day of week nor time of day tokens, and other standard User Language token variations, for example, CYY vs. ZYY, is done by a complex argument setting.

Pattern match tokens

The Sirius datetime formats can contain single-character ("I") or variable length character ("*") match-any tokens in datetime formats. For example, you can specify that a string has an imbedded year, and process that year as a date.

Format-free representations

Non-string datetime values allow you to pass around dates simply as numbers, without the complexities of carrying the corresponding string format (you only need to establish the scale to operate on a value).

Operating on numeric representations

Numeric date values can be operated on directly with FUEL, especially allowing you to add datetime differences (for example, "+"), rather than calling a DATECHG #function and providing a format.

Time

All Sirius datetime #functions allow any reference to a "date" to include time of day. The only standard User Language datetime $function which provides a time of day is $TIME, the current time of day, in one fixed format.

#DATE formats

#DATE allows you to specify any format to return the current date and time; $DATE has only a few numeric codes for a few formats.

Error control args

Fast/Unload provides error handling control that allows you to identify the specific cause of any datetime error.

Error values of numeric date #functions

The #functions that use non-string datetime values provide very uniform error return values: -9.E12 or a null string for numeric or string result #functions, respectively.

CHAPTER 9 *DATESTAT Analysis*

If the DATESTAT statement is present in the FUNIN dataset, *Fast/Unload* will determine which fields in the file contain date values. The determination of whether a field contains a date value is done in two passes; the first pass examines 1000 evenly-distributed records in the file, and the second pass analyzes all records. At the end of each pass, certain fields are retained for date analysis and reporting. A field is retained if:

1. for pass one, the field is not found in the 1000 records

2. at least one value of the field is found in the pass which conforms to a date format, except:

    * a field is not a date if all dates were all numeric, there were more than 5 distinct non-date values, and less than 50% of the instances are dates. This is to attempt to avoid situations in which a field such as zip code could be treated as a 5 digit (YYDDD) date.

Note that Blob fields are not candidates for DATESTAT analysis.

The analysis keeps tracks of various totals that are used in reporting, and also keeps some of the values found in the field. Up to 36 date formats, 20 nondate values, and 120 year samples will be kept. When these overflow, the least-recently-found year and nondate samples are discarded, and the most-recently-found date format sample is discarded. Also, only up to 22 characters of nondate values are kept, so two different values which are the same in the first 22 characters will be considered to be the same value.

The date formats searched for consist of the following list, plus each one of the formats in this list, followed by a blank and time in the form HH:MI:SS, if the date format contains a character other than M, D, or Y, or followed immediately by time in the form HHMMSS otherwise.

```
MON DAY YY
MON DAY YYYY
DAY MON YY
DAY MON YYYY
DAY MONTH YY
DAY MONTH YYYY
DAY MON, YY
DAY MON, YYYY
DAY MONTH, YY
DAY MONTH, YYYY
DDMMYY
DDIMMIYY
DDMMYYYY
DDIMMIYYYY
MONTH DAY YY
MONTH DAY YYYY
MMDDYY
MMIDDIYY
MMDDYYYY
MMIDDIYYYY
YYDDD
YY MON DAY
YY MONTH DAY
YYMMDD
YYIMMIDD
YYYYDDD
YYYY MON DAY
YYYY MONTH DAY
YYYYMMDD
YYYYIMMIDD
```

For the meaning of the components of these formats, see “Datetime Formats” on page 172.

9.1 DATESTAT Reporting

This section describes the reports created by the DATESTAT SUMMARY and DATESTAT DETAIL statements. Both of these statements indicate something about the "quality" of the date data in the field. The purposes of "date field quality" are:

1. If you are running DATESTAT SUMMARY, and you have some values other than **pure**, you may want to do further investigation of the date fields.

2. To indicate how much work might be involved to resolve the various values stored in the field. One tool to resolve the values is to run DATESTAT DETAIL; doing some ad-hoc work with User Language is another approach. The "worse" the quality of a field, the more work is likely to be required to resolve questions about the field values.

The quality is expressed as **pure**, **good**, **fair**, or **poor**; it is an attempt to measure the possible level of effort required to correct data on the file. The terms have the following meanings:

Pure means that there are only date values, and all with a single date format.

Poor means either there are more than 10 distinct non-date values and more than .01% of the field occurrences are non-date values, or the percent of date values which have uncommon formats, times the number of uncommon formats, is greater than .01%, or there are more than 20 date formats.

If there are more than 20 date formats, or if the ratio of occurrences of the most common date format to occurrences of the next most common date format is less than 10 to 1, then *Fast/Unload* prints the string “Common date format not found”.

Fair means either there are more than 5 distinct non-date values and more than .001% of the field occurrences are non-date values, or the percent of date values which have uncommon formats, times the number of uncommon formats, is greater than .001%.

Good is anything else.

9.2 DATESTAT SUMMARY

DATESTAT SUMMARY creates a report with 1-3 lines for each date field, in the following form:

1. *ftag format* (*qual*) *span* field...*n*: *name*

2. *tot* occurrences of field *format count*

3. *nsmp* occurrences of nondate value (len *len*): *sample*

Where:

1. This line is always present; the components are:

   - *ftag* is either "Common date format not found", if there are more than 20 date formats, or "Common format:" otherwise.

   - *format* is the most commonly occurring date format, if *ftag* is "Common format:", or blank otherwise.

   - *qual* is either "pure", "good", "fair", or "poor".

   - *span* is either:

CENTSPAN: *YYYY*
or **No YY occurrences**
The former gives a recommended CENTSPAN, if the field has any 2-digit years, where *YYYY* is the oldest 2-digit year found; the latter occurs if the field does not have any 2-digit years. (2-digit years are interpreted using a CENTSPAN of 1900, or, if running DATESTAT after 1999, a CENTSPAN of -99.)

- *n* is a sequential numbering of the fields; this can be correlated to an FSTATS report.
- *name* is the name of the field.

2. This line is always printed if the *qual* is not **pure**. The components are:
   - *tot* is the total number of occurrences of the field in the file
   - *format count* is either *n* **different formats found**, if *n* is more than 1, or blank otherwise.
3. This line is always printed if there are any nondate values for the field. The line has one of the two following forms:
   a. *nsmp* occurrences of nondate value (*len*): *truncated_value trunc_flag*
   b. *dist* distinct nondate values

   Where *nsmp* is the number of occurrences of a nondate value, if it appears to be exactly one nondate value in the field, and *len* is its length and *truncated_value* is the first 22 characters of its value. *Trunc_flag* is **(first 22 bytes)** if the length is greater than 22. In fact, some of the nondate values in the field may differ, if their first 22 characters are the same.

   If there are 2 or more nondate values in the field, then the second form of this line is presented, where *dist* is the number of different nondate values that Fast Unload has kept as samples (up to the maximum of 20).

Note that for the recommended CENTSPAN, *Fast/Unload* assumes that all 2-digit years occur in the 1900s, or, if running DATESTAT after 1999, 99 years before the date of the run up to and including the date of the run.

9.3 DATESTAT DETAIL

DATEST DETAIL creates a report with 1 page for each date field, which includes:

1. The number of field occurrences.

2. The number of date occurrences, with the minimum and maximum date value, for both 4 digit ("YYYY") and 2 digit ("YY") years.

3. A sample of the discovered date formats.

4. A sample of the year values occurring in the field.

5. A sample of the non-date values occurring in the field.

The sampling rules are described in “DATESTAT Analysis” on page 189; some values may be discarded. An asterisk (*) is printed after a sample if the sample occurrence count is incomplete.

Note that for the recommended CENTSPAN and the min/max values of the DETAIL report, *Fast/Unload* assumes that all 2-digit years occur in the 1900s, or, if running DATESTAT after 1999, 99 years before the date of the run up to and including the date of the run.

CHAPTER 10 Job Statistics

Because *Fast/Unload* is essentially a performance product, it is important to maintain statistics that indicate the cost of performing a *Fast/Unload* and provide information that might be useful in tuning future unloads. These statistics are reported on the report data set. These statistics are reported for the compile step, the unload step, and Ordered Index unload step separately.

One of these stats, “Number of extension pages in base buffer”, along with some of the Table B statistics that are reported as part of FSTATS processing, can also help you detect a need for a reorganization for improving your file's performance with *Model 204*. (See “Description of Table B statistics” on page 51 for an explanation of the Table B statistics generated by FSTATS processing.)

If you use *Fast/Unload* in an MVS environment, you can choose to generate SMF records in addition to the *Fast/Unload* report. See “SMF record format” on page 297 for more information about *Fast/Unload* SMF records. All of the statistics, of course, are reported on SMF records; the printing of statistics on the FUNPRINT dataset, however, is restricted to those statistics which have a value greater than zero.

Many of the statistics refer to I/O processing of the input *Model 204* file (or files, in the case of a group). There are two groups of these I/O statistics:

Base buffer statistics

These occur only during the Unload phase, and are for accessing Table B pages in page number order, that is, accessing the the base records processed during the unload.

Extension page statistics

These can apply during any phase, and are for directly reading pages from the *Model 204* file in no pre-determined order. These are called extension page statistics because one type of direct page access is for extension record access during the Unload phase. There are other types of direct page access, such as reading the Table A field and coded value information during the Compile phase, reading the Existence Bit Map, and reading B-tree, list, and index bitmap pages during the Index unload phase.

Note that the extension pages actually accessed in the unload phase depend on the type of processing in your FUEL program. A UAI operation will access all extensions of all records being unloaded; FSTATS processing will access all extensions of all records processed; some FUEL constructs, such as DELETE or using “#” or “*” for a field

occurrence, cause all extensions of a record to be processed; otherwise, only as much of a logical *Model 204* record is processed as needed to reference the fields used in the FUEL program.

The following is a description of statistics that are reported:

Base buffer reads

Total number of EXCPs issued to read a track or group of tracks into a base buffer. It is directly affected by the SEBUFF parameter and the number of *Model 204* input records (that is, base records) processed.

This statistic is new in *Fast/Unload* version 4.0.

Base buffer waits

Total number of WAITs issued after initiating reads into the base buffers; the ratio of EXCPs to WAITs indicates, to some extent, the degree of overlap of CPU with I/O processing.

This statistic is new in *Fast/Unload* version 4.0.

Extension pg in base buffs

Total number of *Model 204* file pages accessed "directly" which were currently available in pages read into base buffers. This indicates how many extension records were "physically close" to their base record during the Unload phase; specifically, within the SBBUFF times NBBUFF tracks starting with the group of SBBUFF tracks containing the base record. A large number for this indicates that *Model 204* database processing can efficiently access the extension records accompanying a base record, with little physical disk manipulation (or even none, with caching devices). As mentioned in "Description of Table B statistics" on page 51, this information can be used in addition to the FSTATS information about extension records, to help you determine possible benefits obtained by reorganizing a file; if extension records are frequently accessed in *Model 204* processing and a large number of them are not physically close to the base record, performance may suffer to some degree.

This statistic is new in *Fast/Unload* version 4.0.

Extension pg in exten pool

Total number of *Model 204* file pages accessed "directly" which were not currently available in pages read into base buffers but which were available in the pool of extension pages. The larger this number is, the fewer EXCPs are needed to re-read pages, and this statistic will increase (up

to the limit of the number of pages needed) as NEBUFF is increased; you can make the tradeoff of costs for real memory due to increased NEBUFF versus the the costs for I/O.

This statistic is new in *Fast/Unload* version 4.0.

Extension buffer reads

Total number of EXCPs to read pages accessed “directly”. It is the number of such pages that could not be found either in the base buffers or in the extension buffer pool.

This statistic is new in *Fast/Unload* version 4.0.

CPU time

This is the total CPU time used. If data is going out to a sort package, this CPU time value does not include the CPU used by the sort task. If this value is large, there is probably not much you can do about speeding up the unload without making changes to your FUEL program.

Waiting for CPU time

Total real time spent waiting for a CPU to run on. This figure is calculated by subtracting the total of all the wait times and the total CPU time from the total real time for the job. This value could actually include page wait time, or waits for synchronous BSAM I/O under CMS. If you have a high value for waiting for CPU time, it probably indicates severe system-wide contention for CPU or real memory resources.

Report buffer wait time

Total real time spent waiting for a report buffer to be written. This value should always be relatively small unless an unload produces a huge number of reported errors (conversion, hard or missing value). *Fast/Unload* is optimized for placing data into the output data set (FUNOUT). A large value for report buffer wait time indicates a misuse of the report data set for outputting large amounts of data. One can reduce report buffer wait time by using a relatively large block size on the report data sets (FUNPRINT) DD card or FILEDEF statement.

Input wait time

Total real time spent waiting for a program input record to be read in.

Open wait time

Total real time spent waiting for OPEN's of input or output data sets to complete. This would ordinarily be a relatively small value, unless one is using a data set that requires mounting or staging from a removable storage medium such as a magnetic tape.

Output buffer wait time

Total real time spent waiting for an output buffer to be written. If this value is a relatively large chunk of total real time one might be able to get better performance by increasing the blocksize on the output data set (on the output data set DD card or FILEDEF statement) or by using more output buffers (by setting the NOBUFF). If the output data set is on a disk device, better performance might be achieved by using a tape device. If you are going to an external sort routine and this value is large, your efforts might be best spent tuning the sort portion of the job.

Base buffer wait time

Total real time spent waiting for the next base record to be read in. If this value is a relatively large chunk of total real time, one might be able to get better performance by increasing the base record buffer size (by setting SBBUFF) or by increasing the number of base record buffers (by setting NBBUFF). In general, however, one is not likely to see a significant improvement from having more than 2 base record buffers.

Extension buffer wait time

Total real time spent waiting for an extension record to be read in. If this value is a relatively large chunk of total real time one might be able to get better performance by increasing the number of extension record buffers (by setting (NEBUFF) or their size (by setting SEBUFF). If this value is extremely high (more than half of total run time) better performance might be achieved by actually setting the number of base buffers to 1 (setting NBBUFF to 1), preventing base buffer read ahead from blocking extension record retrieval.

PST wait time

Total real time spent waiting for the *Fast/Unload User Language Interface* PST. This can be either to access the next file in the *Model 204* GROUP being unloaded, or to access a *Model 204* page that was marked as being modified when the *Fast/Unload* job started.

Total time

Total time spent in the compile or unload step. This is both figuratively and literally the bottom line for *Fast/Unload* statistics. Since the object of *Fast/Unload* is to unload data as quickly as possible, this figure is a measure of your success. If you get an unexpectedly high total time value, you should examine the individual components of total time and attempt to correct the dominating components.

CHAPTER 11 Fast/Unload User Language Interface

The *Fast/Unload User Language Interface*, purchased as a separate *Fast/Unload* option, allows one to invoke *Fast/Unload* from a User Language program. With this approach, an application builds a set of records to be unloaded, using standard User Language statements; for example, you can reduce unload time by restricting the set of records using indexed *Model 204* fields. There are two ways two invoke *Fast/Unload* from a User Language program:

- By passing the label of the record set or list to the $Funload function.
- By invoking the FastUnload or the FastUnloadTask method against a Recordset object.

$Funload and the FastUnload/FastUnloadTask methods must be linked into the ONLINE or BATCH204 load module as part of the *Sirius Mods* installation process. $Funload or the FastUnload/FastUnloadTask methods send the record set to the *Fast/Unload* PST, which invokes the *Fast/Unload* load module in a subtask (or PST, under CMS) of *Model 204*. The full power of the FUEL language is available, including sorts, UAI, etc.

When invoked via $Funload or the FastUnload/FastUnloadTask methods, *Fast/Unload* only accesses the records in the passed record set, and only issues reads for disk tracks containing those records. Whether or not the *Fast/Unload User Language Interface* is employed, *Fast/Unload* **always** issues reads only for disk tracks containing the records accessed, but, except for records skipped by the EVERY and SKIPREC parameters, a standalone *Fast/Unload* accesses all Table B records in a file until *Fast/Unload* processing ends.

The *Fast/Unload User Language Interface* also provides the ability to unload data from a *Model 204* group.

Note that the *Fast/Unload User Language Interface* allows unloading to be performed (at the thread level) either **synchronously**, so that all processing is completed before control is returned from $Funload or the FastUnload/FastUnloadTask methods, or **asynchronously** so that control is returned to the User Language program as soon as the $function or method parameters are verified and accepted. (*Fast/Unload* processing is **never** synchronous at the Online level: threads not waiting for the completion of a *Fast/Unload User Language Interface* requests will continue to run while one or more unloads are progressing.) When processing is asynchronous (at the thread level) it is up to the user to verify successful completion of the unload much as a user would verify successful completion of a batch job.

A User Language procedure is provided as part of the installation process to monitor and control *Fast/Unload* requests from the ONLINE region. Specifically, a system manager

can cancel or examine *Fast/Unload* requests. In addition, each user can perform the same functions on asynchronous requests that he or she initiated.

To make it easier to diagnose problems where *Fast/Unload User Language Interface* is involved, under *Sirius Mods* version 6.7 and later, messages are also sent to the *Model 204* journal/audit trail at each *Fast/Unload* request made and at each completed by the *Fast/Unload User Language Interface*. The message when the requests are started look like:

```
MSIR.0890: Asynchronous request 2 made by $funload
```

And the message when the requests are completed look like:

```
MSIR.0891: Asynchronous request 2 completed, RC = 0
```

The FastUnload and FastUnloadTask methods are documented in the ***Janus SOAP Reference Manual*** and are available only to customers licensed for *Janus SOAP*. $Funload, and the other $functions which are used for the *Fast/Unload User Language Interface*, are documented in the ***Sirius Functions Reference Manual***.

11.1 When to use the Fast/Unload User Language Interface

If you have the *Fast/Unload User Language Interface*, you can use it, rather then a standalone *Fast/Unload*, to perform a given unload; in fact, you can usually use the *Fast/Unload User Language Interface* and expect it will run as well or better than the comparable standalone *Fast/Unload*. There are many factors which make use of the *Fast/Unload User Language Interface* clearly advantageous over a standalone *Fast/Unload*, such as:

Sparse index-driven record set

If you can use the FIND statement with indexed fields (i.e., no Table B scan) to determine a small fraction of the file which then can be processed by *Fast/Unload*, this will be a significant performance advantage.

Model 204 **Boolean processing**

The capabilities of *Model 204* record list processing, and the ability to combine sets of records in the FIND statement, provide not only efficient index-driven record restriction, but also very powerful application capabilities.

File enqueue

When a file is open for update in an ONLINE environment, the only way to access that file in a standalone job is to bypass file enqueueing (with the NOENQ parameter, as described in "NOEnq" on page 15). A standalone *Fast/Unload* with NOENQ will only examine pages that have been written to disk, and so will not see any modified pages still in the buffer pool.

Performing the unload with the *Fast/Unload User Language Interface* allows you to use normal *Model 204* record-level locking so that a file can be unloaded while it is open for update, obtaining a current image of all pages accessed. There is a tradeoff, of course, involved in holding record locks over the duration of an unload job. You can use an unlocked set of records (record lists, or the User Language FIND WITHOUT LOCKS statement) with the *Fast/Unload User Language Interface*, and the current disk buffer page image is still used. Note, however, that unloading without a locked record set while the file is open for update is suitable for creating some kind of report or other file extract that can tolerate a degree of inconsistent data, but is generally not advisable if you are unloading in order to reorganize the file.

Groups (prior to version 4.4)

The *Fast/Unload User Language Interface* allows you to process a set of records derived from one or more files in a PERManent, TEMPorary, or ad hoc *Model 204* group. Prior to version 4.4, a standalone *Fast/Unload* can only access the single file designated on the *Fast/Unload* OPEN directive.

User Language pre-/post-processing, scheduling

You can use the features of User Language to prepare the *Fast/Unload* input program. The program can be dynamically generated, or static and stored in an Html/Text block or in a separate procedure that's read with $procopn and $procdat. The Html/Text block is particularly useful for generating *Fast/Unload* input programs that are largely static but have some dynamic parts.

The *Fast/Unload User Language Interface* can be used to manipulate and combine the results of one or more $Funload or FastUnload/FastUnloadTask calls. Besides offering the convenience of post-processing with User Language, this can be useful for merging, matching or cross-checking applications. The results can be processed conveniently and efficiently either with Sirius $lists, with the $FunImg function, with Stringlist objects, or with a FastUnloadTask object.

You can combine scheduling and control of all “background” processing in your *Model 204* online with initiating and controlling *Fast/Unload User Language Interface* processing.

The CPU time consumed by the *Fast/Unload* task running under *Model 204* will not interfere with online *Model 204* users: the MVS dispatching priority of the *Fast/Unload* task is 4 less than that of the *Model 204* main task.

In some cases, the only appropriate environment to run *Fast/Unload* is a batch environment, but you can still obtain the relevant advantages of the *Fast/Unload User Language Interface* by invoking it in a single-user *Model 204* job (“BATCH204”). Some of the reasons for running in a batch environment are:

Virtual storage use

The virtual storage requirements of a *Fast/Unload* task running under *Model 204* could significantly affect the virtual storage requirements of the whole region, especially if an external sort is being invoked by *Fast/Unload* as part of its processing. Care should be taken to ensure that these requirements do not cause paging in the *Model 204* region, paging having potentially disastrous performance impact on a multi-user *Model 204* region.

Below-the-line storage

If the below-the-line storage requirements of *Fast/Unload* are large, it may need to run in a separate address space from a *Model 204* online job, and if they are extremely demanding, this might not even permit a BATCH204 environment.

Communication for current buffers

As mentioned above under “File enqueue”, the *Fast/Unload User Language Interface* provides the current copy of any modified pages that the *Fast/Unload* subtask accesses. If a very large number of modified pages are in the buffer pool when $Funload or the FastUnload/FastUnloadTask method is called, this could, in some cases, lead to a noticable amount of overhead; one measure of this would be the **PST wait time**, described in “Job Statistics” on page 195. If this overhead is significant, you might evaluate the tradeoffs in either unloading the file in a standalone unload with the NOENQ parameter, or unloading it when there are fewer dirty pages in the buffer pool, for example, when updating activity is lower.

11.2 Setting up the Fast/Unload User Language Interface environment

To use the *Fast/Unload User Language Interface* the system manager must perform the following steps:

1. Create a custom ONLINE or BATCH204 load module that has the *Sirius Functions* linked in.

2. Either concatenate the load library containing the *Fast/Unload* load module to the load library containing the ONLINE or BATCH204 load module on the STEPLIB DD or copy the *Fast/Unload* load module into the load library containing the ONLINE or BATCH204 load module. For example, if the *Model 204* ONLINE load module resides in M204.LOADLIB and the *Fast/Unload* load module resides in SIRIUS.LOAD then

```
//STEPLIB  DD  DSN=M2Ø4.LOADLIB,DISP=SHR
//         DD  DSN=SIRIUS.LOAD,DISP=SHR
```

   would be an appropriate STEPLIB DD for the ONLINE JCL.

Under CMS, the *Fast/Unload* TEXT file must be on a disk accessed by the *Model 204* ONLINE service machine.

3. Modify the JCL (or EXEC) used to run the *Model 204* ONLINE or BATCH204 so that it invokes the custom load module from step one and so that it contains a DD card (or FILEDEF statement) for the *Fast/Unload* audit trail. This audit trail has DDNAME FUNAUDIT. For initial testing purposes, this DD could simply specify SYSOUT=*.

4. Modify the user0 parm card to indicate the maximum number of concurrent *Fast/Unload* tasks to be allowed. This is specified with the FUNTSKN parameter. This parameter has a default of 0 and a maximum of 64. If this parameter is set to 0, no user will be able to access *Fast/Unload* via $Funload or the FastUnload/FastUnloadtask methods. In addition, it might be necessary to specify the name of the *Fast/Unload* load module (or TEXT file, in CMS). This is done using the FUNPGM parameter. The default value of FUNPGM is 'FUNLOAD'.

   For example, the following would be valid parameters on user0's parm card :

   ```
   FUNTSKN=8,FUNPGM='MYFUN'
   ```

5. The system manager might also need to increase the value of NSUBTKS on user0's parm card. *Fast/Unload* requires a PST and when running under CMS an additional PST for every active *Fast/Unload* task. Thus, the NSUBTKS requirement is increased by 1 for *Fast/Unload* under MVS and 1+FUNTSKN under CMS. The one exception to this rule is that when *Fast/Unload* is to be invoked in a single user run, it is possible to avoid the overhead of using the multi-user scheduler by setting NSUBTKS to 0. This will cause the single user to perform the work of the *Fast/Unload* PST whenever a $FunWait, $FunImg, $FunSStr or $FunSkip is executed.

Once these tasks have been performed by the system manager, *Fast/Unload* is ready for use by the programmer. Communication with *Fast/Unload* is achieved via the Sirius $functions or methods that are provided when a site purchases the *Fast/Unload User Language Interface*. The $functions are documented in the ***Sirius Functions Reference Manual***. The methods are documented in the ***Janus SOAP Reference Manual***.

11.3 System parameters for the Fast/Unload User Language Interface

The parameters described in the following subsections provide useful controls for *Fast/Unload User Language Interface* requests.

11.3.1 FUNPARM

The FUNPARM system parameter is a standard *Model 204* bitmask-style parameter. introduced in *Sirius Mods* version 6.7. Setting the X'01' bit, the only bit currently defined, specifies that a synchronous *Fast/Unload* request is not to be allowed while an updating transaction is active. This is to prevent a *Fast/Unload* request that might take a long time to complete from being run while a user has resources enqueued for an updating transaction. These resources would, of course, include the blocking of checkpoints.

If the FUNPARM X'01' bit is set, and a thread attempts a synchronous *Fast/Unload User Language Interface* request (via $Funload or the FastUnload method of the Recordset class) in the middle of an updating transaction, the transaction is cancelled with a message like the following:

```
CANCELLING REQUEST: MSIR.0561: $FUNLOAD: Synchronous request during
                    update transaction in line 43, procedure
                    FUNTEST, file ALEXPROC
```

11.3.2 FUNMAXT

The FUNMAXT system parameter specifies the maximum amount of time, in seconds, a *Fast/Unload User Language Interface* request is to be given to complete. The timer begins when the *Fast/Unload User Language Interface* is requested, either by $Funload or by the Recordset class FastUnload or FastUnloadTask method.

Introduced in *Sirius Mods* version 6.7, FUNMAXT is a numeric parameter with valid values from 0 to 36000. The default value of 0 means no time limit is placed on *Fast/Unload User Language Interface* requests.

The purpose of FUNMAXT is to prevent user requests from being “hung up” indefinitely while queuing for busy *Fast/Unload* tasks or for unintentionally long-running requests.

To override FUNMAXT for specific requests, you can use either:

- The `MaxTime` named parameter on the FastUnload and FastUnloadTask methods in the Recordset class.
- The sixth parameter on $Funload:

```
* Make sure request completes in one minute
%rc = %rs:fastUnload(%listi, %listo, -
                     parameters='NEBUFF=10', maxTime=60)
 ...
* Make sure request completes in one minute
%rc = $funload('LABEL', %iList, %oList, , 'NEBUFF=10', 60)
```

It is a reasonable strategy to set FUNMAXT to a fairly low value, then to selectively set it higher for requests that need more time. Of course, it can be very difficult to ensure that

short-running requests complete quickly if the Online also has long-running requests that might tie up all the *Fast/Unload* tasks. You improve the odds of quick completion if you specify more *Fast/Unload* tasks (FUNTSKN bigger), but this may still not be enough:

- All tasks might be tied up anyway, if there are many long-running requests.
- Some of the *Fast/Unload* tasks might have trouble getting dispatched, because there are more of them than CPUs to run them.

CHAPTER 12 Using an External Sort Package

Instead of having data for a particular output stream go directly to an output data set, it can be passed first to a SORT routine. This has the following advantages over sorting that data in a separate step:

- A total time savings, because sort processing and record extraction processing are overlapped.
- A disk space savings, because the intermediate output data set that would be used to pass data from *Fast/Unload* to your sort package is eliminated.

Prior to version 4.1, any SORT directives apply to the single output stream, FUNOUT, and there is no “TO destination” qualifier on such directives.

As of version 4.1 and the advent of multiple output streams in a single FUEL program, a SORT directive may indicate the output stream to which it applies:

```
SORT TO destination ...
```

The **destination** ties the SORT directive to the output stream that has the same destination declared in an OUT TO or UAI TO directive. These directives can occur in any order in your FUEL program.

12.1 Specifying the sort

For a UAI stream, you indicate that the output is to be sorted by using the SORT keyword on the UAI statement (see “UNLOAD ALL INFORMATION or UAI” on page 88). The only independent SORT directives you may use are the OPTION statement (at most one per stream) and the PGM statement (at most one per program).

For a non-UAI stream (one declared with an “OUT TO destination” directive), to pass data to a SORT routine you must code two or more SORT statements before the start of the FOR EACH RECORD loop in your FUEL program. *Fast/Unload* interprets the information on a SORT directive as a control statement to be passed to your SORT package (not including the TO destination qualifier).

The SORT FIELDS (“Using SORT FIELDS” on page 208) and SORT RECORD (“Using SORT RECORD” on page 209) statements are required for *Fast/Unload* to pass data to a SORT routine. In addition to these required SORT statements, *Fast/Unload* also supports the following statements:

- MODS

- DEBUG
- ALTSEQ
- SUM
- INCLUDE
- OMIT
- OPTION
- OUTREC
- INREC

In addition:

- When using a 31-bit extended parameter list to pass data to the sort package, any other sort statement accepted by your sort package can be used. For more information on using 31-bit parameter lists see the documentation on the SORTP parameter and also see “Customization of Defaults” on page 291.

- You can identify your sort program by using the following form of the SORT statement:

```
SORT PGM=pgmname
```

  where *pgmname* is the name of your sort program (the default name is SORT). This form of the SORT statement does not allow the “TO destination” qualifier, and it may occur at most once in a FUEL program. You can also customize *Fast/Unload* to change the default SORT program name (see “Default SORT program name” on page 294).

12.2 Using SORT FIELDS

Since SORT FIELDS is a sort statement, one would expect to have to code the *Fast/Unload* version of this as “SORT [TO destination] SORT FIELDS,” where the first “SORT” indicates to *Fast/Unload* that what follows is a sort statement, and the second “SORT” is part of the actual sort statement. While this is, in fact, permitted, *Fast/Unload* will also allow you to specify simply “SORT FIELDS” as a shorthand.

In addition to the standard form of the SORT FIELDS statement (explicitly specifying start position, length, and format), *Fast/Unload* provides a shorthand for specifying field positions in an output record: If you specify a fieldname as part of the SORT FIELDS statement, *Fast/Unload* will replace the name(s) of the field(s) with the starting position, length, and format in the output record.

In addition, you can specify a %variable or a special variable (except for #RECOUT, #OUTPOS, #OUTLEN, and #UPARM) as a shorthand for specifying the position of that %variable or special variable in an output record.

Note: This shorthand is designed for relatively simple FUEL programs. It is **not** available if the FUEL program contains more than one OUT TO stream. In that case, you must use the standard form of SORT FIELDS.

For example, if you code

```
SORT FIELDS=(FIELD1,A)
```

and later in your program you code

```
PUT FIELD1 AT 15 AS STRING(10)
```

Fast/Unload will pass

```
SORT FIELDS=(15,10,CH,A)
```

to the sort package if the record format is fixed, or

```
SORT FIELDS=(19,10,CH,A)
```

if the format is variable. The `19` position in the statement above also illustrates the fact that *Fast/Unload* does not consider the RDW (Record Descriptor Word) as part of the output record, while sorts do consider the RDW as part of the input record.

Constant occurrence numbers are valid with the field names in the SORT FIELDS statement, while (loop control variable or %variable) variable occurrences are not.

12.3 Using SORT RECORD

Fast/Unload uses the SORT RECORD statement to determine the output record format and length.

For example, *Fast/Unload* produces fixed length records with a length of 300 if you code:

```
SORT RECORD TYPE=F,LENGTH=(300)
```

The blocksize used is the largest valid blocksize less than or equal to 4096, or it is the record length, if the record length exceeds 4096. For example, if `TYPE=F,LENGTH=(300)`, the blocksize is 3900. If `TYPE=V,LENGTH=(300)`, the blocksize is 4096. If `TYPE=V,LENGTH=8000`, the blocksize is 8000.

Data passed between *Fast/Unload* and the sort package is buffered, and the number of buffers used is set by the value of NOBUFF.

12.4 Sample code

The following program is an example of the use of an external sort package with *Fast/Unload*.

```
OPEN BIGFILE
SORT TO SBIGFILE FIELDS=(KEY1,A,KEY3,D),EQUALS
SORT TO SBIGFILE RECORD TYPE=F,LENGTH=(100)
SORT TO SBIGFILE ALTSEQ -
   CODE=(F0B0,F1B1,F2B2,F3B3,F4B4,F5B5,F6B6,F7B7,F8B8,F9B9)
OUT TO SBIGFILE DEFAULT
FOR EACH RECORD
  PUT KEY1    AS STRING 10
  PUT KEY2    AS STRING 10
  PUT KEY3    AS FIXED 4
  PUT DATA(*) AS STRING(10)
  OUTPUT
END FOR
```

In this example, the SORT FIELDS statement passed to the sort would be

```
SORT FIELDS=(1,10,CH,A,21,4,BI,D),EQUALS
```

CHAPTER 13 Using Fast/Unload with DBCS data

Fast/Unload understands the format of three varieties of DBCS shift sequences as defined by the hardware vendors IBM, Fujitsu, and Hitachi. You must tell *Fast/Unload* about your DBCS environment with either the DBCS parameter, or by customizing *Fast/Unload*. See "Customization of Defaults" on page 291 for information about customizing *Fast/Unload*. *Fast/Unload* also recognizes the two general types of DBCS data defined by *Model 204*. These are Pure DBCS and Mixed DBCS. *Fast/Unload* always converts DBCS data to the mixed format before output (except for the UAI format). Mixed DBCS means simply EBCDIC and DBCS data are permitted in the field, but all DBCS character strings within the string are enclosed in the appropriate Shift-Out and Shift-In sequences.

When you code your FUEL program to PUT pure DBCS string data, the output length specification must allow for the Shift-Out Shift-In sequences. These are each one byte long for IBM and Fujitsu systems, and two bytes long for Hitachi. Thus, to correctly output a two byte pure DBCS field in an IBM environment, your PUT statement must have a length specification of at least 4 (one byte for the Shift-Out, two bytes for the DBCS character, and one byte for a Shift-In).

Fast/Unload respects DBCS character context when DBCS string truncation occurs. If a DBCS string is truncated, whole DBCS characters are truncated, preserving the Shift-In sequence.

Fast/Unload respects DBCS character context when a start position is specified on a PUT statement (parameter number 4 on the STRING format). That is, the start character counts each DBCS character as a single character whether it is 1 or 2 bytes long. Each Shift-In and Shift-Out sequence also counts as a single character. If the first character to be output is in the middle of a Shift-Out bracket, a Shift-Out is added to the start of the string before output.

When *Fast/Unload* builds the sort key for a UAI SORT, DBCS fields are treated as byte strings (i.e., dshifts are not added to pure DBCS fields, and DBCS characters and shift sequences are not necessarily preserved).

Assignment from a field or %variable preserves the DBCS type of the value.

If a string constant which contains a Shift Out sequence is assigned to a %variable or field (with CHANGE or ADD), or passed as a #function argument, the value has type Mixed DBCS, otherwise it is not DBCS.

The #CONCAT function calculates the appropriate DBCS type and appropriately combines shifts.

For all other #functions, all strings (input and output) are treated as non-DBCS strings.

For example:

```
%LEN = #LEN(PURE_DBCS)                /* Returns number of bytes
%STR = #SUBSTR(PURE_DBCS, 1, %LEN)    /* %STR is non-DBCS
%ST1 = PURE_DBCS                      /* %ST1 has same DBCS type
%ST2 = #CONCAT(PURE_DBCS, PURE_DBCS) /* %ST2 has same DBCS type
```

CHAPTER 14 Customer-written Assembler #Function Packages

You can extend the set of #functions available at your site by writing one or more collections of them and making them available to FUEL. The items enabling you to do this are:

- The FUNCTIONS statement has been added to FUEL to identify the location of #functions you have written.
- An efficient, easy-to-use interface has been designed for matching a #function name to the code implementing that #function.
- An efficient, easy-to-use interface has been designed for performing common operations needed by #functions, such as obtaining the value of arguments and setting the result value and output arguments.
- The interfaces have been designed with compatibility in mind; with new releases of *Fast/Unload* you will not even need to reassemble your #functions.

14.1 Members of SIRIUS.OBJLIB used in coding #Functions

Some assembler language source files are included in your *Fast/Unload* distribution tape to assist you in developing #functions. They are described in this section.

14.1.1 Run-time Interface Symbols: FUNCEQU COPY

This COPY member should be made available in a MACLIB to assemble any #functions you write. The symbols defined in it consist of the values you can pass to the *Fast/Unload* #function service routine, for the various services being requested. The only value defined in this COPY member which will change is the symbol FUNQX, which will increase when additional services are added to the interface.

Note: Prior to the release of version 4.0 of *Fast/Unload*, one of the symbols, FUNQAFS, was incorrectly defined. Be sure you are using the corrected FUNCEQU COPY, which contains the following assembler statement:

```
FUNQAFS  EQU   11    Assign float to arg...
```

14.1.2 Example #Function Package: UFUN ASSEMBLE

This ASSEMBLE program contains the package-searching code, along with one sample #function available in the package. You can modify this program to write a #function package, using your own #function names in the table.

Since future versions of *Fast/Unload* may contain new standard #functions, you should choose a naming convention for your #functions which is not likely to overlap the standard #functions.

14.2 Compiler Call to Package to Locate #Function

When a #function call is compiled in a FUEL program, if the #function name is not one of the standard *Fast/Unload* #functions, all #function packages are dynamically loaded and called, in the order specified, until the #function name is in the package.

The registers passed to the #function package are:

R1 The address of a byte, which contains the length of the #function name, followed by the #function name, blank padded to 255 bytes.

R6 The address of a 4096 byte work area which is passed to all #function packages and to all #functions. This area is initally all binary zeroes and is aligned on a doubleword boundary. *Fast/Unload* does not modify this area.

R13 The address of a 4096 byte work area which is passed only to this #function package and to all #functions within this package. This area is initially all binary zeroes and is aligned on a doubleword boundary. *Fast/Unload* does not modify this area.

R14 The return address.

R15 The entry point of the #function package.

The registers returned by the #function package are:

R0 Must contain 0.

R1 One of the following two cases:

- 0 to indicate the #function is not part of the package.
- A positive number, which is the entry point of the #function.

R2 A bit mask, to indicate which arguments are required. For example, X'A0000000' indicates the first and third arguments are required (a rather strange example).

R3 A bit mask, to indicate which arguments, if specified, must be %variables (i.e., available as output). For example, X'30000000' indicates the third and fourth arguments are output arguments.

R4 Maximum number of arguments the #function can accept.

R5-R15
Need not be restored, can contain any value.

14.3 Run-time Invocation of #Function

This section describes the registers passed to a #function (none need be returned by the #function), and the services that *Fast/Unload* makes available for use by a #function.

The registers passed to a #function are:

R5 The address of a service routine to retrieve from or assign to the #function arguments or perform other services. See the description of the interface to the service routine, presented next in this section.

R6 The address of a 4096 byte work area which is passed to all #function packages and all to #functions. *Fast/Unload* does not modify this area.

R13 The address of a 4096 byte work area which is passed only to a given #function package and to all #functions which were resolved by a call to that package. *Fast/Unload* does not modify this area.

R14 The return address.

R15 The entry point of the #function.

The argument values can be returned by the service routine; various values (**FUNQ2**xxx) are used for different types.

There is no information returned from a #function in the registers. No registers need be restored; they can contain any value. The result value and output arguments of a #function are set by passing various values (**FUNQ2**xxx) to the service routine.

The service routine which is passed to a #function in register R5 is used to retrieve the value of or perform assignment into the #function arguments or result, issue error messages, terminate *Fast/Unload*, and manage memory.

The registers at entry to the service routine are described below; individual services have some exceptions, which are explained in the description of the services in this section.

R0 Unless specified otherwise, the argument number for the service. Argument number 0 designates the #function result.

R1 A code indicating the service required (symbols for this code are defined in FUNCQEU COPY).

R2 As described for individual service descriptions in this section.

F0 (Floating point register 0); as described for individual service descriptions in this section.

R14 The return address.

The registers on return from the service routine are explained in the description of the services in this section. For all services, the registers on return from the service routine are:

R0-R2 Unless specified for an individual service, unpredictable.

R3-R14 Unchanged from values at entry.

R15 Usually a return code; if not specified by the service routine description, unpredictable.

14.3.1 Get information about #function argument(s)

This service obtains the number of arguments to the #function and optionally the type (omitted/present, MISSING value or not, and output or not) of a specified argument.

This service is most often invoked for #functions which have a variable number of arguments (such as #CONCAT). For a different usage example, it can be invoked as follows to see if a particular argument is an output argument (although the function package makes sure that if an argument is declared as output, any function call using that argument is a %variable):

```
LA    RØ,xxx          Get info about arg xxx
LA    R1,FUNQAI       Get service number
BALR  R14,R5          Call service
LTR   R15,R15         Is arg an output arg?
BP    ...             No, handle condition
```

Special input registers for this service are:

R0 Argument number (0 means don't get single arg info).

Output registers for this service are:

R0 Number of arguments (including omitted arguments after a comma) (i.e., 0 if no args, else 1 + number of commas).

Following output registers only apply if this service is called with R0 > 0:

R1 0: Argument has a value.
4: Argument is not present.
8: Argument is present but value is MISSING.

R15 0: Argument is an output arg; assignment will be OK.
4: Argument is not present.
8: Argument present but not an output arg.

14.3.2 Get string value of argument

This pair of services obtains the string value of a specified argument. The strict service (FUNQ2SS) will terminate if the argument is omitted. The conditional service (FUNQ2SC) will allow all argument cases. Either service will reflect exception argument cases by a return code in register R15 and a zero length in register R0.

For example, they can be invoked as follows:

```
   LA    R0,xxx           Get value of arg xxx
   LA    R1,FUNQ2SS       Service number: omitted CANCEL
   BALR  R14,R5           Call service
* Omit following two lines if null for MISSING is OK:
   CH    R15,=H'8'        Is value MISSING?
   BE    ...              Yes, handle condition
   LTR   R15,R0           Is string zero-length?
   BZ    ...              Yes, handle condition
   BCTR  R15,0            No, get length - 1
   EX    R15,MVCA  MVC xx(0,R13),0(R1)  Copy to work area
```

or

```
   LA    R0,xxx           Get value of arg xxx
   LA    R1,FUNQ2SC       Get service number - allow errors
   BALR  R14,R5           Call service
   CH    R15,=H'4'        Is arg present?
   BE    ...              No, handle condition
   LTR   R15,R0           Is string zero-length?
   BZ    ...              Yes, handle condition
   BCTR  R15,0            No, get length - 1
   EX    R15,MVCA  MVC xx(0,R13),0(R1)  Copy to work area
*  Omit following two lines if no DBCS data handled:
   LTR   R2,R2            Is string DBCS?
   BC    ...              Handle condition
```

These services have no special input registers.

Output registers for these services are:

R0 Length of string value (0 if omitted or MISSING value).

R1 Address of string value.

R2 0: Not a DBCS string or not DBCS run
> 0: Mixed DBCS string
< 0: Pure DBCS string

R15 0: Argument contains string value.
4: Argument is not present.
8: Argument present but value is MISSING.

14.3.3 Get float value of argument

This pair of services obtains the 8 byte floating point value of a specified argument. The strict service (FUNQ2FS) will terminate if the argument is omitted, has a non-numeric value, or has a value too large in absolute value to store in an eight byte floating point number (insignificant fractions will be truncated without error). The conditional service (FUNQ2FC) will allow all argument cases. Either service will reflect exception argument cases by a return code in register R15 and a zero in float register F0.

For example, they can be invoked as follows:

```
      LA    RØ,xxx           Get value of arg xxx
      LA    R1,FUNQ2FS       Service num: omit/non-num CANCEL
      BALR  R14,R5           Call service
   * Omit following two lines if zero for MISSING is OK:
      CH    R15,=H'8'        Is value MISSING?
      BE    ...              Yes, handle condition
      STD   FØ,xxx(,R13)     Save value in work area
```

or

```
      LA    RØ,xxx           Get value of arg xxx
      LA    R1,FUNQ2FC       Get service number - allow errors
      BALR  R14,R5           Call service
      B     *+4(R15)         Handle conditions
       B    OK               Handle success
       B    ...              Handle arg omitted
       B    ...              Handle MISSING value
       B    ...              Handle conversion error
   OK STD   FØ,xxx(,R13)     Save value in work area
```

These services have no special input registers.

Output registers for these services are:

F0 Float value (0 if omitted/unconvertible).

R15 0: Argument contains float value.
4: Argument is not present.
8: Argument present but value is MISSING.
12: Argument value can't be converted to float.

14.3.4 Get fixed value of argument

This pair of services obtains the 4 byte binary integer value of a specified argument. The strict service (FUNQ2BS) will terminate if the argument is omitted, has a non-numeric value, or has a value too large (positive) or too small (negative) to store in a four byte signed binary number (fractions will be truncated without error). The conditional service (FUNQ2FC) will allow all argument cases. Either service will reflect exception argument cases by a return code in register R15 and a zero in register R1.

For example, they can be invoked as follows:

```
      LA    RØ,xxx           Get value of arg xxx
      LA    R1,FUNQ2BS       Service num: omit/non-num CANCEL
      BALR  R14,R5           Call service
   * Omit following two lines if zero for MISSING is OK:
      CH    R15,=H'8'        Is value MISSING?
      BE    ...              Yes, handle condition
      ST    R1,xxx(,R13)     Save value in work area
```

or

```
      LA    RØ,xxx           Get value of arg xxx
      LA    R1,FUNQ2BC       Get service number - allow errors
      BALR  R14,R5           Call service
      B     *+4(R15)         Handle conditions
       B    OK               Handle success
       B    ...              Handle arg omitted
       B    ...              Handle MISSING value
       B    ...              Handle conversion error
   OK ST    R1,xxx(,R13)     Save value in work area
```

These services have no special input registers.

Output registers for these services are:

R1 Binary value (0 if omitted/unconvertible).

R15 0: Argument contains integer value (maybe truncated).
4: Argument is not present.
8: Argument present but value is MISSING.
12: Argument value can't be converted to fullword int.

14.3.5 Assign string value to argument

This pair of services sets the value of a specified argument to a specified byte string. The strict service (FUNQASS) will terminate if the argument is omitted or is not an output argument. The conditional service (FUNQASC) will reflect these errors by a return code in register R15.

They are invoked as follows:

```
      LA    RØ,xxx           Set value of arg xxx
      LA    R1,FUNQASS       Get service number - errors CANCEL
      LA    R2,...           Address of string to assign
      set   R3,...           Length of string to assign
   *  Omit following line if no DBCS data handled:
      set   R4,...           Indicate DBCS string type
      BALR  R14,R5           Call service
```

or

```
      LA    RØ,xxx           Set value of arg xxx
      LA    R1,FUNQASC       Get service number - allow errors
      LA    R2,...           Address of string to assign
      LA    R3,...           Length of string to assign
   *  Omit following line if no DBCS data handled:
      set   R4,...           Indicate DBCS string type
      BALR  R14,R5           Call service
      CH    R15,=H'4'        Is arg present/output?
      BC    ...              Handle condition
```

Special input registers for these services are:

R2 Address of value to assign.

R3 Length of value to assign (must be < 256).

R4 If a DBCS run:
0: Not a DBCS string
> 0: Mixed DBCS string
< 0: Pure DBCS string

Output registers for these services are:

R15 0: Assignment successful.
4: Assignment failed: arg omitted.
8: Assignment failed: arg is not output.

For FUNQASS, since the non-0 cases for R15 cause the *Fast/Unload* program to be terminated, if the service returns then R15 will be 0.

14.3.6 Assign float value to argument

This pair of services sets the value of a specified argument to a specified 8 byte floating point value. The strict service (FUNQAFS) will terminate if the argument is omitted or is not an output argument. The conditional service (FUNQAFC) will reflect these errors by a return code in register R15.

They are invoked as follows:

```
LA    RØ,xxx          Set value of arg xxx
LA    R1,FUNQAFS      Get service number - errors CANCEL
LD    FØ,...          Value to assign
BALR  R14,R5          Call service
```

or

```
LA    RØ,xxx          Set value of arg xxx
LA    R1,FUNQAFC      Get service number - allow errors
LD    FØ,...          Value to assign
BALR  R14,R5          Call service
CH    R15,=H'4'       Is arg present/output?
BC    ...             Handle condition
```

Special input registers for these services are:

F0 Value to assign.

Output registers for these services are:

R15 0: Assignment successful.
4: Assignment failed: arg omitted.
8: Assignment failed: arg is not output.

For FUNQAFS, since the non-0 cases for R15 cause the *Fast/Unload* program to be terminated, if the service returns then R15 will be 0.

Note: Prior to the release of version 4.0 of *Fast/Unload*, FUNQAFS was incorrectly defined. Be sure you are using the corrected FUNCEQU COPY, which contains the following assembler statement:

```
FUNQAFS  EQU   11    Assign float to arg...
```

14.3.7 Assign fixed value to argument

This pair of services sets the value of a specified argument to a specified 4 byte binary integer value. The strict service (FUNQABS) will terminate if the argument is omitted or is not an output argument. The conditional service (FUNQABC) will reflect these errors

by a return code in register R15.

They are invoked as follows:

```
LA    RØ,xxx          Set value of arg xxx
LA    R1,FUNQABS      Get service number - errors CANCEL
L     R2,...          Value to assign
BALR  R14,R5          Call service
```

or

```
LA    RØ,xxx          Set value of arg xxx
LA    R1,FUNQABC      Get service number - allow errors
L     R2,...          Value to assign
BALR  R14,R5          Call service
CH    R15,=H'4'       Is arg present/output?
BC    ...             Handle condition
```

Special input registers for these services are:

R2 Value to assign.

Output registers for these services are:

R15 0: Assignment successful.
4: Assignment failed: arg omitted.
8: Assignment failed: arg is not output.

For FUNQABS, since the non-0 cases for R15 cause the *Fast/Unload* program to be terminated, if the service returns then R15 will be 0.

14.3.8 Allocate storage

This pair of services allocates dynamic storage, above (FUNQGMA) or below (FUNQGMB) the 16 meg line, respectively.

They are invoked as follows:

```
LA    RØ,...          Length of storage to allocate
LA    R2,=C'...       Label for storage
LA    R1,FUNQGMB      Get service number - below 16 MB
BALR  R14,R5          Call service
```

or

```
LA    RØ,...          Length of storage to allocate
LA    R2,=C'...       Label for storage
LA    R1,FUNQGMA      Get service number - above 16 MB
BALR  R14,R5          Call service
```

Special input registers for these services are:

R0 Length of storage in bytes.

R2 Address of 8 character eyecatcher for storage.

Output registers for these services are:

R1 Address of storage allocated.

If the storage is not available, the FUEL program is terminated (i.e., the service does not return).

14.3.9 Release storage

This service releases a block of storage which was previously allocated (partial blocks may not be released).

It is invoked as follows:

```
LR    R0,...          Address of storage to release
LA    R1,FUNQFM       Get service number
BALR  R14,R5          Call service
```

Special input registers for this service are:

R0 Address of storage to release.

This service has no output registers.

14.3.10 Issue an error message and/or set return code

This service issues the FUNL0111 message, with the specified text, and/or changes the *Fast/Unload* program return code.

The FUNQMSG service is invoked in a variety of ways. To issue a message without changing the current return code:

```
LA    R0,MSG          Address of message text
LA    R1,FUNQMSG      Get service number
LA    R2,L'MSG        Length of message text
SLR   R3,R3           Indicate no change to return code
BALR  R14,R5          Call service
```

To issue a message and ensure that the return code is set at least as large as a supplied value:

```
LA    RØ,MSG           Address of message text
LA    R1,FUNQMSG       Get service number
LA    R2,L'MSG         Length of message text
LA    R3,value         Set minimum return code
BALR  R14,R5           Call service
ST    R15,xxx(,RD)     Save prior value of return code
```

To ensure that the return code is set at least as large as a supplied value, without printing a message:

```
LA    R1,FUNQMSG       Get service number
SLR   R2,R2            Indicate no message text
BCTR  R2,Ø             Optionally suppress FUNLØ111
LA    R3,value         Set minimum return code
BALR  R14,R5           Call service
ST    R15,xxx(,RD)     Save prior value of return code
```

To force return code to specific value, without regard to previous value, with or without printing a message:

```
LA    R1,FUNQMSG       Get service number
IF    ...              Message desired
 LA   RØ,MSG           Address of message text
 LA   R2,L'MSG         Length of message text
ELSE  ,                No message
 SLR  R2,R2            No message text, but FUNLØ111
 BCTR R2,Ø             Eliminate FUNLØ111 header as well
ENDIF ,
LH    R3,=H'-1'        Change return code in any case
LA    R4,value         Set new return code
BALR  R14,R5           Call service
ST    R15,xxx(,RD)     Save prior value of return code
```

Special input registers for this service are:

R0 Address of message.

R2 Length of message; if less than 0, message not issued. Must be less than 256. Trailing blanks preserved.

R3 0: do not change return code.
> 0: change return code to this value if it is less than that value.
< 0: change return code to value in R4, regardless.

R4 New return code, if R3 less than 0.

Output registers for this service are:

R15 Prior value of return code.

14.3.11 Terminate Fast/Unload, optionally set return code

This service terminates *Fast/Unload*, and optionally changes the *Fast/Unload* program return code.

The FUNQTRM service is invoked in a variety of ways. To terminate the current run without changing the return code:

```
LA    R1,FUNQTRM       Get service number
SLR   R3,R3            Indicate no change to return code
BALR  R14,R5           Call service
```

To terminate the current run and ensure that the return code is at least as high as a provided number:

```
LA    R1,FUNQTRM       Get service number
LA    R3,value         Set minimum return code
BALR  R14,R5           Call service
```

To terminate the current run and set the return code to a given value without regard to its previous value:

```
LA    R1,FUNQTRM       Get service number
LH    R3,=H'-1'        Change return code in any case
LA    R4,value         Set return code
BALR  R14,R5           Call service
```

Special input registers for this service are:

R0 Unused.

R3 0: do not change return code.
> 0: change return code to value in R3 if it is less than that value.
< 0: change return code to value in R4, regardless.

R4 New return code, if R3 less than 0.

This service does not return.

14.4 Example - MVS

This example demonstrates use of the sample #function package contained on the distribution tape, in the MVS environment.

14.4.1 Installing a #Function Package

This step consists of assembling and link-editing a #function package.

```
//ASMLINK EXEC ASMCL
//ASM.SYSIN  DD DISP=SHR,DSN=SIRIUS.LIB(UFUN) Sample package
//ASM.SYSLIB DD DISP=SHR,DSN=SIRIUS.LIB        FUNCEQU macro
//LINK.SYSLMOD DD DISP=(NEW,CATLG),DSN=USER.FUNCPKG
//LINK.SYSIN DD *
 NAME LOCFUNCS(R)
/*
```

14.4.2 Using a #Function Package

Shown here are the changes to the *Fast/Unload* JCL and FUEL needed to use a #function package.

```
//FUNLOAD EXEC PGM=FUNLOAD
//...                                      Normal JCL
//FUNCPKG DD DISP=SHR,DSN=USER.FUNCPKG  Custom package
//FUNIN DD *
FUNCTIONS IN FUNCPKG LOCFUNCS
OPEN PEOPLE
FOR EACH RECORD
   %PALINDROME = #RVRSTR(NAME)
   %PALINDROME = #CONCAT(NAME, %PALINDROME)
   PUT %PALINDROME
   OUTPUT
END FOR
```

14.5 Example - CMS

This example demonstrates use of the sample #function package contained on the distribution tape, in the CMS environment.

14.5.1 Installing a #Function Package

This step consists of assembling a #function package.

```
* Create a macro library with FUNCEQU:
MACLIB GEN FUNLOAD FUNCEQU
* Assemble the package:
GLOBAL MACLIB FUNCEQU
HASM UFUN
```

14.5.2 Using a #Function Package

If the TEXT file created in the preceding step is on an accessed CMS minidisk, you can use the following FUNIN file:

```
FUNCTIONS UFUN
OPEN PEOPLE
FOR EACH RECORD
   %PALINDROME = #RVRSTR(NAME)
   %PALINDROME = #CONCAT(NAME, %PALINDROME)
   PUT %PALINDROME
   OUTPUT
END FOR
```

CHAPTER 15 Using User Exits or Filters

Note: *Fast/Unload* has several extremely efficient features that may reduce the need for output filters: %variables, #functions, arithmetic expressions, FUEL outside FOR EACH RECORD, and the opportunity to write your own assembler language #functions.

Before sending data to an output data set or to a sort package, one can have an output record passed through a user-written load module. This load module must be written in assembler and can be used to modify, replace, or prevent output of a record. In addition, the user exit can request that the run be terminated.

Because of their function, these exits are referred to as **filters**. To have an output record pass through a user filter, one should code the FILTER option on an OUTPUT statement. For example, the statement

```
OUTPUT FILTER HOHO
```

indicates that you would like the current output record to be passed through the filter HOHO.

In the above example HOHO must be a load module contained in a PDS concatenated to STEPLIB for the *Fast/Unload* step.

Under CMS, a HOHO module would have to be linked as a MODULE file on an accessed disk. In addition HOHO would have to have been linked with either the RLDSAVE option or with an ORIGIN that does not overlap the *Model 204* CMS interface or *Fast/Unload*. The CMS interface is typically loaded at address X'20000' and *Fast/Unload* is loaded at X'30000'. Any address greater than X'50000' or the transient area should be all right as an ORIGIN for your load module.

The entry point of the load module indicated by OUTPUT FILTER is entered each time the OUTPUT statement is executed. The registers on entry to the filter load module are:

R0 0 Indicates record is fixed format, 1 indicates record is variable format.

R1 Address of output record. If record format is variable R1 points to the records RDW (record descriptor word).

R2 Length of the record. If record format is variable, this includes the 4 byte length of the RDW.

R3-R12 On first entry these registers are all 0. After first entry these registers are the same as they were on exit from the filter load module. If a filter is used on more than one OUTPUT statement these registers are the same as on exit

from the last time a filter with the same name as the current filter was invoked.

R13 Pointer to a 24 fullword save area.

R14 Return address.

R15 Entry point address for filter load module.

The filter is only responsible for preserving R14 on return to *Fast/Unload*. In addition it must set R0 with a code indicating the required output action for *Fast/Unload*. These codes are:

- 0 - Put the current output record into output stream.
- 1 - Replace current output record with record pointed to by R1.
- 2 - Do not place the current output record into the output stream.

Note that if code 1 is returned and the output format is variable, the record pointed to by R1 must have a valid RDW.

If code 0 is returned the filter may also modify the current output record in the following ways :

- It may modify any part of it.
- It may decrease the length of a variable format part by modifying the RDW.

Note that it may not increase the length of a variable format record by modifying the RDW. To increase the length of a record, code 1 must be returned and the extended record must be copied to a work area pointed to by R1.

Fast/Unload will detect certain errors by a user filter and issue a user ABEND when one is detected. These ABEND codes and errors are:

- 001 - Storage corruption.
- 002 - Record length increased for variable format record.
- 003 - RDW corrupted.
- 004 - Negative code returned in R0.
- 005 - Invalid code returned in R0.

When one of these abends is issued, R3 through R12 are loaded with the registers as they were on exit from the user filter. In addition, R1 on exit from the user filter is moved to R15.

CHAPTER 16 Using Fast/Unload with Model 204 Groups

If you have purchased the *Fast/Unload User Language Interface*, you can directly unload groups with *Fast/Unload*. You can unload permanent groups, temporary groups, or ad hoc groups with this feature. No User Language changes are necessary to use groups with *Fast/Unload*. *Fast/Unload* automatically determines when a found set has a group context, and treats the group as a single database for the duration of the unload. This means that a single output dataset is produced for the entire group unload.

All FUEL statements are supported with the use of groups, except the SORT and HASH options of UAI.

You should be aware of the following special considerations:

- If a field referenced on a PUT statement does not exist in some of the databases in the group, the reference will be treated as a missing occurrence for each record in those files. This means that the PUT statement's MISSING clause will be executed for each record in those files in which the field is not defined.
- The #RECIN variable will always be set to the current record number for the current file. The #RECIN variable is reset at the completion of each file in the group. So, for groups, there can be multiple records unloaded with the same #RECIN value.
- The #RECOUT variable is not reset for the duration of a group unload. It always contains a running total of the number output records produced for the entire unload.
- The #GRPSIZ variable contains the number of files in the group.
- The #GRPMEM variable contains the number of the current file within the group (1 for the first file, etc.).
- If you have any FUEL statements before the FOR EACH RECORD loop, those statements are executed once at the start of the job; they are **not** executed before each member in the group.
- If you have any FUEL statements following the end of the FOR EACH RECORD loop, those statements are executed once at the end of the job; they are **not** executed at the end of each member in the group.
- Field statistics are produced separately for each file in a group.
- Date statistics are produced separately for each file in a group.

- The FRECORD parameter only applies to the first file, if a group unload is being performed.

For example, the following FUEL program prints the number of records in each file and the total number of records in the group (the latter is somewhat silly, since the total number of records is available from the standard *Fast/Unload* summary messages):

```
OPEN INFILES
%GTOT = Ø     /* Initialize counters
%FTOT = Ø
FOR EACH RECORD
   IF %GRP NE #GRPMEM THEN  /* New group member?
      %GTOT = %GTOT + %FTOT
      %FTOT = Ø
      %GRP = #GRPMEM
   END IF
   %FTOT = %FTOT + 1
END FOR
%GTOT = %GTOT + %FTOT    /* Include last file
REPORT %GTOT AND 'records in group'
REPORT #GRPSIZ AND 'files in group'
```

CHAPTER 17 Using Fast/Unload with the Sir2000 Field Migration Facility

For the most part, the presence of a file which has been modified with the *Sir2000 Field Migration Facility* has no effect on *Fast/Unload* processing. The exceptions are as follows:

1. The ADD, CHANGE, and DELETE statements may not be issued for a RELATED field.

2. When doing an unload with UAI, the *Fast/Unload* output contains all of the *Sir2000 Field Migration Facility* definitions, which can be used to recreate them during the *Fast/Reload* LAI operation.

3. The UNLOAD field statement, if it references a RELATED field, will cause the other of the pair of RELATED fields to be unloaded. This ensures that if a partial record is unloaded, both occurrences of a RELATED pair of fields are always unloaded.

4. If ***UAI SORT field_R*** or ***HASH field_R*** causes ***field_R*** to be implicitly unloaded (see "UAI SORT or HASH and field unload order" on page 95), and ***field_R*** is a RELATED field, then any UNLOAD field statement causes the other field of the RELATED pair to also be implicitly unloaded, after ***field_R***.

APPENDIX A

Floating Point Arithmetic and Numeric Conversion

Arithmetic operations in *Fast/Unload* (that is, arithmetic calculations, as defined in "Expressions" on page 32) are performed using the IBM/360 floating point arithmetic instructions. In addition, many operations in *Fast/Unload* involve converting to or from a numeric value, which uses a (power of 2) floating point representation.

Many values that are exactly expressed in base 10 (that is, the usual decimal number system) are not represented exactly as power of 2 floating point values. Also, the floating point arithmetic instructions will, in many cases, produce results that are only approximately equal to the given operation and operands.

To satisfy the requirements of manipulating these numeric values using the "hexadecimal" floating point instructions, certain approaches have been adopted in *Model 204* User Language. In order to produce results in *Fast/Unload* that are identical to the results obtained in User Language, these same approaches have been adopted in *Fast/Unload*. However, this chapter only serves as documentation of float handling in *Fast/Unload*. The similarity of results with User Language has been extensively tested, but there may be edge cases in which *Fast/Unload* and User Language differ.

Note also that *Fast/Unload* contains no exact provision for the floating point value handling provided by the `Image` feature in User Language. Information in this chapter must not be extrapolated to the operation with Images.

Note: The contents of this chapter are refreshed as of *Fast/Unload* 4.3. It is highly recommended that you use at least *Fast/Unload* version 4.3 if your application involves calculations sensitive to the operation of floating point handling.

A.1 Overview

The IBM "hexadecimal" floating point instructions use an exponent based on a power of sixteen. If the value being represented contains a fractional part, or if it is beyond the range of integers that can be exactly represented, then the floating point representation of a value may be only an approximation of the value. For example, the value "one-tenth" is represented in base 16 as the infinite hexadecimal series .1AAAAAAA... These approximations can lead to surprising results, especially when dealing with **decimal** fractions that one commonly considers as having an exact representation, for example, the base 10 number .1 for the value one-tenth.

To address this problem, the approach to floating point manipulation in *Fast/Unload* is based on the following:

Decimal external inputs and outputs

For external numeric inputs and outputs of *Model 204* applications, the original source (for example, data entry fields) and final destination (for example, printed values) is expressed in decimal (base 10) notation.

Arithmetic results mirror decimal operations

When two numeric values are operated upon, in particular with addition and subtraction, the resulting value, when expressed in decimal, is as close as practical to the value that would occur if the operation were performed in decimal.

The principles described above are accomplished using the following algorithms:

15-digit decimal significance

An 8-byte floating point value, in the IBM 360 architecture, contains 56 bits of significance for the fraction part. 56 bits can represent the numbers from 0 to approximately 7.2E16 (7.2 times 10 to the 16th power). Therefore, the maximum significance, in decimal, of an 8-byte FLOAT is 16 (decimal) digits. User Language uses a more conservative limit to significance, and uses 15 decimal digit significance with numeric operations.

Float value preserved if target has same length

A float value is exactly preserved if copied to a float target that has the same length. "Length-converting PUT statements" on page 240 specifies the rules for copying a float value to a float target that has a different length.

Float length 4 conversions differ from length 8 or 16

When a float value of length 4 is used in any context other than copying, it is converted as described in "Using a float value, with decimal digit precision" on page 237. When a float value of length 8 or 16 is used in any context other than copying or arithmetic, it is converted as described in "Using a float value, with decimal digit precision" on page 237.

FLoat length 8 or 16 values in arithmetic expressions

Float values of length 8 or 16 use the high order 8 bytes, without any modification, when they are used as entities in an arithmetic expression.

The purpose of the above rules is to achieve (approximately) the same results for float numeric operations as would be given by operations with decimal numbers.

A.1.1 Primitive operations

As a brief background, note the following:

- Floating point values use the IBM hexadecimal floating point representation, which is a one-bit sign, a 7-bit base 16 exponent, and a binary fraction whose length is either 3 bytes (FLOAT LEN 4), 7 bytes (FLOAT LEN 8), or 14 bytes (FLOAT LEN 16).

- In a **normalized** floating point number, the high-order nibble (the first four bits) of the fraction has a non-zero value.

- The normalized 8-byte add (AD/R) and subtract (SD/R) instructions are used for addition and subtraction in arithmetic expressions; the 8-byte multiply (MD/R) and divide (MD/R) instructions are used for multiplication and division. These instructions produce normalized results (except at the limits of normalized values) and do not round.

  See also “Arithmetic expressions” on page 241, which explains that after every float addition or subtraction, there is a decimal rounding step to preserve the proper number of significant digits.

A.1.2 Using a float value, with decimal digit precision

Except for the following cases:

- when an 8- or 16-byte floating point value is the input to an arithmetic expression (described in “Arithmetic expressions” on page 241);

- when a floating point value is the input to a PUT statement or is the right side of an assignment statement (described in “Assignments and length-preserved PUT statements” on page 239 and “Length-converting PUT statements” on page 240);

- when the argument of the #FLOAT8 function is a 4-byte floating point value (described in “#FLOAT8: Get 8-byte float, padding 4-byte input with 0” on page 127);

whenever FUEL requires the ***value*** of a **floating point** value, the value obtained is approximately the closest 8-byte floating point representation of a value, which depends on the length of the “input” floating point value:

LEN 4 The result is the floating point value approximately closest to the **decimal** number with **6** significant digits closest to the LEN 4 float input. For example, if a FLOAT LEN 4 field contains the following value, shown in hexadecimal:

```
41100004
```

which in decimal is:

```
1.000003814697265625
```

then the nearest **6**-digit decimal value is:

```
1.00000
```

so that value is used, represented exactly by the 8-byte float:

```
4110000000000000
```

LEN 8 or 16

The result is the floating point value approximately closest to the **decimal** number with **15** significant digits closest to the first 8 bytes of the float input. For example, if a FLOAT LEN 8 field contains the following value, shown in hexadecimal:

```
4110000400000000
```

which in decimal is:

```
1.000003814697265625
```

then the nearest **15**-digit decimal value is:

```
1.00000381469727
```

so that value is used, represented by the 8-byte float that is approximately the nearest:

```
4110000400000013
```

Note: The low order 8 bytes of a 16-byte float are ignored.

A.1.3 Obtaining numeric values from non-floats

When a numeric value is required in FUEL from a **string or constant that is a decimal number**, the value obtained is approximately the closest 8-byte floating point representation of the decimal value to 15 significant digits. For example, after this FUEL fragment:

```
%T = '1.000003814697265625' /* Note: string value
%T = '1.000003814697265625' /* Note: string value
CHANGE MYFIELD = %T         /* So field set to string value
%Z = MYFIELD * 1            /* %Z has float value
%Y = %T * 1                 /* %Y has (same) float value
PUT MYFIELD                 /* Line 1: From string, 19 digits
OUTPUT
PUT %Y                      /* Line 2: From float, 15 digits
OUTPUT
IF %T EQ '1.000003814697265625' THEN  /* No conversion here
   PUT 'String comparison 1 EQ, of course'
   OUTPUT
END IF
IF %T NE %X THEN  /* Here converting %X->string: 15 digits
   PUT 'String comparison 2 should be NE'
   OUTPUT
END IF
IF %T EQ +%X THEN /* Here converting %T->float
   PUT 'Float comparison should be EQ'
   OUTPUT
END IF
```

The output file will contain:

```
1.000003814697265625
1.00000381469727
String comparison 1 EQ, of course
String comparison 2 should be NE
Float comparison should be EQ
```

And %X, %Y, and %Z will each contain the 8-byte floating point number `X'4100000400000013'`.

A.2 Assignments and length-preserved PUT statements

Assignments between fields and %variables

Whenever a FLOAT field occurrence is assigned to a %variable, the entire 4, 8, or 16 bytes are copied exactly to the %variable. Whenever a %variable that contains a floating point value is assigned (via the CHANGE or ADD[C] statement) as the value of a field occurrence, the %variable's entire 4, 8, or 16 bytes are copied exactly to the field occurrence.

Length-preserved PUT AS FLOAT

Whenever a field occurrence or %variable that contains a floating point value is used in a PUT AS FLOAT statement, and the FLOAT format specifies a length that is the same as that of the occurrence or %variable, the entire 4, 8, or 16 bytes are copied exactly to the ouput file.

For example, if field FLT4 contains a 4-byte float value that in hexadecimal is X'41000004', then the following FUEL fragment:

```
PUT FLT4 AS FLOAT(4)
OUTPUT
%X = FLT4
PUT %X AS FLOAT(4)
OUTPUT
```

places two lines to the output file, each containing the 4 bytes that are hexadecimal `X'41000004'`.

A.3 Length-converting PUT statements

Other than obtaining the value of a float (for example, as part of an IF statement comparison or as an argument to a #function), which is explained in “Using a float value, with decimal digit precision” on page 237, the only context in FUEL in which a float value is transformed to a float value with a different length is in the PUT statement with a FLOAT format whose format length differs from the length of the float “input.” The cases are shown below:

AS FLOAT(4) The first (and only, if the input is length 8) 8 bytes of the input are copied to a 4-byte float using the LEDR instruction, which produces the first 4 bytes of the input 8 bytes, or those 4 bytes plus 1 (times the sign of the value) if the high order bit of the second 4 bytes is 1.

Note that there is no normalization of the input value prior to rounding; thus the low-order 31 fraction bits of an 8-byte float are ignored, and the low-order 31 + 56 fraction bits of a 16-byte float are ignored.

LEN 16 -> 8 The AS FLOAT(8) clause for a 16-byte float input is obtained by taking the first 8 bytes of the 16-byte float value.

Note that there is no rounding, as there is when converting from 8-byte to 4-byte floats, and there is no normalization; thus the low order 56 fraction bits of the 16-byte float are ignored.

LEN 4 -> 8/16 When a 4-byte float value is the input to AS FLOAT(8) or AS FLOAT(16), the 4-byte input float value is converted to an 8-byte value that is approximately nearest the input value expressed as the nearest 6-significant-digit decimal number, as described in the **LEN 4** case in “Using a float value, with decimal digit precision” on page 237. This is the result when the PUT format length is 8; an additional 8 bytes of zeroes are added when it is 16.

LEN 8 -> 16 The AS FLOAT(16) clause for a 8-byte float input is obtained by appending 8 additional bytes of zeroes to the unchanged 8-byte value.

A.4 Arithmetic expressions

The result of an arithmetic expression is always an 8-byte float; these are produced as described in "Primitive operations" on page 236. Also, after every addition or subtraction in the expression, a step is performed to ensure that the correct significance is retained as the result of that operation. This significance is based on the magnitude of the inputs and the result of the addition or subtraction.

For example, after performing the SDR to operate on the following two values:

```
%X =  123456789.1234    -   123456789
 /*   =   X'4775BCD151F97247'
 /*     - X'4775BCD150000000'
 /*   =   X'401F972470000000'
 /*   =        .123399998992682  (rounded to 15 digits)
```

Fast/Unload examines the magnitude of the absolute values of the result and addends, and it determines that 4 significant digits should be retained, so it rounds the result to 4 significant digits:

```
 /*            .1234  (result rounded to 4 digits)
 /*   =   X'401F972474538EF3' (float nearest to .1234)
```

A.5 Example

Following is one example of the behavior of *Fast/Unload*. Versions prior to 4.0 obtain the different result shown below.

```
%T2 = 1
FOR J FROM 1 TO 7   /* Get 1E-7
   %T2 = %T2 / 1Ø
END FOR
PUT %T2
OUTPUT
FOR J FROM 1 TO 5   /* Get .Ø1
   %T2 = %T2 * 1Ø
END FOR
PUT %T2
OUTPUT
%T = Ø
FOR J FROM 1 TO 1Ø  /* Get .1
   %T = %T + %T2
END FOR
PUT %T
OUTPUT
```

Results (starting with version 4.0):

```
Ø.ØØØØØØ1
Ø.ØØ9999999999999999
Ø.1
```

Results (prior to version 4.0):

```
Ø.ØØØØØØ1
Ø.ØØ9999999999999999
Ø.Ø999999999999999
```

APPENDIX B Messages

FUNL0002 **Invalid parameter specification '*parm*'.**

This indicates a parameter specified with the PARM option on the EXEC card under MVS or a parameter after the open parenthesis on the M204CMS command under CMS is invalid. Verify your job stream or EXEC against the documentation for *Fast/Unload* parameters.

FUNL0003 **GETMAIN request failed, RC=*n*. Run terminated.**

This indicates that there is not enough storage in the region or address space to satisfy or free storage request. This could be caused by running in too small of a region or address space, or requesting too many buffers. Either increase the amount of virtual storage available to *Fast/Unload* or decrease the number of required buffers. Parameters which significantly affect the amount of virtual storage required are NOBUFF, NEBUFF and NBBUFF.

FUNL0004 **End of Fast/Unload *abend info***

This message indicates the termination of *Fast/Unload* and is a normal message. If you have used the ABENDERR parameter to request that *Fast/Unload* end with an ABEND if the return is greater than or equal to some value, and the return code is greater than that value, then a suffix is added to this message (***abend info***) indicating this.

FUNL0005 **Unable to open input data set (FUNIN).**

This message is usually the result of a missing or invalid DD statement for FUNIN.

FUNL0007 **First statement not OPEN, compile not performed.**

The first statement of every *Fast/Unload* program running in batch mode must be the OPEN statement followed by the name of the input database file. Correct the input program to contain the appropriate OPEN statement. Note that no OPEN statement is required when *Fast/Unload* is invoked via $FUNLOAD.

FUNL0008 **Invalid OPEN DDNAME '*ddname*', compile not performed.**

Either the OPEN statement is not followed by a data set DDNAME or the DDNAME is longer than 8 characters long. Put a valid DDNAME after the OPEN statement.

FUNL0009 **Unable to open '*ddname*', compile not performed.**

Either the data set indicated by the OPEN dataset or one of the secondary datasets that make up the logical database file could not be opened. This could be caused by an invalid DD card (FILEDEF under CMS). Validate and correct the DD cards or FILEDEFs for the input database file.

FUNL0010 **Unmatched quotes in input data.**

A quote was found in the input program (FUNIN) without a corresponding close quote. Correct the statement so that all quotes have a matching close quote. Note that if you wish to place a quote character in a literal string you should double the quote. For example, to place the literal **THAT'S LIFE** in a FUEL program you must code it as **'THAT''S LIFE'**.

FUNL0011 **Logical input record is too long.**

A logical program input (FUNIN) record, the record created after processing continuation characters, is too long. You can either break the offending statement into 2 or more statements, or you can increase the value of the LIBUFF parameter.

FUNL0012 **Start of compile**

This message indicates the start of the compile phase of *Fast/Unload*. In the phase, the input program (FUNIN) is compiled to machine language code.

FUNL0013 **End of compile. Number of errors = *n***

This message indicates the end of the compile phase of *Fast/Unload*. If number of errors is indicated to be greater than 0, the unload phase will not be performed.

FUNL0014 **Invalid device type for *ddname*, compile not performed.**

Either the data set indicated by the OPEN dataset or one of the secondary datasets that make up the logical database file was on a device not supported by *Fast/Unload*. This is probably caused by an incorrect DD card or FILEDEF. Validate and correct the DD cards or FILEDEFs for the input database file. Note that Fast/Unload will only work against database files on Count Key Data disk devices.

FUNL0015 **No extents defined for *ddname*.**

Either the data set indicated by the OPEN dataset or one of the secondary datasets that make up the logical database file was empty. This is probably caused by an incorrect DD card or FILEDEF. Validate and correct the DD cards or FILEDEFs for the input database file.

FUNL0016 **DDNAME '*ddname*' not found.**

Either the data set indicated by the OPEN dataset or one of the secondary datasets that make up the logical database file could not be opened. This could be caused by an missing DD card (FILEDEF under CMS). Validate the DD cards or FILEDEFs for the input database file against those use to process the file with *Model 204* and correct the DD statements to match those used with *Model 204*.

FUNL0017 **I/O error on *ddname*.**

An I/O error occurred trying to read the FPL page. Either the input database file is not formatted as a *Model 204* database file, or there is a hardware problem with the disk pack on which the FPL resides. Verify the format of the database file by opening it with *Model 204*.

FUNL0018 **DDNAME '*ddname*' does not match internal name *intname***

The DDNAME indicated on the OPEN statement did not match the database file name on the FPL page. Verify the name of the database file by opening it with *Model 204*.

FUNL0019 **Invalid version for *ddname*.**

The database file was created by a release of *Model 204* later than the latest release which *Fast/Unload* supports. Call Sirius Software Support to obtain the latest version of *Fast/Unload*.

FUNL0020 ***ddname* already enqueued exclusive by *jobname stepname*.**

The input database file is being updated by a *Model 204* job and hence cannot be enqueued in exclusive mode by *Fast/Unload*.load. Either wait until the updating job is completed or use the NOENQ *Fast/Unload* parameter to process the file without any enqueueing. Note that the latter option could result in errors in the unload phase of *Fast/Unload*.

FUNL0021 **No room in file enqueueing table for *ddname*.**

The input database file is enqueued in share mode by the maximum number of jobs. Either wait until one or more of the other enqueued jobs is completed or use the *Fast/Unload* NOENQ parameter to run against the file without any enqueueing. Note that the latter option could result in errors in the unload phase of *Fast/Unload*.

FUNL0022 **Error writing FPL page for *ddname*.**

An I/O error occurred trying to write the FPL page. Either the input database file has been overwritten after *Fast/Unload* has opened it or there is a hardware problem with the disk pack on which the FPL resides. Verify the integrity of the database file by opening it with *Model 204*.

FUNL0023 **Error dequeueing *ddname*, file might be broken.**

The FPL page does not contain the *Fast/Unload* job's enqueueing data. Either the input database file has been overwritten after *Fast/Unload* has opened it or there is a bug in *Fast/Unload Model 204*. Verify the integrity of the database file by opening it with *Model 204*.

FUNL0024 **No multi-track reads allowed on *ddname*. SBBUFF and SEBUFF must be 1.**

The indicated DDNAME is not on cylinder boundaries but you have specified an SEBUFF or SBBUFF greater than 1. Either move the data set to cylinder aligned extents (using IEBGENER or DUMP/RESTORE) or use SEBUFF and SBBUFF values of 1.

FUNL0025 **Tracks per cylinder not divisible by *n*.**

You have specified an SEBUFF or SBBUFF greater than 1 which is not an even divisor of the number of tracks per cylinder for the disk device holding at least one of the physical files making up the logical database file. For example, 3380's and 3390's have 15 tracks per cylinder. Hence, SEBUFF and SBBUFF must be 1, 3, 5 or 15 if you are using 3380's and/or 3390's.

FUNL0026 ***parm* is too large.**

You have specified a value for NBBUFF or NEBUFF that would require an impossibly large amount of storage (more than 2**31 bytes). Correct the value for the offending parameter.

FUNL0027 **Truncated file - *ddname*.**

A physical file making up a logical database file does not have enough space allocated to hold the pages that are indicated by the FPL data set list. This could be caused by inadvertent truncation of a physical file by a utility program such as IEBGENER or DUMP/RESTORE. This could also be caused by an invalid DD card or FILEDEF. Verify the correctness of your DD card or FILEDEF and, if correct, try to recover the file from a backup.

FUNL0028

This message not used starting with version 3.0

FUNL0029

This message not used starting with version 3.0

FUNL0030 **Premature end of program.**

The *Fast/Unload* compiler had not read the **END FOR** to match the **FOR EACH RECORD** statement before the end of the input program (FUNIN). Correct the FUEL program to contain the appropriate END FOR statement to terminate the FOR EACH RECORD clause.

FUNL0031 **Invalid statement.**

The *Fast/Unload* compiler encountered an unrecognized statement. Check the spelling of the indicated statement and if correct, verify the statement syntax in the ***Fast/Unload Reference Manual***.

FUNL0032 **Program too large to run.**

The compiled FUEL program would compile to more than 4 megabytes of object code. This is the current absolute size limit for compiled FUEL programs. Try to eliminate unnecessary statements, convert IF/ELSEIF/ELSE clauses to SELECT clauses or try to split the program into multiple programs.

FUNL0033

This message not used starting with version 3.0

FUNL0034 **Extra code at end of statement is invalid.**

The current FUEL statement contained extra, unexpected data at the end. This could be caused by misspelling, unintentional continuation characters, or syntax errors. Verify the syntax of the indicated statement.

FUNL0035 **Too many nesting levels.**

The FUEL program contains a nesting level deeper than 256. This is the current absolute nesting limit for compiled FUEL programs. Try to convert nested IF clauses to IF/ELSEIF/ELSE clauses.

FUNL0036 **Invalid syntax for *stype* statement.**

A syntax error was encountered in the indicated statement. Verify the syntax of the statement indicated by ***stype*** in the ***Fast/Unload Reference Manual***.

Prior to version 4.0 of *Fast/Unload*, this message often indicates a mis-spelled field name.

FUNL0037 **END *token1* expected but *token2 token3* was found.**

The *Fast/Unload* compiler was expecting a particular kind of END statement and instead got something else. An example is a FUEL program with an IF clause terminated with an END FOR statement. Verify the nesting levels in your FUEL program and correct it to contain the appropriate END statement.

FUNL0038 **Invalid PUT format specification.**

An invalid format was indicated in the specified PUT statement. Verify the PUT statement format in the ***Fast/Unload Reference Manual***.

FUNL0039 **Unable to convert constant to output format.**

A PUT statement was encountered where the indicated constant could not be converted to the required format. For example, the statement PUT 'ABC' AS FLOAT(4) would receive this error message. Correct either the constant or the output format on the PUT statement.

FUNL0040 **Output statements invalid with UAI.**

A statement or special variable which directly manipulates the *Fast/Unload* output file was included in a UAI type of unload. These are incompatible with UAI, since UAI determines the format and order of the output records. The incompatible statements and special variables are:

#OUTLEN
#OUTPOS
OUTPUT
PAI
PUT
SORT (but see below)

Beginning with version 4.1, with Multiple-Output support, the statements above can be used with OUTPUT streams in a program that also includes UAI streams. This error message has therefore been retired as of version 4.1.

Note that SORT PGM=***sort program*** can be used (once) in any FUEL program. Moreover, even with a UAI stream, an associated SORT OPTION ***string*** statement can be used (at most one per UAI stream).

FUNL0041 **UNLOAD statement only valid with UAI.**

An UNLOAD statement was encountered in a non-UAI type of unload. Either insert the UAI statement after the OPEN statement, or remove the UNLOAD statement.

Beginning with version 4.1, with Multiple-Output support, an UNLOAD statement can be used with UAI streams in a program that also includes OUTPUT streams. This error message has therefore been retired as of version 4.1.

FUNL0042 I/O error for *ddname*, Table *t* Page *p-lastpg*, at cylinder *cyl* track *trk*.

An I/O error occurred for the database indicated by ***ddname*** at the indicated cylinder and track, trying to read table ***t*** (0=FCT, 1=Table A, 2=Table B, 3=Table C, 4=Table D) and page ***p***. This could be caused by an erroneous DD card or FILEDEF command, hardware problems, or a corrupted database file. Verify the DD card or FILEDEF command. If they seem valid try to examine the indicated extents with either *Model 204* or some other utility (DFDSS or DDR). Check your EREP logs for DASD I/O errors. For more help call Sirius Software Technical Support.

If the read is for multiple Table B pages (e.g., a track of pages when SBBUF=1), the number of the first page being read (***p***) is followed by the last page number in the group of pages. This may be able to assist you in diagnosing the problem in the file. For example, the following message:

```
I/O error for MYFIL, Table B Page 451-457, at cylinder 2165
track Ø.
```

indicates that the error may have been in any of the 7 pages 451 through 457. You can use *Model 204* to try to examine the records on these pages: multiply the page numbers by the BRECPPG parameter of the file, and the result is the mimimal and maximal numbers of the *Model 204* records which may be contained in the group of pages.

FUNL0043 Unable to load *loadmod*.

The indicated output filter program could not be loaded. Verify the spelling of the output filter name. Under MVS, verify that the output filter program is in a library concatenated to STEPLIB for the Fast/Unload job step. Under CMS, verify that the output filter program is a MODULE file on a disk accessed in the virtual machine running *Fast/Unload*.

FUNL0044

This message not used starting with version 3.0

FUNL0045 Loop control variable already in use.

A FOR statement was encountered inside of a FOR clause which used the same loop control variable. An example of this is a **FOR A FROM ...** occurring inside of a **FOR A FROM ...** clause. Correct the inner or outer loop to use a different loop control variable to eliminate the conflict.

FUNL0046 **Unable to open '*ddname*'.**

The sequential output file could not be opened. This could be caused by a missing or invalid DD card (FILEDEF under CMS). Validate and correct the DD card or FILEDEF for the output file. When *Fast/Unload* is invoked as a standalone load module, a DD is required for each of the sequential data sets declared as an output stream. In versions of *Fast/Unload* prior to 4.1, the output file is always identified by the FUNOUT DD. If *Fast/Unload* is invoked via $FUNLOAD, the DDname of the output file is indicated by the fourth argument of the $FUNLOAD function, although this argument need not indicate an existing data set if there are multiple output streams.

FUNL0047 **Too many buffers for '*ddname*'.**

You have specified a value for NOBUFF that would require an impossibly large amount of storage (more than 2**31 bytes). Correct the value for the NOBUFF parameter.

FUNL0048 **Invalid format for '*ddname*' (*reason*).**

The sequential output file format is incorrectly defined. This may be the result of improper settings for one or more of the output file RECFM, LRECL, or BLKZSIZE parameters (for example, UAI output requires a VB record format and a minimum LRECL value that depends on the UAI options). Review the requirements for these parameter settings and correct at least the specific problem cited in the ***reason*** in the message.

This message could also be caused by a missing or invalid DD card (FILEDEF under CMS). Validate and correct the DD card or FILEDEF for the output file.

When Fast/Unload is invoked as a standalone load module, a DD is required for each of the sequential data sets declared as an output stream. In versions of *Fast/Unload* prior to 4.1, the output file is always identified by the FUNOUT DD. If *Fast/Unload* is invoked via $FUNLOAD, the DDname of the output file is indicated by the fourth argument of the $FUNLOAD function, although this argument need not indicate an existing data set if there are multiple output streams.

The output file must have format F, FB, V or VB.

FUNL0049 **Record *recno* unexpectedly missing.**

A record that was found in the existence bit map (or found set) was missing from the database file. This could be the result of running with the NOENQ parameter against a database file that is being updated, or of running against an unenqueued found set or list when using the *Fast/Unload User Language Interface*. You can change the action to be taken when this situation is encountered by setting the HARDERR parameter. The default action is to cancel the run.

FUNL0050 Record *recno* unexpectedly missing extension.

An extension record was missing from the database file. This could be the result of running with the NOENQ parameter against a database file that is being updated, or of running against an unenqueued found set or list when using the *Fast/Unload User Language Interface*. You can change the action to be taken when this situation is encountered by setting the HARDERR parameter. The default action is to cancel the run.

FUNL0051 PUT *errtype* on line *lineno* for record *recno* in file *filename*, output record *outrec*.

A conversion error or an unexpected missing value was encountered when executing a PUT statement. ***errtype*** will either indicate ERROR if there was a conversion error or MISSING if the value was missing. ***lineno*** indicates the line number of the compiled PUT statement being executed. ***recno*** indicates the *Model 204* record number of the record being processed. This record could be examined under *Model 204* with the **FOR RECORD NUMBER** statement or using the **POINT$** facility. ***filename*** indicates the name of the file containing the record. This will be the name of the file being unloaded unless a group is being unloaded via the *Fast/Unload User Language Interface*, in which case the indicated file is one of the files in the group. ***outrec*** is the output record number on which the error occurred. If you wish to suppress these messages for a particular PUT statement use the NOREPORT option on the PUT statement. If this message is unexpected, verify the contents of the indicated record using either *Model 204* or *Fast/Unload* (with the FRECORD and MAXREC parameters).

FUNL0052 Cancelling run because of PUT condition.

The run is being cancelled because of the CANCEL option on the MISSING or ERROR clause of a PUT statement. Verify the contents of the indicated record using either *Model 204* or *Fast/Unload* (with the FRECORD and MAXREC parameters). If you do not wish the run to be cancelled when the indicated PUT error is encountered, eliminate the CANCEL option from the PUT statement.

FUNL0053 *operation* started.

This is a normal message which indicates the initiation of a scan of the records for *Fast/Unload*. The values of ***operation*** can be:

- **Unload**, to indicate the unload phase *Fast/Unload* not in group context.
- **Group unload**, to indicate the unload phase of the first file of a group found set passed by the *Fast/Unload User Language Interface*.
- **Date scan pass 1**, to indicate the first pass of DATESTAT processing.

This message is only issued if the *Fast/Unload* compiler processed the FUEL program without any errors.

FUNL0054 ***n* input records processed.**

This message is issued at the end of the unload phase of *Fast/Unload* and is a normal message. ***n*** indicates the number of *Model 204* records processed in the unload phase.

FUNL0055 ***steptype* statistics :**

This message is issued at the end of the compile and unload phases of a *Fast/Unload* run and is a normal message. This message uses ***steptype*** to indicate either compilation or unload statistics.

FUNL0056 **Unknown field in record *recno* (*hexDiag*).**

A record was found containing an unknown field code. This could be the result of running with the NOENQ parameter against a database file that has just had a fieldname or fieldnames added or of running against an unenqueued found set or list when using the *Fast/Unload User Language Interface*. This can also be an indication of an integrity problem in your database file. You can change the action to be taken when this situation is encountered by setting the HARDERR parameter. The default action is to cancel the run.

Starting with version 4.4 of *Fast/Unload*, the end of this message shows a hexadecimal diagnostic code which may be useful if Sirius Software support needs to diagnose the situation.

FUNL0057 **Cancelling run because of *reason*.**

This message indicates that a *Fast/Unload* run is being cancelled for the indicated reason. ***reason*** can be any of the following:

- INPUT error condition.

The message follows either message FUNL0049, FUNL0050, FUNL0056, FUNL0083, or FUNL0100. See the description of these errors for more detail; for some of them, you can change the action to be taken when the indicated situation is encountered by setting the HARDERR parameter. The default action is to cancel the run.

- CHECK conditions

The message follows message FUNL0131. See the description of message FUNL0131 for more detail. You can change the action to be taken for various conditions by use of the **CHECK**xxx statement.

FUNL0058 ***n* output records created.**

This message is issued at the end of the unload phase of *Fast/Unload* and is a normal message. This message indicates the number of output records written to the output data set (FUNOUT in batch mode).

FUNL0059 **Mixed page sizes in *ddname*.**

The database represented by "ddname" has unlike page sizes in one or more of the physical files that comprise the logical file. *Fast/Unload* can only unload data from databases with like page sizes.

FUNL0060 **Unsupported parameter '*parm*'.**

A parameter was specified which requested an extra feature of *Fast/Unload* that is not installed at your site. For example, requesting field statistics via the FSTATS parameter will produce this message if the FSTATS feature is not installed at your site.

FUNL0061 **Error reading input data set (FUNIN)**

This message is usually the result of an invalid DD statement for FUNIN or a corrupted input file. Verify the DD statement and format of the input program.

FUNL0062 **No *statement1* associated with *statement2* statement.**

This indicates that a FUEL statement which should only occur inside a particular clause was encountered outside the appropriate clause. Examples of this include ELSE statements occurring outside of an IF clause or a WHEN statement occurring outside of a SELECT clause. Verify the nesting of your FUEL statements and check the syntax of the indicated statement in the ***Fast/Unload Reference Manual***.

FUNL0063 **Can't have *statement2* after *statement1* clause.**

This indicates that a FUEL statement inside a particular clause was encountered in incorrect order. Examples of this are ELSEIF statements occurring after an ELSE statement or a WHEN statement occurring after an OTHERWISE statement. Verify the nesting of your FUEL statements and check the syntax of the indicated statement in the ***Fast/Unload Reference Manual***.

FUNL0064 **Fast/Unload Version *vnum* at *site* on *MM/DD/YYYY time job_ID* CPU *CPU_ID*.**

This informational message is issued at the start of a *Fast/Unload* run. It indicates the version of *Fast/Unload* that you are running, your site ID for Sirius product support and distribution, and the date and time the *Fast/Unload* run started. Starting with version 3.2, the MVS job and step name, or the CMS user ID, will be shown, followed the CPU identifier, as the last portions of this message (***job_ID ... CPU_ID***).

FUNL0065 **Parameter settings :**

This informational message is issued at the start of a *Fast/Unload* run. It indicates the settings of all *Fast/Unload* parameters that will be used for the run.

FUNL0066 **Unmatched parentheses in *statement* statement.**

An open parenthesis was found in the input program (FUNIN) without a corresponding close parenthesis. Correct the statement so that the open parenthesis has a corresponding close parenthesis or eliminate the open parenthesis.

FUNL0067 **Statement too complex.**

The indicated statement was too complex for the Fast/Unload compiler to process it. Try to break the statement up into multiple statements.

FUNL0068 **Unmatched quotes in *statement* statement.**

A quote was found in the input program (FUNIN) without a corresponding close quote. Correct the statement so that all quotes have a matching close quote. Note that if you wish to place a quote character in a literal string you should double the quote. For example, to place the literal **THAT'S LIFE** in a FUEL program you must code it as **'THAT''S LIFE'**.

FUNL0069 **Error linking to *name*.**

The indicated SORT program could not be LINKed. Verify the name on a SORT PGM statement and ensure the appropriate SORT program is accessible from *Fast/Unload*. Under MVS this requires that the appropriate SORT load module be in the STEPLIB or JOBLIB concatenation or within the LINKLIB or LPALIB search order. Under CMS, this requires the appropriate SORT program in a TXTLIB made available with the GLOBAL TXTLIB statement.

FUNL0070 **SORT terminated prematurely.**

The sort program to which data was being passed terminated prematurely. This could be caused by any number of SORT program errors. Check your job log and the sort program's report data set (usually the SYSOUT DD).

FUNL0071 **Invalid SORT RECORD statement.**

A SORT RECORD statement was encountered that could not be interpreted by *Fast/Unload*. Check the format of the SORT RECORD statement in your sort program's reference manual.

FUNL0072 **Multiple SORT *statement* statements.**

A particular SORT statement was encountered more than once. The interface between Fast/Unload and your sort package does not allow this. Eliminate the extra occurrence of the identified SORT statement.

FUNL0073 **SORT FIELDS statement replaced by -**

This informational message is issued at the end of the compilation phase if you had specified a SORT FIELDS statement that used field names rather than column positions. Because sort packages require column positions, *Fast/Unload* must convert field names to column positions before passing the SORT FIELDS statement to the sort package.

FUNL0074 **Unable to resolve *fieldname* in SORT FIELDS statement.**

This message is issued at the end of the compilation phase if you had specified a SORT FIELDS statement that used field names rather than column positions and *Fast/Unload* could not resolve the reference to ***fieldname*** into a column position. Because sort packages require column positions, *Fast/Unload* must convert field names to column positions before passing the SORT FIELDS statement to the sort package. This message occurs if the indicated field is not placed in an output record, or if the indicated field can occur at a variable position in the output record.

FUNL0075 **SORT *statement* statement missing.**

This message is issued at the end of the compilation phase if you had specified one or more SORT statements but failed to code either a SORT FIELDS or a SORT RECORD statement. Either add the required statement or eliminate all SORT statements from your FUEL program.

FUNL0076 **Invalid data in coded field dictionary.**

Invalid data was encountered in the field code dictionary for the current *Model 204* file. This is the area that stores values for *Model 204* fields that have the CODED attribute. This is an indication of a severe data integrity problem. Contact Sirius Software Technical Support for assistance.

FUNL0077 **Cancelling run because of CANCEL statement.**

A CANCEL statement in the FUEL program resulted in the *Fast/Unload* run being cancelled.

FUNL0078 **Unable to format *type* value.**

A MISSING or ERROR constant was encountered in a PUT statement where the indicated constant could not be converted to the required format. For example, the statement:

```
PUT FIELD AS FLOAT(4) ERROR 'ABC'
```

would receive this error message. Correct either the constant or the output format on the PUT statement.

FUNL0079 **Open DD *ddname* does not match found set DD *ddname*.**

This message indicates that an OPEN statement in the input FUEL program did not match the DDNAME associated with the found set passed to $FUNLOAD. This message can only occur when running the *Fast/Unload User Language Interface*.

FUNL0080 **Online run terminated at Online region request.**

The *Fast/Unload* job that had been initiated with $FUNLOAD was terminated either because the initiating user had been bumped, as the result of a $FUNPURG function or as the result of the termination of the *Model 204* **ONLINE** or **BATCH204** that initiated the request.

FUNL0081 **List output error.**

The *Fast/Unload* job that had been initiated via $FUNLOAD was sending output data to a $list (the fourth parameter of $FUNLOAD) and an error was encountered. This is most likely either the result of running out of CCATEMP or exceeding the internal limit on the maximum size of a $list (currently about 6 megabytes).

FUNL0082 **Online run terminated because of intercepted program check.**

The *Fast/Unload* job initiated via $FUNLOAD was terminated because of an intercepted program check in *Fast/Unload*. This message only occurs when running the *Fast/Unload User Language Interface* under CMS. Contact Sirius Software Technical Support for assistance.

FUNL0083 **UAI SORT field truncation error occurred for item *n[/fieldname]*; input record number *recno*.**

You specified a LENGTH value for a UAI SORT unload that was smaller than the length of an occurrence of the SORT field. Change the LENGTH parameter to the appropriate value, remove it from the UAI statement, or add the TRUNC option. If you remove the LENGTH parameter, the default of 255 will be used, which is probably excessive.

n refers to the item number in the SORT fields specified on the UAI statement; that is, **1** indicates the first item, **2** indicates the item after the first **AND** of the UAI statement, and so on. If the item in question is a field name, ***n*** will be followed by a slash (*/*) and the fieldname. ***recno*** indicates the record number in the *Model 204* file being unloaded.

FUNL0084 **Unable to initialize SORT task.**

Fast/Unload tried to start the SORT task, but it terminated. Your SORT package may have written diagnostic information to SYSOUT, or to SORTDIAG, if the DD card was present. Refer to these messages and your SORT package documentation. If there are no apparent SORT related errors, contact Sirius Software Technical Support for assistance.

FUNL0085 **MAXREC out of range.**

For a UAI unload, the variable length records must be greater than or equal to 271 plus the length of any SORT fields, and less than 32756. You can increase the LRECL of the FUNOUT DD (or MAXREC, if specified on the UAI SORT statement) to a larger value. For the best SORT performance, MAXREC should be set to the approximate average length of a Table B record, plus the length of any SORT fields. (See "UNLOAD ALL INFORMATION or UAI" on page 88.)

FUNL0086 **BSIZE required for UAI HASH.**

For a UAI HASH unload, you didn't specify the BSIZE for the target file. If the UAI data will be loaded to a file with a different Table B size, the Table B size must be specified in the UAI statement so that correct hash key values are generated.

FUNL0087 **Can't mix SORT statement and UAI.**

Sort specifications for a UAI program must be part of the UAI statement. Remove the SORT statements and check the syntax of the UAI statement.

FUNL0088 **Cancelling run because of attempt to *action* record *recnum* twice.**

This message indicates that your FUEL program attempted to "unload" a record (numbered ***recnum*** in the input *Model 204* file) twice. The unload ***action*** that attempted this is either the PAI or the UNLOAD statement, as shown in the message. To ensure database integrity, *Fast/Unload* disallows this.

Note that in the case of UNLOAD, the prior statement that was executed is one without a field occurrence; issuing UNLOAD *with* a field occurrence does not prevent subsequent UNLOAD statements. Change your FUEL program so that your IF or SELECT conditions are always mutually exclusive when they enclose a PAI statement, and a normal UNLOAD followed by a normal or field UNLOAD.

FUNL0089

This message not used starting with version 3.0

FUNL0090 **Index unload started.**

This informational message indicates the initiation of the index unload phase of a *Fast/Unload* run. This message is only issued if you specified the OINDEX option on a UAI statement.

FUNL0091 **Terminated because of 0 *code* Abend; PSW 0 FUNL 0**

Fast/Unload intercepted an ABEND condition. The unload terminated prematurely. ***type*** can be either **System** or **User**. For System codes, check your MVS messages and codes manual for the specific cause of the abend. If a dump was suppressed for the system abend, this message will be followed by FUNL0150. Program checks, (System codes 0C1 through 0CF) generally indicate a *Fast/Unload* internal error; for these, or if you are otherwise unable to correct the problem, you should contact Sirius Software Technical Support for assistance. Provide Sirius with the *Fast/Unload* version number, a copy of the report dataset (FUNPRINT), the *Fast/Unload* link map, if you have it, and the abend dump. ***Psw*** is the Program Status Word at the time of error, and ***load_addr*** is the memory address of the start of the *Fast/Unload* load module.

FUNL0092 **Unloading index data for *fieldname*..**

This is an informational message that indicates *Fast/Unload* has started unloading index information for a particular field.

FUNL0093 **Cancelling run because of OUTPUT error in record *recnum*.**

An error was detected writing the output file and the output record error action was set to CANCEL. ***Recnum*** indicates the output record sequence number.

FUNL0094 **Group *groupname* does not match found set group name *groupname*.**

This message indicates that an OPEN statement for a group in the input FUEL program did not match the group name associated with the found set passed to the *Fast/Unload User Language Interface*.

FUNL0095

This message not used starting with version 3.0

FUNL0096 **UAI SORT or HASH not allowed in group context.**

This error message indicates that *Fast/Unload User Language Interface* was used to process a found set from a *Model 204* group, and the FUEL program attempted to unload the file with a UAI statement that contained the SORT or HASH keyword. *Fast/Unload* currently only allows a group to be unloaded via UAI if no SORT or HASH key is specified.

FUNL0097 ***database* is internally inconsistent.**

Invalid data was encountered in field dictionary portion of the identified *Model 204* file. This is an indication of a severe data integrity problem with ***database***. Contact Sirius Software Technical Support for assistance.

FUNL0098 ***product* has expired.**

Your trial period for *Fast/Unload*, or for some component of it, has expired. Contact Sirius Software for further information.

FUNL0099 ***product* not authorized for CPU *CPUID*.**

Fast/Unload, or some component of it, is running on a CPU with an ID that does not match an authorized CPU ID. Contact Sirius Software for further information.

FUNL0100 **UAI SORT field conversion error occurred for item *n[/field]*; input record number *recnum***

This indicates that *Fast/Unload* was unable to convert an item value to the datatype specified in a UAI SORT statement.

n refers to the ordinal item number in the SORT fields specified on the UAI statement; that is, **1** indicates the first item, **2** indicates the item after the first **AND** of the UAI statement, and so on. If the item in question is a field name, ***n*** will be followed by a slash (*/*) and then the field name.

recnum indicates the internal record number for the *Model 204* record being unloaded.

FUNL0101 **Invisible field invalid in UAI SORT: *fieldname***

This indicates that Fast/Unload was unable to sort the unloaded data in sequence by the indicated field, since ***fieldname*** is INVISIBLE.

FUNL0102 **Unknown or invalid *item*: *phrase***

This indicates that a FUEL program contained ***phrase*** which is not valid for the ***item*** indicated. ***item*** can be **special variable** or **#function**.

FUNL0103 **Invalid expression: *phrase***

This indicates that a FUEL program contained ***phrase*** in a context in which an expression is required, but that ***phrase*** is not a valid expression.

FUNL0104 **Call Sirius Software for a new authorization.**

This indicates a problem with authorization of *Fast/Unload* on your CPU. Please contact Sirius Software.

FUNL0105 ***Type* error loading #function package *module* from *DDname*.**

This indicates that a FUEL program contained a #function call, and that Fast/Unload was unable to load the #function package named ***module***, either from the specified ***DDname*** or, if ***** is shown, from the STEPLIB/JOBLIB/link area in MVS, or from a TEXT file on an accessed CMS disk.

Type is either **LOAD**, indicating the LOAD macro failed, or **OPEN**, indicating an OPEN macro failed for ***DDname***.

If the #function can be found in another package, processing continues.

FUNL0106 **Unexpected return from #function package *module* in *DDname*: *info*; Fast/Unload cancelled.**

This message indicates that *Fast/Unload* is trying to compile a FUEL program that contains a call to a #function. The FUEL compiler calls various $function packages to locate the #function for the call being compiled. The #function package identified by ***module*** returned invalid information to the FUEL compiler, as noted by ***info***.

If ***module*** is ****STDFUN**, then the #function is a Sirius-provided function and this bug should be reported to Sirius Software. Otherwise, the #function package is user provided. The package was loaded as ***module***, from either the specified ***DDname*** or, if ***** is shown, the STEPLIB/JOBLIB/link area in MVS, or a TEXT file on an accessed CMS disk. See "Customer-written Assembler #Function Packages" on page 213 for a description of the requirements and coding conventions for *Fast/Unload* #function packages. Contact Sirius Software Technical Support for assistance.

FUNL0107 **Invalid *symptom* to service routine: *value*; Fast/Unload cancelled.**

This indicates that during execution of a #function call, the *Fast/Unload* #function services routine was called with invalid information. ***symptom*** indicates the type of information, and ***value*** specifies, in hexadecimal form, the value provided.

The *Fast/Unload* FUEL program is ended when this error occurs.

See "Customer-written Assembler #Function Packages" on page 213 for a description of the requirements of the *Fast/Unload* #function services routine.

FUNL0108 **Too many arguments: *arg*.**

This indicates that a FUEL program contained a #function call that provided more arguments than allowed by that #function. The parameter in error is identified by ***arg***.

FUNL0109 **Attempt to extract *type* value from argument *arg_number* - *cause*; Fast/Unload cancelled.**

During execution of a #function call, the *Fast/Unload* #function services routine encountered an error while trying to extract the value for argument ***arg_number*** as ***type***, which can be **string**, **float**, or **fixed**. The reason for the error is indicated by ***cause***, which will either be **omitted**, which indicates that the argument was not supplied, or **conversion error**, which indicates that the value of the argument is incompatible with ***type***.

The *Fast/Unload* run is ended when this error occurs.

FUNL0110 **Attempt to assign value to argument *arg_number* - *cause*; Fast/Unload cancelled.**

During execution of a #function call, the *Fast/Unload* #function services routine encountered an error while trying to set the value for argument ***arg_number*** (the #function result is argument **0**). The reason for the error is indicated in ***cause***, which will be set to either **missing**, which indicates that the argument was not supplied, or **input only**, which indicates that the argument is defined to be input only.

The Fast/Unload program is ended when this error occurs.

Please call Support at Sirius Software if this error occurs during execution of a #function provided by Sirius Software. See "Customer-written Assembler #Function Packages" on page 213, for a description of how to use the #function service routine and how the arguments of a #function are designated as input only.

FUNL0111 ***customer #function message***

A customer-provided #function called the *Fast/Unload* #function services routine to issue the message shown as ***customer #function message***.

FUNL0112 **Invalid argument number *arg_number*: it must be *requirement*.**

The FUEL compiler detected an invalid #function call, the *Fast/Unload* run will be cancelled. The argument number (zero for the #function result) is given in ***arg_number***, while ***requirement*** indicates the detected requirement that was violated, as follows:

supplied the #function declared the argument as required, but none was provided, or

a %variable the #function declared the argument as an output argument, which means it must be a %variable.

FUNL0113 Bad page trailer for *ddname*, Table *t* Page *p*, at cylinder *n1* track *n2*: *Xtrlr DD Xtrlr tpg*.

A page was read from the indicated database ddname, cylinder and track, which was expected to be page number ***p*** of table ***t***: (**0**=FCT, **1**=Table A, **2**=Table B, **3**=Table C, **4**=Table D). Each *Model 204* page contains a trailer that includes the file name (DD name) for the file and the *Model 204* table and page number. The trailer for the page read does not match the expected values for these fields. Rather, the page trailer contained the name ***Xtrlr DD*** (shown in hexadecimal) and the table concatenated with page number was ***Xtrlr tpg***, also in hexadecimal. This could be caused by reading a database file that is being updated, or could be caused by an erroneous DD card or FILEDEF command, hardware problems, or a corrupted database file.

If the error is in Table D, ensure that you are not using the UAI INVISIBLE nor UAI OINDEX statements with a file that is having the Ordered Index updated. Verify the DD card or FILEDEF command. Check your EREP logs for DASD I/O errors. Call Sirius Software Support for assistance.

If *Fast/Unload* is able to continue in spite of this error, one of the messages FUNL0049, FUNL0050, or FUNL0056 will follow; the action taken for these messages depends on the HARDERR parameter. If Fast/Unload is unable to continue, it will abend with a diagnostic dump.

FUNL0114 ***update error* in field *fieldname*.**

An error occurred performing an update to the field ***fieldname***. The type of error is indicated by ***update error*** as follows:

Attempt to ADD MISSING value

This indicates that an ADD statement was executed, and the value on the right hand side is MISSING (for example, a %variable that had not been assigned to). If this does not indicate an error in your FUEL program, you can either test for the missing value explicitly, using the IF entity EXISTS phrase, or you can use the ADDC statement, which simply results in a no-op for the MISSING value.

Attempt to CHANGE to MISSING value

This indicates that a CHANGE statement was executed, and the value on the right hand side is MISSING (for example, a %variable that had not been assigned to).

Attempt to CHANGE non-existing occurrence

This indicates that a CHANGE statement was executed, and the occurrence subscript of the field on the left hand side is greater than the number of occurrences of the designated field in the current record.

Attempt to DELETE non-existing occurrence

This indicates that a DELETE statement was executed, and the occurrence subscript of the field is greater than the number of occurrences of the designated field in the current record. If this does not indicate an error in your *Model 204* file or FUEL program, you can either test for the occurrence explicitly, using the IF field EXISTS phrase, or you can use the DELETEC statement, which simply results in a no-op for a missing field.

This message indicates a FUEL programming error which must be corrected. The *Fast/Unload* run is cancelled as a result of this error.

FUNL0115 **Error converting entity to number in line *linenum*.**

This indicates that the FUEL program contained a reference to a %variable or field occurrence that required a numeric value (for example, in the right hand side of an arithmetic assignment statement). The line number of the FUEL program being executed is shown in ***linenum***.

FUNL0116 **Value is non-numeric or is out of range in line *linenum*; Fast/Unload cancelled.**

This indicates that the FUEL program referenced an entity in a context that required a numeric value within a particular range, yet the value in the entity referenced was not within the required range. This error could occur, for example, when a %variable is used as a field occurrence number or as the FROM clause in a FOR statement, both of which must be numbers greater than or equal to 1.

The line number of the FUEL program being executed is shown in ***linenum***.

FUNL0117 Duplicate field name.

This indicates that the field name specified on a NEW statement is either a field name already defined in the *Model 204* file, or has previously been defined using the NEW statement.

FUNL0118 Field dictionary full.

This indicates that you attempted to define a new field using the NEW statement, but the number of fields in the file is already at the maximum allowed by *Model 204*.

FUNL0119 Error performing arithmetic in line *linenum*: *intcode* opcode *op*.

This indicates that the FUEL program contained an arithmetic expression which resulted in an error, such as division by zero. The line number of the failing FUEL statement is indicated by ***linenum*** The interruption code for the error is given by ***intcode***. The operation being performed is indicated by ***opcode***.

FUNL0120 Unexpected END statement.

This indicates that an END statement occurred with no FOR, IF, nor SELECT statement to match.

FUNL0121 String truncated in line *linenum*; Fast/Unload cancelled.

This indicates an attempt to create a string longer than 255 bytes.

The line number of the FUEL program being executed is shown in ***linenum***.

FUNL0122 Invalid datetime or datetime format in line *linenum*; Fast/Unload cancelled.

This indicates that a FUEL program called a #function which has datetime and datetime format arguments, and that the datetime or the datetime format is invalid. This error could occur in a call to #C2DATE, for example, if a constant for the format argument is incorrectly coded, or if the date string argument does not match the format (assuming the fourth argument is omitted; supplying a %variable for the fourth argument allows you to test for errors).

The line number of the FUEL program being executed is shown in ***linenum***.

Refer to "Run-time errors during standard #function calls" on page 100 for a discussion of error return arguments in #function calls. Refer to "Datetime Formats" on page 172 for an explanation of valid datetime formats and valid dates.

FUNL0123 Invalid argument in line *linenum* (*designation*); Fast/Unload cancelled.

This indicates that the FUEL program contains a call to a #function and that an argument has an invalid string value. ***Designation*** specifies the purpose of the argument, in terms shown in the documentation of the #function being invoked; it can indicate one of the following purposes:

option	For example, the third argument to #VERPOS must be a string starting with either uppercase 'M' or 'N'.
character	For example, the third argument to #LEFT must be a single character.
required argument(s)	For example, #TRANSLATE requires either argument 2, 3, or 4.

The line number of the FUEL program being executed is shown in ***linenum***.

FUNL0124 Feature not installed.

This indicates that a *Fast/Unload* statement was entered which is not supported by the configuration of *Fast/Unload* being executed. Ensure that the statement is allowed by one of the features you have purchased and contact Sirius Software Support for assistance.

FUNL0125 File *filename*: *n* date fields detected.

This message is issued at the end of the first pass of DATESTAT processing by *Fast/Unload* and is a normal message if the DATESTAT statement is present. This message indicates the number of fields in a file which have been determined to contain dates.

The name of the file being processed is shown by ***filename***, while the number of fields is shown as ***n***.

FUNL0126 Fast/Unload cancelled by customer-written #function.

This message indicates that a customer-written #function used the *Fast/Unload* #function service routine to terminate a *Fast/Unload* run.

FUNL0127 Fast/Unload cancelled unloading input record number *recno* in file *filename*.

This message indicates that some event caused *Fast/Unload* to terminate prematurely during the unload phase of processing a file.

The run was terminated while processing record number ***recno*** from the *Model 204* file ***filename***.

FUNL0128 ***N* argument positions passed to #function; values :**

This message indicates that a #function was active during the premature termination of a *Fast/Unload* run, that there were ***n*** comma-separated arguments supplied to the #function, and that the display of the arguments follows as a series of FUNL0129 messages.

FUNL0129 ***argno argval*.**

This message is used to display an argument to the active #function. It consists of the argument number, ***argno***, followed by an indication of the argument value, ***argval***, which will be one of the following:

=value The value of the argument follows the equal sign.

(MISSING)
The value of the argument is MISSING; for example, a field occurrence was coded for the argument, but the given occurrence is not present on the record.

(omitted) The argument is omitted; that is, the argument was not coded but a comma was coded to indicate the omission of the argument. For example, in the following statement:

```
%OFF = #VERPOS(%STR, '1234567890', , 10)
```

argument 3 is omitted.

FUNL0130 **Fast/Unload cancelled during execution of line *linenum*.**

This message indicates that some event caused *Fast/Unload* to terminate prematurely during the unload phase of processing a file.

The line number of the FUEL program being executed is shown in ***linenum***.

FUNL0131 **Check failed - *[*Cancel: *condition ...] [*Warn: *condition ...].***

This message indicates that one or more conditions specified (or defaulted) on a CHECK statement were present. If CANCEL was the specified or default action for any detected condition, the *Fast/Unload* run is terminated with a minimum completion code of 8. If WARN was the specified or default action for all detected conditions, the *Fast/Unload* run will be allowed, but the completion code will be set to a minimum value of 4.

condition ... is a condition specified on the CHECK statement (see "CHECK condition ... CANCEL | WARN | ALLOW" on page 41). The detected conditions are grouped and prefixed by their resulting action (**Cancel:** or **Warn:**).

The CHECK statement can be used to override the default conditions that are checked for a *Fast/Unload* run (see "CHECK statement defaults" on page 43). These default conditions can also be customized at your site (see "Default CHECK conditions and actions" on page 293).

FUNL0132 To avoid, *primary-suggestion* re-run with CHECK *condition* ALLOW.

This message follows message FUNL0131 and suggests how to proceed given the conditions stated in FUNL0131. You may receive more than one instance of this message if multiple conditions are found in the file.

The job step return code is set to a minimum of 4 when this message is issued.

Note: If your FUNL0131 message did **not** specify a `Cancel:` action, and you don't need to follow the ***primary-suggestion*** right away, then this message can be used for reference in future *Fast/Unload* jobs.

primary-suggestion provides information for handling the condition detected by *Fast/Unload* and reported as ***condition***. The conditions that can be detected include all those that can be specified on the CHECK statement (see "CHECK condition ... CANCEL | WARN | ALLOW" on page 41).

If a ***primary-suggestion*** is not present or is not applicable:

- You can force *Fast/Unload* to ignore the condition by specifying the CHECK ***condition*** ALLOW statement.
- The defaults in effect at your site can be customized (see "Default CHECK conditions and actions" on page 293).
- If ***condition*** is `PROCS`, this message indicates that the file has *Model 204* procedures, and you have specified CHECK PROCS WARN or CHECK PROCS CANCEL.

  Unloading procedures, invoked for UAI unloads by default or by including the PROCS option on the UAI statement, is supported as of *Fast/Unload* version 4.2.

  To complete the unload and not unload procedures, use UAI NOPROCS (and do not use CHECK PROCS CANCEL). To unload procedures, use UAI PROCS.

primary-suggestion can be any of the following, some of which are version-dependent:

- **run M204 recovery or**

  This occurs when ***condition*** is either `BROKE-PHYS` or `BROKE-LOGIC`, which means that the value of the *Model 204* FISTAT parameter indicates some corrective action needs to be taken to the data and/or indices stored in the file. If you are able to run *Model 204* recovery, the "broken" file status should be corrected.

- **run M204 Z command or**

  This occurs when ***condition*** is `DUPDT`, which means that the value of the *Model 204* FISTAT parameter indicates that there may be deferred index updates which need to be applied to the file. If you are able to run the *Model 204* process for applying

the deferred updates, the index will be complete for unloading.

- **may save procs via M204 COPY PROC command or Disp to USE dataset, then:**

  This occurs when ***condition*** is `PROCS`, which indicates that the file has *Model 204* procedures; versions of *Fast/Unload* prior to 4.2 cannot unload procedures. To workaround:

  1. Ensure that any needed procedures have been saved, using either of the following:

     - Copy the procedures to a work file using the *Model 204* **COPY PROC** command.
     - Copy the procedures to a **USE** dataset using the **DISPLAY PROC** command.

  2. Insert the following FUEL statement to avoid the cancellation or warning:

     ```
     CHECK PROCS ALLOW
     ```

  Note: If you are unloading for the purpose of a file reorganization, the COPY PROCEDURE command cannot copy an ALIAS and "reconnect" it to its underlying procedure.

- **use UAI INV | OINDEX to unload ORD INVIS fields, or**

  This occurs when ***condition*** is `INVIS`, which indicates that you are unloading data from a *Model 204* file *without* the **OINDEX** or **INV** UAI options, and the file contains some ORDERED INVISIBLE fields. If you are performing a **UAI** job, these invisible fields can be unloaded by adding **OINDEX** or **INV**. If you are not running a **UAI** job, or you don't need to unload the fields, the cancellation or warning can be avoided by the suggested CHECK statements.

- **use FUEL ADD for derived INVIS non-ORD fields, {or | and then}**

  This occurs when ***condition*** is `INVIS`, which indicates that you are unloading a file with INVISIBLE non-ORDERED fields. *Fast/Unload* cannot unload these fields directly. However, if the invisible fields are deriveable from other visible fields in the file, and you have installed the *Fast/Unload Fuel Compiler*, you can use the **ADD** statement to create the values and then they can be unloaded by UAI, PAI, or the PUT method of unload.

  The end of this text, **{or | and then}**, is determined as follows:

  or — If you did not code a CHECK INVIS WARN or CANCEL statement, your program does not contain a FOR EACH RECORD loop (and your site must have customized the CHECK defaults). In the process of inserting one or more ADD statements, you will need to enclose them within a FOR EACH RECORD loop, which, in the absence of

CHECK INVIS WARN or CANCEL, prevents the CHECK of invisible fields. You can add CHECK INVIS ALLOW if the ADD statement is not applicable to solving the problem.

and then If you did code a CHECK INVIS WARN or CANCEL statement, you must remove it or change it to ALLOW to avoid the WARN or CANCEL with this file.

FUNL0133 Checking file *filename* initialized and [Cancel: *condition ...*] [Warn: *condition ...*].

This message indicates the various conditions being checked for ***filename*** in the current *Fast/Unload* run. The various conditions, ***condition ...*** consist of the names of conditions that can be specified on the **CHECK** statement. See "CHECK condition ... CANCEL | WARN | ALLOW" on page 41.

FUNL0134 Inconsistent mixed DBCS string; Fast/Unload cancelled.

This indicates that a mixed DBCS string had a missing or spurious **Shift In** or **Shift Out** sequence.

FUNL0135 Invalid hexadecimal string in line *linenum*; Fast/Unload cancelled.

This indicates that the FUEL program contains a call to a #function and that an argument has an invalid hexadecimal string. This error could occur, for example, if the first argument passed to the #X2C function contains the string '123M'.

The line number of the FUEL program being executed is shown in ***linenum***.

FUNL0136 SIRFIELD block missing for '*filename*', compile not performed.

This indicates that SIRFIELD information has been added to the file named ***filename*** but the information is not available in Table D. Please call Sirius Software if you get this message.

FUNL0137 '*Filename*' has been updated with a version *version* of the Sir2000 Field Migration Facility.

This message indicates that *filename* has been updated with an incompatible release of the *Sir2000 Field Migration Facility*. *Fast/Unload* must upgraded before it can open the indicated file. This message is followed by FUNL0138 which indicates the release of *Sir2000 Field Migration Facility* that had last updated *filename*.

FUNL0138 Version *version* of the Sir2000 Field Migration Facility is incompatible with this release of Fast/Unload.

This indicates that the SIRFIELD information stored in Table D of the file can not be processed by the version of Fast/Unload in use. You should upgrade to a version of Fast/Unload that was released at or after the release of the Sirius Mods that updated the file. ***Version*** is an internal version number for use by Sirius Software.

FUNL0139 ***Type name* referenced in *file* and CANCEL set by SIRFIELD command.**

This indicates that the FUEL program contains a reference to the field or alias ***name***, and that references to ***name*** have been set to CANCEL by a SIRFIELD command for file ***file***. ***Type*** will indicate whether ***name*** is a field or an alias.

FUNL0140 ***Type name* referenced in *file* and WARN set by SIRFIELD command.**

This indicates that the FUEL program contained a reference to the field or alias ***name***, and that references to ***name*** have been set to WARN by a SIRFIELD command for file ***file***. ***Type*** will indicate whether ***name*** is a field or an alias.

FUNL0141 ***Stmt* not supported for SIRFIELD RELATED field.**

This indicates that the FUEL program contains an ADD, DELETE, or CHANGE statement (as indicated by ***stmt***) which references a field which has been related using the SIRFIELD command. ADD, CHANGE, and DELETE of related fields is not supported in version 3.1 of Fast/Unload.

FUNL0142 ***Stmterr* statement may not occur outside FOR EACH RECORD loop.**

This indicates that the FUEL program contains a ***stmterr*** statement, and it may only occur within the FOR EACH RECORD loop. You must either delete the ***stmterr*** statement or move it inside the FOR EACH RECORD loop.

For example, an UNLOAD statement may only occur within the FOR EACH RECORD loop.

FUNL0143 **Fast/Unload program must have UAI or FOR EACH RECORD.**

This indicates that the FUEL program contains neither a UAI, UNLOAD ALL INFORMATION, nor FOR EACH RECORD statement. One of these statements is required to specify the processing for each record in the input. Note that FOR EACH RECORD can be combined with UAI (or UNLOAD ALL INFORMATION, which is synonymous with UAI).

FUNL0144 ***PTCH usage*.**

This indicates the offsets in the PTCH CSECT which have been changed from the initial value of zero, which approximates the maintenance level of the executing load module. ***PTCH usage*** can have one of the two following forms:

1. **No PTCH used** , which indicates that the entire patch space is zero.

2. **PTCH usage:** ***beg-end ...*** , which indicates the ranges (in hexadecimal) in the patch area which have been changed from zero.

FUNL0145 **Authorization zap produced *date time*.**

This informational message is issued at the start of a *Fast/Unload* run. It shows the date and time at which the *Fast/Unload* authorization zap was produced.

FUNL0146 **Invalid checksum in authorization zap.**

This message indicates that the *Fast/Unload* authorization zap was incorrectly entered. If the zap was manually entered, double check the contents from the original copy. If the zap was received electronically from Sirius Software, either it was modified during receipt or was incorrectly transmitted. You should double check your steps for receiving it or obtain a new zap from Sirius Software.

FUNL0147 ***Authorization status feature*.**

This informational message is issued at the start of a *Fast/Unload* run. It indicates the ***authorization status*** (such as **Permanently authorized**) of the indicated ***feature*** (such as **Fast/Unload Extraction Language**).

FUNL0148 **Multiple *stmt* statements.**

The statement indicated by ***stmt*** has occurred more than once in your FUEL program, and this is not allowed for that particular statement. Remove the incorrect statement(s).

FUNL0149 **Error in line *linenum*:**

An error has occurred in your FUEL program, and the source listing is suppressed, either due to the NOLIST parameter, or because you are using the *Fast/Unload User Language Interface* and you did not use the ALLMSG parameter.

The number, within the FUEL program, of the line in error is shown as ***linenum***, and the message that applies to the error will follow FUNL0149.

FUNL0150 **Cause of error: *description*:**

A system error has occurred, and the dump has been bypassed. The cause of the error is shown in ***description***; if you need more help determining the action to take to correct the error, see the IBM documentation for the abend code (which is shown in message FUNL0091) and for any messages that we generated by the operating system and accompany the job.

FUNL0151 ***Item* expected.**

This indicates the FUEL compiler expected a particular ***item*** in the input program, but it was not found. For example, this message is issued if you place a statement other than WHEN or OTHERWISE immediately after a SELECT statement.

FUNL0152 **Attempt to UNLOAD *sibinfo* SORT or HASH field.**

This message indicates that the FUEL program contained the UAI SORT or UAI HASH statement, and it also contained an UNLOAD statement with a field, and that field is either:

1. the HASH field or the first SORT field (in this case the string ***sibinfo*** is omitted from the message).

2. the field RELATED to the HASH field or the first SORT field (in this case the string ***sibinfo*** is **is related to**).

Since the unload process produces the HASH field (or in cases the first sort field) at the beginning of each record, the order of unloading that field cannot be changed.

For UAI HASH, you may not control the order of the HASH field, so you must not have an UNLOAD field statement (other than UNLOAD field(*)) which references the HASH field occurrence. For SORT, if you need to use the UNLOAD field statement and it may reference the first SORT field occurrence, you can use AS PLACED on the UAI statement, indicating that the field will not automatically be unloaded first.

If the UNLOAD field statement uses a %variable or a loop control variable as the occurrence, and you are sure that the value of that variable will not be the same as the HASH field or first SORT field, you may use AS FIRST on the UAI statement, which will cause that duplicate unloading to be checked at run time.

See “UAI SORT or HASH and field unload order” on page 95.

FUNL0153 **Attempt to UNLOAD non-existent occurrence *occ* of field *field*.**

This message indicates that you attempted to unload occurrence number ***occ*** of the field named ***field*** using the UNLOAD field statement, and that this occurrence does not exist in the current record. If this does not indicate an error in your *Model 204* file or FUEL program, you can either test for the occurrence explicitly, using the IF field EXISTS phrase, or you can use the UNLOADC statement, which allows specification of a missing field occurrence.

FUNL0154 **Attempt to re-UNLOAD occurrence *occ* of field *field*.**

This message indicates that you used the UNLOAD field statement to unload a field occurrence in a record that had already been unloaded, or the UNLOAD followed a NOUNLOAD for the field. This is not allowed; you must correct your FUEL program to remove the duplicate field unload.

FUNL0155 Partial UNLOAD, ordered index invalidated.

This message indicates that you attempted a partial unload of a record, in a UAI OINDEX or UAI INVISIBLE unload. If the purpose of the job is to unload partial records, you must remove OINDEX or INVISIBLE from the UAI command. Otherwise, you should ensure that a normal UNLOAD statement, which unloads the entire record, is executed for every *Model 204* record which has any UNLOAD field statement executed.

FUNL0156 Attempt to UNLOAD SORT or HASH field: occurrence *occ* of field *field*.

This message indicates that you used the UNLOAD field statement to unload the HASH or SORT field occurrence in a record which has already been implicitly unloaded. This is not allowed; you must correct your FUEL program to remove the UNLOAD field statement with this field and occurrence, or change the field and/or occurrence of the UAI HASH or SORT statement, or use the AS PLACED clause, if you are using UAI SORT.

If you use AS PLACED and you are unloading any partial records, note that the first SORT field will not be unloaded unless you explicitly execute an UNLOAD field statement for it.

See "UAI SORT or HASH and field unload order" on page 95.

FUNL0157 Integer truncated in line *linenum*; Fast/Unload cancelled.

This indicates an attempt to create a number with more significant digits for the integer part than the number of places provided.

The line number of the FUEL program being executed is shown in ***linenum***.

FUNL0158 Null extension record (*extnum*) of record number *basenum*.

This informational message indicates that a zero-length extension record (at number ***extnum***) occurs in the chain of extensions of base record ***basenum***. This message is only produced if you are using the FSTATS parameter to *Fast/Unload*, and it indicates an unnecessary condition in the *Model 204* file which can have negative impact on performance.

FUNL0159 Invalid *construct* in *stmt* statement: *string*.

This message indicates a syntax error in a statement in the FUEL program. The type of statement with the error is shown as ***stmt***. ***string*** is the portion of the statement which is invalid, and ***construct*** shows the type of FUEL construct that is expected.

See the documentation of the type of statement (***stmt***) for a description of the valid constructs that you may use for ***construct***.

FUNL0161 **Only one SORT PGM= statement allowed.**

To override the default and specify an alternate sort package, you may provide a SORT PGM=*sortprogramname* statement. However, only one such statement is allowed. All sorted output streams will use the same *sortprogramname* for sorting.

FUNL0162 **SORT statement for *destination* has no matching OUT TO or UAI TO declaration.**

The compiler encountered a SORT statement that: a) specified an invalid (that is, undeclared) ***destination***, or b) in a FUEL program with multiple destinations, had no TO qualifier indicating the stream to which it applied.

FUNL0163 **Premature EOF while in an #IF block.**

While in a #IF block, the program ended before it encountered the #END IF line.

FUNL0164 **Illegal nested #IF.**

While in a #IF block, the program encountered another #IF line. #IF blocks cannot be nested.

FUNL0165 **Ill-formed #END IF.**

A line beginning with #END contained something other than IF.

FUNL0166 **#END IF without #IF.**

A #END IF line was encountered while outside of any #IF blocks. This is possibly due to an error on the #IF line itself.

FUNL0167 **Ill-formed #ELSE.**

Something beside #ELSE was found on the #ELSE line.

FUNL0168 **#ELSE without #IF.**

A #ELSE line was encountered while outside of any #IF blocks. This is possibly due to an error on the #IF line itself.

FUNL0169 **#ELSEIF without #IF.**

A #ELSEIF line was encountered while outside of any #IF blocks. This is possibly due to an error on the #IF line itself.

FUNL0170 **Ill-formed #IF or #ELSEIF.**

#IF and #ELSEIF lines must contain a field name followed by one of the keywords DEFINED or UNDEFINED. A line beginning with #IF or #ELSEIF did not match this syntax.

FUNL0171 Illegal preprocessor (#) line.

A line beginning with # did not contain a valid preprocessor statement (#IF/#ELSEIF/#ELSE/#END IF).

FUNL0172 Invalid destination name '*string*'.

The ***string*** specified as a destination name was ill-formed (longer than 8 characters), or it was never declared in an OUT TO or UAI TO statement.

FUNL0173 Destination '*string*' already used.

The ***string*** specified as a destination name in an OUT TO or UAI TO declaration was already used. Each output stream must have a unique name.

FUNL0174 More than one default destination specified.

The DEF[AULT] attribute was specified on more than one OUT TO or UAI TO declaration. At most one default destination may be specified for all the OUT TO streams, and at most one may be specified for all the UAI to streams.

FUNL0175 No default destination specified.

A PUT or OUTPUT statement was encountered without a TO clause, but no default OUT TO destination was declared. Or, an UNLOAD statement without a TO clause was encountered, but no UAI TO default destination was declared.

FUNL0176 Invalid syntax in TO *dest* prefix.

A TO ***dest*** clause was specified but the rest of the statement was either missing or invalid.

FUNL0177 Invalid statement with TO *dest* prefix.

A TO ***dest*** prefix was specified but the rest of the statement was not one that allows a destination. The statements that do allow a destination are

PUT
OUTPUT
PAI
PRINT ALL INFORMATION
UNLOADC
UNLOAD

FUNL0178 Must specify TO *dest* for all UAIs if more than one output stream is declared.

If the FUEL program has more than one output stream (UAI or OUT TO), all UAI declarations must have the TO ***dest*** qualifier.

FUNL0179 Only one UAI allowed with empty program.

If the FUEL program has no FOR EACH RECORD loop, it must have exactly one UAI declaration and no other output stream declarations.

FUNL0180 *number* output records created on destination '*destination*'.

Issued at the end of the unload phase of *Fast/Unload*, this informational message indicates that ***number*** output records were written to the output data set ***destination***.

FUNL0181 Cannot have multiple outputs in online mode if output is directed to a list.

The multiple-output feature is not supported when running *Fast/Unload* in online mode with output going to a list.

FUNL0182 Cannot exceed 2048 output destinations.

The maximum number of output declarations is 2048.

FUNL0183 With multiple destinations, #RECOUT must specify destination.

The #RECOUT special variable can be used with both OUT TO and UAI TO streams. If the FUEL program has more than one output stream, #RECOUT must be qualified with the destination, as in #RECOUT(***destination***), even if default destinations are declared.

FUNL0184 TO * but no {OUT|UAI} destinations declared.

You specified `TO` * on an UNLOAD[C] statement, but you did not declare any UAI output streams. Or, you specified `TO` * with one or more PUT, OUTPUT, PAI, or PRINT ALL INFORMATION statements, but you did not declare an OUT stream.

FUNL0185 Can't sort by field name (fieldname) with multiple OUT TO streams.

If your FUEL program declares more than two streams for PUT/OUTPUT operations (that is, OUT TO streams), you cannot use the sort-by-field-name feature. (See “Using SORT FIELDS” on page 208). You must specify all sort fields with "start,length,type" notation.

FUNL0186 Attempt to UNLOAD[C] [field] after a blanket NOUNLOAD.

After a “blanket” NOUNLOAD statement is executed for a particular output stream, no further UNLOADs can be executed for that stream, neither an UNLOAD nor UNLOADC of specified field occurrence(s) nor a “blanket” unload.

FUNL0187 Procs, Aliases not unloaded on destination '*destName*'.

This indicates that the file has been unloaded with UAI, and that procedures in the file were not unloaded.

FUNL0188 ***numberProc* Procs, *numberAlias* Aliases unloaded on destination '*destName*.'.**

This indicates that the file has been unloaded with UAI, with the indicated number of procedures unloaded, and the indicated number of procedure aliases unloaded.

FUNL0189 **Procs, Aliases not unloaded.**

This indicates that the file has been unloaded with UAI, and that procedures in the file were not unloaded.

FUNL0190 ***numberProc* Procs, *numberAlias* Aliases unloaded.**

This indicates that the file has been unloaded with UAI, with the indicated number of procedures unloaded, and the indicated number of procedure aliases unloaded.

FUNL0191 **Unable to enqueue *destinationname*: *reason*.**

This indicates that the file has been unloaded with UAI, with the indicated number of procedures unloaded, and the indicated number of procedure aliases unloaded. ***reason*** is one of the following:

- Can't find DDname
  The destination data set specified or implied in the FUEL program is not defined or its DDname does not match the DDname of a defined data set.

- Model 204 ALLOCATE command required
  A *Model 204* DEFINE DATASET command was issued for the destination data set, but no ALLOCATE command was issued.

- Enqueue failed
  Some other process has already enqueued the data set.

The FUNLOAD program validates that a data set that is to be the output destination for *Fast/Unload* unloaded data exists (is defined and allocated), is referenced within the FUEL program by a single, unique DDname, and is available for exclusive enqueue.

FUNL0192 Can't use multiple outputs with this version of *product*

The text that replaces ***product*** in this message depends on your versions of *Fast/Unload* and *Sirius Mods*. You receive the message if both the following are true:

- The FUEL program specifies an output stream that has a destination name that differs from the one passed as the fourth argument in the $FUNLOAD call.
- You are using a combination of product versions other than *Fast/Unload* 4.2 (or higher) and *Sirius Mods* 6.5 (or higher).

To use multiple output streams and the *Fast/Unload User Language Interface*, you must be running (at least) *Fast/Unload* 4.2 and (at least) *Sirius Mods* 6.5. The multiple output feature was introduced in *Fast/Unload* 4.1, but not supported for the *Fast/Unload User Language Interface* until *Fast/Unload* version 4.2.

FUNL0193 SORTOUT[D] cannot be used with PROCS or with OINDEX/INV. SORT/HASH forced FUNOUT on destination *destinationname*.

You get the FUNL0193 informational message in version 4.3 or later if the following are all true:

- For one or more output streams, UAI SORT (or UAI HASH) is specified and NOPROCS is not.
- The run-time parameter is SORTOUT or SORTOUTD.
- The file to be unloaded contains procedures.

The *destinationname* reported in FUNL0193 is the specified or implied destination for the UAI SORT or UAI HASH statement that provoked the message.

To prevent *Fast/Unload* from sending procedure and alias records to the sort program, which the sort would be likely to mis-arrange, *Fast/Unload* automatically forces SORTOUT or SORTOUTD to FUNOUT *for and only for* the UAI SORT or UAI HASH stream or streams. If the FUNOUT parameter is used instead of SORTOUT or SORTOUTD, only Table B records go through the sort, and all records are ultimately output to their destination data set (the FUNOUT DD) by *Fast/Unload*, not by the sort program.

Prior to version 4.2, if the SORTOUT or SORTOUTD program parameters were specified or the default for UAI SORT or UAI HASH streams, *all* file records ultimately went through the sort program, which output them to their destination data set.

This forcing of FUNOUT is also in effect (but “silent,” without notification) for UAI OINDEX and UAI INV streams, if those indexes are indeed present in the file.

Note: If FUNOUT is forced ON for one or more output streams, the *Fast/Unload* report data set will still show the SORTOUT or SORTOUTD parameter setting as `ON` and `FUNOUT = OFF`, since these parameters apply to the whole run, for all streams where possible. Message FUNL0193, generated after the report data set is written, is citing the exceptional streams where SORTOUT or SORTOUTD is overridden.

FUNL0194 24-bit sort parm list forced FUNOUT on destination *destinationname*.

You get this informational message if the following are all true:

- For one or more output streams, UAI SORT (or UAI HASH) is specified.
- The run-time parameter SORTOUT or SORTOUTD is specified or implied.
- The specified or implied setting of the run-time parameter SORTP is 24 (31 is the shipped default).

If SORTOUT or SORTOUTD is in effect, the sort program and not *Fast/Unload* is responsible for outputting *all* file records to their destination sequential data set. In some cases (see FUNL0193, for example), the sort program cannot properly handle this task, and the sorted records are passed back to *Fast/Unload* for outputting. This passing back is, in effect and equivalent to, a resetting of the SORTOUT or SORTOUTD parameter to the FUNOUT parameter.

FUNL0194 marks another instance where FUNOUT is forced in place of SORTOUT or SORTOUTD: if a SORTP parameter setting of 24 is also specified for the *Fast/Unload* run.

FUNL0195 Unknown type, field *name* cannot be processed: *xxxxxx*.

This message indicates that a field definition is either in error or has attributes that are not processed by this version of *Fast/Unload*. The field name is ***name***, and ***xxxxxx*** is the hexadecimal value of the definition of the field in Table A.

This is an informational message; the *Fast/Unload* job can continue if the following are all true:

- There is no UAI nor PAI output.
- The field is not referenced.
- Neither FSTATS nor DATESTATS is requested.
- The NEW statement is not present.

FUNL0196 *stmt* invalid; file contains field of unknown type.

This message indicates that the operation denoted by ***stmt*** was attempted in the FUEL program, but the operation cannot be performed, because a field definition is in error.

For example, ***stmt*** can be `UAI`, indicating that a UAI operation was specified, but it cannot be performed because one or more unknown fields cannot be successfully unloaded.

FUNL0197 FPL version used by online is too new: *xx*.

In the online issuing a $Funload invocation, the default database file version that was created is later than the latest version that *Fast/Unload* supports. Call Sirius Software Technical Support to obtain the latest release of *Fast/Unload*.

FUNL0198 **Illegal reference to BLOB/CLOB field '*fld*' in file *filNam*.**

The value of the BLOB or CLOB field named ***fld*** in file ***filNam*** is referenced in an illegal context, for example, as a #function argument which is limited to strings of length 255, or in a comparison in an IF statement.

FUNL0199 **BUG evaluating #function at line *n*: *symptom*.**

An error occurred while evaluating a standard #function in line ***n*** of the FUEL program. Please call Sirius Support and have the listing of the FUEL program available (from the FUNPRINT output), and provide the contents of the line in error.

FUNL0200 **Getmain failure attempting to append to long string at line *n*; Fast/Unload cancelled.**

The most likely cause of this error is a very large number of %variables being assigned strings longer than 255, or a very large number of CHANGE, or especially, ADD, statements for BLOB or CLOB fields. While it is possible that increasing the region size available to *Fast/Unload* will correct the program, you should first examine your FUEL program logic.

The error occurred while executing line ***n*** of the FUEL program, although if there is an error in your logic, it may be at a different line which updates a %variable or BLOB or CLOB field.

FUNL0204 **Truncation of %variable with length exceeding 255 *operation*.**

An attempt to use a %variable to perform some operation (described as ***operation***; for example, assigning to a non-Lob field) is disallowed, because the %variable contains a string longer than 255.

FUNL0205 **Attempt to use %variable with length exceeding 255 as *context*.**

An attempt was made to use a %variable whose length exceeds 255 in a context that does not allow this. ***Context*** indicates the usage that was attempted; for example, “operand of IF/ELSEIF statement:”.

FUNL0206 **Attempt to PUT or convert %variable with length exceeding 255.**

An attempt was made to use a %variable whose length exceeds 255 as the operand of a PUT statement.

FUNL0207 **Attempt to ADD or CHANGE non-Lob field with string longer than 255.**

An attempt was made to update a field, which is neither a BLOB nor a CLOB, with a string whose length is greater than 255.

FUNL0208 **Record *pageNum* missing page for Lob field.**

While obtaining the value of a BLOB or CLOB field, one of the Table E pages that should contain part of the field's value could not be read. The identifier of the missing page is shown in the message as ***pageNum***. This indicates a physical inconsistency in the file, and an error should also occur in trying for the same page, if you attempt to obtain the same field occurrence with *Model 204*.

FUNL0209 **Assignment with Lob field on right hand side followed by extraneous token.**

The most likely cause of this error is an attempt to use a BLOB or CLOB field as part of an arithmetic expression.

FUNL0210 **Terminated because of System Abend in line *n*: *abendCode*.**

An error occurred while executing line ***n*** of the FUEL program. Please call Sirius Support and have the listing of the FUEL program available (from the FUNPRINT output), and provide both the contents of the line in error and the value of the abend shown as ***abendCode***.

FUNL0211 **Nonsense IF condition due to *explanation*.**

A condition in an IF or ELSEIF statement is illegal, and the explanation is shown in the message as ***explanation***. For example, a test such as `+%X IS FLOAT` will be rejected with the explanation “type comparison against forced type”, because using “+” to obtain the float value of “%X” would always give a float value. If you were using such a statement to determine whether the value of %X conformed to a float value, use the above test without the “+”.

FUNL0212 **Invalid customization zap: one of X'*illegal*' bits on at PTCH+X'&bitalic(offset)'.**

The load module being executed has had a customization zap applied to the offset in PTCH shown as ***offset***, setting to the value `1` a bit that is not defined for *Fast/Unload* customization. The subset of the byte at ***offset*** that is not allowed to be `1` is shown in hexadecimal as ***illegal***.

FUNL0213 **File '*ddName*' is duplicated in ad-hoc group.**

This indicates that the OPEN command specified multiple DD names, two of which are the same.

FUNL0214 **Multiple file names in single-file 'OPEN FILE' command.**

This indicates that the OPEN FILE command contained two or more DD names.

FUNL0215 **Comma missing between *ddName* and following file name in OPEN command.**

This indicates that the OPEN command contained two consecutive DD names without an intervening comma.

FUNL0216 **Trailing comma in OPEN command.**

This indicates that the OPEN command contained a comma without a subsequent DD name.

FUNL0217 **Missing filename before comma in OPEN command.**

This indicates that the OPEN command contained either two consecutive commas, or a comma before the first DD name.

FUNL0218 **Found set contained in *nullOrGroup foundNumberFiles* file(s), OPEN command specifies *openNumberFiles* file(s).**

This indicates that *Fast/Unload* was invoked with the *Fast/Unload User Language Interface* which passed a record set contained either:

- in a group of files (in which case ***nullOrGroup*** is the phrase "`Group of`"), and the number of files in the group (***foundNumberFiles***) differs from the number of DD names in the OPEN command (***openNumberFiles***), or,
- in a single file (in which case ***nullOrGroup*** is the null string), and there is more than one DD name in the OPEN command.

FUNL0219 **Found set file number *memberNum* (*foundFileName*) does not match corresponding in OPEN command (*ddName*).**

This indicates that *Fast/Unload* was invoked with the *Fast/Unload User Language Interface* which passed a record set from one or more files, and that the file name of the member number ***memberNum*** in the found set (***foundFileName***) differs from the corresponding DD name in the OPEN command (***ddName***).

FUNL0220 **Starting unload for *fileName*.**

This indicates that *Fast/Unload* is unloading a group of files, and is about to unload the file indicated in the message.

FUNL0221 ***numberRecs* input records processed for member of group.**

This indicates that *Fast/Unload* is unloading a group of files, and has just read the indicated number of records from the current file, whose name was reflected in message FUNL0220.

FUNL0222 **-------- Group totals --------'**

This indicates that *Fast/Unload* is unloading a group of files, and that the following sets of messages (e.g., FUNL0221 and FUNL0220) apply to individual members in the group.

APPENDIX C Return Codes

Fast/Unload will terminate with one of the following program return codes:

0 *Fast/Unload* successfully completed.

4 *Fast/Unload* encountered a potentially non-fatal error.

8 *Fast/Unload* encountered a probably fatal error.

12 An invalid PARM card value was encountered.

16 Insufficient storage.

32 *Fast/Unload* trial has expired or is not authorized for CPU.

40 *Fast/Unload* could not open the FUNPRINT DD.

48 Insufficient storage to even intialize *Fast/Unload*.

64 *Fast/Unload User Language Interface* called an incompatible *Fast/Unload* load module.

128 *Fast/Unload* abend.

-1 *Fast/Unload* PST became unavailable after request accepted.

n A *Fast/Unload* `CANCEL n` statement was executed.

APPENDIX D Installation

Fast/Unload can be installed from a product tape or can be installed from an object deck downloaded from the Sirius Software web site. Installation from tape is described in "MVS Installation" on page 288 and "CMS Installation" on page 289 while installation from the web is described in "Installation from the web". In any case, all *Fast/Unload* distributions come with *all* maintenance pre-applied: up to the time the tape was cut for tape installations and up to the time the object deck was downloaded for web installations.

If you are a user of a previous version of *Fast/Unload*, see the **Release Notes** for the version you are installing. This document is available on the Sirius Software web site Documentation page (http://sirius-software.com/maint/manlist). The Release Notes highlight changes in the new version, and list any compatibility issues with the previous version.

D.1 Installation from the web

You can download the *Fast/Unload* object files from the Sirius Software web site (**http://sirius-software.com**). The download process requires a userid and password.

1. Click the `Support` navigation link to go to the Customer Service page (**http://sirius-software.com/support.html**), a public page that explains how to get the required userid and password and that also contains links to the various download pages.

2. Click the `Download object files` link to go to a protected page (**https://sirius-software.com/maint/objlist**) that contains a dynamically-generated list of the various Sirius products that you may download.

   The page also contains a `Click here` link that causes the page to be re-displayed with detailed download and installation instructions. These instructions supercede any information in this manual.

3. On the `Download object files` page, select the link to the currently available *Fast/Unload* object file for the version you require. This file is used for all the versions of *Model 204* that are currently supported by *Fast/Unload*.

D.2 MVS Installation

Fast/Unload is distributed on a magnetic tape which also contains the all the other Sirius products you purchased. If you are a user of a previous version of *Fast/Unload*, see the **Release Notes** accompanying the tape, highlighting the changes in the new version, and listing any compatibility issues with the previous version.

If you have purchased the *Fast/Unload User Language Interface*, please refer to the ***Sirius Mods Installation Guide*** when you have completed the basic *Fast/Unload* install. In addition, your tape will contain the *SirZap* facility; use of *SirZap* is optional. *SirZap* usage and installation instructions are described in the ***Rocket Model 204 SirZap User's Guide and Reference Manual***.

To install *Fast/Unload* you must simply load two files from tape onto disk. Use the following JCL to perform the load: Note: This job will also load any other Sirius products you have purchased.

```
//FUNTAPEL JOB  (Ø),'McBain',MSGCLASS=A,CLASS=C,NOTIFY=MCBAIN
//*
//*        Load Sirius products from tape
//*
//IEBCOPY  EXEC PGM=IEBCOPY,REGION=ØM
//T1       DD   UNIT=TAPE,VOL=SER=SIRIUS,LABEL=(1,SL),DISP=(OLD,PASS),
//             DSN=SIRIUS.LIB
//T2       DD   UNIT=TAPE,VOL=SER=SIRIUS,LABEL=(2,SL),DISP=(OLD,PASS),
//             DSN=SIRIUS.LOAD
//T3       DD   UNIT=TAPE,VOL=SER=SIRIUS,LABEL=(3,SL),DISP=(OLD,PASS),
//             DSN=SIRIUS.ULSPF
//*
//D1       DD   DISP=(,CATLG),DSN=SIRIUS.LIB,UNIT=SYSDA,
//             SPACE=(CYL,(1Ø,Ø,5))
//D2       DD   DISP=(,CATLG),DSN=SIRIUS.LOAD,UNIT=SYSDA,
//             SPACE=(CYL,(5,Ø,2))
//D3       DD   DISP=(,CATLG),DSN=SIRIUS.ULSPF,UNIT=SYSDA,
//             SPACE=(CYL,(25,Ø,2))
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   *
 COPY I=T1,O=D1
 COPY I=T2,O=D2
 COPY I=T3,O=D3
/*
```

You must, of course, change the job card and possibly the output data set names to conform to your local standards.

After loading these files, *Fast/Unload* is ready to use. The second tape file contains the *Fast/Unload* load module and may be all you need for running *Fast/Unload*. After unloading the *Fast/Unload* load module, you can customize it to your site's needs, if necessary; see “Customization of Defaults” on page 291.

You may need the first and third tape files for other Sirius products you have installed. The first tape file also contains some *Fast/Unload* sample material; the members are:

- **FUNCEQU** - a macro instruction to be used if you write any assembler language #functions. See “Customer-written Assembler #Function Packages” on page 213.

- **FUNLIST** - sample User Language procedure which provides a simple facility to monitor *Fast/Unload* requests.
  You should simply load this into a *Model 204* procedure file. Note that this procedure can be placed into a subsystem as a pre-compiled request.

- **FUNLOAD** - input to LINKFUN.
  See the description below of the LINKFUN member

- **LINKFUN** - a sample link job.
  This job can be used to re-link the *Fast/Unload* load module should this become necessary. It should not be necessary since the second tape file contains a library which contains a fully linked *Fast/Unload* load module.

- **UFUN** - a sample program which you can use as a starting point for writing assembler language #functions. See "Customer-written Assembler #Function Packages" on page 213.

D.3 CMS Installation

Fast/Unload is distributed on a magnetic tape. This magnetic tape always contains all the Sirius products you have purchased. If you are a user of a previous version of *Fast/Unload*, see the **Release Notes** accompanying the tape, highlighting the changes in the new version, and listing any compatibility issues with the previous version.

If you have purchased the *Fast/Unload User Language Interface*, please refer to the ***Sirius Mods Installation Guide*** when you have completed the basic *Fast/Unload* install. In addition, your tape will contain the *SirZap* facility; use of *SirZap* is optional. *SirZap* usage and installation instructions are described in the ***Rocket Model 204 SirZap User's Guide and Reference Manual***.

The files that will be loaded are:

- FUNLOAD MODULE
- FUNLOAD MAP
- FUNLOAD TEXT
- FUN EXEC
- FUN FUNLOAD
- FUNLIST CCAIN

To install *Fast/Unload* you must simply

1. Have the *Fast/Unload* tape mounted on a tape drive attached at virtual address 181.

2. Issue the command 'VMFPLC2 REW' to ensure the tape is properly positioned.

3. Issue the command 'VMFPLC2 LOAD' to load the object, utility files, and the load module to your 'A' disk. Allocate a minidisk with sufficient space to contain all the Sirius products you ordered. Refer to the tapemap shipped with your installation package for the actual number of blocks required, or allocate a minidisk large enough to hold all Sirius products; about 3,400 4K blocks.

4. Customize FUN EXEC to conform to local standards.

5. Place the customized FUN EXEC and FUNLOAD MODULE on a disk accessible to users who you want to have access to *Fast/Unload*. Note that *Fast/Unload* requires access to the *Model 204* CMS interface (typically called M204CMS MODULE).

Note that the distributed FUN EXEC takes as its only parameter the filename of a file containing the FUEL program. The filetype of the FUEL program is assumed to be FUNLOAD. FUN FUNLOAD is sent as an example of a simple FUEL program and can be run by issuing the command

```
FUN FUN
```

Note that FUN EXEC and FUN FUNLOAD both refer to the CLIENTS database distributed with *Model 204*. To use it against another database you must customize both FUN EXEC and FUN FUNLOAD. You might simply want to use a separate exec for each unload application or you might want to write a generalized exec that suits your environment.

After unloading FUNLOAD MODULE and FUNLOAD TEXT, you can customize them to your site's needs, if necessary; see “Customization of Defaults” on page 291.

APPENDIX E Customization of Defaults

In the descriptions below of the customization patches, you should note that each of the VER (VERify) commands presume that no prior patching has been done (i.e., they require the particular byte to be all zero). They are shown this way merely as a template and this is obviously wrong if you've made a prior customization run.

If you are doing all your patches in one run, you can indeed verify that the byte in question is all zero and then provide only the necessary REP commands after that one VER command. If you have in fact made any prior runs, you must replace the zero-byte in the VER command with the logical-or of whatever flag bits you turned on in prior patches.

E.1 Sort Parameter List

Fast/Unload communicates with an external sort package via a parameter list or **plist**. There are two basic standards for sort parameter lists:

- The 24-bit “old-fashioned” parameter list. This is supported by virtually all sort packages.
- The 31-bit extended parameter list. This is supported by DFSORT, SYNCSORT, and several other "modern" sort packages.

You can use the SORTP parameter to tell *Fast/Unload* which type of parameter list to try to use. Note, however, that *Fast/Unload* makes no attempt to verify that your sort package actually supports the requested type of plist.

The 31-bit extended parameter lists are more flexible and allow a greater variety of SORT statements to be passed to the sort package. For this reason, the default value for SORTP is 31.

If you do not have a sort package that supports 31-bit extended parameter lists at your site, you may want to change the SORTP default. To do so, simply apply the following ZAP to *Fast/Unload*.

```
NAME FUNLOAD PTCH
VER  1200 00
REP  1200 80
```

E.2 Changing the default sort parameter

The parameter that controls the selection of the destination data set for sorted output is set by *Fast/Unload* by default to SORTOUTD. Prior to version 4.1, the default was SORTOUT. You can change this default setting to be either FUNOUT or SORTOUT. These parameters are described in "FUnout" on page 11 and "SORTOut | SORTOUTD" on page 17.

To make FUNOUT the default at your site, apply the following ZAP to *Fast/Unload*:

```
NAME FUNLOAD PTCH
VER  1200 00
REP  1200 20
```

To make SORTOUT the default at your site, apply the following ZAP to *Fast/Unload*:

```
NAME FUNLOAD PTCH
VER  1214 00
REP  1214 10
```

E.3 Default for ERROR clause on PUT statement

Fast/Unload by default sets the ERROR clause to be the same as the MISSING clause on the PUT statement (except when AS STRING is specified, in which case TRUNCATE is the default for ERROR).

If you want CANCEL to be the default for the ERROR clause at your site, apply the following ZAP to *Fast/Unload*:

```
NAME FUNLOAD PTCH
VER  1200 00
REP  1200 01
```

This will be the default whether AS STRING is specified or not.

E.4 Default for MISSING clause on PUT statement

Fast/Unload by default sets the MISSING clause to be -1 on the PUT statement (except when AS STRING is specified, in which case blank fill is the default for MISSING).

To make 0 the default for the MISSING clause at your site, apply the following ZAP to *Fast/Unload*:

```
NAME FUNLOAD PTCH
VER  1200 00
REP  1200 10
```

This will not change the default if AS STRING is specified.

E.5 Default CHECK conditions and actions

During an individual run of *Fast/Unload*, you can use the CHECK statement to specify the conditions to check and the action to take if a condition is found. The CHECK statement overrides the default checks, if any, for the conditions it specifies. The defaults shipped by Rocket Software are shown in "CHECK statement defaults" on page 43.

To change any of the defaults for CHECK at your site, you can ZAP the defaults. Each of the defaults consists of a single byte, with the following values:

0 or 1	IGNORE
2	WARN
3 or greater	CANCEL

The PTCH offsets for the 6 default bytes are:

X'1202'	BROKE-PHYS
X'1203'	BROK-LOGIC
X'1204'	DUPDT in *Fast/Unload* run with neither UAI OINDEX nor UAI INV
X'1205'	PROCS in *Fast/Unload* run with no FOR EACH RECORD statement
X'1206'	INVIS in *Fast/Unload* run with no FOR EACH RECORD statement
X'1207'	DUPDT in *Fast/Unload* run with UAI OINDEX or UAI INV

Example customization of FISTAT actions:

- To unload any file, regardless of FISTAT, without affecting the job step return code:

```
NAME FUNLOAD PTCH
VER  1202 0303,0303,0003
REP  1202 0000,0000,0000
```

- To set the job step return code to a minimum of 4 for *Fast/Unload* runs where the file contains definitions of either procedures or INVISIBLE fields that are not unloaded:

```
NAME FUNLOAD PTCH
VER  1202 0303,0303,0000
REP  1202 0303,0302,0200
```

E.6 CENTSPAN and SPANSIZE

If you want to change the default value of CENTSPAN used in your load module, calculate the hexadecimal value (as a signed 2-byte number) and place that value (*vvvv*) into the following zap:

```
NAME FUNLOAD PTCH
VER  1208 FFCE  Change default CENTSPAN of -50
REP  1208 vvvv                          to ...
```

If you want to change the value of SPANSIZE used in your load module, calculate the hexadecimal value (as a 2-byte number - it must be between 1 and 100) and place that value (*vvvv*) into the following zap:

```
NAME FUNLOAD PTCH
VER  120A 005A  Change SPANSIZE of 90
REP  120A vvvv                  to ...
```

E.7 Default SORT program name

If you want to change the default name used for external sorts, calculate the hexadecimal value of the 8-character EBCDIC program name, and place that value into the following zap:

```
NAME FUNLOAD PTCH
VER  120C E2D6D9E3,404040  Change name from SORT
REP  120C vvvvvvvv,vvvvvv       to ...
```

For example, to change the name to XSORT, use the following zap:

```
NAME FUNLOAD PTCH
VER  120C E2D6D9E3,404040  Change name from SORT
REP  120C E7E2D6D9,E34040       to XSORT
```

E.8 Setting NOLIST as default

If you want to suppress the listing of FUEL programs by default at your installation, you can use the following zap:

```
NAME FUNLOAD PTCH
VER  1214 00    Unzapped byte
REP  1214 80    NOLIST is default
```

Note that when the *Fast/Unload User Language Interface* is used, NOLIST is the default unless the ALLMSG parameter is specified.

E.9 Setting default FSTATS processing

If you want to change the default behavior of FSTATS processing to be that performed by FSTATS MINMAX, use the following zap:

```
NAME FUNLOAD PTCH
VER  1214 ØØ    Unzapped byte
REP  1214 2Ø    MINMAX is default for FSTATS
```

E.10 Setting default ABENDERR

If you want to change the default of the ABENDERR parameter, you can use the following zap:

```
NAME FUNLOAD PTCH
VER  1216 ØØØØ  Old default was ABENDERR=Ø
REP  1216 ØØvv  New default is vv in hex
```

For example, the following zap sets the default ABENDERR to 8:

```
NAME FUNLOAD PTCH
VER  1216 ØØØØ  Old default was ABENDERR=Ø
REP  1216 ØØØ8  New default is ABENDERR=8
```

Note that when the *Fast/Unload User Language Interface* is used, ABENDERR=0 is always the default.

E.11 DBCS Environment

If your database files frequently contain DBCS data, you may want to tell *Fast/Unload* to automatically assume a specific DBCS environment. This way, you can avoid coding the DBCS parameter on all your *Fast/Unload* runs. The distribution version of *Fast/Unload* defaults to DBCS=NONE.

E.11.1 IBM DBCS Environment

To set an IBM default DBCS environment, apply the following zap.

```
NAME FUNLOAD PTCH
VER  12ØØ ØØ
REP  12ØØ Ø2
```

E.11.2 Fujitsu DBCS Environment

To set a Fujitsu default DBCS environment, apply the following zap.

```
NAME FUNLOAD PTCH
VER  1200 00
REP  1200 04
```

E.11.3 Hitachi DBCS Environment

To set a Hitachi default DBCS environment, apply the following zap.

```
NAME FUNLOAD PTCH
VER  1200 00
REP  1200 08
```

APPENDIX F *SMF record format*

Offset	Length	Type	Description
000	2	Signed Binary	Record length
002	2	Reserved	Block Descriptor
004	1	Bit	System indicator
005	1	Unsigned Binary	SMF record type
006	4	Signed Binary	Time in 1/100ths seconds since midnight
010	4	Packed	Date (00YYDDDF)
014	4	Character	System ID
018	8	Character	Job ID
026	10	Character	Originating Model 204 userid
036	10	Character	Originating Model 204 account name
046	1	Character	*Fast/Unload* record type 'U' Unload record 'I' Index Unload record 'C' Compile record
047	5	Reserved	
052	4	Signed Binary	User Language Interface request number
056	4	Signed Binary	Return code Times are in milliseconds
060	4	Signed Binary	Real time used (total job time)
064	4	Signed Binary	Wait for CPU time
070	4	Signed Binary	CPU Time
072	4	Signed Binary	Base buffer wait time
076	4	Signed Binary	Extension buffer wait time
080	4	Signed Binary	Output buffer wait time
084	4	Signed Binary	Report buffer wait time
088	4	Signed Binary	Input buffer wait time
092	4	Signed Binary	Open wait time
096	4	Signed Binary	PST wait time
100	4	Signed Binary	Maximum 24-bit storage used
104	4	Signed Binary	Maximum 31-bit storage used
108	4	Signed Binary	Number of base buffer reads
112	4	Signed Binary	Number of base buffer waits
116	4	Signed Binary	Number of extension pgs in base buffer
120	4	Signed Binary	Number of extension pgs in exten pool
124	4	Signed Binary	Number of extension buffer reads
128	4	Signed Binary	Number of Longstring items acquired

Note: The statistics for “Number of base buffer reads” through “Number of extension buffer reads” are new in *Fast/Unload* version 4.0; prior to that, the record length of the *Fast/Unload* SMF records was 105 bytes. The statistic for “Number of Longstring items acquired ” is new in *Fast/Unload* version 4.3.

Index

A

B

C

D

E

F

H

I

J

L

M

N

O

P

U

W

Z