

---

# Fast/Unload Release Notes Version 4.3

**May, 2006**

---



Sirius Software, Inc.  
875 Massachusetts Avenue, Suite 21  
Cambridge, MA 02139

Telephone: (617) 876-6677  
FAX: (617) 234-1200  
E-mail: [support@sirius-software.com](mailto:support@sirius-software.com)  
World Wide Web: <http://sirius-software.com>

September 6, 2006

© 2006 Sirius Software, Inc.

---

## *Proprietary Notices*

The following products:

- *Fast/Reload*
- *Fast/Unload*
- *Fast/Unload User Language Interface*
- *Sirius Mods*

are proprietary products of Sirius Software, Inc.:

**Sirius Software, Inc.**  
**875 Massachusetts Avenue, Suite 21**  
**Cambridge, Massachusetts 02139**  
**USA**

**Model 204™** is a proprietary product of Computer Corporation of America:

**Computer Corporation of America**  
**500 Old Connecticut Path**  
**Framingham, Massachusetts 01701**  
**USA**

---

# Contents

|  |            |
|--|------------|
| <b>Proprietary Notices</b> . . . . .                             | <b>ii</b>  |
| <b>Contents</b> . . . . .  | <b>iii</b> |
| <b>Chapter 1: Introduction</b> . . . . .                         | <b>1</b>   |
| <b>Chapter 2: New Features</b> . . . . .                         | <b>3</b>   |
| Lob field examples . . . . .                                     | 4          |
| Creating a NEW Lob field . . . . .                               | 4          |
| Structured unload of Lob field . . . . .                         | 5          |
| New or changed FUEL statements . . . . .                         | 6          |
| NEW fieldname WITH BLOB   CLOB . . . . .                         | 6          |
| New or changed #functions . . . . .                              | 6          |
| #CONCAT supports long string arguments and result . . . . .      | 6          |
| #LEN supports a long string argument . . . . .                   | 6          |
| #SUBSTR supports a long string argument and result . . . . .     | 7          |
| #FLOAT8: Get 8-byte float, padding 4-byte input with 0 . . . . . | 7          |
| Other FUEL changes . . . . .                                     | 8          |
| %Variables containing strings longer than 255 . . . . .          | 8          |
| Permitted use of long string values and Lobs . . . . .           | 8          |
| Statistics improvements . . . . .                                | 10         |
| <b>Chapter 3: Float Rules</b> . . . . .                          | <b>11</b>  |
| Background . . . . .   | 11         |
| Primitive operations . . . . .                                   | 11         |
| Using a float value, with decimal digit precision . . . . .      | 12         |
| Obtaining numeric values from non-floats . . . . .               | 13         |
| Assignments and length-preserved PUT statements . . . . .        | 14         |
| Length-converting PUT statements . . . . .                       | 15         |
| Arithmetic expressions . . . . .                                 | 16         |
| <b>Chapter 4: Compatibility/Fixes</b> . . . . .                  | <b>17</b>  |
| Backwards compatibility with Fast/Unload 4.2 . . . . .           | 17         |
| Type of fields created by NEW statement . . . . .                | 17         |
| IS FIXED/FLOAT with +/\$ . . . . .                               | 17         |
| Float handling . . . . .   | 17         |
| Fixes in Fast/Unload 4.3 but not in 4.2 . . . . .                | 18         |
| Fast/Unload User Language Interface . . . . .                    | 18         |
| Float handling fixes . . . . .                                   | 18         |

---

**Contents**

---

IS FIXED/FLOAT with +/- \$ . . . . . 19  
Version co-requisites . . . . . 20

---

**CHAPTER 1** ***Introduction***

This document lists the enhancements and other changes contained in the newest release of *Fast/Unload*: version 4.3. The previous generally released version of *Fast/Unload*, 4.2, was released in March, 2004.



---

**CHAPTER 2** *New Features*

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 were introduced with V6R1 of *Model 204*. The following features support this:

- Adding CLOB or BLOB to the NEW field declaration statement (“NEW fieldname WITH BLOB | CLOB” on page 6), for example, to create a Lob field by concatenating “old” non-Lob fields.
- Support for #functions that may both accept arguments and produce results in excess of 255 bytes (see “#CONCAT supports long string arguments and result” on page 6, “#LEN supports a long string argument” on page 6, and “#SUBSTR supports a long string argument and result” on page 7).
- Support for FUEL %variables that may contain strings longer than 255 bytes (see “%Variables containing strings longer than 255” on page 8).
- Support for Lobs and for string values longer than 255 bytes in specified statements, #functions, and directives (see “Permitted use of long string values and Lobs” on page 8).
- Reporting Table E page usage for each Lob field (see “Statistics improvements” on page 10).

The specific features are discussed in the rest of this chapter, following “Lob field examples” on page 4, which is a group of examples using these features.

## **2.1 Lob field examples**

### **2.1.1 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.

## 2.1.2 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.

## **2.2 New or changed FUEL statements**

This section lists new statements that may be used in *Fast/Unload*, or changes to existing statements.

### **2.2.1 NEW fieldname WITH BLOB | CLOB**

The NEW statement, which allows you to define a field name that does not exist in the input file(s), now allows you to specify that the field 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 that there is also a slight incompatibility introduced when using the NEW statement; this is explained in [“Type of fields created by NEW statement” on page 17](#).

## **2.3 New or changed #functions**

In version 4.3 of *Fast/Unload*, three #functions have been changed to support strings longer than 255 bytes. They are described in the following three subsections; See [“Lob field examples” on page 4](#) for examples of using these #functions with long string values.

After those subsections is a description of a new #function, #FLOAT8.

### **2.3.1 #CONCAT supports long string arguments and result**

The arguments of #CONCAT 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.

### **2.3.2 #LEN supports a long string argument**

The first argument of #LEN 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.

### **2.3.3 #SUBSTR supports a long string argument and result**

The first argument of #SUBSTR 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.

### **2.3.4 #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)
```

See [“Float Rules” on page 11](#) 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.

**Note:** 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 wish 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.

## 2.4 Other FUEL changes

### 2.4.1 %Variables containing strings longer than 255

The value of a %variable may now be a string longer than 255 bytes. This can arise as the result of:

|                          |   |
|--------------------------|---|
| <b>%v1 = %v2</b>         | assignment from another %variable that contains a string longer than 255 bytes        |
| <b>%v = fld</b>          | assignment from a Lob field   |
| <b>%v = #SUBSTR(...)</b> | assignment from a substring of a string value longer than 255 bytes                   |
| <b>%v = #CONCAT(...)</b> | assignment from the concatenation of strings, whose lengths total more than 255 bytes |

See [“Permitted use of long string values and Lobs”](#) for a list of contexts in which a %variable may be used if it contains a string longer than 255 bytes.

### 2.4.2 Permitted use of long string values and Lobs

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
```

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.

The following contexts allow reference to any field, Lob or not:

- The UNLOAD(C) statement
- The DELETE(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
```

## 2.5 Statistics improvements

For each Lob field, the total number of pages used in Table E is shown.

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

---

**CHAPTER 3** *Float Rules*

This chapter contains a complete specification of the handling of floating point values in *Fast/Unload*. This chapter will also be carried forward into the *Fast/Unload Reference Manual*.

Float handling in *Fast/Unload* observes the float handling in User Language. 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. Also, *Fast/Unload* contains no exact provision for the floating point value handling provided by the *Image* feature in User Language, and any information in this chapter should certainly not be extrapolated to the operation with Images.

### 3.1 Background

The general approach to numbers in *Fast/Unload* is that:

- A float value is exactly preserved if copied to a float target that has the same length.
- “[Length-converting PUT statements](#)” on page 15 specifies the rules for copying a float value to a float target that has a different length.
- 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 12.
- 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.
- 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 12.

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.

#### 3.1.1 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 16](#), which explains that after every float addition or subtraction, there is a decimal rounding step to preserve the proper number of significant digits.

### 3.1.2 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 16](#));
- 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 14](#) and [“Length-converting PUT statements” on page 15](#));
- 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 7](#));

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:

1.000000

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.

### 3.1.3 Obtaining numeric values from non-floats

When a numeric value is required in FUEL from a **string or literal 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:

```
%X = 1.000003814697265625
%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'.

## **3.2 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 output 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'.

### 3.3 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 12](#), 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 12](#). 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.

### 3.4 Arithmetic expressions

The result of an arithmetic expression is always an 8-byte float; these are produced as described in “Primitive operations” on page 11. 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'4775BCD1500000000'
/*  = X'401F9724700000000'
/*  = .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)
```

This section lists any compatibility issues with *Fast/Unload*, and any fixes contained in this version of *Fast/Unload* but not, as of the date of this release, in the immediately prior version (4.2).

## **4.1 Backwards compatibility with Fast/Unload 4.2**

This section lists any differences in processing that result from execution with *Fast/Unload* version 4.3, as compared with the same inputs to *Fast/Unload* version 4.2 at current maintenance levels.

### **4.1.1 Type of fields created by NEW statement**

In version 4.2, if you define a field in a UAI unload with the NEW statement, and then load the file with the LAI command of *Fast/Reload* without a preceding DEFINE FIELD command for the field, the field will be defined as `FRV KEY CODED UPDATE AT END`. This was an oversight, so almost certainly, anyone using the NEW statement for a field provides a DEFINE FIELD command for that field in the LAI job stream.

Now that the NEW statement allows specifying some attributes of a field (that is, CLOB or BLOB), the default field type is changed to `NFRV NKEY NCOD UPDATE IN PLACE`.

### **4.1.2 IS FIXED/FLOAT with +/\$**

In version 4.2, if you force a type conversion with the plus sign (+) or dollar sign (\$) while using the IS FIXED/FLOAT clause on an IF/ELSEIF statement, you can cause the result to be independent of the value of a field occurrence or %variable. Since Sirius believes that such FUEL code is more likely to be a coding error than intended to express a desired result, these constructs are now illegal in FUEL.

More detailed information is given in [“IS FIXED/FLOAT with +/\\$” on page 19](#).

### **4.1.3 Float handling**

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 fixes have been made to float handling that can produce different results. See [“Float handling fixes” on page 18](#) for a general description of the changes.

**Note:** Although Sirius does not believe these changes affect the operation of *Fast/Unload* at any customer site, it is strongly recommended that all customers upgrade to version 4.3 to obtain the revised, correct float handling now in place.

For a complete specification of the handling of floating point values in *Fast/Unload*, see “[Float Rules](#)” on page 11. That chapter will be carried forward into the *Fast/Unload Reference Manual*.

## 4.2 Fixes in Fast/Unload 4.3 but not in 4.2

This section lists other fixes to features existing in *Fast/Unload* version 4.2 but which, in the absence of customer problems, have not been fixed in that version (as of the date of the release).

### 4.2.1 Fast/Unload User Language Interface

Previously, in unusual circumstances, certain kinds of updates to a file being unloaded by the *Sirius Mods* \$Funload function can fail to be reflected in the unloaded file.

Only one of these updates impacts the “data” output (for example, the FUNOUT DD): For the very first addition in the file of an ORDERED field (that is, creating the first index entry for that field), if a UAI OINDEX/INVIS is being performed, the index entry may be missing from the unload.

In addition, some of the Table B statistics in the unload may be incorrect, such as the number of Table B pages in use, and the number of base records in the file.

This problem is fixed in version 4.3 of *Fast/Unload*, together with a fix in the *Sirius Mods* (for example, ZAP66D6, or the commercially released version 6.7 or later).

### 4.2.2 Float handling fixes

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.

The principal problem was that, prior to version 4.3, the results of a FUEL program could 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 have worked 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 did 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, could have produced the above errors — and possibly others, as well (particularly if FLOAT LEN 16 fields are explicitly referenced in a FUEL program).

**Note:** There are other odd cases in which float handling was performed incorrectly prior to version 4.3. Again, Sirius does not believe the changes in version 4.3 affect the operation of *Fast/Unload* at any customer site, but it is strongly recommended that all customers upgrade to version 4.3 to obtain the revised, correct float handling now in place.

As mentioned in “[Float handling](#)” on page 17, this fix represents a small incompatibility.

See “[Float Rules](#)” on page 11 for a complete specification of the handling of floating point values in *Fast/Unload*; that chapter has been carried forward into the *Fast/Unload Reference Manual*.

### 4.2.3 IS FIXED/FLOAT with +/-

There are two clauses on the IF and ELSEIF statements that can be used to check the type of a field occurrence or of a %variable's contents:

IS FIXED

and

IS FLOAT

In addition, a **comparison** of an entity in an IF or ELSEIF statement can be performed using the value of the entity converted to a fixed (that is, integer) or floating point (for example, fractional) number. A fixed conversion is signified by use of a dollar sign (\$) prefixing the entity, and a float conversion is signified by use of a plus sign (+) prefixing the entity.

These conversions are available to change the default comparison type of a field or %variable to another field or %variable, which is done using a string comparison. For example, with the following FUEL fragment:

```
%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

In version 4.2, these two mechanisms (type checking and type conversion) could be combined in a single test. Most combinations of +/\$ type conversion with the IS FIXED/FLOAT clause on an IF/ELSEIF statement cause the result to be independent of the value of a field occurrence or %variable. Since Sirius believes that such FUEL code is more likely to be a coding error than intended to express a desired result, these constructs are now illegal in FUEL.

As mentioned in [“IS FIXED/FLOAT with +/\\$” on page 17](#), this fix represents a small incompatibility.

### **4.3 Version co-requisites**

This section lists any restrictions on usage of various products (including *Fast/Unload* itself) which will be imposed by use of version 4.3 of *Fast/Unload*.

Strictly speaking, there are no version co-requisites. However, in order to obtain the fix described in [“Fast/Unload User Language Interface” on page 18](#), you should be using a version of the *Sirius Mods* containing the fix mentioned in that section.