

---

# Fast/Unload Release Notes Version 4.6

**Preliminary Version**

**??, 2012**

---



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>

February 21, 2012

© 2012 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, a wholly-owned subsidiary of Rocket Software, Inc., which owns the trademark:

**Rocket Software Corporate Office**  
**M204 Division**  
**275 Grove Street**  
**Suite 3-410**  
**Newton, Massachusetts 02466-2272**  
**USA**

---

## Contents

<b>Proprietary Notices</b> . . . . .	<b>ii</b>
<b>Contents</b> . . . . .	<b>iii</b>
<b>Chapter 1: Introduction</b> . . . . .	<b>1</b>
Differences between versions 4.5 and 4.6 . . . . .	1
z/OS or z/CMS required for DATETIME fields . . . . .	2
<b>Chapter 2: Maintenance and Support</b> . . . . .	<b>3</b>
Improved error messages . . . . .	3
Documentation . . . . .	3
<b>Chapter 3: New Features</b> . . . . .	<b>5</b>
Support for FILEORG X'100' files - summary . . . . .	5
Record structure in FILEORG X'100' files, context and other fieldgroup concepts . . . . .	6
Nested fieldgroups and FG * . . . . .	8
FOR FIELDGROUP blocks . . . . .	10
Statements which cannot reference fieldgroup members . . . . .	10
References to fieldgroup members not in fieldgroup context . . . . .	11
First occurrence . . . . .	12
EXACTLY-ONE fields . . . . .	13
Handling of DEFAULT-VALUE fields . . . . .	13
Handling of missing AT-MOST-ONE fields . . . . .	14
Nested FOR FIELDGROUP blocks . . . . .	15
Nested fieldgroups . . . . .	16
Support for V7R4 . . . . .	16
New or changed FUEL statements . . . . .	17
FOR FIELDGROUP fgrpName[(occ)] . . . . .	17
FOR FIELDGROUP fgrpName = id . . . . .	18
FOR EACH FIELDGROUP fgrpName . . . . .	19
LEAVE FIELDGROUP . . . . .	20
PAI . . . . .	21
New or changed #functions and special variables . . . . .	21
#FIELDGROUPLD . . . . .	21
#SHADIGEST: SHA-1 digest ("hash") of string . . . . .	21
Other FUEL changes . . . . .	22
Asterisk in PUT statement syntax . . . . .	22
Revised syntax for the PUT statement . . . . .	23

MISSING and ERROR clauses . . . . .	24
Pre-4.5 defaults for the MISSING and ERROR clauses . . . . .	27
PUT examples . . . . .	28
Statistics improvements . . . . .	28
FSTATS for FILEORG X'100' files . . . . .	28
<b>Chapter 4: Compatibility/Fixes . . . . .</b>	<b>29</b>
Backwards compatibility with Fast/Unload 4.4 and 4.5 . . . . .	29
Detect PUT syntax error immediately after FIXED/DECIMAL/ZONED . . . . .	29
Disallow PUT constant>255 AS FIXED(1) . . . . .	30
Handle quotes in MISSING clause of UAI SORT . . . . .	30
Revised description of UAI MISSING . . . . .	31
#IF errors which are now detected . . . . .	31
Backwards compatibility with Fast/Unload 4.5 . . . . .	32
Differences in PUT ONE DV fields with no MISSING literal . . . . .	32
Fixes in Fast/Unload 4.6 but not in 4.5 . . . . .	32
UAI of EXACTLY-ONE fields . . . . .	33
PUT of ONE DV xx field AS non-STRING with no MISSING clause . . . . .	33
[NO]UNLOAD of outer EXACTLY-ONE field . . . . .	33
References to AT-MOST-ONE fieldgroup members . . . . .	33
DEFAULT-VALUE for AT-MOST-ONE field in UAI SORT . . . . .	34
Bug in PAI if fieldgroup present and outer field added . . . . .	34
Preventing illegal access to UTF8 fields . . . . .	34
Fixes in Fast/Unload 4.6 but not in 4.4 . . . . .	34
Allow MISSING or ERROR immediately after FIXED/DECIMAL/ZONED in PUT . . . . .	34
Long string access to FLOAT LEN 4 field . . . . .	35
Separate extension record stats for each file in group . . . . .	35
Fixes to #IF . . . . .	35
Version co-requisites . . . . .	36

---

**CHAPTER 1** *Introduction*

**This is a work-in-progress document describing possible contents of a software release.**

**Until the the commercial release of the software, we reserve the right to remove or change anything described herein.**

This document lists the enhancements and other changes contained in the newest release of *Fast/Unload*: version 4.6, which is still under development. The previous generally released version of *Fast/Unload*, 4.4, was released in October, 2007, and an interim version., 4.5, was released in May, 2010.

## 1.1 Differences between versions 4.5 and 4.6

Much of the information in these, the "Version 4.6 Release Notes", also applies to changes which were made in the interim version 4.5 of *Fast/Unload*. Changes listed in these release notes which do **not** apply to version 4.5, are listed in this section.

Version 4.5 does not support the following statements:

- FOR FIELDGROUP fgrpName = id [on page 18](#)
- FOR EACH FIELDGROUP fgrpName [on page 19](#)
- LEAVE FIELDGROUP [on page 20](#)

In addition, although the FOR FIELDGROUP fgrpName[(occ)] statement ([on page 17](#)) is allowed in version 4.5, it has little practical use. In 4.5 it does not establish context for fieldgroup members; it only establishes context for #FIELDGROUPID ([on page 21](#)).

Other differences between version 4.5 and version 4.6 are:

1. In 4.5, the UAI output for a record with any EXACTLY-ONE field which is not physically present will contain the default value (either the DEFAULT-VALUE or, if none, the null string). In 4.6, only physically present EXACTLY-ONE occurrences are output for UAI. This is primarily a performance and file size issue, but also, as described in "UAI of EXACTLY-ONE fields" [on page 33](#), there are some UAI/LAI bugs fixed by this change.
2. Asterisks in the syntax of the PUT statement (see "Asterisk in PUT statement syntax" [on page 22](#)) are not available in 4.5.

3. Although the DEFAULT-VALUE of an AT-MOST-ONE field serves as the value for PUT of the first occurrence, if it is missing and there is no MISSING clause, the other handling of missing AT-MOST-ONE DEFAULT-VALUE fields (as specified in [“Handling of missing AT-MOST-ONE fields” on page 14](#)) is not performed in version 4.5.

Finally, in addition to the above functional differences, bugs were fixed in version 4.6 that have not been fixed (that is, via zaps) in version 4.5; these are listed in [“Fixes in Fast/Unload 4.6 but not in 4.5” on page 32](#).

## **1.2 z/OS or z/CMS required for DATETIME fields**

If the FUEL program contains any statement which requires conversion of a DATETIME field (for example, any PUT statement referencing the field, any comparison, the PAI statement, and so on), *Fast/Unload* must be running under z/OS or z/CMS.

---

**CHAPTER 2** *Maintenance and Support*

These enhancements to *Fast/Unload* do not affect its intrinsic functionality, but rather affect the way Sirius Software delivers maintenance and support for the product.

## 2.1 Improved error messages

The following *Fast/Unload* error messages have been improved:

### Improved I/O error message

Error message FUNL0042 now contains the ending page number, if the read is for multiple Table B pages (for example, a track of pages when SBBUF=1). In that case, the last page number in the group of pages being read is shown after the first page being read. This may be of assistance in diagnosing the problem in the file.

### Line number on PUT condition error message

Error message FUNL0052 now contains the line number being executed in the FUEL program when a PUT condition (for example, `PUT ... ERROR CANCEL`) cancels *Fast/Unload*.

## 2.2 Documentation

The changes to the documentation of the PUT statement, described in [“Revised syntax for the PUT statement” on page 23](#), apply to *Fast/Unload* prior to versions 4.6 and 4.5, except for those aspects which are new, namely:

- the "asterisk placeholder" (`STRING(*)`, `MISSING *`, and `ERROR *` — it also is an "override" for `ERROR`);
- the treatment of AT-MOST-ONE DEFAULT-VALUE fields.



---

**CHAPTER 3** *New Features*

All new features in version 4.6 of *Fast/Unload* are presented in the subsections of this chapter.

### 3.1 Support for FILEORG X'100' files - summary

This version of *Fast/Unload* supports files with FILEORG X'100', which were introduced in version V7R2 of *Model 204*. The various aspects of this support are:

**UAI** UAI FUEL programs may be used to unload FILEORG X'100' files, for reloading with *Fast/Reload*.

**PAI** The PAI statement in FUEL supports FILEORG X'100' files; see [“PAI” on page 21](#).

**FSTATS** The field statistics produced by the FSTATS option or statement include information about fields and fieldgroups in a FILEORG X'100' file, including the new field attributes. See [“FSTATS for FILEORG X'100' files” on page 28](#).

#### **FOR FIELDGROUP blocks**

New FUEL statements are provided to establish a fieldgroup context for processing members of the fieldgroup. An overview of these statements is provided in [“FOR FIELDGROUP blocks” on page 10](#).

#### **Occurrences of EXACTLY-ONE and AT-MOST-ONE fieldgroup members**

You may refer to occurrences of non-nested, non-FG-\*, non-REPEATABLE fields without an enclosing FOR FIELDGROUP block. This facility is explained in [“References to fieldgroup members not in fieldgroup context” on page 11](#).

#### **EXACTLY-ONE fields**

EXACTLY-ONE fields are new in V7R2; their handling in FUEL is explained in [“EXACTLY-ONE fields” on page 13](#).

#### **Reference to the number of occurrences of a fieldgroup**

Just as you can refer to the number of occurrences of a field as an entity in a FUEL program, you can also refer to the number of occurrences of a fieldgroup, by coding the fieldgroup name followed by the “#” token in parentheses:

SOMEFIELDGROUP(#)

### Handling of DEFAULT-VALUE fields

This is explained in “[Handling of DEFAULT-VALUE fields](#)” on page 13. Also, for PUT of an AT-MOST-ONE DEFAULT-VALUE field which is MISSING, see “[Handling of missing AT-MOST-ONE fields](#)” on page 14.

### Handling of missing AT-MOST-ONE fields

This is explained in “[Handling of missing AT-MOST-ONE fields](#)” on page 14.

There are many other field attributes introduced in FILEORG X'100' files, and they do not have any impact on version 4.6 of *Fast/Unload*. For example, *Fast/Unload* does not enforce the DATETIME format restriction in the ADD or CHANGE statement.

## 3.2 Record structure in FILEORG X'100' files, context and other fieldgroup concepts

A record in a *Model 204* file with FILEORG X'100' consists of a sequence of field and/or fieldgroup occurrences; these occurrences are called **outer** occurrences. A fieldgroup occurrence consists of a sequence of field and/or fieldgroup occurrences; these occurrences are called **member** occurrences. Each fieldgroup occurrence has a numeric ID, which is different from all other fieldgroup occurrence IDs in the same record.

A fieldgroup is defined in a file by using the DEFINE FIELDGROUP command. Note that this is different from the FIELDGROUP attribute which can be specified on either the DEFINE FIELD or DEFINE FIELDGROUP command. Note also that the FIELDGROUP attribute can be abbreviated as FG, which is how we will refer to it in this document.

The DEFINE command for a field or fieldgroup can specify that it is a member of a specific fieldgroup by using the FG attribute with the name of the fieldgroup. When a field or fieldgroup is defined as a member of a specific fieldgroup, that field or fieldgroup can only occur as a member within some occurrence of the fieldgroup named in its FG attribute.

Fields and fieldgroups defined without the FG attribute cannot occur within a fieldgroup, so when they occur in a record, they occur as outer occurrences.

These features are powerful and enable definition of a straightforward approach to repeating field groups, as shown in this example:

```
DEFINE FIELD OUTFLD
DEFINE FIELDGROUP GRP
DEFINE FIELD EXOMEM WITH FG GRP
DEFINE FIELD EXOMEM2 WITH FG GRP
DEFINE FIELD REPMEM WITH REPEATABLE FG GRP
```

An example PAI of a record in a file with these definitions is:

```
OUTFLD = OUT01
OUTFLD = OUT02
\GRP = 20
  INGRP = MEM01
  INGRP2 = MEM02
/GRP = 20
OUTFLD = OUT03
\GRP = 5
  INGRP = MEM03
  INGRP2 =
  REPMEM = REPMEM01
/GRP = 5
OUTFLD = OUT07
```

Notes:

- The order of fieldgroup IDs, as in this example (20 and 5) need not correspond to the order of the fieldgroups in the record. The ID of a fieldgroup is assigned when it is added to the record, and is equal to one more than the highest fieldgroup ID which had been used in the record (even if the fieldgroup with that ID has been deleted).
- The default repeatability for (non-FG \*) fieldgroup member **fields** is EXACTLY-ONE. In the second occurrence of GRP above, whether or not INGRP2 has been stored cannot be determined; it is treated in all respects as if it had been stored with a value of the null string.

In order for *Fast/Unload* to operate on an occurrence of a fieldgroup member, the fieldgroup containing the member must be identified. This is obtained in either of two ways:

- by referring to the member of the fieldgroup within a FOR FIELDGROUP block which identifies an occurrence of the fieldgroup — this fieldgroup occurrence is the **fieldgroup context** for the fieldgroup member;
- for an AT-MOST-ONE or EXACTLY-ONE non-FG \* field member of a non-nested fieldgroup, a reference outside the context of the fieldgroup implicitly identifies the fieldgroup occurrence; see [“References to fieldgroup members not in fieldgroup context” on page 11](#).

The following FUEL program explains both cases of fieldgroup member reference, assuming that the record shown in the above PAI output is the current record:

```
FOR EACH RECORD
  FOR FIELDGROUP GRP(1)
    PUT EXOMEM /* In context of GRP(1): MEM01
  END FOR
  FOR FIELDGROUP GRP(2)
    PUT EXOMEM /* In context of GRP(2): MEM03
    PUT REPMEM /* In context of GRP(2): REPMEM01
  END FOR
  PUT EXOMEM(1) /* Out of context: MEM01
  PUT EXOMEM(2) /* Out of context: MEM03
  * REPMEM illegal here; it requires a fieldgroup context
END FOR
```

In addition to fieldgroup context:

- **Record context** is always available for fields and fieldgroups defined without the FG attribute.

### 3.2.1 Nested fieldgroups and FG \*

Two advanced aspects of the fieldgroup feature, not explored in [“Record structure in FILEORG X'100' files, context and other fieldgroup concepts”](#) on page 6, are:

1. The DEFINE FIELDGROUP command allows the FG attribute, which indicates that an occurrence (or more, if AT-MOST-ONE is not specified) of the fieldgroup being defined may be contained within an occurrence of the fieldgroup specified in the FG attribute. A fieldgroup occurrence contained within another fieldgroup is called a **nested** fieldgroup occurrence.
2. The FG attribute of the DEFINE FIELD or DEFINE FIELDGROUP command can specify FG \*; this indicates that the field or fieldgroup can occur either or both as an outer occurrence or as a member of any occurrence of any fieldgroup in a record. A field or fieldgroup defined with the FG \* attribute is called a FG \* field or fieldgroup, respectively.

Here is a contrived example illustrating the possibilities using the FG attribute:

```
DEFINE FIELD OUTFLD
DEFINE FIELD STARFLD WITH FG *
DEFINE FIELDGROUP STARGRP WITH FG *
DEFINE FIELD INSTAR WITH FG STARGRP
DEFINE FIELDGROUP GRP
DEFINE FIELD INGRP WITH FG GRP
DEFINE FIELDGROUP NEST WITH FG GRP
```

An example PAI of a record in a file with these definitions is:

```
OUTFLD = OUT01
OUTFLD = OUT02
STARFLD = STAROUT01
STARFLD = STAROUT02
\STARGRP = 15
  INSTAR = INSTAR01
  STARFLD = STARMEM01
  STARFLD = STARMEM02
  \STARGRP = 30
  INSTAR =
  STARFLD = STARMEM03
  STARFLD = STARMEM04
  /STARGRP = 30
/SSTARGRP = 15
STARFLD = STAROUT03
OUTFLD = OUT03
\GRP = 20
  INGRP = MEM01
  STARFLD = STARMEM05
  \STARGRP = 35
  INSTAR =
  STARFLD = STARMEM06
  STARFLD = STARMEM07
  /STARGRP = 35
/GRP = 20
\GRP = 5
  INGRP = MEM02
/GRP = 5
```

Note that since FG \* fields and fieldgroups can exist as both outer and member occurrences, that a reference to *fstar*, where *fstar* is defined with FG \*, will be to the occurrence of *fstar* within the fieldgroup occurrence established by the closest containing FOR FIELDGROUP block, or to the outer occurrence of *fstar* if there is no enclosing FOR FIELDGROUP block. For example, if the current record is the one described by the above PAI output:

```
FOR EACH RECORD
  PUT STARFLD /* In record context: STAROUT01
  FOR FIELDGROUP GRP /* In record context: ID=20
    PUT STARFLD /* In fieldgroup context: STARMEM05
    FOR FIELDGROUP STARGRP /* In fieldgroup context: ID=35
      PUT STARFLD /* In fieldgroup context: STARMEM06
    END FOR
  END FOR
END FOR
```

See [“Nested fieldgroups” on page 16](#) for further discussion of nested fieldgroups.

### 3.3 FOR FIELDGROUP blocks

Three new statements, all with syntax "FOR ... FIELDGROUP ...", create **FOR FIELDGROUP blocks**, which are terminated by `END FOR`. Within each block, a fieldgroup occurrence (one for each iteration, in the case of `FOR EACH FIELDGROUP`) is used as the context for references to its members. These statements are described in the following sections:

- `FOR FIELDGROUP fgrpName[(occ)]` [on page 17](#)
- `FOR FIELDGROUP fgrpName = id` [on page 18](#)
- `FOR EACH FIELDGROUP fgrpName` [on page 19](#)

In addition, within any of the above blocks there is a new statement for terminating execution of the block (`LEAVE FIELDGROUP` [on page 20](#)) and a new special variable for obtaining the fieldgroup ID of a fieldgroup occurrence (`#FIELDGROUPID` [on page 21](#)).

Note that in addition to this new syntax for referencing fieldgroup members, some fieldgroup members can also be referenced outside `FOR FIELDGROUP` blocks in certain circumstances, as described in "[References to fieldgroup members not in fieldgroup context](#)" [on page 11](#).

### 3.4 Statements which cannot reference fieldgroup members

For a field in the context of an occurrence of its fieldgroup (established by a containing `FOR FIELDGROUP` block), or for a fieldgroup member which can be used outside its fieldgroup context (as described in "[References to fieldgroup members not in fieldgroup context](#)" [on page 11](#)), that field can be used in any `FUEL` statement, **except** the following:

- in the `UNLOAD[C] <field>` or `NOUNLOAD <field>` statement
- in the `DELETE[C] <field>` statement
- in the sort or hash specification of the `SORT` or `UAI` statement

### 3.5 References to fieldgroup members not in fieldgroup context

A fieldgroup member can always be referenced within the context of an occurrence of its fieldgroup established by a FOR FIELDGROUP block, as described in “FOR FIELDGROUP blocks” on page 10 (although, as listed in “Statements which cannot reference fieldgroup members” on page 10, certain statements may not reference fieldgroup members).

In addition to the fieldgroup context established by FOR FIELDGROUP blocks, you may make references to certain fieldgroup members without a containing context. A fieldgroup member can only be referenced outside its fieldgroup context if it is EXACTLY-ONE or AT-MOST-ONE, and it is non-FG \*, and its containing fieldgroup is not nested.

Stated another way, the following references are only allowed within fieldgroup context for fieldgroup members or, for non-fieldgroup members, within record context:

- REPEATABLE member of fieldgroup
- FG \* field
- Field in nested fieldgroup
- Nested fieldgroup

For fieldgroup members which allow it, an "out of context" reference to an occurrence is actually a reference to occurrence number 1 of that field within the specified occurrence of its containing fieldgroup. For example, consider the following field definitions and FUEL program:

```
DEFINE FIELDGROUP GRP
DEFINE FIELD F00 WITH FG GRP
...
//FUNIN DD *
OPEN ...
FOR EACH RECORD
  PUT F00(10)
  OUTPUT
END FOR
```

In the above, F00(10) is a reference to occurrence number 1 of F00 within occurrence number 10 of fieldgroup GRP.

Further, an "out of context" reference to the occurrence count of an EXACTLY-ONE (or AT-MOST-ONE, as well) member is actually a reference to the occurrence count of its

containing fieldgroup. For example (assuming the same definitions as above), consider the following FUEL fragment:

```
FOR I FROM 1 TO FOO(#)
  ...
END FOR
```

In the above fragment, the FOR loop is executed as many times as the occurrence count of fieldgroup GRP.

### 3.5.1 First occurrence

In many discussions of EXACTLY-ONE and AT-MOST-ONE fields, the term *first occurrence* is used. This refers to any of the following:

- occurrence number 1 of an outer EXACTLY-ONE or AT-MOST-ONE field.
- occurrence number 1 of an EXACTLY-ONE or AT-MOST-ONE fieldgroup member, in the context of the containing fieldgroup.
- any occurrence *n* of an EXACTLY-ONE or AT-MOST-ONE fieldgroup member, not in the context of the containing fieldgroup, **if** occurrence *n* of the fieldgroup exists.

Hence, given the following definitions within a file:

```
DEFINE FIELD OUTER WITH AT-MOST-ONE
DEFINE FIELDGROUP GRP
DEFINE FIELD INNER WITH AT-MOST-ONE FG GRP
DEFINE FIELD BOTH WITH AT-MOST-ONE FG *
```

Then the following FUEL program contains comments illustrating which references are to the first occurrence (all of the comments are also true if any AT-MOST-ONE above is changed to EXACTLY-ONE):

```
FOR EACH RECORD
  PUT OUTER      /* This is first occurrence
  PUT OUTER(3)   /* This is NOT first occurrence
  PUT BOTH       /* This is first occurrence
  PUT BOTH(3)    /* This is NOT first occurrence
  PUT INNER      /* This is first occurrence, if fieldgroup
                  /* GRP has at least one occurrence
  PUT INNER(3)   /* This is first occurrence, if fieldgroup
                  /* GRP has at least three occurrences
  FOR FIELDGROUP GRP
    PUT BOTH      /* This is first occurrence
    PUT BOTH(3)   /* This is NOT first occurrence
    PUT INNER     /* This is first occurrence
    PUT INNER(3)  /* This is NOT first occurrence
  END FOR
END FOR
```

### 3.6 EXACTLY-ONE fields

One of the new field attributes in *Model 204 V7R2* is the EXACTLY-ONE "repeatability" attribute, which designates that there is one and only one occurrence of the field, either within its containing fieldgroup occurrence or within the record:

- An EXACTLY-ONE field which is a fieldgroup member occurs exactly once in every occurrence of that fieldgroup.
- An EXACTLY-ONE field which is **not** a fieldgroup member (that is, an outer field) occurs exactly once in every record in the file.

EXACTLY-ONE is the default repeatability attribute for a **field** defined as a member of a (specific) fieldgroup. EXACTLY-ONE is not allowed in combination with FG \*, nor is it allowed on the DEFINE FIELDGROUP command. The other repeatability attributes are REPEATABLE, AT-MOST-ONE, and OCCURS. For fieldgroups, fields which are not fieldgroup members, and FG \* fields, REPEATABLE is the default repeatability attribute.

*Fast/Unload* handling of EXACTLY-ONE fields is straightforward:

- The DELETE statement is not allowed with an EXACTLY-ONE field.
- The ADD statement is not allowed with an EXACTLY-ONE field (note that the analogous constraint for AT-MOST-ONE fields, which **is** enforced in User Language, is **not** enforced in FUEL).
- The first occurrence (see [“First occurrence” on page 12](#)) of an EXACTLY-ONE field is never MISSING (that is, on the PUT statement, the SORT clause of the UAI statement, or on the IF/ELSEIF statements - and it always EXISTS on IF/ELSEIF).
- Within its context, the occurrence count of an EXACTLY-ONE field is always one.
- See [“References to fieldgroup members not in fieldgroup context” on page 11](#) for a discussion of an EXACTLY-ONE fieldgroup member outside its context.

In the above, *its context* is either the context of an occurrence of its fieldgroup, for fieldgroup members, or record context, for outer fields.

### 3.7 Handling of DEFAULT-VALUE fields

The DEFAULT-VALUE attribute, allowed in `FILEORG X'100'` files, specifies the value of the first occurrence (see [“First occurrence” on page 12](#)) of an EXACTLY-ONE or AT-MOST-ONE field if it has not been stored.

For an EXACTLY-ONE field, the result of this in FUEL is that any reference to the value of the first occurrence of the field, if it has not been stored, results in the DEFAULT-

VALUE (and, note that there is no way in User Language nor in FUEL to determine whether or not the field has been stored).

For an AT-MOST-ONE DEFAULT-VALUE field, any reference to the value of the first occurrence of the field, if it has not been stored, also generally results in the DEFAULT-VALUE but, since the field occurrence may also be MISSING, the PUT statement result may be something other than the DEFAULT-VALUE, as explained in [“Handling of missing AT-MOST-ONE fields”](#).

### **3.8 Handling of missing AT-MOST-ONE fields**

Missing field processing is done for:

1. the MISSING/EXISTS clauses on the IF/ELSEIF statements
2. the PUT statement
3. the #ERROR special variable
4. retrieving a value
5. the UAI SORT statement

In addition, all of the above processing is performed for a %variable to which a field occurrence has been assigned.

Prior to the introduction of FILEORG X'100' files, the handling of all missing field occurrences was the same. However, when the DEFAULT-VALUE attribute is used with an AT-MOST-ONE field, if the first occurrence (see [“First occurrence” on page 12](#)) of the field is missing, it still has a value (the value of the DEFAULT-VALUE attribute).

When an AT-MOST-ONE field has a DEFAULT-VALUE, and that field is missing, then the following take place when the **first** occurrence of the field is referenced:

1. The MISSING test for the first occurrence is true.
2. When the first occurrence is used in a PUT statement which **does not** contain the MISSING clause other than `MISSING *`:
  - If the DEFAULT-VALUE is convertible to the PUT AS format (a DEFAULT-VALUE longer than the length of a PUT AS STRING statement is **not** convertible), then the DEFAULT-VALUE is output.
  - Otherwise, an ERROR condition occurs (**in addition** to the MISSING condition). See [“Revised syntax for the PUT statement” on page 23](#) for a description of the ERROR clause in the PUT statement.

3. After a PUT of the first occurrence, #ERROR is 1 or, if the DEFAULT-VALUE is not convertible to the output format as just described, it is 3.
4. The value used for the field in UAI SORT is the DEFAULT-VALUE.

This same behavior will take place for a %variable, if the %variable has been assigned from the missing first occurrence of an AT-MOST-ONE DEFAULT-VALUE field.

The DEFAULT-VALUE of a field (either AT-MOST-ONE or EXACTLY-ONE) has no effect on references to occurrences other than the first.

### **3.9 Nested FOR FIELDGROUP blocks**

A FOR FIELDGROUP block may contain other FOR FIELDGROUP blocks, and the context established by a containing block "remains active" within a contained block (unless the fieldgroup specified on the contained block is the same as that specified in a containing block). For example, given the following definitions:

```
DEFINE FIELDGROUP PERSON
DEFINE FIELD NAME WITH FG PERSON
DEFINE FIELD PERSON.SOCSECNUM WITH FG PERSON
DEFINE FIELDGROUP BANKACCT
DEFINE FIELD BALANCE WITH FG BANKACCT
DEFINE FIELD ACCT.SOCSECNUM WITH FG BANKACCT
```

The following FUEL program aggregates each person's bank balances:

```
FOR EACH RECORD
FOR EACH FIELDGROUP PERSON
  %BAL = 0
  FOR EACH FIELDGROUP BANKACCT
    IF ACCT.SOCSECNUM EQ PERSON.SOCSECNUM THEN
      %BAL = %BAL + BALANCE
    END IF
  END FOR
  PUT NAME
  PUT ' balance='
  PUT %BAL
  OUTPUT
END FOR /* EACH FIELDGROUP PERSON
END FOR
```

Notice that IF ACCT.SOCSECNUM EQ PERSON.SOCSECNUM references field ACCT.SOCSECNUM in the inner (BANKACCT) block and references field PERSON.SOCSECNUM in the outer (PERSON) block.

In the above example, and in many applications, nested FIELDGROUP blocks are not processing nested fieldgroups, but whenever processing nested fieldgroups, nested FOR FIELDGROUP blocks are required, as shown in “[Nested fieldgroups](#)”.

### 3.10 Nested fieldgroups

A **nested fieldgroup** is a fieldgroup which is itself defined to be a member of another fieldgroup; for example:

```
DEFINE FIELDGROUP PERSON
DEFINE FIELD NAME WITH FG PERSON

DEFINE FIELDGROUP MEDICALEXAM WITH FG PERSON
DEFINE FIELD EXAM.DATE WITH FG MEDICALEXAM
```

In the above example, fieldgroup MEDICALEXAM is nested within, that is, it is a member of, fieldgroup PERSON.

To access occurrences of nested fieldgroups, you must use nested FIELDGROUPS; for example:

```
FOR EACH RECORD
FOR EACH FIELDGROUP PERSON
  PUT NAME
  %NEED_EXAM = '(no exams)'
  FOR EACH FIELDGROUP MEDICALEXAM
    PUT EXAM.DATE AT 30
    OUTPUT
    %NEED_EXAM = ''
  END FOR
  IF %NEED_EXAM NE '' THEN
    PUT %NEED_EXAM AT 30
    OUTPUT
  END FOR
END FOR /* EACH FIELDGROUP PERSON
END FOR
```

### 3.11 Support for V7R4

A few changes in V7R4 of *Model 204* required changes to *Fast/Unload*:

- Using the *Fast/Unload User Language Interface* with *Model 204 V7R4* requires *Fast/Unload* version 4.6.

- Processing a file with BLOB or CLOB fields which have the DEFAULT-VALUE attribute requires *Fast/Unload* version 4.6.
- *Fast/Unload* is not equipped to "fetch", ADD, or CHANGE UTF-8 fields, and so such statements are not allowed in FUEL. If UTF8FLD has the UTF8 attribute, only the following types of references to it are allowed:
  - UTF8FLD(#)
  - DELETE UTF8FLD or DELETE UTF8FLD(occur)
  - DELETEC UTF8FLD or DELETEC UTF8FLD(occur)
  - UNLOAD UTF8FLD or UNLOAD UTF8FLD(occur)
  - UNLOADC UTF8FLD or UNLOADC UTF8FLD(occur)
  - NOUNLOAD UTF8FLD or NOUNLOAD UTF8FLD(occur)

Also, the PAI statement may not be used if any of the files which are input to *Fast/Unload* contains a field with the UTF8 attribute.

Version 4.5 does not enforce these UTF8 field restrictions, as noted in [“Preventing illegal access to UTF8 fields” on page 34](#).

## 3.12 New or changed FUEL statements

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

These statements are all in support of fieldgroups in FILEORG X'100' files.

### 3.12.1 FOR FIELDGROUP fgrpName[(occ)]

The following block creates a context within which references to fieldgroup members and #FIELDGROUPID use the given fieldgroup occurrence:

```
FOR FIELDGROUP fgrpName[(occ)]
  . . . statements which use fgrpName[(occ)] as the
  . . . fieldgroup context
END FOR
```

The **fgrpName** clause in the statement can be replaced by the name of any fieldgroup defined in the file.

The **(occ)** clause in the statement is optional and specifies the occurrence number of the fieldgroup being used as the context; just as with the occurrence number of a regular field in FUEL, it may be a positive integer count, a loop control variable, or a %variable. If it is a %variable, the value of the %variable must be a number greater than or equal to 1; any fractional part is dropped.

The given occurrence of the specified fieldgroup, within the current context, is the context for any members of that fieldgroup. If the **(occ)** clause is omitted, the first occurrence of the fieldgroup is used. If the occurrence of the fieldgroup does not exist within the current context, the entire block is skipped.

For example, assuming the following field definitions:

```
DEFINE FIELD FIELDA (FG GRP)
DEFINE FIELD FIELDB (FG GRP)
```

the following statements put, on the output stream, the values of the fields in the second occurrence of fieldgroup GRP:

```
FOR FIELDGROUP GRP(2)
  PUT FIELDA
  PUT FIELDB
  OUTPUT
END FOR
```

Note: if GRP is not nested, and each of FIELDA and FIELDB is either EXACTLY-ONE, or is AT-MOST-ONE and not FG \*, the following statements are equivalent to the above:

```
PUT FIELDA(2)
PUT FIELDB(2)
OUTPUT
```

There is no significant difference in processing time between the above two approaches.

### 3.12.2 FOR FIELDGROUP fgrpName = id

The following block creates a context within which references to fieldgroup members and #FIELDGROUPID use the fieldgroup occurrence whose ID is specified:

```
FOR FIELDGROUP fgrpName = id
  . . . statements which use fgrpName with the
  . . . specified id as the fieldgroup context
END FOR
```

The **fgrpName** clause in the statement can be replaced by the name of any fieldgroup defined in the file.

The **id** clause in the statement specifies the fieldgroup ID to be matched for the fieldgroup being used as the context. It may be a positive integer count, a loop control variable, a special variable, or a %variable. If it is a special variable or a %variable, its value must be a number greater than or equal to 1; any fractional part is dropped.

The occurrence of the specified fieldgroup, within the current context, whose fieldgroup ID is equal to the specified **id**, is the context for any members of that fieldgroup. If an

occurrence of the fieldgroup does not exist with the specified *id* within the current context, the entire block is skipped.

For example, assuming the following field definitions:

```
DEFINE FIELD FIELDA (FG GRP)
DEFINE FIELD FIELDB (FG GRP)
```

the following statements put on the output stream the values of the fields in the occurrence of fieldgroup GRP which has the ID '4':

```
FOR FIELDGROUP GRP = 4
  PUT FIELDA
  PUT FIELDB
  OUTPUT
END FOR
```

Note that if both the fieldgroup ID and fieldgroup occurrence number are available, it is faster to access a fieldgroup occurrence using "FOR FIELDGROUP fgrpName(occ)" than to access it using "FOR FIELDGROUP fgrpName = id".

### 3.12.3 FOR EACH FIELDGROUP fgrpName

The following block loops over all occurrences of a fieldgroup in the current context. Within the block it creates a context within which references to fieldgroup members and #FIELDGROUPID use, in order, each of the occurrences of the fieldgroup within the current context:

```
FOR EACH FIELDGROUP fgrpName
  . . . statements which use consecutive occurrences of
  . . . fgrpName as the fieldgroup context
END FOR
```

The *fgrpName* clause in the statement can be replaced by the name of any fieldgroup defined in the file.

Within the current context, each occurrence of the specified fieldgroup is the context for any members of that fieldgroup.

For example, assuming the following field definitions:

```
DEFINE FIELD FIELDA (FG GRP)
DEFINE FIELD FIELDB (FG GRP)
```

the following statements put on the output stream the values of the fields in all occurrences of fieldgroup GRP:

```
FOR EACH FIELDGROUP GRP
  PUT FIELDA
  PUT FIELDB
  OUTPUT
END FOR
```

Note: if GRP is not nested, and each of FIELDA and FIELDB is either EXACTLY-ONE or is AT-MOST-ONE, and is not FG \*, the following statements are equivalent to the above (except for the use of the loop control variable I):

```
FOR I FROM 1 TO FIELDA(#)
  PUT FIELDA(I)
  PUT FIELDB(I)
  OUTPUT
END FOR
```

There is no significant difference in processing time between the above two approaches.

Also note that the following FUEL statements:

```
FOR EACH FIELDGROUP GRP
  . . body
END FOR
```

are processed, for all intents and purposes (for example, with the same performance) the same as the following:

```
FOR I FROM 1 TO GRP(#)
  FOR FIELDGROUP GRP(I)
    . . body
  END FOR
END FOR
```

The latter approach may be better in your FUEL program if you need to access the occurrence number of the fieldgroup.

### **3.12.4 LEAVE FIELDGROUP**

This statement will terminate execution of the current FOR FIELDGROUP block, causing execution to resume at the statement after the end of its matching END FOR.

### 3.12.5 PAI

The PAI statement in FUEL supports FILEORG X'100' files, providing the same information that PAI in User Language can provide. There are no operands of the FUEL PAI statement, to control display of automatic fields or LOB fields, as there are in the User Language PAI statement.

Fieldgroups are displayed with a 'fieldgroup name = ID' line at the start and end of the fieldgroup; so, for example, repeating the example PAI output shown in [“Record structure in FILEORG X'100' files, context and other fieldgroup concepts”](#) on page 6:

```

OUTFLD = OUT01
OUTFLD = OUT02
\GRP = 20
  INGRP = MEM01
  INGRP2 = MEM02
/GRP = 20
OUTFLD = OUT03
\GRP = 5
  INGRP = MEM03
  INGRP2 =
  REPMEM = REPMEM01
/GRP = 5
OUTFLD = OUT07

```

## 3.13 New or changed #functions and special variables

The following sections describe new or changed #functions or special variables in version 4.6 of *Fast/Unload*.

### 3.13.1 #FIELDGROUPID

This special variable returns the fieldgroup ID of the current occurrence of the fieldgroup specified on the containing FOR FIELDGROUP block.

### 3.13.2 #SHADIGEST: SHA-1 digest ("hash") of string

This function returns the 20-byte (always) binary string that is the SHA-1 digest of the argument.

```
%hashval = #SHADIGEST(string)
```

**#SHADIGEST syntax**

## Syntax Terms

**%hashval** A %variable to receive the SHA-1 digest of the argument string.

**string** The input string to be hashed; this may be a long string (that is, it may exceed 255 bytes in length).

## Usage Notes

- SHA (Secure Hash Algorithm) is a set of cryptographic hashing functions; #SHADIGEST provides SHA-1, the most commonly used. A complete explanation of SHA hashing can easily be found on the internet.
- This FUEL #function operates the same as the Janus SOAP ULI SHAdigest method.

## Examples

The 20-byte SHA-1 hash of a string is typically expressed as a 40-digit hex value. In the following example, the output string from #SHADIGEST is converted to hex using the #C2X function:

```
%HSH = #SHADIGEST('this is a simple test')
%XHSH = #C2X(%HSH)
PUT %XHSH
OUTPUT
```

The result is:

```
BC38AA2D6769639946806616C14AF0C69477AABE
```

## 3.14 Other FUEL changes

### 3.14.1 Asterisk in PUT statement syntax

The syntax for the PUT statement has been enhanced to support the use of the asterisk (\*) in the following:

#### **MISSING \* or ERROR \***

The MISSING and ERROR clauses now allow, in addition to a value or action keyword (immediately following MISSING or ERROR), an asterisk (\*), which specifies that the default should be taken for the value or action and that an ERROR condition is exempted from the effect of the customization zap for the PUT ERROR clause [see the appropriate section] so that CANCEL is not performed for the conversion error. It also allows you to specify REPORT or NOREPORT after the asterisk.

**STRING(\*[,other format specification])**

Using an asterisk (\*) as the length specification for the STRING output format causes the value to be placed in the output using as many bytes as there are in the value; that is, the output length is variable.

The use of STRING with a length of zero (its default) also specifies variable length output, but it has an undesirable effect on the MISSING clause, and so many forms of the STRING format are deprecated.

You should use STRING(\*[,..]) for variable length output. STRING with length zero causes the constant specified in the MISSING clause to be ignored; for example:

```
PUT MIDDLE.NAME AS STRING MISSING '(none)'
```

If field MIDDLE.NAME is missing for a record, nothing will be put to the output. The desired result, placing the string (none) in the output for a missing field, is obtained if you use:

```
PUT MIDDLE.NAME AS STRING(*) MISSING '(none)'
```

The following formats are deprecated and cause a warning message to be issued; after the deprecated format, the preferred format is shown:

- STRING(0[,...])  
STRING(\*[,...]) /\* This is preferred
- STRING()  
STRING(\*) /\* This is preferred
- STRING([...])  
STRING(\*,[...]) /\* This is preferred
- STRING not immediately followed by a parenthesis, and followed by a MISSING clause with a constant, for example:

```
PUT AS STRING ERROR CANCEL MISSING '!'
PUT AS STRING(*) ERROR CANCEL MISSING '!' /* This is preferred
```

The syntax for the PUT statement is shown in the following two sections.

**3.14.1.1 Revised syntax for the PUT statement**

As described above, the asterisk (\*) can be used in the STRING, MISSING, and ERROR clauses. This section and the next include this as part of the syntax description. This revised syntax description is a replacement for the PUT description in the *Fast/Unload Reference Manual*.

The syntax diagram for the PUT statement is:

```
[TO destination] -  
  PUT info -  
    [AT loc] -  
    [AS format -  
      [MISSING missActOrVal [repOrNot]] -  
      [ERROR errActOrVal [repOrNot]]  
    ]
```

The current doc for 'TO destination', 'info', 'AT loc', and 'AS format' is adequate, except:

- Negative *loc* is allowed and causes "overlay" of previous bytes.
- We need to remove or alter some language about the missing value from the various formats; also we should insert square brackets to indicate optional format parameters. The default length for FLOAT is 4, and for DECIMAL and ZONED is 32.
- The following sentence should be added in the 'AS format' section:  
If there is no AS format ... clause, the default format is `STRING(*)`.

The items in the definition list would be better served as subsections of the PUT section, for example, 'AT loc', etc., with a final subsection titled 'PUT examples'.

The MISSING and ERROR sections will be combined, as shown in the next section.

### 3.14.1.2 MISSING and ERROR clauses

The MISSING and ERROR clauses are allowed only if *inf* is a field occurrence or a %variable. The clauses may occur in either order. The terms in the MISSING and ERROR clauses are:

#### **missActOrVal**

one of the following:

- a constant value, which is placed in the output record if *info* is MISSING;
- either of the keywords SKIP or CANCEL, as described below;
- an asterisk (\*) as a placeholder (causing *missActOrVal* to be the same as it would be without any MISSING clause); this is allowed starting with version 4.6.

If the MISSING clause is not specified, or if `MISSING *` is specified, the value put when *info* is MISSING depends on whether *info* has a value:

**If info has no value**

Then the default MISSING handling for a STRING format is to fill the output area with blanks; for a numeric format, the default MISSING handling is to output either -1 or, if the customization zap for MISSING default [see the appropriate section] is applied, 0.

**If info has a value**

A MISSING field occurrence or %variable has a value if, and only if, it is the first occurrence (see “[First occurrence](#)” on page 12) of an AT-MOST-ONE field which has a DEFAULT-VALUE, or is a %variable, which has been assigned from such a field occurrence. In either of these cases, the default *missActOrVal* is as follows:

- If the value is convertible (and does not exceed the format length, for a STRING format), the value is placed in the output area.
- Otherwise, an ERROR condition occurs (**in addition** to the MISSING condition).

(We will also update the doc for #ERROR, noting that it can be 3 if both the ERROR and MISSING conditions occur).

**errActOrVal**

one of the following:

- a constant value, which is placed in the output record if a conversion error occurs on the PUT;
- either of the keywords SKIP or CANCEL, as described below;
- an asterisk (\*) as a placeholder/override. This is allowed starting with version 4.6 and has the following effect:
  1. It overrides the effect of the customization zap for the PUT ERROR clause [see the appropriate section] so that CANCEL is not performed for a conversion error on this PUT statement.
  2. It does not affect the *repOrNot* keyword, but otherwise it handles the conversion error just as the PUT statement would handle it (except that it overrides the zap) if there were no MISSING or ERROR clauses (see the section below for MISSING and ERROR defaults). Note that this effect is already the default for a STRING format, and so the only reason to use it with a STRING format, other than reason (1) above, is to specify `ERROR * NOREPORT` (or equivalently `ERROR TRUNC NOREPORT`).
  3. It can simply serve as a placeholder, so that you can specify REPORT or NOREPORT in the ERROR clause, without an explicit constant, SKIP, or CANCEL.
- either of the keywords TRUNCATE or TRUNC (only for a STRING format); this is the same as using an asterisk (\*).

### repOrNot

For both the MISSING and ERROR clauses, you may add a trailing REPORT or NOREPORT keyword. NOREPORT indicates that the condition **is not** reported on the report data set; this is the default for the MISSING clause unless CANCEL or SKIP is specified as *missActOrVal*. REPORT indicates that the condition **is** reported on the report data set; this is the default (starting with version 4.6) for the ERROR clause, and is the default for the MISSING clause if CANCEL or SKIP is specified as *missActOrVal*.

The ERROR *repOrNot* default is distinctly different prior to version 4.6; see [“Pre-4.5 defaults for the MISSING and ERROR clauses” on page 27](#).

The SKIP and CANCEL keywords are handled as follows:

**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.

The SKIP keyword in the MISSING clause causes REPORT to be the default for MISSING.

**CANCEL** This means the entire *Fast/Unload* job is terminated. Use this value if the MISSING or ERROR condition indicates a severe logic error in your data file structure.

The CANCEL keyword in the MISSING clause causes REPORT to be the default for MISSING.

Starting with version 4.6, the defaults for *missActOrVal* and *errActOrVal* are as follows:

#### **missActOrVal default**

The default for *missActOrVal* is as described at the start of this section; see [“MISSING and ERROR clauses” on page 24](#).

#### **errActOrVal default**

The default for *errActOrVal* is as follows:

- If the customization zap for the PUT ERROR clause [see the appropriate section] is applied, the default is **CANCEL**.
- Otherwise, if the format is **STRING**, the default is the truncated string value.
- Otherwise (numeric format and no PUT ERROR customization zap):

- If *missActOrVal* is specified and is not \*, then the default for *errActOrVal* is the same as the specified *missActOrVal*.
- Otherwise, the default *errActOrVal* is -1, or, if the customization zap for MISSING default [see the appropriate section] is applied, 0.

As can be seen, there is some asymmetry between the ERROR defaults for STRING versus numeric formats if the customization zap for the PUT ERROR clause [see the appropriate section] has not been applied:

- For numeric formats, if there is a MISSING clause (with *missActOrVal* other than \*), the default for *errActOrVal* is whatever was specified for *missActOrVal*; if there is not a non-\* MISSING clause, the default *errActOrVal* either is -1 or, if the customization zap for MISSING default [see the appropriate section] is applied, it is 0.
- For STRING formats, the default for *errActOrVal* is the truncated string value, regardless of what may be specified for *missActOrVal*.

For *Fast/Unload* versions prior to 4.6, see “[Pre-4.5 defaults for the MISSING and ERROR clauses](#)” for a description of the defaults for the MISSING and ERROR clauses.

### 3.14.1.3 Pre-4.5 defaults for the MISSING and ERROR clauses

This section describes the defaults for the MISSING and ERROR clauses for *Fast/Unload* versions 4.4 and earlier (version 4.5 is the same as version 4.4, except that it uses the DEFAULT-VALUE, if any, for the missing first occurrence of an AT-MOST-ONE field).

**missActOrVal** The default *missActOrVal* for a STRING format is to fill the output area with blanks; for a numeric format, the default *missActOrVal* is to output either -1 or, if the customization zap for MISSING default [see the appropriate section] is applied, 0.

This is the same as described above for version 4.6, but the above description is simpler, since the DEFAULT-VALUE attribute is not supported prior to version 4.5.

**MISSING repOrNot** NOREPORT is the default unless CANCEL or SKIP is specified as *missActOrVal*, in which case REPORT is the default.

This is exactly as described above for version 4.6.

**ERROR clause** The default for the ERROR clause is as follows:

- If the customization zap for the PUT ERROR clause [see the appropriate section] is applied, the default is **ERROR CANCEL REPORT**.

- Otherwise, if the format is **STRING**, the default is the truncated string value, and **REPORT** is the default for *repOrNot*.
- Otherwise (numeric format and no **PUT ERROR** customization zap):
  - *repOrNot* for **ERROR** defaults to **REPORT**, if an **ERROR** clause is present; if there is no **ERROR** clause, *repOrNot* for **ERROR** defaults to **NOREPORT**, unless **REPORT**, **CANCEL**, or **SKIP** is specified on the **MISSING** clause, in which case it defaults to **REPORT**.
  - If *missActOrVal* is specified, then the default for *errActOrVal* is the same as the specified *missActOrVal*.
  - Otherwise, the default *errActOrVal* is -1, or, if the customization zap for **MISSING** default [see the appropriate section] is applied, 0.

#### **3.14.1.4 PUT examples**

SORELY NEEDED! Maybe rather than a section, just an example with each of the tortuous cases above.

### **3.15 Statistics improvements**

#### **3.15.1 FSTATS for FILEORG X'100' files**

Field and fieldgroup statistics are produced for FILEORG X'100' files as follows:

- Occurrence counts for fieldgroup members are shown per fieldgroup, as opposed to per record, for non-fieldgroup members.
- Occurrence counts for **EXACTLY-ONE** fields refer to physically stored occurrences.
- The length calculated for a fieldgroup is based on the actual length (including the fieldgroup header) of physical fieldgroup items, which never exceed 511. For example, if the total length of the fields physically stored in a fieldgroup is 700, the fieldgroup occurrence will be split into multiple fieldgroup items, and the individual lengths of these items is used in the fieldgroup length statistics.
- The occurrence count calculated for a fieldgroup is based on “logical” fieldgroup occurrences. For example, if the total length of the fields physically stored in a fieldgroup is 700, even though this is physically stored as multipoe fieldgroup items, this is treated as one fieldgroup occurrence.

This chapter lists any compatibility issues with prior versions of *Fast/Unload*, in two sections — one covering compatibility with version 4.4, and one covering compatibility with version 4.5. We also present a section listing any bugs which have been fixed in this version of *Fast/Unload* but had not, as of the date of this release, been fixed in version 4.5 (all 4.6 fixes have also been fixed in version 4.4).

In general, backward incompatibility means that an operation which was previously performed without any indication of error, now operates, given the same inputs and conditions, in a different manner. We may not list as backwards incompatibilities those cases in which the previous behaviour, although not indicating an error, was “clearly and obviously” incorrect, and which are introduced as normal bug fixes (whether or not they had been fixed with previous maintenance).

## **4.1 Backwards compatibility with Fast/Unload 4.4 and 4.5**

This section lists any differences in processing that result from execution with *Fast/Unload* version 4.6, as compared with the same inputs to *Fast/Unload* versions 4.4 and 4.5 at current maintenance levels. In some cases zaps have been delivered to change the behavior to be the same as the version 4.6 behavior; these cases are explicitly listed.

See “[Backwards compatibility with Fast/Unload 4.5](#)” on page 32 for compatibility issues which exist only between versions 4.6 and 4.5 of *Fast/Unload*.

### **4.1.1 Detect PUT syntax error immediately after FIXED/DECIMAL/ZONED**

Prior to version 4.6 of *Fast/Unload*, a single-character “garbage” character was allowed and ignored after the FIXED, DECIMAL, or ZONED formats in the PUT statement; for example:

```
PUT AMOUNT AS FIXED 2 MISSING -999
```

This invalid syntax is no longer allowed. Detecting this syntax error was part of the fix described in “[Allow MISSING or ERROR immediately after FIXED/DECIMAL/ZONED in PUT](#)” on page 34.

### 4.1.2 Disallow PUT constant>255 AS FIXED(1)

Previously, a FUEL statement was **not** treated as an error if it specified an integer constant whose value is greater than 255 with a format of FIXED(1). For example:

```
PUT 256 AS FIXED(1)
```

This should be reported as an error, because the value (256) does not fit in the length (1 byte) specified.

Any PUT statement which specifies an integer constant whose value is greater than 255, and with a format of FIXED(1), is now reported as an error.

This fix was also delivered as a maintenance zap to versions 4.4 and 4.5 of *Fast/Unload*.

### 4.1.3 Handle quotes in MISSING clause of UAI SORT

Previously, the contents of the MISSING clause in a UAI SORT statement were used exactly as is, without stripping quotes (and also with a couple of other problems); consequently:

- MISSING values which should be reported as syntax errors, namely, mismatched quotes and values which are longer than 255 bytes, are not reported in error. Take this FUEL program, for example:

```
UAI SORT CHAD MISSING 'X
```

This should be reported as an invalid statement due to the missing trailing quote, but it is not; the output of this program is actually (because of another glitch) the same as if MISSING 'X' were specified.

- The MISSING value stored in the sort key of UAI records contains an incorrect value, so that such records are not sorted in the proper sequence. Take this FUEL program for example:

```
UAI SORT CHAD MISSING 'X'
```

If a record has a missing occurrence of CHAD, and another has an occurrence with the single character 'X', these will **not** be sorted together. Rather, if there is a record with the single character '@' and another with the single character '=', the missing occurrence record will be sorted between them. As a little glitch, the final quote of the MISSING value is dropped if it is the last character in the FUEL statement.

The handing of UAI's MISSING clause has been fixed, as reflected by the revised start of the description of that clause, as shown in ["Revised description of UAI MISSING" on page 31](#).

#### 4.1.3.1 Revised description of UAI MISSING

As stated in “[Handle quotes in MISSING clause of UAI SORT](#)” on page 30, a fix has been made to the handling of the MISSING clause of the UAI statement. The rest of this section presents the new syntax of that clause, as a replacement for the start of its documentation.

- 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** is the string of characters between the MISSING keyword and the following keyword (usually AND) or the end of the line, if there is no additional keyword on the UAI statement. If there are no quotation marks (actually, apostrophe characters; the terms 'apostrophe' and 'quote' mean the same thing in this section) in this string of characters, then that string (which can be numeric) is the value to be used for a missing field or %variable (multiple consecutive blanks are collapsed to a single blank). Quotes are **necessary** only if the value is to contain a UAI statement keyword which normally terminates the value, or if leading, trailing, or multiple consecutive blanks are required in the value. If there are quotes in the string, then quote normalization is repetitively performed, as follows:

- Balanced** There must be an even number of apostrophes.
- Start quote** The first quote is discarded, and encloses a quoted region which begins after that quote and continues until a close quote.
- Combine doubled** Within a quoted region, if a quote is immediately followed by another, then the two quotes are replaced with a single one.
- Close quote** An "undoubled" quote terminates the quoted region, and is discarded.

All characters within a quoted region (after undoubling of internal quotes) are appended to the preceding portion of the value.

**mvalue** must (after quote normalization as above) be less than 256 characters in length.

**mvalue** must be convertible to the sort key data type. For example, ... (rest of MISSING clause description as before)

This fix was also delivered as a maintenance zap to versions 4.4 and 4.5 of *Fast/Unload*.

#### 4.1.4 #IF errors which are now detected

As mentioned in “[Fixes to #IF](#)” on page 35, several problems in the #IF statement have been fixed. Some of these problems allowed erroneous #IF statements to compile without any indication of error; in version 4.6 these errors are now detected:

- `#IF DEFINED` (as a complete statement) was allowed, and it always operated as if checking for a field which is not found. This statement is no longer allowed.
- `#ELSE` and `#ELSEIF` were allowed within an `#IF` block after an `#ELSE`. These are no longer allowed.

## 4.2 Backwards compatibility with Fast/Unload 4.5

This section lists any differences in processing that result from execution with *Fast/Unload* version 4.6, as compared with the same inputs to *Fast/Unload* version 4.5 at current maintenance levels, although in some cases a version 4.5 zap has been delivered to change the behavior to be the same as the version 4.6 behavior; these cases are explicitly listed.

See also “[Backwards compatibility with Fast/Unload 4.4 and 4.5](#)” on page 29 for additional compatibility issues between versions 4.6 and 4.5 of *Fast/Unload*.

### 4.2.1 Differences in PUT ONE DV fields with no MISSING literal

As described in “[Handling of missing AT-MOST-ONE fields](#)” on page 14, in version 4.6 the PUT statement of the first occurrence of a missing AT-MOST-ONE DEFAULT-VALUE field (or a %variable which has been assigned from the missing first occurrence of such a field), if the PUT statement does not have the MISSING keyword followed by a literal, will

1. result in the MISSING condition
2. output the DEFAULT-VALUE, if convertible
3. **Also** result in the ERROR condition, if non-convertible

In version 4.5, a non-convertible DEFAULT-VALUE **does not** result in an ERROR condition.

## 4.3 Fixes in Fast/Unload 4.6 but not in 4.5

This section lists fixes to functionality existing in *Fast/Unload* version 4.5 but which, due to the absence of customer problems, have not, as of the date of the release, been fixed in that version.

None of the problems listed in this section pertain to *Fast/Unload* version 4.4. The fixes in *Fast/Unload* 4.6 that are not available in version 4.4 are listed in “[Fixes in Fast/Unload 4.6 but not in 4.4](#)” on page 34.

### **4.3.1 UAI of EXACTLY-ONE fields**

In *Fast/Unload* version 4.5, the UAI output for a record with any EXACTLY-ONE field which is not physically present will contain the default value (either the DEFAULT-VALUE or, if none, the null string). In 4.6, only physically present EXACTLY-ONE occurrences are output for UAI.

For the most part the only consequences of this are that UAI can run more slowly in 4.5 than in 4.6, and that a subsequent LAI using the UAI output can increase table B/X usage if EXACTLY-ONE fields with DEFAULT-VALUE are STORE-DEFAULT LIT or EXACTLY-ONE fields without DEFAULT-VALUE are STORE-NULL LIT. However, it also will cause LAI to fail (with a field constraint violation) if the default value violates a constraint on the field.

### **4.3.2 PUT of ONE DV xx field AS non-STRING with no MISSING clause**

Previously, a PUT statement of an AT-MOST-ONE field is not allowed if the PUT statement uses a non-STRING format, does not contain a MISSING clause, and the DEFAULT-VALUE of the field is not convertible to a number. This is now allowed; if a field occurrence is missing in the above situation, the default non-string MISSING value (that is, -1, or 0 with the customization zap for MISSING default — see the appropriate section) is placed in the output.

### **4.3.3 [NO]UNLOAD of outer EXACTLY-ONE field**

In version 4.5 of *Fast/Unload*, the UNLOAD and NOUNLOAD statements are not performed correctly for outer EXACTLY-ONE fields (in both versions 4.5 and 4.6, they are not allowed for EXACTLY-ONE fieldgroup members). For version 4.5, NO/UNLOAD of an outer EXACTLY-ONE field produces a compilation error message (when maintenance ZAP4538 is applied).

Version 4.6 correctly performs NO/UNLOAD of outer EXACTLY-ONE fields.

### **4.3.4 References to AT-MOST-ONE fieldgroup members**

In version 4.5, references to AT-MOST-ONE fieldgroup members is not supported. That is, version 4.5 does not provide FOR FIELDGROUP blocks for field references, and references to non-nested, non-FG-\* AT-MOST-ONE fieldgroup members without a FOR FIELDGROUP block (which are supported in 4.6 as described in [“References to fieldgroup members not in fieldgroup context” on page 11](#)), result (with ZAP4540) in a compilation error in 4.5.

### **4.3.5 DEFAULT-VALUE for AT-MOST-ONE field in UAI SORT**

In version 4.5, when an AT-MOST-ONE field is used for the SORT key in a UAI SORT unload, the DEFAULT-VALUE of the field is **not** used as the default for the MISSING clause of the sort specification. In version 4.6, as described in [“Handling of missing AT-MOST-ONE fields” on page 14](#), the DEFAULT-VALUE **is** used.

### **4.3.6 Bug in PAI if fieldgroup present and outer field added**

In version 4.5 of *Fast/Unload*, the following FUEL fragment:

```
ADD REP = #RECIN
PUT '*'
OUTPUT
PAI
```

is not handled correctly; for example, it may produce a "FUNL0056 Unknown field" error message and cancel *Fast/Unload*.

### **4.3.7 Preventing illegal access to UTF8 fields**

*Fast/Unload* is not equipped to "fetch", ADD, or CHANGE UTF8 fields (doing so requires, at the least, conversion between EBCDIC and Unicode). Version 4.6 of *Fast/Unload* ensures that these unsupported accesses are not allowed, by prohibiting the FUEL statements which would give rise to them.

Version 4.5 does not prohibit those statements, so it should not be used with files containing UTF8 fields.

## **4.4 Fixes in Fast/Unload 4.6 but not in 4.4**

This section lists fixes, delivered in *Fast/Unload* 4.6, to functionality existing in *Fast/Unload* version 4.4 but which, due to the absence of customer problems, have not, as of the date of the release, been fixed in that version. Some of the problems in this section have also been fixed in version 4.5; such cases are indicated.

The *Fast/Unload* 4.6 fixes to problems introduced in version 4.5 that are not available in version 4.5 are listed in [“Fixes in Fast/Unload 4.6 but not in 4.5” on page 32](#).

### **4.4.1 Allow MISSING or ERROR immediately after FIXED/DECIMAL/ZONED in PUT**

The FIXED, DECIMAL, and ZONED formats of the PUT statement all have default lengths, which allow omission of the parenthesized format; for example, the following is allowed and puts a 4-byte binary integer to the output:

```
PUT AMOUNT AS FIXED
```

However, prior to version 4.6, you were not able to specify a MISSING or ERROR clause immediately after the format type, for example:

```
PUT AMOUNT AS FIXED MISSING -999
```

This is now allowed. Note that when this bug was fixed, it also fixed another problem, as described in [“Detect PUT syntax error immediately after FIXED/DECIMAL/ZONED” on page 29](#).

#### **4.4.2 Long string access to FLOAT LEN 4 field**

When a FLOAT LEN 4 field is passed as an argument to certain #functions which accept string values longer than 255 bytes, an error may occur (specifically noted in version 4.4 has been an operation exception at FUNS + X'3870'). The following FUEL statement exhibits such a usage:

```
%X = #CONCAT('xyz', FLOAT4)
```

#### **4.4.3 Separate extension record stats for each file in group**

In version 4.4 of *Fast/Unload*, the table B statistics for the files within a group have two incorrect lines:

```
Maximum extension chain length processed  
Non-adjacent extension records processed
```

In version 4.4, they are cumulative for all files in the group. In version 4.6 (and version 4.5 with ZAP4545 applied), they are separated for each of the files within the group.

#### **4.4.4 Fixes to #IF**

There are a number of fixes to the #IF statement, which are present in version 4.6 of *Fast/Unload* but not in version 4.4 nor version 4.5:

1. The preprocessor statements (`#...`) did not allow the "trailing comments" (`/* ...`).
2. Sometimes the error messages were misleading.

3. The parsing of #IF and #ELSEIF did not allow fieldnames which have the word 'DEFINED' or 'UNDEFINED' in them. There is a workaround for this, using quotes around part of the field name.
4. An erroneous #IF DEFINED (as a complete statement) was allowed, and it always operated as if checking for a field which is not found.
5. #ELSE and #ELSEIF were allowed within an #IF block after an #ELSE.

Items (4) and (5) above are both noted as incompatibilities; see [“#IF errors which are now detected” on page 31](#).

## 4.5 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 4.6 of *Fast/Unload*.

If a UAI operation is performed on a `FILEORG X'100'` file, the LAI requires at least version 7.7 of the *Sirius Mods*, although in general, if the file contains `FILEORG X'100'` features, especially fieldgroups, the LAI must be done with *Model 204 V7R2* or greater, and so also requires a version of *Sirius Mods* which supports *Model 204 V7R2* or greater.