Rocket Model 204 Fast/Reload

Reference Manual

Notices

Edition

Publication date: November 2014
Book number: FRL-0705-RM-01
Product version:

Copyright

Trademarks

Examples

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

License agreement

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

Corporate Information

Rocket Software, Inc. develops enterprise infrastructure products in four key areas: storage, networks, and compliance; database servers and tools; business information and analytics; and application development, integration, and modernization.

Website: www.rocketsoftware.com

Rocket Global Headquarters
77 4th Avenue, Suite 100
Waltham, MA 02451-1468
USA

Contacting Technical Support

If you have current support and maintenance agreements with Rocket Software and CCA, contact Rocket Software Technical support by email or by telephone:

Email: m204support@rocketsoftware.com

Telephone :

North America   +1.800.755.4222

United Kingdom/Europe   +44 (0) 20 8867 6153

Alternatively, you can access the Rocket Customer Portal and report a problem, download an update, or read answers to FAQs. You will be prompted to log in with the credentials supplied as part of your product maintenance agreement.

To log in to the Rocket Customer Portal, go to:

www.rocketsoftware.com/support

Contents

Contents

Figures

Summary of Changes

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

Fast/Reload Version 6.9

The following changes correspond to changes in *Fast/Reload* since version 6.8:

- Although this fix has been provided via maintenance to all supported releases, it is worth noting that the handling of float values in *Fast/Reload* has been improved. The most significant aspect of this is that using UAI/LAI to reorganize a file with, in particular, FLOAT LEN 4 fields, and changing them to FLOAT LEN 8, now produces values which are correct according to standard *Model 204* float handling.

  The explanation of float field processing has been elaborated in "Float Handling" on page 29.

Fast/Reload Version 6.7

The following changes correspond to changes in *Fast/Reload* since version 6.5:

- Support is added for the loading of files processed under *Model 204* version 6.1, including files that contain *Model 204* Large Object fields (Lobs). This support requires *Model 204* V6R1 and for Table E space to be defined in the target file; Lobs can be added by either of the following:

  - The *Fast/Reload* D statement will process PAI output from a file that has Lob fields defined.
  - With the LAI statement; LAI support for Lobs requires UAI output of Lobs, first delivered in version 4.3 of *Fast/Unload*.

- A new keyword, ERRCONT, is added to the OPTIONS statement (see "The OPTIONS statement" on page 6). ERRCONT lets you complete a reload despite errors that would normally cause a MSIR.0316 user restart.

- Buffer pointer validation is added, along with a new option, NOVALIDATE, to turn this validation off ("NOVAlidate" on page 13).

Fast/Reload Version 6.5

The following changes correspond to changes in *Fast/Reload* since version 5.4:

- Support for reloading of procedures and procedure aliases (see "Procedures" on page 25).

Fast/Reload Version 5.4

The following changes correspond to changes in *Fast/Reload* since version 5.0:

- New OPTION BAVAIL statement.

Fast/Reload Version 5.0

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

- Support for *Sir2000 Field Migration Facility*.

CHAPTER 1   *Introduction*

Fast/Reload is a plug compatible replacement for the *Model 204* FLOD and FILELOAD utilities. It consists of replacement and new object decks that are linked into your BATCH204 or ONLINE load module. After building these new load modules you will derive immediate benefits from *Fast/Reload* because existing FLOD and FILELOAD programs will then be compiled and executed by *Fast/Reload*.

In addition to these immediate benefits, *Fast/Reload* provides some extra functional enhancements to the FLOD and FILELOAD utilities. Primary among these is the ability to efficiently load data unloaded by *Fast/Unload* using a single command (LAI). This greatly simplifies and speeds up the reorganization process. The combination of *Fast/Unload* with UAI and *Fast/Reload* with LAI is called *Fast/Reorg*.

Fast/Reload has its own FILELOAD language compiler which converts FILELOAD language programs to machine language. In reloads that involve simply appending data to the end of an empty or existing file, *Fast/Reload* will bypass the *Model 204* disk buffer monitor and instead perform full track I/O to write the table B pages being loaded.

Fast/Reload uses multi-buffered BSAM input for the input sequential data file. This allows overlap of BSAM input and other processing. In addition, this allows *Fast/Reload* to take advantage of chained scheduling under MVS. Both of these benefits can significantly reduce the real time required to perform a FLOD or FILELOAD operation.

Fast/Reload will run under both MVS and CMS with any release of *Model 204* at or after 2.2.

CHAPTER 2 *Invoking Fast/Reload*

Fast/Reload is invoked exactly the same way that the FLOD or FILELOAD utility is invoked with a standard *Model 204* load module.

<u>CHAPTER 3</u>   *Fast/Reload Statements*

All *Fast/Reload* programs must begin with the FILELOAD or FLOD command and end with an END statement. All statements between the FILELOAD or FLOD and the END are converted into machine code by the *Fast/Reload* compiler or are used to set environmental parameters by the *Fast/Reload* compiler. An example of a very simple *Fast/Reload* program is:

```
FILELOAD -1,-1,0,1000000,10000,10000,,68
G
 DUMMY=1,0,X'8000'
 FIELD1=1,10
END
```

Fast/Reload supports all the statements supported by FLOD and FILELOAD except the following:

- The L statement.
- The ENDL statement.
- The F statement.
- The ENDF statement.
- The UP statement.
- The DOWN statement.
- The DELETE statement.

In addition, the X'4000' mode bit used to delete a field occurrence or occurrences is not currently supported. Finally, FLOD exits are not currently supported.

If you have FILELOAD or FLOD programs that contain statements not supported by *Fast/Reload*, you can still run the programs with the same load module you use for *Fast/Reload* programs. The FLODX and FILELOADX commands invoke the original *Model 204* FLOD and FILELOAD programs instead of *Fast/Reload*.

Fast/Reload attempts to copy the FLOD/FILELOAD program to the temporary procedure indicated by the FRELPREV system parameter (which defaults to -1). If *Fast/Reload* determines that the FLOD or FILELOAD program cannot be processed by *Fast/Reload* but can be processed by standard FLOD or FILELOAD, the FRELPREV temporary procedure is opened for input, and the program is passed to standard FLOD or FILELOAD to be read again.

For more information on both supported and unsupported statements, see http://m204wiki.rocketsoftware.com/index.php/Category:File_management. In addition to the standard statement set, *Fast/Reload* supports the extra statements described in the following subsections: "The OPTIONS statement" on page 6 and "The LAI statement" on page 14.

Note that each statement must begin in column one, since a blank in column one of a FLOD statement indicates the "read-and-load-a-field" statement.

3.1 The OPTIONS statement

This statement can appear anywhere inside a *Fast/Reload* program and is used to set environmental parameters that affect the way the load will be performed. The OPTIONS statement can be abbreviated to its first three or more characters. For example, the following are all valid OPTIONS statements:

```
OPTIONS TAPEI FOOFOO
OPTION TAPEI FOOFOO
OPTI TAPEI FOOFOO
OPT TAPEI FOOFOO
```

Multiple parameters can be set via the OPTIONS statement. They are described in the following subsections. The minimum required abbreviation for each is indicated in uppercase.

3.1.1 ANYorder

This option only has an effect when data is being loaded into a hash or sort key file. When data is being loaded into one of these files, performance will generally be good if the data is being loaded in hash or sort key order. If the data being loaded is not in this order, load performance could be disastrously affected. *Fast/Reload* normally does not continue in such a case. The ANYORDER option overrides this behavior and lets the file load continue.

In general, it is better to determine the reason records are out of order and to correct the problem, than to use the ANYORDER option to ignore it.

If you are using LAI, the ordering can be done by the HASH or SORT option of the UAI statement in *Fast/Unload* (see the **Fast/Unload Reference Manual**).

Otherwise, to obtain the full track I/O benefits of *Fast/Reload*, you should ensure that your input records are in the correct order. You must sort the TAPEI dataset in order by the SORT key, or by the **hash code** of the HASH key. This can be achieved using the M204HASH utility, as described in http://m204wiki.rocketsoftware.com/index.php/Hash_key_files.

3.1.2   BAVail [page_type] size

Size is -1 (the default), or a non-negative number.

This option has the following two beneficial effects:

1. If **size** is non-negative (and the file is not SORTED), the use of the BRESERVE=**bres** parameter is changed during the file load. A record will start within **remain <= size+bres** bytes at the end of the page if, and only if, the record fits in its entirety on that page; that is, the total record size is less than or equal to **remain**. This ensures that no base record is split with BRESERVE bytes or fewer in the base record, while using more available space on a table B page than the default algorithm.

2. If **size** is greater than zero, *Fast/Reload* ensures, during the file load, that at least **size** bytes are unused on a table B page. This provides for some record growth by subsequent updates, for example, during an ONLINE run, without shifting fields to an extension record.

BAVAIL is ignored if *Fast/Reload* is not operating using full track I/O. A **base** record is the first part of a table B record, as opposed to the subsequent parts of a table B record, which are called **extension** records.

Page_type can be any of the following:

ALL   This applies to all table B pages. If **size** is non-negative, then *Fast/Reload* guarantees that every table B page contains at least **size** unused bytes.

BASE   This applies to any page which contains one or more base records. If **size** is non-negative, then *Fast/Reload* guarantees that any page with a base record contains at least **size** unused bytes.

EXT   This applies to any page which contains an extension record. If **size** is non-negative, then *Fast/Reload* guarantees that any page with an extension record contains at least **size** unused bytes.

If **page_type** is omitted, **size** applies to all table B pages.

Multiple OPTIONS BAVAIL specifications will override the value(s) previously specified; for example:

```
OPTIONS BAVAIL 100 BAVAIL BASE 150
```

will set aside 150 bytes on each page with a table B base record, and 100 bytes on all other table B pages, but

```
OPTIONS BAVAIL BASE 150 BAVAIL 100
```

will set aside 100 bytes on all table B pages.

The default BAVAIL for each page type is -1. If the value of all of them is -1, *Fast/Reload* does not specifically reserve any table B space, but uses the normal *Model 204* space algorithm determined by BRESERVE, ensuring that a record does not start within the last BRESERVE bytes of a page.

If the file being loaded is not a SORTED file (FILEORG X'01'), a non-negative setting for any page type will modify the effect of the BRESERVE parameter for all table B pages. Let *BA* be the maximum of all BAVAIL values; if it is non-negative, then if an entire table B record is less than BRESERVE bytes long it can be stored within the last *BA* + BRESERVE bytes on the page, but otherwise a table B record is not started in that area.

Therefore, you may want to set a temporary value for BRESERVE during the file load which differs from the value used during online updating. For example:

```
OPEN FILE mumble
RESET BRESERVE 300
FILELOAD -1,-1,0,1000000,1000000
OPTIONS BAVAIL 400
LAI
END
RESET BRESERVE 800
```

Note: The correct values for BRESERVE and BAVAIL are file-specific, and depend on characteristics such as record sizes, the cost to your applications of extension records, the importance of conserving table B space, and updating characteristics. Like any tuning exercise, this requires analysis and experimentation. The numbers in the above example could be wrong for your file, in fact, you may do better with an online BRESERVE that is smaller than the file load BRESERVE, and BAVAIL may be smaller than both, larger than both, or in between.

In the following examples:

- The file is a simple ENTRY ORDER file (FILEORG X'00').

- There are no preallocated fields.

- Each page contains 6140 usable bytes. (A page has 6144 bytes prior to the trailer, but there are 2 bytes used to determine the number of records on a page, and 2 used for the amount of unused space.)

- **DataIn** means the total length of data in a record, **basln** means the space used for a base record, and **extln** means the space used for an extension record.

- Each base record or extension record requires a 2-byte page locator, and contains a 3-byte area which is zero or has the record number of a subsequent extension record. So, for example, if a record is completely stored in a base record, basln = dataln + 5.

- Each example illustrates a case starting with a base record on an empty page.

- The resulting space allocation shows each page as the letter "P" and a page number, followed by extension and/or base records and free space enclosed in square brackets. For example

```
Rec0 dataln=100
Rec1 dataln=300
P0 [Rec0 basln=105] [Rec1 basln=305]
   [free=5730]
```

shows two records stored on one page.

- Extension records are shown with the base record ID followed by a period and extension number, for example:

```
Rec0 dataln=12570
P0 [Rec0 basln=6140]
P1 [Rec0.1 extln=6140]
P2 [Rec0.2 extln=305] [free=5835]
```

shows one record stored on three pages.

- Results are shown for different values for BAVAIL. With the default value of BAVAIL=-1, *Fast/Reload* uses the standard *Model 204* algorithm for allocating table B space during a file load. This standard algorithm is always used when *Fast/Reload* is not using full track I/O.

The following example shows that BAVAIL=0 can be used for maximum packing on a page, and BAVAIL>0 can be used to ensure free space on a page; each of them ensure a base record is not split in the first BRESERVE bytes:

```
Rec0 dataln = 6000
Rec1 dataln = 130
BRESERVE = 300

using BAVAIL = -1:
 P0 [Rec0 basln=6005] [free=135]
 P1 [Rec1 basln=135] [free=6005]

using BAVAIL = 0:
 P0 [Rec0 basln=6005] [Rec1 basln=135]

using BAVAIL = 135:
 P0 [Rec0 basln=6005] [free=135]
 P1 [Rec1 basln=135] [free=6005]
```

The following example shows that without BAVAIL, you cannot guarantee free space on a page:

```
RecØ dataln=13Ø
Rec1 dataln=6ØØØ (6Ø 1ØØ-byte fields)
BRESERVE = 3ØØ

using BAVAIL = -1:
 PØ [RecØ basln=135] [Rec1 basln=6ØØ5]

using BAVAIL = Ø:
 PØ [RecØ basln=135] [Rec1 basln=6ØØ5]

using BAVAIL = 135:
 PØ [RecØ basln=135] [Rec1 basln=58Ø5] [free=2ØØ]
 P1 [Rec1.1 extln=2Ø5] [free=5935]
```

The following example shows tighter packing and fewer extension records:

```
RecØ dataln=6ØØØ (6Ø 1ØØ-byte fields)
Rec1 dataln=1ØØ
Rec2 dataln=61ØØ (61 1ØØ-byte fields)
Rec3 dataln=1ØØ
Rec4 dataln=6ØØØ (6Ø 1ØØ-byte fields)
BRESERVE = 2ØØ

using BAVAIL = -1:
 PØ [RecØ basln=6ØØ5] [free=135]
 P1 [Rec1 basln=1Ø5] [Rec2 basln=6ØØ5] [free=3Ø]
 P2 [Rec2.1 extln=1Ø5] [Rec3 basln=1Ø5]
    [Rec4 basln=59Ø5] [free=25]
 P3 [Rec4.1 extln=1Ø5] [free=6Ø35]

using BAVAIL = Ø:
 PØ [RecØ basln=6ØØ5] [Rec1 basln=1Ø5] [free=3Ø]
 P1 [Rec2 basln=61Ø5] [free=35]
 P2 [Rec3 extln=1Ø5] [Rec4 basln=6ØØ5] [free=3Ø]
```

The following example shows use of BAVAIL which increases the extension records at file load time, but by leaving space on **all** pages, records can grow without producing extension records which are stored on pages "far away" from the base page:

```
Rec0 .. Rec5 dataln=4000 (40 100-byte fields)
BRESERVE = 300

using BAVAIL = -1:
 P0 [Rec0 basln=4005] [Rec1 basln=2105]
    [free=30]
 P1 [Rec1.1 extln=1905] [Rec2 basln=4005]
    [free=230]
 P2 [Rec3 basln=4005] [Rec4 basln=2105]
    [free=30]
 P3 [Rec4.1 extln=1905] [Rec5 basln=4005]
    [free=230]

using BAVAIL = 300:
 P0 [Rec0 basln=4005] [Rec1 basln=1805]
    [free=330]
 P1 [Rec1.1 extln=2205] [Rec2 basln=3605]
    [free=330]
 P2 [Rec2.1 extln=405] [Rec3 basln=4005]
    [Rec4 basln=1405] [free=325]
 P3 [Rec4.1 extln=2605] [Rec5 basln=3205]
    [free=330]
 P4 [Rec5.1 extln=805] [free=5335]
```

The smallest value of **size** for any BAVAIL option is -1; the largest value is 6130 minus the length of all preallocated fields, minus 3 if FILEORG is SORTED, minus 1 if FILEORG is HASHED, minus 4 if FILEORG is UNORDERED.

3.1.3   DKOnly

In many cases, *Fast/Reload* bypasses the *Model 204* disk buffer monitor for table B pages and instead uses its own full track buffers for output. This can, in certain instances, result in data being loaded into different table B pages than standard FLOD or FILELOAD would use. An example of this is non-empty reuse record number files. You can force *Fast/Reload* to use the *Model 204* disk buffer monitor by specifying the DKONLY option. This option can result in a significant increase in the real and CPU time required to perform a file load.

3.1.4   ERRCont

Added in *Sirius Mods* version 6.7, this parameter lets you complete a reload despite errors that would normally cause a MSIR.0316 user restart. This can be useful, for example, in a case where you know you have some input data that will cause an error, but you want to use a reload for reformatting purposes despite the problematic fields.

If you specify ERRCONT to "continue on error" a *Fast/Reload* program, you probably need to increase the *Model 204* ERMX parameter setting to prevent a session termination because of excessive errors.

Note: Using ERRCONT may place your data at risk, so use this parameter only in cases where you know your data and have taken due precautions and backups beforehand.

The following FILELOAD code uses the ERRCONT parameter:

```
FILELOAD -1,-1,0,1000000,10000,10000,10000,50
OPTIONS ERRCONT
LAI
END
```

3.1.5   FTOnly

In many cases, *Fast/Reload* bypasses the *Model 204* disk buffer monitor for table B pages and instead uses its own full track buffers for output. In certain cases, though, *Fast/Reload* will automatically drop back to using the *Model 204* disk buffer monitor. When it does this, a file load might take considerably longer than expected. If you do not wish *Fast/Reload* to perform this automatic dropback, simply specify the FTONLY option. FTONLY will result in the file load being terminated when an automatic dropback would ordinarily be done.

3.1.6   NObuff n_out_buffs

NOBUFF must be followed by a number indicating the number of full track output buffers *Fast/Reload* is to use. When *Fast/Reload* is bypassing the *Model 204* disk buffer monitor and performing full track I/O to table B, it uses a certain number of full track buffers. These buffers must each be large enough to hold the data that will fit on a single track of the disk devices containing the database file. This means that the buffers will each be 43,288 bytes long if you are using 3380's, 49,472 bytes long if you are using 3390's and 49,472 bytes long if you are using both. Ordinarily *Fast/Reload* uses 3 full track output buffers. This is probably sufficient for most purposes. On rare occasions, it might be possible to get some speed improvements by using more than 3 output buffers. In these cases simply specify the NOBUFF option followed by a number greater than 3. NOBUFF must always be greater than or equal to 3.

3.1.7   NXBuff n_tableX_buffs

NXBUFF must be followed by a number indicating the number of full track table X buffers *Fast/Reload* is to use. NXBUFF must always be greater than or equal to 3.

3.1.8   NOVAlidate

NOVAlidate indicates that no buffer pointer validation is to be performed. For example, in the following statement, buffer pointer validation would make sure that the input record was at least 107 bytes long:

```
FIELDA=99,9
```

Without buffer pointer validation, the above statement would simply load data off the end of the input buffer, which would result in FIELDA being loaded with garbage, or perhaps in certain cases, would result in an addressing exception because the area past the end of the buffer is not allocated. Buffer pointer validation is probably more important with index register based references to the input buffer, because these types of references are more likely to have bad values in the index register and so go outside the current input buffer:

```
FIELDC=4|1,20
```

With buffer pointer validation, if a buffer reference is determined to be invalid, an error message is issued indicating the input record number as well as the invalid buffer offset and length, as in the following example:

```
MSIR.0892: Input error: input record number: 1, start position/
           length: 99/9
```

This option is only available under *Sirius Mods* 6.7 and later. Before *Sirius Mods* 6.7, buffer pointer validation was never performed, which is compatible with *Model 204* versions before V5R1. Standard FILELOAD started performing buffer pointer validation in *Model 204* V5R1.

While buffer pointer validation is probably a good idea, it does have a slight performance cost. If there appears to be little chance of a buffer pointer error in a FILELOAD program, and performance is at an absolute premium, you can use the NOVALIDATE option to eliminate the cost of the buffer pointer validity checks.

Since LAI programs have no explicit buffer references, the NOVALIDATE option has no effect on LAI programs.

3.1.9   STAtc stats_intvl

STATC must be followed by a number indicating the number of database file records that *Fast/Reload* will create before updating *Model 204* statistics. To speed up processing, *Fast/Reload* does not ordinarily update *Model 204* statistics (since last, system, etc.) until the end of a file load. If for some reason you wish these statistics to be updated more frequently you can specify a STATC value. If you specify OPTION STATC 10, *Model 204* statistics will be updated for every 10th record created in the database file. OPTION STATC 0 and OPTION STATC 1 are equivalent.

3.1.10 TAPei ddname

TAPEI must be followed by string indicating the DDNAME to be used instead of TAPEI for the input sequential file containing the data to be loaded. For example if you wish to load data from DDNAME HOMER, simply specify OPTION TAPEI HOMER. This allows more than one input stream to be used to load data in a single BATCH204 or ONLINE run.

3.2 The LAI statement

The LAI statement allows data created with the *Fast/Unload* UAI (Unload All Information) statement to be loaded into a database file. LAI stands for Load All Information and has several optional parameters.

```
LAI [DElfield] [FAMsplit] [NEWfgid] [NOfdef]
    [NOIndex] [NOProcs]
```

<div align="center">

LAI statement syntax

</div>

where the parameters can be abbreviated with only the part shown in uppercase, and where:

DElfield   means that if a field that is not defined in the target file is found in the input dataset, that is, was unloaded by UAI, that field should simply be deleted. If this parameter is not specified, the presence of an undefined field would terminate the LAI with an error. This statement provides an efficient way of deleting fields during a reorg.

Note: This parameter has no effect if the NOFDEF parameter is not also specified. This is because if NOFDEF is not specified, any unloaded field will be automatically defined in the target file.

FAMsplit   means that if any two names (two fields, a field and an alias, or two aliases) were part of a single *Sir2000 Field Migration Facility* family in the unloaded file but are now either separate fields or part of separate *Sir2000 Field Migration Facility* families, the values should be loaded into each field or family. Effectively, this "splits" the original family into multiple components. If *FAMSPLIT* is not specified, an attempt to do such a split causes the FLOD or FILELOAD to be terminated.

The most likely use of this parameter is to convert a file that is being controlled by *Sir2000 Field Migration Facility* into one that is not, but where any fields or aliases that were in the original file could still be referred to with User Language programs. This might be useful for sending a non-*Sir2000 Field Migration Facility* version of a file to a site that does not have

Sir2000 Field Migration Facility. Since *Sir2000 Field Migration Facility* makes sure that related fields and aliases automatically stay in synch, it is generally not a good idea to update the non-*Sir2000 Field Migration Facility* version of the file.

See the chapter titled "File Reorganizations" in the **Sir2000 Field Migration Facility Reference Manual** for considerations about preserving SIRFIELD information using *Fast/Reorg*.

means that fieldgroup IDs from the TAPEI input file are **not** copied to created fieldgroups, but rather that fieldgroup IDs within each record are created starting with 1.

NEWfgid

NOfdef   means that field definitions unloaded by UAI are not to be used to automatically define a field if it does not exist in the new file. If this parameter is not specified, all fields that have not been explicitly defined in the new database file but were defined in the old database file (unloaded with UAI) will be defined with exactly the same attributes that they had in the old database file.

NOIndex   means that ordered index data unloaded by UAI is not to be loaded by LAI. When this option is used, visible ordered index data will be regenerated and sorted by LAI and invisible ordered index data will be lost. This parameter has no effect if the OINDEX parameter was not specified on the UAI command for the unload.

NOProcs   means that any procedures and procedure aliases unloaded by UAI are not to be loaded by LAI. Procedure and alias records in the input dataset will be skipped. For more information about loading procedures, see "Procedures" on page 25.

Fast/Reload programs using LAI are quite simple. For example, the following is a valid *Fast/Reload*: program:

```
FILELOAD -1,-1,0,1000000
LAI
END
```

The LAI statement has these limitations:

- The LAI statement must be the only statement that appears in a *Fast/Reload* program other than the OPTION statement.

- LAI cannot be used on files that were created by a release of *Model 204* before release 8.

3.2.1   LAI compared to FILELOAD/FLOD

When using LAI, some of the parameters on the FILELOAD or FLOD statement have a slightly different meaning than when doing a standard FLOD. These parameters are:

- Parameter 1

    This parameter has the same meaning as in standard FILELOAD/FLOD. That is, this is the maximum number of records that will be loaded into the database file.

- Parameter 2

    This parameter is ignored when doing an LAI.

- Parameter 3

    In standard FILELOAD/FLOD programs this indicates the number of physical input records in TAPEI to be skipped. When doing an LAI, this indicates the number of unloaded table B records to be skipped. This parameter, along with parameter 1 allows you to split a reload into pieces. For example, the following statements load the first million records unloaded via UAI, then the next million records and finally the remaining records.

```
FILELOAD 1000000,-1,0,100000,100000
LAI
END
FILELOAD 1000000,-1,1000000,100000,100000
LAI
END
FILELOAD -1,-1,2000000,100000,100000
LAI
END
```

All other parameters for the FILELOAD statement have the same meaning as when doing a non-LAI FILELOAD.

3.2.2   Using LAI with UAI

LAI is closely related to the UAI statement in *Fast/Unload*. Ordinarily, the interaction between these two is quite simple. However, in cases where sorting of the data unloaded by *Fast/Unload* is desired and/or if the data is being loaded into a hash or sort key file, the interaction can become more complex.

When loading data into a hash or sort key file with the LAI statement, the data must have been unloaded with a UAI statement that specified the HASH or SORT parameter and specified the new file's hash or sort key as the first key on the UAI statement. If this is not the case, the LAI will not load any data. If the UAI statement is coded correctly, the data unloaded by UAI will be in the order that the data will eventually be in the database

file being loaded. In these cases, *Fast/Reload* will never have to make more than one pass through table B and will actually be able to use full track I/O to write the table B pages if the data is being loaded into an empty hash or sort key file.

If the UAI statement is coded incorrectly, however, the unloaded data will be in an order different from that which the data will loaded into the destination database file. The impact of this on performance could be disastrous. Consequently, *Fast/Reload* will not ordinarily continue in the case when data is detected out of order. You can override this behavior with the ANYORDER option, but at least in the case of reorganizations, it is generally faster to unload the data again with the correct parameters rather than trying to load the data in incorrect order.

The types of errors on the UAI statement that can cause data to be in incorrect order are:

- Specifying SORT on the UAI and then attempting to load the data into a hash key file.

- Specifying HASH on the UAI and then attempting to load the data into a sort key file.

- Specifying a BSIZE on the UAI HASH statement different from the BSIZE in the hashed file being loaded.

Note if you are loading the data into a sorted file, there are several cases of the UAI SORT statement which render the sort key unusable. See the **Fast/Unload Reference Manual**.

Note that if you are loading the data into a non-hash key, non-sort key file, there is nothing wrong with loading data unloaded with a UAI command that had a SORT or HASH key specified. In fact, if you wish to maintain good locality for data with similar values for a particular field without incurring the overhead of maintaining a sort key file, you can specify that field as a sort key on a UAI and simply load the data into a non-sort key file.

3.3   Loading null data

Standard *Model 204* FLOD and FILELOAD do not allow null data (string data with a length of 0) to be loaded into the database file. That is, if after stripping blanks and zeros the resulting field has length zero, no data will be loaded into the database file. This is the case, even if you are trying to load null data that had been dumped using *Model 204*'s PAI command. Thus, it is extremely difficult if not impossible to reorganize a database file containing null data using FLOD or FILELOAD.

Fast/Reload allows you to do this in a couple of ways:

- Any null data unloaded via *Fast/Unload*'s UAI command will be loaded as null data by *Fast/Reload*'s LAI command.

- The X'0001' mode bit lets you load null data as a zero length string into the database file.

For example, the following FILELOAD program will load null data dumped via the PAI command back into the new database file as null data:

```
FILELOAD -1,-1,0,1000000,10000,10000,,68
I 3
#1
G
=4,5,*
I 1
#2
=3,6|1,=
I 1,,,1|1
=2
#3
I 2,1,2,-7,-1|1
D 5,0|1=8|1,0|2,X'0801'
=1
#4
 PROC.ID=1,0,X'8000'
END
```

The key part of the above example is the X'0001' mode bit on the D statement. Note also, that for PAI type reloads, it is generally a good idea to turn off blanks stripping. This is controlled by the X'0800' mode bit on the D statement.

CHAPTER 4 *Incompatibilities with standard FLOD/FILELOAD*

In general, every effort was made to maintain compatibility between *Fast/Reload* and standard *Model 204* FLOD and FILELOAD. In some specific instances, however, it was determined that compatibility was undesirable. These instance are described in this chapter.

4.1 Extension records in hash key files

When *Model 204* determines that an extension record must be started while loading data into a hash key file, it starts the extension record on a random page. *Fast/Reload*, when operating using full track I/O, on the other hand, tries to start the extension record on the next page. This is because the full track feature requires that pages be written in ascending order. This has a side benefit that when retrieving the extension record, head movement and caching characteristics are likely to be improved. If you wish *Fast/Reload* to use the standard *Model 204* extension record allocation, you should specify the DKONLY option in the FLOD program.

4.2 Adding data to non-empty unordered files

When loading data into unordered files that have had reusable space created because of record or field deletions, *Model 204* will ordinarily try to put data in the reusable areas before appending data after the last record in the file. *Fast/Reload*, on the other hand will attempt to add data after the last record in the file. This ensures that records will be allocated in ascending order allowing full track I/O to be used for the loaded table B pages. If you wish *Fast/Reload* to use the standard *Model 204* record allocation for unordered files, you should specify the DKONLY option in the FLOD program.

4.3 Ordered index layout when field codes change

Data in the ordered index is maintained in field code order. A field code is an internal, numeric (between 0 and 4095) representation of a field name. It is possible that a field code might change between a UAI and LAI. Since UAI simply loads the data in the order in which it was unloaded, this can produce a slightly different layout of the non-leaf nodes in the ordered index. The effect on FLOD/FILELOAD or online performance is likely to be unmeasurable. If you wish to minimize the chances of field code changes you should

- Explicitly define fields in the same order they were defined in the original database file.

- Use the same value of ATRPG in the new database file as used in the old.

4.4 Ordered index layout when mixing generated and unloaded index data

When the old database file had ordered index data unloaded via UAI OINDEX or UAI INVISIBLE, and new ordered index fields are defined in the new database file, the layout of non-leaf nodes will be slightly different than if the ordered index data had all been regenerated and resorted via FLOD. This is because the ordered index data is loaded in a new step in the FLOD process called the "FLOD" step, before the Z step. Thus the ordered index data is loaded in a different order than it would have been if all the data had been sorted and the loaded in the Z step. The effect on FLOD/FILELOAD or online performance is likely to be unmeasurable.

4.5 Table B space allocation

As described in "BAVail [page_type] size" on page 7, setting OPTION BAVAIL to a non-negative number causes a change in the way space is reserved on table B pages.

1. *Fast/Reload* guarantees (if full track I/O is in use) that BAVAIL bytes are unused on each table B page.

2. If full track I/O is in use, and the file is not SORTED (FILEORG X'01'), *Fast/Reload* will allow records to be started with less than BRESERVE + BAVAIL bytes available on the page, as long as the entire record fits on the page.

See "BAVail [page_type] size" on page 7 for a complete discussion and some examples.

CHAPTER 5   *Performance*

In general UAI/LAI is the fastest way of performing a database file reorganization. There are two key options that effect both FILELOAD/FLOD performance and online performance:

1.   Whether the reload/unload is to be split up into passes.

2.   Whether ordered index data is to be unloaded via UAI OINDEX and reloaded via LAI.

In general, for optimal online performance, you should try to unload and reload data in a single pass.

•   This minimizes the number of list pages that will be used for a particular fieldname-value pair.

•   This ensures that index data associated with a particular fieldname-value pair will tend to be localized on disk, minimizing head movement and maximizing cache hit ratios.

If it is not possible to unload and reload the data in a single pass because of limitations in intermediate sort work space, then there are a few things to keep in mind:

•   If you are not sorting the data on the UAI, it is more efficient to unload the data into multiple separate data sets and then load these data sets in multiple reload steps. By doing this, you avoid having to skip input records after the first reload step. In addition, if unloading ordered index data, you eliminate the need to skip intermediate records to get to the ordered index data in each reload step. Finally, you reduce the amount of ordered index data that must be scanned on each reload step, since each step must scan all ordered index data unloaded.

•   If you must sort the data on the UAI, you might actually get better performance by regenerating the ordered index data rather than unloading it via UAI OINDEX and then reloading it. This is especially true if you have a large amount of ordered index data and you are using many reload passes. You might consider doing a UAI INVISIBLE in this case, if you wish to preserve invisible ordered index values.

When unloading and reloading data in a single pass, it will generally be more efficient to use the UAI OINDEX parameter to unload the ordered index data than to regenerate the ordered index data because the ordered index data will already be in sorted order in the old database file. You must be aware however, that using the UAI OINDEX or UAI INVISIBLE feature can place significant real memory requirements on the reload because the reload must map old record numbers to new record numbers. The table

that is used for this mapping must reside in real storage for adequate performance since the entries will tend to be accessed in arbitrary order. Any paging for this mapping table could have disastrous effects on reload performance.

The amount of storage required for this mapping table depends on two factors. The first factor is the range of record numbers being loaded. The number of entries in the mapping table will equal the range of record numbers unloaded. This will generally be BSIZE times BRECPPG for hash key files and BHIGHPG times BRECPPG for other files. For example, if you are loading data unloaded from an entry order file with BHIGHPG of 15,000 and BRECPPG of 100, the mapping table will have 1,500,000 entries. By splitting the unload or reload into multiple passes you will reduce the number of entries in the mapping table. The second factor influencing the size of the record mapping table is whether the data was sorted on the UAI. If the data was sorted, the record number mapping will tend to be random and thus require 3 bytes per entry in the record mapping table. If the data was not sorted the data will have a structure that allows the record mapping table to require only 1 byte per entry.

Thus, the maximum possible storage requirement for the record mapping table would be 16 million (maximum record numbers allowed in a 204 file) times 3 bytes or 48 million bytes.

_____

CHAPTER 6   *Invisible fields*

One of the features of *Fast/Reorg* is to allow ordered invisible field values to be
preserved over a file reorganization in a generic manner. That is, once you define and
load all of your invisible fields as ORDERED INVISIBLE or ORDERED NUMERIC
INVISIBLE, you will never again have to worry about losing invisible fields over a
reorganization.

In addition to preserving ordered invisible fields over a file reorganization, *Fast/Reorg*
does some cleanup of the invisible ordered index data. Specifically, index entries
associated with non-existent records are deleted over the reorganization process. This
is accomplished in two steps.

- First, *Fast/Unload* does not unload any index data associated with records outside
  the range of record numbers unloaded. Thus, if an unload is done in two passes,
  the index data unloaded in the first pass will contain record numbers in the range of
  records unloaded in the first pass.

- Second, *Fast/Reload* would discard index data associated with deleted records.

For example, if you unload record numbers 0 through 1,422,633 in an unload, the
invisible index value associated with record number 2,422,666 would not be unloaded.
Note that if record 944,433 had been deleted but had ordered index invisible data in
table D, this data would be unloaded by UAI. When LAI attempts to convert record
number 944,433 to a new record number, it would notice that the record had not been
reloaded (because it had never been unloaded) and hence would discard the ordered
index data for that record. This cleanup of invisible ordered index data ensures that
table D is not wasted by invisible ordered index data for deleted records.

Note that if you are using invisible fields (ordered or non-ordered) with reuse record
number files, there is nothing that prevents a deleted record from being replaced by
another record, thus rendering the invisible index value for that record number
erroneously "valid" again. This is a generic problem with reuse record number files and
invisible fields and is not exacerbated by the use of *Fast/Reorg*. In fact, *Fast/Reorg*
gives you an option of eliminating the use of reuse record number files in an application
requiring the use of invisible fields, and instead relying on the speed and ease of use of
Fast/Reorg to recover record numbers via frequent reorganizations.

Invisible fields

_____
CHAPTER 7   *Procedures*

As of *Fast/Unload* version 4.2 and *Sirius Mods* 6.5, the UAI and LAI statements support the reorganization of procedures and procedure aliases. Procedures and aliases present in the TAPEI input data set for a *Fast/Reload* run are loaded as is by LAI by default. The UAI unload and LAI reload preserve their associations.

If procedure or alias names clash with names in the target file, the reload stops. If you want the unloaded procedures and aliases not to be reloaded, you specify the NOPROCS option of the LAI statement ().

The procedure reloading feature likely entails some environmental adjustments (especially more storage buffers), and it offers an automatic tuning of the target file's procedure dictionary settings.

7.1   Preparing the environment

When you are LAI-reloading a file that has procedures, take into consideration the following recommendations:

●   More CSECTs are required than in the typical BATCH204 module.

●   Instead of maintaining separate BATCH204 and ONLINE modules, combine their CSECTs into a single ONLINE module, and use that for *Fast/Reload*.

●   Ensure sufficient disk buffers to minimize the elapsed time for the reload:

     If your storage capacity is plentiful, 10,000 to 15,000 buffers should ensure maximum throughput.

     ▪   Set the *Model 204* parameter MINBUF to at least 10000.
     ▪   Set the LDKBMWND parameter to at least 30.

     Otherwise, if storage capacity is a concern, set the MAXBUF parameter to the number of pages in the unloaded procedure dictionary, plus some overhead:

```
MAXBUF = PDpgs + LDKBMWND + 5Ø +
         min(1Ø, nprocs) * avgProcpgs
```

Where:

- PDpgs (total number of pages in the unloaded procedure dictionary), nprocs (number of unloaded procedures), and avgProcpgs (average procedure length in pages) are available from the FSTATS statistics in the UAI report data set.

- The LDKBMWND parameter is set to at least 30.

7.2   Setting PDSIZE

Procedure loading includes an automatic PDSIZE-setting feature. This auto-sizing is weighted toward creating a new procedure dictionary whose total size is nearly the same as the old, but which typically is a product of a larger PDSIZE and fewer procedure dictionary chunks (blocks of contiguous pages) than the old.

The sizing feature is invoked only when you are loading into a file that has no procedures, the file's PDSIZE value is the default size (3), and the number of pages of UAI unloaded procedures and aliases is greater than the default PDSIZE.

When the preceding conditions exist, the PDSIZE of the reload target file is set to one of the following:

- The number of pages in the unloaded file's procedure dictionary, if that number of pages is less than or equal to 255.
- The smallest number less than 255 that will yield the smallest number of page chunks, if the number of pages in the unloaded procedure dictionary exceeds 255.

  To arrive at these smallest numbers, (the rounded-up, integral result of) the number of pages in the unloaded procedure dictionary is divided by the initial number of chunks in the new dictionary. This chunks value is the quotient of the old page count and 255, rounded up to the next integral value.

For example, for an unloaded dictionary that has 800 pages, the new chunk count would be 4 (800/255, then rounded up), and the new PDSIZE would be 200 (800/4). For an unloaded dictionary that has 1332 pages, the new chunk count would be 6, and the new PDSIZE would be 222.

The target file's existing value for PDSIZE is used in any of the following cases:

- A procedure dictionary already exists.
- PDSIZE is already set to a value other than 3.
- The unloaded procedures and aliases used no more than 3 pages.

Note: The PDSIZE optimization feature is only activated if the *Fast/Unload* user has the FSTATS feature enabled. If FSTATS is **not** enabled, necessary optimization information is not written to the unloaded data set.

CHAPTER 8   *DBCS support*

Fast/Reload provides support for mixed and pure DBCS fields. This includes support of the X'0080', X'0040', X'0020' and X'0010' FLOD mode bits. For more information on these bits see the **Model 204 DBCS Programmer's Guide**.

In addition, *Fast/Reload* will perform certain DBCS conversions automatically when loading data in UAI/LAI format. These conversions are performed automatically when a field's DBCS definition has changed. The possible DBCS conversions are:

Pure DBCS to mixed DBCS
> No conversion is performed.

Pure DBCS to mixed DBCS
> Data is wrapped in shift/out and shift/in.

Pure DBCS to non-DBCS
> The data is stored as a string with no modification.

Mixed DBCS to pure DBCS
> The data must be wrapped in shift/out and shift/in with no intervening shift/out or shift/in. If this is the case, the shift/out and shift/in are stripped. If not, an error message is issued and the field is not loaded.

Mixed DBCS to mixed DBCS
> No conversion is performed.

Mixed DBCS to non-DBCS
> The data is stored as a string with no modification.

Non-DBCS to pure DBCS
> This conversion is invalid so an error message is issued and the field is not loaded in this case.

Non-DBCS to mixed DBCS
> The data is stored as a string with no modification other than conversion of non-string types to string.

If DBCS data is stored in a field that was not defined as a DBCS field, it is possible to tell *Fast/Reload* to treat the field as if it had been a DBCS field. This is done with the **PURE** and **MIXED** statements. The PURE statement tells *Fast/Reload* to treat the unloaded data for a field as if it were pure DBCS data regardless of its actual definition. The MIXED statement tells *Fast/Reload* to treat the unloaded data for a field as if it were MIXED DBCS data regardless of its actual definition. Both the PURE and the MIXED

statements must come after the LAI statement and are invalid in non-LAI FLOD programs.

For example:

```
FILELOAD -1,-1,0,1000000,10000,10000,,68
LAI
PURE DBCS_STRING
MIXED OTHER_STRING
END
```

indicates that *Fast/Reload* is to treat data for field DBCS_STRING as pure DBCS data and data for field OTHER_STRING as mixed DBCS data.

APPENDIX A *Float Handling*

There are two contexts in which floating point values are processed by *Fast/Reload*:

● When the field being loaded into is defined as a FLOAT field.

● When the input value is a floating point value and the field being loaded into is defined as any type other than FLOAT.

These two contexts are discussed in the following two sections.

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.

A.1   Loading into FLOAT fields

An input value to be loaded into a field defined as FLOAT is processed in three different ways, depending on whether the input is provided with the X'0080' mode bit, or, if not, whether the input is a non-floating point source or is a floating point source:

X'0080' mode

In a non-LAI load, if the value being loaded is due to one of the file load statements with the X'0080' bit set (indicating float input), the floating point value is used as it is. If the input value is shorter than the field being loaded into, the fraction part of the field is padded on the right with binary zeros. If the input value is longer than the field being loaded into, the fraction part of the field is truncated to the length of the field. Otherwise (lengths the same) it is copied unchanged to the field occurrence.

Non-float

Input that is a non-float value (in either an LAI or non-LAI load) is first converted to the 8-byte floating point value that is closest to the 15-digit decimal representation of the input. Then it is treated as an 8-byte float input value, as described next.

Float If the input is a float value **not** due to the X'0080' mode bit (such an input can only occur in a LAI load), if the length of the input is the same as the length of the field, it is copied unchanged to the field occurrence. Otherwise, it is processed according to the length of the input value and the length of the field:

4-byte input

To store a 4-byte input float value (in an LAI load) into a length 8 or 16 FLOAT field, the 4-byte value is converted to an 8-byte value which is closest to the 6-digit decimal value closest to the 4-byte float input. This 8-byte value is stored unchanged in a FLOAT LEN 8 field, or the fraction is padded on the right with binary zeroes to store into a FLOAT LEN 16 field.

8-byte or 16-byte input to FLOAT LEN 4

To store FLOAT LEN 4 from an 8 or 16-byte input, the first 8 bytes of the input are used; this float value is then "hex rounded" to a FLOAT LEN 4, **without normalization**. That is, the first 24 bits of the fraction are used, incremented by 1 if the 25th fraction bit is 1; if this overflows the 24 bit fraction, the exponent is adjusted, otherwise it is copied.

8-byte input to FLOAT LEN 16

To store an 8 byte input float value into a FLOAT LEN 16 field, eight bytes of binary zero are padded on the right.

16-byte input to FLOAT LEN 8

To store a 16 byte input float value into a FLOAT LEN 8 field, the first 8 bytes of the input are used.

A.2   Loading float values into non-FLOAT fields

Loading a floating point input value into a **non-FLOAT** field depends on whether the input value is 4 bytes long or not:

4-byte float input

A 4-byte floating point input value being loaded into a **non-FLOAT** field is first converted to a 6-digit decimal number closest to the value of the floating point input.

8-byte or 16-byte float input

An 8-byte or 16-byte floating point input value being loaded into a **non-FLOAT** field is first converted to a 15-digit decimal number closest to the value of the floating point input.

The resulting decimal number is then stored into the field being loaded, according to the rules for the defined field type.

APPENDIX B *Installing Fast/Reload*

Fast/Reload was part of the *Sirius Mods*. Until version 7.5 of Model 204, see the **Sirius Mods Installation Guide** for installation instructions.

Fast/Reload logs FLOD and FILELOAD programs to a temporary procedure. This is so that if it decides to pass the FLOD or FILELOAD program to standard FLOD or FILELOAD, the statements it has scanned are not lost. Ordinarily, the temporary procedure used for this purpose is -1. If this is inconvenient, the temporary procedure number to be used can be changed by setting the FRELPREV system parameter. This parameter should be set to 0 or a negative number greater than the value of he NORQS system parameter. Setting FRELPREV to a positive number turns off the copying of FLOD/FILELOAD programs to a temporary procedure and hence makes it impossible to have *Fast/Reload* automatically pass of to standard FLOD or FILELOAD.

APPENDIX C   *Messages*

Please refer to
http://m204wiki.rocketsoftware.com/index.php/M204wiki_main_page#Messages for
messages related to *Fast/Reload*.

APPENDIX D *Date Processing*

This chapter presents date processing issues for *Fast/Reload*. The only use of dates within *Fast/Reload* is to examine the CPU clock (as returned by the STCK hardware instruction) to determine the current date, in case *Fast/Reload* is under a rental or trial agreement. Please note that *Fast/Reload* itself does not produce any results which depend on the content of any data which may be date values. However, since it does load content into *Model 204* fields, if that content contains two digit year date values, the customer must ensure that any application using that data has an algorithm or rule for unambiguously determining the correct century for the values.

You must examine all uses of date values in your applications to ensure that each of your applications produces correct results. Furthermore, both the operating system and *Model 204* must correctly process and transmit dates beyond 1999 in order for *Fast/Reload* to operate properly.

Index

A

ANYorder option ... 6

B

BAVail option ... 7
BRESERVE ... 8, 20

D

DBCS support ... 27
DELETE statement ... 5
DElfield parameter ... 14
DKOnly option ... 11, 19
DOWN statement ... 5

E

ENDF statement ... 5
ENDL statement ... 5
ERRCont option ... 11
Errors, ignoring ... 11

F

F statement ... 5
FAMsplit parameter ... 14
Fast/Reorg ... 1
Field definitions ... 15
Fields, deleting ... 14
FILELOAD parameters ... 16
FLOD parameters ... 16
FRELPREV parameter ... 5, 31
FTOnly option ... 12

H

Hash key files ... 6, 16

I

Invisible fields ... 23

L

L statement ... 5
LAI ... 1, 14, 17

M

Multiple step reloads ... 16

N

NEWfgid parameter ... 15
NObuff n_out_buffs option ... 12
NOfdef parameter ... 15
NOIndex parameter ... 15
NOProcs parameter ... 15
NOVAlidate option ... 13
Null data ... 17
NXBuff n_tableX_buffs option ... 12

O

OPTIONS statement ... 6
Ordered index ... 19-20

P

PDSIZE parameter, Model 204 ... 26
Performance ... 21
Procedure dictionary ... 25
Procedure support ... 25

S

Sort key files ... 6, 16
splitting a Sir2000 FMF family ... 14
Splitting an reload into multiple steps ... 16
STAtc stats_intvl option ... 13

T

TAPei ddname option ... 14

U

UAI ... 1, 14, 16-17
UP statement ... 5

Index

Figures

Figures

Index