



GT.M

Release Notes

V7.1-004

Empowering
the Financial World

FIS

Contact Information

GT.M Group
Fidelity National Information Services, Inc.
347 Riverside Drive
Jacksonville, FL 13220
United States of America

GT.M Support for customers: gtmsupport@fisglobal.com
Automated attendant for 24 hour support: +1 (484) 302-3248
Switchboard: +1 (484) 302-3160

Legal Notice

Copyright ©2024 Fidelity National Information Services, Inc. and/or its subsidiaries. All Rights Reserved.






Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts and no Back-Cover Texts.

GT.M™ is a trademark of Fidelity National Information Services, Inc. Other trademarks are the property of their respective owners.

This document contains a description of GT.M and the operating instructions pertaining to the various functions that comprise the system. This document does not contain any commitment of FIS. FIS believes the information in this publication is accurate as of its publication date; such information is subject to change without notice. FIS is not responsible for any errors or defects.

Revision History		
Revision 1.0	27 June 2024	V7.1-004

Table of Contents

V7.1-004	1
Overview	1
Conventions	1
Platforms	2
Platform support lifecycle	5
Additional Installation Instructions	5
Recompile	6
Rebuild Shared Libraries or Images	6
Compiling the Reference Implementation Plugin	6
Re-evaluate TLS configuration options	7
Upgrading to V7.1-004	7
Stage 1: Global Directory Upgrade	8
Stage 2: Database Files Upgrade	8
Stage 3: Replication Instance File Upgrade	14
Stage 4: Journal Files Upgrade	14
Stage 5: Trigger Definitions Upgrade	14
Managing M mode and UTF-8 mode	14
Setting the environment variable TERM	16
Installing Compression Libraries	16
Change History	17
V7.1-004	17
Database	19
Language	21
System Administration	23
Other	25
Error and Other Messages	27
MLKHASHRESIZE 	27
MLKHASHRESIZEFAIL 	27
MLKREHASH 	27
MUTEXLCKALERT 	27
TPNOTACID 	29

This page is intentionally left blank.

V7.1-004

Overview

V7.1-004 provides additional database statistics (GVSTATS) as well as numerous changes aimed at improving ease of use and also a number of fixes.

Items marked with the 🟢 symbol document new or different capabilities.

Please pay special attention to the items marked with the 🟡 symbol, as those document items that have a possible impact on existing code, practice or process. Please be sure to recompile all objects to ensure all the updates are in place.



Note

While FIS keeps message IDs and mnemonics quite stable, message texts change more frequently as we strive to improve them, especially in response to user feedback. Please ensure you review any automated scripting that parses GT.M messages.

Conventions

This document uses the following conventions:

Flag/Qualifiers	- (dash)
Program Names or Functions	upper case. For example, MUPIP BACKUP
Examples	lower case. For example: mupip backup -database ACN,HIST /backup
Reference Number	A reference number enclosed between parentheses () used to track software enhancements and support requests.
Platform Identifier	Where an item affects only specific platforms, the platforms are listed in square brackets, e.g., [AIX]



Note

The term UNIX refers to the general sense of all platforms on which GT.M uses a POSIX API. As of this date, this includes: AIX and GNU/Linux x86_64.

Effective V6.0-000, GT.M documentation adopted IEC standard Prefixes for binary multiples. This document therefore uses prefixes Ki, Mi and Ti (e.g., 1MiB for 1,048,576 bytes). Over time, we'll update all GT.M documentation to this standard.

✔ denotes a new feature that requires updating the manuals.

⚠ denotes a new feature or an enhancement that may not be upward compatible and may affect an existing application.

✖ denotes deprecated messages.

⚠ denotes revised messages.

➕ denotes added messages.

Platforms


Over time, computing platforms evolve. Vendors obsolete hardware architectures. New versions of operating systems replace old ones. We at FIS continually evaluate platforms and versions of platforms that should be Supported for GT.M. In the table below, we document not only the ones that are currently Supported for this release, but also alert you to our future plans given the evolution of computing platforms. If you are an FIS customer, and these plans would cause you hardship, please contact your FIS account executive promptly to discuss your needs.

Each GT.M release is extensively tested by FIS on a set of specific versions of operating systems on specific hardware architectures, we refer to the combination of operating system and hardware architecture as a platform. We deem this set of specific versions: Supported. There may be other versions of the same operating systems on which a GT.M release may not have been tested, but on which the FIS GT.M Group knows of no reason why GT.M would not work. We deem this larger set of versions: Supportable. There is an even larger set of platforms on which GT.M may well run satisfactorily, but where the FIS GT.M team lacks the knowledge to determine whether GT.M is Supportable and therefore deem them: Unsupported. Contact FIS GT.M Support with inquiries about your preferred platform.

As of the publication date, FIS supports this release on the hardware and operating system versions below. Contact FIS for a current list of Supported platforms. The reference implementation of the encryption reference plugin has its own additional requirements.

Platform	Supported Versions	Notes
IBM Power Systems AIX	7.1 TL 5, 7.2 TL 5, 7.3 TL 1	<p>Only 64-bit versions of AIX with POWER7 as the minimum required CPU architecture level are Supported.</p> <p>While GT.M supports both UTF-8 mode and M mode on this platform, there are problems with the AIX ICU utilities that prevent FIS from testing 4-byte UTF-8 characters as comprehensively on this platform as we do on others.</p> <p>Running GT.M on AIX 7.1 requires APAR IZ87564, a fix for the POW() function, to be applied. To verify that this fix has been installed, execute instfix -ik IZ87564.</p>

Platform	Supported Versions	Notes
		Only the AIX jfs2 filesystem is Supported. Other filesystems, such as jfs1 are Supportable, but not Supported. FIS strongly recommends use of the jfs2 filesystem on AIX; use jfs1 only for existing databases not yet migrated to a jfs2 filesystem.
x86_64 GNU/Linux	Red Hat Enterprise Linux 7.9, 8.9, 9.4; Ubuntu 20.04 LTS, and 22.04 LTS; Amazon Linux 2	<p>To run 64-bit GT.M processes requires both a 64-bit kernel as well as 64-bit hardware. As of V7.1-001, GT.M on x86-64 requires hardware/virtualized support for AVX instructions.</p> <p>GT.M should also run on recent releases of other major Linux distributions with a contemporary Linux kernel (2.6.32 or later), glibc (version 2.12 or later) and ncurses (version 5.7 or later).</p> <p>Due to build optimization and library incompatibilities, GT.M versions older than V6.2-000 are incompatible with glibc 2.24 and up. This incompatibility has not been reported by a customer, but was observed on internal test systems that use the latest Linux software distributions from Fedora (26), Debian (unstable), and Ubuntu (17.10). In internal testing, processes either hung or encountered a segmentation violation (SIG-11) during operation. Customers upgrading to Linux distributions that utilize glibc 2.24+ must upgrade their GT.M version at the same time as or before the OS upgrade.</p> <p>GT.M requires a compatible version of the libtinfo library. On Red Hat, the ncurses-libs and ncurses-compat-libs packages contain the libtinfo library. On Debian/Ubuntu, libtinfo5 and libncurses5 packages contain the libtinfo library. If any of these packages is not already installed on your system, please install using an appropriate package manager.</p> <p>To support the optional WRITE /TLS fifth argument (the ability to provide / override options in the tlsid section of the encryption configuration file), the reference implementation of the encryption plugin requires libconfig 1.4.x or later.</p> <p>Only the ext4 and xfs filesystems are Supported. Other filesystems are Supportable, but not Supported. Furthermore, if you use the NODEFER_ALLOCATE feature, FIS strongly recommends that you use xfs. If you must use NODEFER_ALLOCATE with ext4, you must ensure that your kernel includes commit d2dc317d564a46dfc683978a2e5a4f91434e9711 (search for d2dc317d564a46dfc683978a2e5a4f91434e9711 at https://www.kernel.org/pub/linux/kernel/v4.x/ChangeLog-4.0.3). The Red Hat Bugzilla identifier for the bug is 1213487. With NODEFER_ALLOCATE, do not use any filesystem other than ext4 and a kernel with the fix, or xfs.</p>

Platform	Supported Versions	Notes
		<p>Our testing has shown an interaction between glibc 2.36 and all versions of GT.M on Linux/x86_64 systems without AVX2 support. This can cause segmentation violations (SIG-11) in processes performing concurrent updates to the same database block, which terminate the process, but do not damage the database. The issue is due to the way glibc performs certain memory operations when using SSE2 instructions. The glibc behavior was subsequently modified to avoid this issue, and the change was included in glibc 2.37. Linux/x86_64 systems with support for AVX2 instructions are not vulnerable, as glibc chooses its AVX2 implementation, when available, over its SSE2 implementation, and the problematic behavior is specific to SSE2. Note, depending on how CPU virtualization is configured, that virtual environments may not support AVX2 even if the underlying hardware does.</p> <p>Ubuntu 24.04 LTS is Supportable.</p> <div><p>Note</p><p>FIS recommends recompiling the reference encryption plugins to match the target platform. See Compiling the Reference Implementation Plugin section for instructions.</p><p>OpenSSL 3.0 by default does not allow client-side initiated TLSv1.2 renegotiation requests due to potential DoS attacks. Because of this, the reference TLS implementation in GT.M versions before V7.0-004 do not use the appropriate OpenSSL 3.0 API to enable support for client-side initiated TLSv1.2 renegotiation. Customers needing to replicate to/from GT.M versions before V7.0-004 with OpenSSL 3.0 must use -RENEGOTIATE_INTERVAL=0 in the Source Server startup. This limitation only affects database replication and not SOCKET devices.</p></div>



Important

Effective V7.0-003, GT.M is no longer Supportable on the 32 bit x86 platform. Please contact your FIS account manager if you need ongoing support for GT.M on this platform.

Platform support lifecycle

FIS usually supports new operating system versions six months or so after stable releases are available, and we usually support each version for a two-year window.

We support GT.M releases in a rolling support model based on two years of certified releases. A release becomes no longer officially supported once a given release is more than one release beyond the two year window. Historically we have produced GT.M releases on a quarterly basis, subject to change. Note: customers always get the best support by staying current with releases as they are made available.

FIS will continue to attempt to support any release of GT.M in use by a Profile customer under that client's maintenance agreement, while that agreement is still in effect. FIS's ability to provide an appropriate level of support may become increasingly costly to the client. In other words, FIS may need to enact a special maintenance agreement to continue to provide support. The additional costs required would be maintain client release level specific servers, operating systems and other ancillary software for a given and reasonable time frame beyond the normal window.

FIS policy is only to provide remediation, in the current release, for identified issues in generally available and supported releases. It is not FIS policy to provide ongoing support of client specific release levels of unsupported software.

GT.M cannot be patched, and bugs are only fixed in new releases of software.


Additional Installation Instructions

To install GT.M, see the "Installing GT.M" section in the GT.M Administration and Operations Guide. For minimal down time, upgrade a current replicating instance and restart replication. Once that replicating instance is current, switch it to become the originating instance. Upgrade the prior originating instance to become a replicating instance, and perform a switchover when you want it to resume an originating primary role.



Caution

Never replace the binary image on disk of any executable file while it is in use by an active process. It may lead to unpredictable results. Depending on the operating system, these results include but are not limited to denial of service (that is, system lockup) and damage to files that these processes have open (that is, database structural damage).

- FIS strongly recommends installing each version of GT.M in a separate (new) directory, rather than overwriting a previously installed version. If you have a legitimate need to overwrite an existing GT.M installation with a new version, you must first shut down all processes using the old version. FIS suggests installing GT.M V7.1-004 in a Filesystem Hierarchy Standard compliant location such as /usr/lib/fis-gtm/V7.1-004_arch (for example, /usr/lib/fis-gtm/V7.1-004_x86_64 on Linux systems). A location such as /opt/fis-gtm/V7.1-004_arch would also be appropriate.
- Use the appropriate MUPIP command (e.g. ROLLBACK, RECOVER, RUNDOWN) of the old GT.M version to ensure all database files are cleanly closed.
- Make sure gtmsecshr is not running. If gtmsecshr is running, first stop all GT.M processes including the DSE, LKE and MUPIP utilities and then perform a **MUPIP STOP *pid_of_gtmsecshr***.
- Starting with V6.2-000, GT.M no longer supports the use of the deprecated \$gtm_dbkeys and the master key file it points to for database encryption. To convert master files to the libconfig format, please click  to download the CONVDBKEYS.m program and follow instructions in the comments near the top of the program file. You can also download CONVDBKEYS.m from <http://tinco.pair.com/bhaskar/gtm/doc/articles/downloadables/CONVDBKEYS.m>. If you are using \$gtm_dbkeys for database encryption, please convert master key files to libconfig format immediately after upgrading to V6.2-000 or later. Also, modify your environment scripts to include the use of gtmcrypt_config environment variable.

Recompile

- Recompile all M and C source files.

Rebuild Shared Libraries or Images

- Rebuild all Shared Libraries after recompiling all M and C source files.
- If your application is not using object code shared using GT.M's auto-relink functionality, please consider using it.

Compiling the Reference Implementation Plugin

If you plan to use the example / reference implementation plugin in support of database encryption, TLS replication, or TLS sockets, you must compile the reference plugin in order to match the shared library dependencies specific to your platform. The instructions for compiling the Reference Implementation plugin are as follows:

1. Install the development headers and libraries for libgcrypt, libgpgme, libconfig, and libssl. On Linux, the package names of development libraries usually have a suffix such as -dev or -devel and are available through the package manager. For example, on Ubuntu_x86_64 a command like the following installs the required development libraries:

```
sudo apt-get install libgcrypt11-dev libgpgme11-dev libconfig-dev libssl-dev
```

Note that the package names may vary by distribution / version. For example, on RHEL 9 the libraries required to recompile the reference implementation encryption plugin are libgcrypt-devel, gpgme-devel, libconfig-devel, and openssl-devel.

2. Unpack \$gtm_dist/plugin/gtmcrypt/source.tar to a temporary directory.

```
mkdir /tmp/plugin-build
cd /tmp/plugin-build
cp $gtm_dist/plugin/gtmcrypt/source.tar .
tar -xvf source.tar
```

3. Follow the instructions in the README.

- Open Makefile with your editor; review and edit the common header (IFLAGS) and library paths (LIBFLAGS) in the Makefile to reflect those on your system.
 - Define the gtm_dist environment variable to point to the absolute path for the directory where you have GT.M installed
 - Copy and paste the commands from the README to compile and install the encryption plugin with the permissions defined at install time
4. When reinstalling or upgrading GT.M, stop existing gpg-agents. The agents may be working with information about the prior GT.M installation, such as GNUPGHOME, that will not work with the new version. Additionally, if the process deletes the GPG agent's socket, proper operation requires a new agent.
 5. It is a good idea to read the Administration and Operations Guide section entitled "Special note - GNU Privacy Guard and Agents" and re-evaluate the GPG configuration options in use.

Re-evaluate TLS configuration options

The GT.M TLS reference encryption plugin implements a subset of options as documented in the OpenSSL man page for SSL_set_options which modify the default behavior of OpenSSL. Future versions of the plugin will enable new options as and when the OpenSSL library adds them. To enable options not supported by the GT.M TLS reference plugin, it is possible to create an OpenSSL configuration for GT.M processes. See the OpenSSL man page for "config".

<para/>

Upgrading to V7.1-004



Before you begin

GT.M supports upgrade from V5*, V6.* and V7.* versions to V7.1-004.

GT.M does not support upgrading from V4* versions. Please upgrade V4 databases using instruction in the release notes of an appropriate GT.M V6.* version.

The GT.M database consists of four types of components- database files, journal files, global directories, and replication instance files.

GT.M upgrade procedure for V7.1-004 consists of 5 stages:

- Stage 1: Global Directory Upgrade
- Stage 2: Database Files Upgrade
- Stage 3: Replication Instance File Upgrade
- Stage 4: Journal Files Upgrade
- Stage 5: Trigger Definitions Upgrade

Before starting, read the upgrade instructions of all stages carefully. Your upgrade procedure for GT.M V7.1-004 depends on your GT.M upgrade history and your current version.

Stage 1: Global Directory Upgrade

FIS strongly recommends you back up your Global Directory file before upgrading. There is no one-step method for downgrading a Global Directory file to an older format.

To upgrade from any previous version of GT.M:

- Open your Global Directory with the GDE utility program of GT.M V7.1-004.
- Execute the EXIT command. This command automatically upgrades the Global Directory.
- If you inadvertently open a Global Directory of an old format with no intention of upgrading it, execute the QUIT command rather than the EXIT command.

If you inadvertently upgrade a global directory, perform the following steps to downgrade to an old GT.M release:

- Open the global directory with the GDE utility program of V7.1-004.
- Execute the SHOW -COMMAND -FILE=file-name command. This command stores the current Global Directory settings in the file-name command file. If the old version is significantly out of date, edit the command file to remove the commands that do not apply to the old format. Alternatively, you can use the output from SHOW -ALL or SHOW -COMMAND as a guide to manually enter equivalent GDE commands for the old version.

An analogous procedure applies in the reverse direction.

Stage 2: Database Files Upgrade

Before starting the database file upgrade, use the prior GT.M version to perform an appropriate MUPIP action (i.e. ROLLBACK, RECOVER, RUNDOWN) to remove abandoned GT.M database semaphores and release any IPC resources.

There are three upgrade paths available when you upgrade to V7.1-004.

V7 Upgrade Path 1: In-place Upgrade

To upgrade from GT.M V7*:

There is no explicit procedure to upgrade a V7 database file when upgrading to a newer V7 version. After upgrading the Global Directory, opening a V7 database with a newer V7 GT.M process automatically upgrades the fields in the database file header.

To upgrade from GT.M V6* (or V5*):

There are two phases to upgrade from V6 to V7:

- Phase 1: MUPIP UPGRADE phase; requires standalone access
- Phase 2: MUPIP REORG -UPGRADE (GVT Index Block Upgrade); may optionally run with concurrent access if performance is acceptable

Both phases operate once per region. Phase 1 is not restartable. Phase 2 is restartable.

While these are the basic steps, customers must integrate them with appropriate operational practice and risk mitigating procedures, such as comprehensive testing, backup, integrity checks, journal and replication management, and so on based on their environments and risk tolerance. FIS strongly recommends performing a MUPIP INTEG (-FAST), of the database and creating a backup prior to upgrade. Customers must test these utilities against copies of their own production files, using their planned procedures, before undertaking the conversion of current production files.

Using MUPIP UPGRADE and MUPIP REORG -UPGRADE should be a significantly faster alternative to using MUPIP EXTRACT and LOAD. FIS favors using a "rolling" upgrade using a replicated instance. Whatever the approach you choose, FIS requests capturing all logs in case there are issues or questions leading to support requests.

Phase 1: Standalone MUPIP UPGRADE

MUPIP UPGRADE performs Phase 1 actions of upgrading a database to V7. The format of the UPGRADE command is:

```
MUPIP UPGRADE {-FILE <file name>; | [-REGION] <region list>}
```

As the GT.M version upgrade changes the journal format to support 64-bit block pointers, MUPIP UPGRADE does not maintain journal files or replication; configured journaling and replication resumes for activity after MUPIP UPGRADE.

UPGRADE:

- Requires standalone access
- Turns off journaling and replication for the duration of UPGRADE

- When encountering an error where the command specifies multiple regions, UPGRADE moves on to the next region, while for a single file/region, it terminates; avoid any unnecessary <CTRL_C> or MUPIP STOP (or kill) of an active MUPIP UPGRADE process, as such an action leaves the database region effectively unusable
- Estimates and reports the space required for its work
 - UPGRADE estimates are intended to be generous, and, particularly for small databases, they may seem unnecessarily large
 - If MUPIP is not authorized to perform a required file extension, that is, the extension amount is defined as zero (0), it produces an error before it does anything that would damage the selected database file
- Moves blocks from immediately after the existing master map to make room for a V7 master map
 - Depending on the block size and the GT.M version with which it was created, the new starting Virtual Block Number (VBN), the location of block zero for the database file, may exceed the starting VBN for a database created with V7, which causes a minor waste of space
 - UPGRADE relocates blocks in multiples of 512 to align blocks with their local bitmaps
- Eliminates any globals that previously existed, but have been KILL'd at the name level; these global variable trees (GVTs) contain only a level one (1) root block and an empty data (level zero) block and are "invisible" to the GT.M process run-time
- Stores the offset GT.M must apply to the original block pointers as a consequence of the relocation of the starting VBN
- Upgrades the directory tree (DT) block pointers from 32- to 64-bits; this requires splitting any blocks that do not have sufficient space to accommodate the larger block pointers
- Ensures that all its work is flushed to secondary storage
- Reports completion of its activity on a database file with a "MUPIP MASTERMAP UPGRADE completed" message

At this point, after a successful MUPIP UPGRADE:

- All DT blocks are in V7m format and all GVT index blocks remain in V6/V6p format
- Subsequent activity that updates index blocks for existing GVTs implicitly converts any V6 index blocks to V6p format after applying the offset
- No process other than MUPIP REORG -UPGRADE converts GVT index blocks from V6p format to V7m format; in other words, adding new nodes does not create GVT index blocks with V7 format
 - adding new nodes splits existing index blocks and such block splits retain the pre-existing block format
- Newly created GVTs, storing new global names, have V7m format

- Data blocks, at level zero (0), and local bit map blocks have the same format in V6 and V7, so, for consistency, normal updates also give those blocks a V7m format designation

These database changes are physical rather than logical, and thus do not require replication beyond noting the increase in transaction numbers.

Phase 2: MUPIP REORG -UPGRADE (GVT Index Block Upgrade)

MUPIP REORG -UPGRADE performs Phase 2 actions of upgrading a database to V7 format. The format of MUPIP REORG -UPGRADE is:

```
MUPIP REORG -UPGRADE {-FILE <file_name> | [-REGION] <region_list>}
```

Before image journaling with MUPIP REORG upgrade provides maximum resiliency. MUPIP REORG -UPGRADE reports it has completed its actions for a region with a MUPGRDSUCC message, at which point all index blocks have V7m format with 64-bit block pointers. You can resume and complete a MUPIP REORG -UPGRADE stopped with a MUPIP STOP (or <Ctrl-C>); avoid a kill -9, which carries a high risk of database damage.

MUPIP REORG -UPGRADE:

- Requires standalone access
- Runs on an entire region; as a result, MUPIP REORG -UPGRADE prevents multiple concurrent REORG -UPGRADE runs per region
- Stops execution when a concurrent Online ROLLBACK is detected because that operation changes the block content of the database
- Can be subject to stopping and restarting at any point
- Processes the GVTs within a database file
 - Splitting any index blocks that do not have sufficient space to accommodate the block pointer upgrade from 32 to 64 bits
 - Updating the block pointers from 32 to 64 bits, also changing the version of the block to V7m
 - Journaling its work as before images (if so configured) and INCTN records

Phase 3: Optional GVT Data and Local Bit Map Block Upgrade

While it makes no operational or processing difference, GT.M does not consider the database "fully upgraded" until the block version format of all data blocks becomes V7m. Any of the following operations upgrade some or all of the remaining data blocks:

- MUPIP REORG

Because this operation may not visit every block in the database it may fail to upgrade static/unchanging blocks

- MUPIP REORG -ENCRYPT
- MUPIP INTEG -TN_RESET

This operation requires standalone access and resets the transaction number on all blocks in the database.

Failure to perform Phase 3 has **NO** implications for V7.1-004 but might be an issue for any as-yet unplanned further enhancement.



Important

Taking the steps in the following list that use MUPIP REORG -MIN_LEVEL=1 significantly reduce upgrade time.

The following lists the recommended ordered steps for the full upgrade process:

1. Offline Upgrade instance to use new GT.M V7.1-002+ version - at this point, customers can use the upgraded the GT.M version without any DB changes
2. Online MUPIP SET -INDEX_RESERVED_BYTES=n - where n is 1/3 the block size
3. Online MUPIP REORG -MIN_LEVEL=1 -NOSWAP - free up space in all index blocks to ease the block reference change from 32bits (4bytes) to 64bits (8bytes); this operation alters only index blocks (-MIN_LEVEL=1), and so generates a much lower volume of before image journal records.
4. Offline MUPIP UPGRADE -move blocks around to make space for the expanded master bitmap and upgrade the index blocks in the directory tree (tree of Global names).
5. Online MUPIP REORG -UPGRADE - upgrade the remaining index blocks
6. Online MUPIP SET -INDEX_RESERVED_BYTES=0 - remove the previously applied reservation as it is no longer needed; some application may find it produces a continuing performance benefit.
7. (optional) Online REORG -MIN_LEVEL=1 -NOSWAP -NOSPLIT - coalesce the index blocks to leave index blocks in a less fragmented state

V7 Upgrade Path 2: EXTRACT and LOAD

Two commonly used mechanisms are as follows. We recommend you use replication to stage the conversion and minimize down time.

- MUPIP EXTRACT -FREEZE followed by a MUPIP LOAD

Using MUPIP EXTRACT with -FREEZE ensures that the V6 database files are frozen at the point of the extract, preventing updates without administrative action to unfreeze the database. MUPIP LOAD the extracts into newly created V7 database files

Use this operation when there is insufficient space to make a database extract

- MERGE command with two global directories and Extended References

Using this approach to transfer data from a V6 database file to a V7 database, administrators must take some action to prevent updates during the transfer

This operation consumes less disk space and disk I/O. As a result the operation is faster than an EXTRACT and LOAD.



If you are using triggers, extract the triggers from the V6 database and load them in the new V7 database.

V7 Upgrade Path 3: No change

Continue using your V6 databases with GT.M V7.1-004. In case you do not wish to operate with files of differing format, specify the -V6 qualifier when invoking MUPIP CREATE.

Choosing the right upgrade path

Choose V7 Upgrade Path 1 or 2 if you anticipate a database file to grow to over 994Mi blocks or require trees of over 7 levels as V7.1-004 supports 16Gi blocks and 11 levels. Note that the maximum size of a V7 database file having 8KiB block size is 114TiB (8KiB*16Gi).

Choose the V7 Upgrade Path 3 if you do not anticipate a database file to grow beyond the V6 database limit of 994Mi blocks or a tree depth limit of 7 levels. Note that the maximum size of a V6 database file having 8KiB block size is 7TiB (8KiB*992Mi).

Other than the new maximum database file size and greater tree depth that comes with V7 Upgrade Path 1 and 2, there is no difference between V7 Upgrade Path 1 and 2 and V7 Upgrade Path 3. You can choose V7 Upgrade Path 3 first and then later choose V7 Upgrade Path 1 or 2 if a need arises.

For additional details on differences in factors involved in the V6 to V7 upgrade refer to Appendix G in the GT.M Administration and Operations Guide.

Database Compatibility Notes

- Changes to the database file header may occur in any release. GT.M automatically upgrades database file headers as needed. Any changes to database file headers are upward and downward compatible within a major database release number, that is, although processes from only one GT.M release can access a database file at any given time, processes running different GT.M releases with the same major release number can access a database file at different times.
- Databases created with V5.3-004 through V5.5-000 can grow to a maximum size of 224Mi (234,881,024) blocks. This means, for example, that with an 8KiB block size, the maximum database file size is 1,792GiB; this is effectively the size of a single global variable that has a region to itself and does not itself span regions; a database consists of any number of global variables. A database created with GT.M versions V5.0-000 through V5.3-003 can be upgraded with the V5 version of MUPIP UPGRADE to increase the limit on database file size from 128Mi to 224Mi blocks.

- Databases created with V5.0-000 through V5.3-003 have a maximum size of 128Mi (134, 217,728) blocks. GT.M versions V5.0-000 through V5.3-003 can access databases created with V5.3-004 and later as long as they remain within a 128Mi block limit.
- Database created with V6.0-000 through V6.3-014 have a maximum size of 1,040,187,392 (992Mi) blocks.
- Database created with V7.0-000 and up have a maximum size of 17,179,869,184 (16Gi) blocks.

Stage 3: Replication Instance File Upgrade

GT.M V7.1-004 does not require new replication instance files when upgrading from any version after V6.0-000.

Stage 4: Journal Files Upgrade

On every GT.M upgrade:

- Create a fresh backup of your database
- Generate new journal files (without back-links), typically by turning journaling OFF and then back ON



Important

This is necessary because MUPIP JOURNAL cannot use journal files from a release other than its own for e.g. RECOVER, ROLLBACK, or EXTRACT.

MUPIP UPGRADE temporarily disables journaling and replication settings for the duration of its activity. Once complete, MUPIP UPGRADE restores prior settings.

Stage 5: Trigger Definitions Upgrade

GT.M V7.1-004 does not require trigger definition upgrade when upgrading GT.M from any version after V6.3-000. If upgrading from a prior GT.M release, please see the instructions in the release notes for V6.3-014.

Managing M mode and UTF-8 mode

With International Components for Unicode® (ICU) version 3.6 or later installed, GT.M's UTF-8 mode provides support for Unicode® (ISO/IEC-10646) character strings. On a system that does not have ICU 3.6 or later installed, GT.M only supports M mode.

On a system that has ICU installed, GT.M optionally installs support for both M mode and UTF-8 mode, including a utf8 subdirectory of the directory where GT.M is installed. From the same source file, depending upon the value of the environment variable gtm_chset, the GT.M compiler generates

an object file either for M mode or UTF-8 mode. GT.M generates a new object file when it finds both a source and an object file, and the object predates the source file and was generated with the same setting of `$gtm_chset/$ZCHset`. A GT.M process generates an error if it encounters an object file generated with a different setting of `$gtm_chset/$ZCHset` than that processes' current value.

Always generate an M object module with a value of `$gtm_chset/$ZCHset` matching the value processes executing that module will have. As the GT.M installation itself contains utility programs written in M, their object files also conform to this rule. In order to use utility programs in both M mode and UTF-8 mode, the GT.M installation ensures that both M and UTF-8 versions of object modules exist, the latter in the `utf8` subdirectory. This technique of segregating the object modules by their compilation mode prevents both frequent recompiles and errors in installations where both modes are in use. If your installation uses both modes, consider a similar pattern for structuring application object code repositories.

GT.M is installed in a parent directory and a `utf8` subdirectory as follows:

- Actual files for GT.M executable programs (`mumps`, `mupip`, `dse`, `lke`, and so on) are in the parent directory, that is, the location specified for installation.
- Object files for programs written in M (GDE, utilities) have two versions - one compiled with support for UTF-8 mode in the `utf8` subdirectory, and one compiled without support for UTF-8 mode in the parent directory. Installing GT.M generates both versions of object files, as long as ICU 3.6 or greater is installed and visible to GT.M when GT.M is installed, and you choose the option to install UTF-8 mode support. During installation, GT.M provides an option that allows placing the object code in shared libraries in addition to individual files in the directory.
- The `utf8` subdirectory has files called `mumps`, `mupip`, `dse`, `lke`, and so on, which are relative symbolic links to the executables in the parent directory (for example, `mumps` is the symbolic link `../mumps`).
- When a shell process sources the file `gtmprofile`, the behavior is as follows:
 - If `$gtm_chset` is "m", "M" or undefined, there is no change from the previous GT.M versions to the value of the environment variable `$gtmroutines`.
 - If `$gtm_chset` is "UTF-8" (the check is case-insensitive),
 - `$gtm_dist` is set to the `utf8` subdirectory (that is, if GT.M is installed in `/usr/lib/fis-gtm/gtm_V7.1-004_i686`, then `gtmprofile` sets `$gtm_dist` to `/usr/lib/fis-gtm/gtm_V7.1-004_i686/utf8`).
 - On platforms where the object files have not been placed in a `libgtmutil.so` shared library, the last element of `$gtmroutines` is `$gtm_dist($gtm_dist/..)` so that the source files in the parent directory for utility programs are matched with object files in the `utf8` subdirectory. On platforms where the object files are in `libgtmutil.so`, that shared library is the one with the object files compiled in the mode for the process.

For more information on `gtmprofile`, refer to the Basic Operations chapter of GT.M Administration and Operations Guide.

Although GT.M uses ICU for UTF-8 operation, ICU is not FIS software and FIS does not support ICU.

Setting the environment variable TERM

The environment variable TERM must specify a terminfo entry that accurately matches the terminal (or terminal emulator) settings. Refer to the terminfo man pages for more information on the terminal settings of the platform where GT.M needs to run.

- Some terminfo entries may seem to work properly but fail to recognize function key sequences or fail to position the cursor properly in response to escape sequences from GT.M. GT.M itself does not have any knowledge of specific terminal control characteristics. Therefore, it is important to specify the right terminfo entry to let GT.M communicate correctly with the terminal. You may need to add new terminfo entries depending on your specific platform and implementation. The terminal (emulator) vendor may also be able to help.
- GT.M uses the following terminfo capabilities. The full variable name is followed by the capname in parenthesis:

```
auto_right_margin(am), clr_eos(ed), clr_eol(el), columns(cols), cursor_address(cup),
cursor_down(cud1), cursor_left(cub1), cursor_right(cuf1), cursor_up(cuu1),
eat_newline_glitch(xenl), key_backspace(kbs), key_dc(kdch1),key_down(kcud1),
key_left(kcub1), key_right(kcuf1), key_up(kcuu1), key_insert(kich1),
keypad_local(rmkx),keypad_xmit(smkn), lines(lines).
```

GT.M sends keypad_xmit before terminal reads for direct mode and READs (other than READ *) if EDITING is enabled. GT.M sends keypad_local after these terminal reads.

Installing Compression Libraries

If you plan to use the optional compression facility for replication, you must provide the compression library. The GT.M interface for compression libraries accepts the zlib compression libraries without any need for adaptation. These libraries are included in many UNIX distributions and are downloadable from the zlib home page. If you prefer to use other compression libraries, you need to configure or adapt them to provide the same API as that provided by zlib.

If a package for zlib is available with your operating system, FIS suggests that you use it rather than building your own.

By default, GT.M searches for the libz.so shared library in the standard system library directories (for example, /usr/lib, /usr/local/lib, /usr/local/lib64). If the shared library is installed in a non-standard location, before starting replication, you must ensure that the environment variable LIBPATH (AIX) or LD_LIBRARY_PATH (GNU/Linux) includes the directory containing the library. The Source and Receiver Server link the shared library at runtime. If this fails for any reason (such as file not found, or insufficient authorization), the replication logic logs a DLLNOOPEN error and continues with no compression.

Although GT.M uses a library such as zlib for compression, such libraries are not FIS software and FIS does not support any compression libraries.

Change History

V7.1-004

Fixes and enhancements specific to V7.1-004:

Id	Prior Id	Category	Summary
GTM-9141	GTM-F135289	Admin	🔴 Improve Source Server memory management
GTM-9272	GTM-F135238	Admin	🔴 Create database files with permissions from umask
GTM-9653	GTM-DE201377	Other	Include region and database file names in MLKHASHRESIZE, MLKHASHRESIZEFAIL and MLKREHASH messages
GTM-9775	GTM-DE532807	Other	Avoid GTMASSERT due to a high rate of MUPIP INTRPTs
GTM-9812	GTM-DE557990	Other	Handle asynchronous events more appropriately
GTM-9815	GTM-DE562037	Other	Address issue with Direct Mode handling of MUPIP INTRPT and similar signals
GTM-9831	GTM-DE575840	Other	🔴 Address interaction with C compiler optimization
GTM-9833	GTM-DE579226	Language	Improve \$STORAGE
GTM-9836	GTM-DE582271	DB	Manage online rollback interaction with concurrent kills appropriately
GTM-9839	GTM-DE583881	Admin	Correct encrytion interaction with MUPIP REORG
GTM-9841	GTM-DE585411	Language	Compiler Optimization
GTM-9855	F132375	Language	READ * from a socket returns -1 if the read fails
GTM-10071	GTM-F238637	DB	Improve wait for database block read in progress
GTM-10076	GTM-F239380	DB	Add database index block statistics 🟢
GTM-10081	GTM-F240798	Admin	Include a timestamp in GETADDRINFO error message. 🟢

Id	Prior Id	Category	Summary
GTM-10098	GTM-F249269	DB	🔴 Limit TPNOACID retries 🟢
GTM-10099	GTM-F249271	DB	🔴 Check ZEDIT and external calls for TPNOACID issues 🟢
GTM-10123	GTM-F266122	DB	Add \$INCREMENT() statistic 🟢
GTM-10150	GTM-7163	Language	Recognize and report DBROLLEDABACK errors in \$TEXT() processing
GTM-10541	-	DB	Manage the Journal pool resource appropriately after stressful events
GTM-10554	-	DB	Add sleep timing statistics 🟢
GTM-10586	-	Other	Eliminate a rare issue with asynchronous events
GTM-10587	-	Admin	Improve connection try management and IPC management at replication shutdown
GTM-10588	-	Other	🔴 Improve memory management
GTM-10591	-	Other	Prevent a timer expiration that could cause processes to loop
GTM-10597	-	Other	Prevent rare indefinite looping due to MUPIP STOP or other process kill
GTM-10603	-	DB	🔴 Provide millisecond accuracy and 60 second limit for \$ZMAXTPTIME 🟢
GTM-10615	-	Other	Protect against asynchronous events causing LOCKing processes to subsequently terminate with a GTMASSRT2
GTM-10623	-	DB	Inhibit epoch tapering for lone/single updater process
GTM-10670	-	Admin	Deal appropriately with decreases in tree depth in MUPIP REORG
GTM-10679	-	Language	🔴 Maintain \$REFERENCE to the last node it successfully processed for ZWRITE of a gvn 🟢
GTM-10684	-	Admin	Fix minor MUPIP INTEG buffer mismangement issue
GTM-10687	-	Admin	Provide a more appropriate value in errors concerning the maximum journal file extension setting
GTM-10688	-	Language	Prevent possible garbage assignments to \$ZTRAP and device EXCEPTIONs

Database

- GT.M properly manages concurrent process kills during an online rollback. Previously, a concurrent kill might have induced unwanted transaction restarts during the online rollback. This was only seen in development environments and was never reported by a customer. (GTM-9836)
- GT.M avoids unnecessary delays when one process waits for another process to complete the read from secondary storage of a block in which it is interested. Previously, such cases could result in slowdowns in processing; for example, when the replication update process repeatedly waits for reads initiated by reader helpers. (GTM-10071)
- For regions with BG access method, GT.M provides GVSTATS IDXH which gives the number of index block hits and IDXN which gives the number of index block misses. These statistics may help in determining appropriate sizing for a region's global buffers. The hit ratio may be determined as: $\text{IDXH}/(\text{IDXH}+\text{IDXN})$. Index blocks hold metadata GT.M uses to rapidly locate application data. They constitute a very small, but heavily used, portion of a database file. Finding them in the global buffers causes a higher positive impact on database performance than finding data blocks. While these statistics are useful in understanding the role of global buffers, because processes use their own caching of paths to recently referenced data, these statistics underrepresent actual index block usage. IDXN is a subset of DRD. The hit ratio for data blocks may be approximated by: $(\text{DTA}+\text{GET}+\text{KIL}+\text{ORD}+\text{QRY}+\text{SET}+\text{ZPR}-\text{DRD})/(\text{DTA}+\text{GET}+\text{KIL}+\text{ORD}+\text{QRY}+\text{SET}+\text{ZPR})$. Previously, GT.M did not provide separate index block statistics. (GTM-10076) 🟢
- By default, TPNOTACID retries for a transaction exceed 16, GT.M issues a trappable TPNOTACID error in order to limit livelock. The limit may be adjusted to between four (4) and 100, or set to zero (0) by appropriately defining the gtm_tpnacidtries environment variable; specifying a value outside the supported values silently reverts to the default. The setting of zero (0) acts like an audit by reporting all encountered operations capable of causing TPNOTACID detection to the syslog but setting the actual error limit to the default. In addition, GT.M suppresses TPNOTACID syslog warning messages that originate from the same action as the last prior message from a given process. Previously, GT.M did not enforce any limit on TPNOTACID retries, had no "audit" capability and could be noisy with informational messages for TPNOTACID retries. (GTM-10098) 🟡🟢
- GT.M uses TPNOTACID behavior for all external calls and for ZEDIT; previously, external calls and the ZEDIT command were exempted from TPNOTACID handling. (GTM-10099) 🟡🟢
- \$VIEW "GVSTAT", ZSHOW "G" and %YGBLSTAT report the count of \$INCREMENT() operations labeled INC. \$INCREMENT() operations are still included in the SET count. Previously, these operations were only included in the SET count. (GTM-10123) 🟢
- GT.M appropriately manages the journal pool resource control (ftok semaphore IPC) after tracked events lose accuracy for example after simultaneous access of a journal pool by more than more than 32Ki processes. Previously, such a situation could cause inappropriate EINVAL errors when subsequently accessing the journal pool. This was only seen in development environments and was never reported by a customer. (GTM-10541)

- GT.M includes six additional counters tracking milliseconds of time slept while waiting on another process to finish a task. The counters are available as part of all forms of GVSTATs, and have the following mnemonics and descriptions:
 - JCS - milliseconds of time slept in crit waiting for another process to finish a write into the journal file.
 - JNS - milliseconds of time slept outside of crit waiting for another process to finish a write into the journal file.
 - WPCS - milliseconds of time slept in crit waiting for another process to finish flushing a block to disk.
 - RCS - milliseconds of time slept in crit waiting for another process to finish reading in a block from disk.
 - WPCS - milliseconds of time slept in crit while performing a flush of one or more blocks to disk, usually in an epoch.
 - GCS - milliseconds of time slept in crit while acquiring a global buffer and waiting for another process to finish writing a block to disk.

These counters can help identify the cause of performance degradation. High values per second can indicate I/O throughput issues that are forcing GT.M to perform excessive coordination activity.

(GTM-10554) 🟢

- \$ZMAXTPTIME and its initializing environment variable, \$gtm_zmaxtptime, maintain precision to milliseconds (three digits after the decimal point). Also, both ignore values exceeding a maximum of 60 seconds. Previously \$ZMAXTPTIME only provided whole seconds, and, while it limited the initial value to 60 seconds, it did not limit the SET value. (GTM-10603) 🟡 🟢
- GT.M inhibits epoch tapering when epoch tapering is enabled and only one process actively does updates. Previously, in this situation, epoch tapering could reduce performance. (GTM-10623)

Language

- \$STORAGE reports the correct value when there is a restriction on the data segment limit (RLIMIT_DATA) of a process. Previously, \$STORAGE inappropriately returned 2GiB in such a case. (GTM-9833)
- GT.M avoids processing certain unneeded compiler operations. The optimization originally appeared in V6.3-012, but was ineffective. (GTM-9841)
- READ * from a socket returns minus one (-1) if the read fails; previously it returned a zero (0) in that case, which was inconsistent with other devices and with the standard. (GTM-9855)
- \$TEXT() recognizes and reports DBROLLEDABACK error. Previously \$TEXT() ignored and did not report DBROLLEDABACK errors. This was only seen in development environments and was never reported by a customer. (GTM-10150)
- ZWRITE of a gvn maintains \$REFERENCE to the last node it successfully processed. If you use ZWRITE as a debugging tool, or unlikely, in application code, in a circumstance where preserving \$REFERENCE has importance, note \$REFERENCE and restore it, as you would for any other command or function that references a global. Previously, ZWRITE restored the \$REFERENCE to the value it had prior to the ZWRITE. (GTM-10679) 🍷🟢
- SET \$ZTRAP appropriately validates the code string provided as a new value. GTM-9480/GTM-DE304273 in V7.0-004, which addressed an issue with \$ZTRAP maintenance, introduced a possibility that the compilation of a \$ZTRAP during a "SET \$ZTRAP=x" operation could result in assigning a garbage value to \$ZTRAP. A similar issue, dating to V6.3-001A, also affects certain I/O exception handlers specified as exception device parameters. If your application always uses code vectors rather than labels, a workaround for both issues is to set the gtm_ztrap_form environment variable to "adaptive", which prevents the assignment operation from validating the code. (GTM-10688)

This page is intentionally left blank.

System Administration

- A Source Server more reliably returns memory used to manage journal files to the operating system as it moves from older to newer journal files for a region. The Source Server uses the `mmap()` and `munmap()` services rather than `malloc()` and `free()`, which may affect the memory limits for the process or the system. Previously, the Source Server deallocated memory for older journal files, but `malloc()` and `free()` rely on a system heap implementation that can interfere with attempts to return memory from the process to the operating system. (GTM-9141) 🟡
- MUPIP CREATE gives database files permissions based on the `umask` setting of the target directory. Previously new database files always took 0666 as their permission regardless `umask` setting. (GTM-9272) 🟡
- GT.M processes started at the same time as a REORG ENCRYPT correctly load the new encryption key. Previously, processes started at the same time as REORG ENCRYPT could fail to acquire the new key and exit with a CRYPTOPFAILED error message. This issue was only seen in development and not reported by a customer. (GTM-9839)
- Both the Source and Receiver Servers include a timestamp in any GETADDRINFO error messages and the Receiver Server follows that message with a REPLCOMM message. Previously, neither Server included a timestamp in the GETADDRINFO error message, and the Receiver Server omitted the REPLCOMM message. (GTM-10081) 🟢
- On replication shutdown GT.M allows sufficient time for all processes to terminate before attempting to clean up shared IPC resources. Previously running peer processes could prevent the resources from being released. This was seen only in internal testing and was not reported in production. Also for replication retries GT.M takes care to check the retry interval. Previously, if an operating system sleep operation returned slightly before expected, GT.M might delay connection retries. This was only seen in internal testing and not reported by any customer. (GTM-10587)
- MUPIP REORG correctly handles concurrent reduction in the depth of a global tree caused by concurrent REORGs or full-global kills. Previously, in rare cases where this situation occurred in a small window, REORG could terminate unexpectedly due to a GVFAIL error or segmentation violation (SIG-11) without any database damage. This issue was only discovered in testing and not reported by any customer. (GTM-10670)
- MUPIP INTEG appropriately manages reporting details of an error in a database block. Previously, INTEG would overwrite between one (1) and four (4) bytes beyond the end of the message buffer when preparing such an error message. This issue was only discovered in testing and not reported by any customer; we not been unable to identify any user-visible symptoms or any security exploit associated with this issue. (GTM-10684)
- GDE and MUPIP report a more appropriate maximum value for journal file extension. This does not change the way the extension logic works, only the value reported by the GDE VALTOOBIG or MUPIP JNLINEXT messages. Previously the messages reported a maximum that had no relationship with the maximum size of a journal file. (GTM-10687)

This page is intentionally left blank.

Other

- MLKHASHRESIZE, MLKHASHRESIZEFAIL and MLKREHASH messages include region and database file names. Previously, these error messages did not provide the region and database file names. (GTM-9653)
- GT.M appropriately handles a high rate of MUPIP interrupts. Previously, under rare circumstances where a process received many MUPIP INTRPTs in rapid succession, the process could fail with an GTMASSERT error. This issue was only seen in development and not reported by any customer. (GTM-9775)
- GT.M appropriately manages processing of asynchronous events, such as MUPIP INTRPT. Previously, when GT.M handled such events it used certain non-reentrant system calls, which, if those calls were coincidentally already in use, could in rare cases lead to memory corruption, resulting in bad behavior typically a missed event or a segmentation violation (SIG-11). (GTM-9812)
- Processes in Direct Mode handle local variables appropriately when interrupted by SIGINT or SIGUSR1 signals. Previously, processing such signals in Direct Mode leaked stack space, potentially causing STACKCRIT and/or STACKOFLOW errors. (GTM-9815)
- GT.M properly handles the effects of advanced optimization under newer C compilers during asynchronous signal handling. This issue does not affect any known distributed binaries of GT.M, but can affect versions of GT.M compiled from source with compiler optimizations enabled. The issue can cause database corruption, application-level inconsistency in the form of partially-complete transactions, and hangs. Users of previous versions who compiled from source with compiler optimizations enabled can reduce the likelihood of encountering this issue by increasing the database flush timer. This issue was found in development and was never reported by any customer. (GTM-9831) 🍷
- Eliminate a rare issue with asynchronous events; previously such an event at in a window that widened in V7.1-003, could cause a process termination with a SIG6. (GTM-10586)
- GT.M more reliably returns physical and virtual memory used to manage its heap to the operating system, using the mmap() and munmap() services rather than malloc() and free(). The change may affect the memory limits for the system. Previously, GT.M deallocated memory when its heap needed to expand, but used malloc() and free() which rely on a system heap implementation that can interfere with attempts to return memory to the operating system. (GTM-10588) 🍷
- GT.M protects itself against a case where a timer could expire in a vulnerable window of several instructions and cause a process performing a journal operation to loop indefinitely. This issue did not compromise database integrity. Recovery required shutting down processing and restarting journaling. (GTM-10591)
- GT.M protects itself against a rare case where a MUPIP STOP or other process kills could occur in a vulnerable window of a handful of instructions and create an inconsistent state in the journal buffer which could cause GT.M processes to loop indefinitely while trying commit an update. This issue did not compromise database integrity. (GTM-10597)

Other

- GT.M provides protection from asynchronous events during small windows performing state changes to LOCK shared memory. Previously, asynchronous events like MUPIP STOP at inopportune times could leave LOCK shared memory in an out-of-design state, causing subsequent attempts at using LOCKs to terminate with a GTMASSERT2. The workaround was to close and restart the instance. (GTM-10615)

Error and Other Messages

MLKHASHRESIZE

MLKHASHRESIZE, LOCK hash table for region rrrr (dddd) increased in size from aaaa to bbbb and placed in shared memory (id = mmmm)

Operator log Information: GT.M needed to expand a hash table for region rrrr mapped to database file dddd used for managing LOCK information.

Action: No user action is required, but shared memory monitoring will show an additional shared memory segment with id mmmm.

MLKHASHRESIZEFAIL

MLKHASHRESIZEFAIL, Failed to increase LOCK hash table for region rrrr (dddd) size from aaaa to bbbb. Will retry with larger size.

Operator log Warning: GT.M needed to expand a hash table for region rrrr mapped to database file dddd used for managing LOCK information needed to be expanded, but the initial attempt failed, necessitating a retry.

Action: A subsequent MLKHASHRESIZE indicates that the retry succeeded and no user action is required.

MLKREHASH

MLKREHASH, LOCK hash table rebuilt for region rrrr (dddd) (seed = ssss)

Run Time Information: GT.M has detected an issue with the LOCK hash table for region rrrr mapped to database file dddd and regenerated it using a new seed value ssss.

Action: This information message confirms the success of the rehash operation.

MUTEXLCKALERT

MUTEXLCKALERT, Mutual Exclusion subsystem ALERT - Lock attempt threshold crossed for region rrrr. Process pppp is in crit cycle cccc.

Run Time Error: This warning indicates that a process could not obtain a critical section lock for region rrrr even after waiting longer than the GT.M determined threshold (approximately 32 seconds) because the critical section lock was held that entire time by another process pppp. cccc is the crit cycle count which GT.M increases by one every time it successfully grants the mutual exclusion (mutex) lock to a process. cccc provides a measure of the frequency of mutex lock use. MUTEXLCKALERT messages indicate that process pppp is blocking access to region rrrr for inappropriately long periods of time and thereby impacting performance for other processes needing access that region.

GT.M produces this warning when:

- A process owning a critical section dies (most likely because of a kill -9) and the OS gives its PID to another process. To reclaim the inappropriately held critical section, GT.M first checks whether the process is alive and whether it holds the critical section. On finding that the process is alive but does not hold the critical section, GT.M concludes that it is not safe to free the critical section and alerts the operator with this message.
- The process holding the critical section is using a non-Isolated command such as ZSYSTEM, BREAK or a timed command in a way that creates a deadlock or a live-lock. GT.M attempts to limit this by limiting the time a process using one of these commands can hold a critical section, but your use of non-Isolated commands and your settings for \$ZMAXTPTIM and / or the environment variable \$gtm_tpnocidtime may be such that you get MUTEXLCKALERT messages.
- There is an IO bottleneck that caused GT.M to slow down: GT.M detects that process pppp is currently using the critical section lock.



Note

GT.M blocks signals during MUTEXLCKALERT warnings. This means that GT.M defers error handling as a result of TP timeout (\$ZMAXTPTIME), interrupt handler invocation, \$ztimeout action, MUPIP STOP, etc. until the mutex is released. For example, a process may have a 10 seconds \$ZMAXTIMEOUT but GT.M may execute the error handler at a materially later time, until after the MUTEXLCKALERT condition has cleared.

Action: Monitor the system to determine whether there is a process with process id pppp and whether that process is a GT.M process.

Implement a script to get a stack trace for process pppp or take other appropriate action and use the \$gtm_prodstuckexec environment variable to activate it before the block process sends the MUTEXLCKALERT message.

Identify and terminate process pppp to release control of that resource. If the process is a GT.M process, use a MUPIP STOP to terminate it. If a process of another application, use an appropriate mechanism to stop it.

If this message is due to an IO bottleneck, adopt a strategy that reduces IO. Some of the IO reducing strategies are:

- Revisit your database configuration parameters (especially block size, number of global buffers, journal buffers, and so on) to see if you can make improvements.
- Create separate region (database) for temporary globals and do not replicate them.
- Consider whether a different database access method and journaling strategy could improve throughput while satisfying your operational needs.

- Consider tuning your filesystem
- For application configurations with large numbers of concurrent processes and/or large process memory footprints, consider placing object code in shared libraries on GT.M editions that support it. This may free system memory which the OS can use for its file system cache, or which you can use to increase the number of global buffers.



Do not apply IO reduction strategies all at once. Try them one at a time and always verify/measure the results of each strategy.

TPNOTACID

TPNOTACID, tttt at xxxx in a final TP retry violates ACID properties of a TRANSACTION and could exceed ssss milliseconds and rrrr RESTARTs have occurred

Run Time Error: GT.M issues this message as a warning to the syslog if it is executing a TP TRANSACTION in the final retry and control gets transferred out of GT.M due to any one of the following conditions:

- a ZEDIT or ZSYSTEM command
- Entering direct mode (e.g. due to a BREAK command)
- An external call
- a long running command (those which accept timeout specifications) encountered potentially indefinite restarts.

The xxxx indicates the \$ZPOSITION where the transfer of control occurred and the condition that caused this is identified in tttt.

GT.M issues this message as an error when it reaches the default limit of 16 or the limit setable by the gtm_tpnnotacidtries environment variable

Action: Review your code to determine whether the non-Isolated commands can be moved outside of transaction encapsulation. Alternatively, ensure that they are minimally disruptive by using \$ZMAXTPTIME to prevent transactions from running unreasonably long times and setting gtm_tpnnotacidtime to specify the wait period for long running commands before GT.M logs a TPNOTACID message.

This page is intentionally left blank.