GT.M

Release Notes

V6.3-002

Empowering
the Financial World

FIS

Contact Information

GT.M Group
Fidelity National Information Services, Inc.
200 Campus Drive
Collegeville, PA 19426
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
Website: http://fis-gtm.com

Legal Notice

Revision History

Revision 1.3	04 July 2022	Added GTM-5150 and GTM-8712.
Revision 1.2	5 February 2019	Updated the Platforms section to add AIX 7.1 TL 4 and AIX 7.2 as supported versions; Correct the maximum V6 database size.
Revision 1.1	11 September 2017	• In Platforms, removed the paragraph on recompiling the reference implementation of the encryption plugin. • Made content improvements for GTM-8740 and GTM-8415.

		• Specified that: - GTM-8436 is a performance enhancement for applications that are limited by journal writes with BG access method. - GTM-5250 includes JOB, LOCK, OPEN, READ, and $gtm_tpnotacidtime that support fractional timeout values to millisecond precision.
Revision 1.0	21 August 2017	V6.3-002

This page is intentionally left blank.

Table of Contents

This page is intentionally left blank.

V6.3-002

Overview

V6.3-002 includes multiple optimizations for performance, some applicable to all platforms.

GT.M now fully supports asynchronous IO as an option for databases using the BG access method. Unlike traditional database IO, which performs synchronous IO through the file system cache, asynchronous IO bypasses the file system cache. The performance characteristics of asynchronous IO are likely to be quite different from traditional sequential IO. Although asynchronous IO in theory should be more efficient than synchronous IO by eliminating the need for the UNIX file buffer cache and thereby eliminating certain file system locks (e.g., file systems mounted with AIX's CIO mount option), in practice asynchronous IO is likely to emerge from the starting gate under-performing synchronous IO because of the years that synchronous IO has been the common IO model operating systems and file systems have had used by applications. Please anticipate extensive benchmarking and tuning for your application to achieve the best performance it can with asynchronous IO. For more information, refer to GTM-6838 in the V6.3-001A Release Notes.

GT.M provides a mechanism to restrict certain facilities (GTM-8694). This release and going forward, the release includes a kit for installing a gtmpcat that matches the release.

As always, the release bring numerous smaller enhancements, and fixes. See the Change History below. Please pay special attention to the items marked with the symbols ✅ or ❗

Note

Messages are not part of the GT.M API whose stability we strive to maintain. Make sure that you review any automated scripting that parses GT.M messages.

Conventions

This document uses the following conventions:

Flag/Qualifiers	-
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 is used to track software enhancements and support requests. It is enclosed between parentheses ().
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 on x86 (32- and 64-bits).

The following table summarizes the new and revised replication terminology and qualifiers.

Pre V5.5-000 terminology	Pre V5.5-000 qualifier	Current terminology	Current qualifiers
originating instance or primary instance	-rootprimary	originating instance or originating primary instance. Within the context of a replication connection between two instances, an originating instance is referred to as source instance or source side. For example, in an B<-A->C replication configuration, A is the source instance for B and C.	-updok (recommended) -rootprimary (still accepted)
replicating instance (or secondary instance) and propagating instance	N/A for replicating instance or secondary instance. -propagateprimary for propagating instance	replicating instance. Within the context of a replication connection between two instances, a replicating instance that receives updates from a source instance is referred to as receiving instance or receiver side. For example, in an B<-A->C replication configuration, both B and C can be referred to as a receiving instance.	-updnotok
N/A	N/A	supplementary instance. For example, in an A->P->Q replication configuration, P is the supplementary instance. Both A and P are originating instances.	-updok

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 (the combination of operating system and hardware architecture is referred to as a platform). This set of specific versions is considered 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. This larger set of versions is considered 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. These are considered 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 plugin has its own additional requirements, should you opt to use it as included with GT.M.

Platform	Supported Versions	Notes
IBM Power Systems AIX	7.1 TL 4, 7.2	Only 64-bit versions of AIX with POWER7 as the minimum required CPU architecture level are Supported. 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. 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.** AIX 7.1 TL 5 is Supportable.

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 7.3; Ubuntu 16.04 LTS	To run 64-bit GT.M processes requires both a 64-bit kernel as well as 64-bit hardware. 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). GT.M requires the libtinfo library. If it is not already installed on your system, and is available using the package manager, install it using the package manager. If a libtinfo package is not available: ● Find the directory where libncurses.so is installed on your system. ● Change to that directory and make a symbolic link to libncurses.so.<ver> from libtinfo.so.<ver>. Note that some of the libncurses.so entries may themselves be symbolic links, for example, libncurses.so.5 may itself be a symbolic link to libncurses.so.5.9. 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. Although GT.M itself does not require libelf, the geteuid program used by the GT.M installation script requires libelf (packaged as libelf1 on current Debian/Ubuntu distributions and elfutils-libelf on RHEL 6 & 7). 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 ⊠⊠⊠ 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.

Platform	Supported Versions	Notes
x86 GNU/Linux	Debian 9 (Stretch)	This 32-bit version of GT.M runs on either 32- or 64-bit x86 platforms; we expect the x86_64 GNU/Linux version of GT.M to be preferable on 64-bit hardware. Running a 32-bit GT.M on a 64-bit GNU/Linux requires 32-bit libraries to be installed. The CPU must have an instruction set equivalent to 586 (Pentium) or better. Please also refer to the notes above on x86_64 GNU/Linux.

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. GT.M releases are also normally supported for two years after release. While FIS will attempt to provide support to customers in good standing for any GT.M release and operating system version, our ability to provide support diminishes after the two year window.

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

32- vs. 64-bit platforms

The same application code runs on both 32-bit and 64-bit platforms; however there are operational differences between them (for example, auto-relink and the ability to use GT.M object code from shared libraries exist only on 64-bit platforms). Please note that:

- You must compile the application code separately for each platform. Even though the M source code is the same, the generated object modules are different - the object code differs between x86 and x86_64.
- Parameter-types that interface GT.M with non-M code using C calling conventions must match the data-types on their target platforms. Mostly, these parameters are for call-ins, external calls, internationalization (collation) and environment translation, and are listed in the tables below. Note that most addresses on 64-bit platforms are 8 bytes long and require 8 byte alignment in structures whereas all addresses on 32-bit platforms are 4 bytes long and require 4-byte alignment in structures.

Call-ins and External Calls

Parameter type	32-Bit	64-bit	Remarks
gtm_long_t	4-byte (32-bit)	8-byte (64-bit)	gtm_long_t is much the same as the C language long type.
gtm_ulong_t	4-byte	8-byte	gtm_ulong_t is much the same as the C language unsigned long type.

Parameter type	32-Bit	64-bit	Remarks
gtm_int_t	4-byte	4-byte	gtm_int_t has 32-bit length on all platforms.
gtm_uint_t	4-byte	4-byte	gtm_uint_t has 32-bit length on all platforms

Caution

If your interface uses gtm_long_t or gtm_ulong_t types but your interface code uses int or signed int types, failure to revise the types so they match on a 64-bit platform will cause the code to fail in unpleasant, potentially dangerous, and hard to diagnose ways.

Internationalization (Collation)

Parameter type	32-Bit	64-bit	Remarks
gtm_descriptor in gtm_descript.h	4-byte	8-byte	Although it is only the address within these types that changes, the structures may grow by up to 8 bytes as a result of compiler padding to meet platform alignment requirements.

Important

Assuming other aspects of code are 64-bit capable, collation routines should require only recompilation.

Environment Translation

Parameter type	32-Bit	64-bit	Remarks
gtm_string_t type in gtmxc_types.h	4-byte	8-byte	Although it is only the address within these types that changes, the structures may grow by up to 8 bytes as a result of compiler padding to meet platform alignment requirements.

Important

Assuming other aspects of code are 64-bit capable, environment translation routines should require only recompilation.

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 V6.3-002 in a Filesystem Hierarchy Standard compliant location such as /usr/lib/fis-gtm/V6.3-002_arch (for example, /usr/lib/fis-gtm/V6.3-002_x86 on 32-bit Linux systems). A location such as /opt/fis-gtm/V6.3-002_arch would also be appropriate. Note that the ***arch*** suffix is especially important if you plan to install 32- and 64-bit versions of the same release of GT.M on the same system.

- 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 database encryption and TLS replication, you must compile the reference implementation plugin to match the shared library dependencies unique 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.

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

Caution

There are separate steps to compile the encryption plugin for GT.M versions V5.3-004 through V6.3-000 when OpenSSL 1.1 is installed and OpenSSL 1.0.x libraries arestill available.

- Download the most recent OpenSSL 1.0.x version
- Compile and install (default installs to /usr/local/ssl)

```
./config && make install
```

- Adjust the configuration : Move the newly installed libraries out of the way

```
mv /usr/local/ssl/lib /usr/local/ssl/lib.donotuse
```

- Adjust the configuration : Create another /usr/local/ssl/lib and symlink the existing 1.0.x library into it as the default. This ensures that the encryption plugin is compiled using the compatible OpenSSL 1.0.x library. Adjust the path below as necessary.

```
mkdir /usr/local/ssl/lib && ln -s /path/to/existing/libssl.so.1.0.x /usr/
local/ssl/libssl.so
```

- Recompile the encryption plugin following existing directions above
- Remove /usr/local/ssl to avoid future complications

Upgrading to GT.M V6.3-002

The GT.M database consists of four types of components- database files, journal files, global directories, and replication instance files. The format of some database components differs for 32-bit and 64-bit GT.M releases for the x86 GNU/Linux platform.

GT.M upgrade procedure for V6.3-002 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

Read the upgrade instructions of each stage carefully. Your upgrade procedure for GT.M V6.3-002 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 V6.3-002.

- Execute the EXIT command. This command automatically upgrades the Global Directory.

To switch between 32- and 64-bit global directories on the x86 GNU/Linux platform:

1. Open your Global Directory with the GDE utility program on the 32-bit platform.
2. On GT.M versions that support SHOW -COMMAND, execute SHOW -COMMAND -FILE=file-name. This command stores the current Global Directory settings in the specified file.
3. On GT.M versions that do not support GDE SHOW -COMMAND, execute the SHOW -ALL command. Use the information from the output to create an appropriate command file or use it as a guide to manually enter commands in GDE.
4. Open GDE on the 64-bit platform. If you have a command file from 2. or 3., execute @file-name and then run the EXIT command. These commands automatically create the Global Directory. Otherwise use the GDE output from the old Global Directory and apply the settings in the new environment.

An analogous procedure applies in the reverse direction.

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 V6.3-002.
- 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.

Stage 2: Database Files Upgrade

To upgrade from GT.M V5.0*/V5.1*/V5.2*/V5.3*/V5.4*/V5.5:

A V6 database file is a superset of a V5 database file and has potentially longer keys and records. Therefore, upgrading a database file requires no explicit procedure. After upgrading the Global Directory, opening a V5 database with a V6 process automatically upgrades fields in the database fileheader.

A database created with V6 supports up to 992Mi blocks and is not backward compatible. V6 databases that take advantage of V6 limits on key size and records size cannot be downgraded. Use MUPIP DOWNGRADE -VERSION=V5 to downgrade a V6 database back to V5 format provided it meets the database downgrade requirements. For more information on downgrading a database, refer to Downgrading to V5 or V4.

Important

A V5 database that has been automatically upgraded to V6 can perform all GT.M V6.3-002 operations. However, that database can only grow to the maximum size of the version in which it was originally created. A database created on V5.0-000 through V5.3-003 has maximum size of 128Mi blocks. A database created on V5.4-000 through V5.5-000 has a maximum size of 224Mi blocks. A database file created with V6.0-000 (or above) can grow up to a maximum of 992Mi blocks. This means that, for example, the maximum size of a V6 database file having 8KiB block size is 7936GiB (8KiB*992Mi).

Important

In order to perform a database downgrade you must perform a MUPIP INTEG -NOONLINE. If the duration of the MUPIP INTEG exceeds the time allotted for an upgrade you should rely on a rolling upgrade scheme using replication.

If your database has any previously used but free blocks from an earlier upgrade cycle (V4 to V5), you may need to execute the MUPIP REORG -UPGRADE command. If you have already executed the MUPIP REORG -UPGRADE command in a version prior to V5.3-003 and if subsequent versions cannot determine whether MUPIP REORG -UPGRADE performed all required actions, it sends warnings to the syslog requesting another run of MUPIP REORG -UPGRADE. In that case, perform any one of the following steps:

- Execute the MUPIP REORG -UPGRADE command again, or
- Execute the DSE CHANGE -FILEHEADER -FULLY_UPGRADED=1 command to stop the warnings.

Caution

Do not run the DSE CHANGE -FILEHEADER -FULLY_UPGRADED=1 command unless you are absolutely sure of having previously run a MUPIP REORG -UPGRADE from V5.3-003 or later. An inappropriate DSE CHANGE -FILEHEADE -FULLY_UPGRADED=1 may lead to database integrity issues.

You do not need to run MUPIP REORG -UPGRADE on:

- A database that was created by a V5 MUPIP CREATE
- A database that has been completely processed by a MUPIP REORG -UPGRADE from V5.3-003 or later.

For additional upgrade considerations, refer to Database Compatibility Notes.

To upgrade from a GT.M version prior to V5.000:

You need to upgrade your database files only when there is a block format upgrade from V4 to V5. However, some versions, for example, database files which have been initially been created with V4

(and subsequently upgraded to a V5 format) may additionally need a MUPIP REORG -UPGRADE operation to upgrade previously used but free blocks that may have been missed by earlier upgrade tools.

- Upgrade your database files using in-place or traditional database upgrade procedure depending on your situation. For more information on in-place/traditional database upgrade, see Database Migration Technical Bulletin.
- Run the MUPIP REORG -UPGRADE command. This command upgrades all V4 blocks to V5 format.

Note

Databases created with GT.M releases prior to V5.0-000 and upgraded to a V5 format retain the maximum size limit of 64Mi (67,108,864) blocks.

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 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 or above have a maximum size of 1,040,187,392(992Mi) blocks.
- For information on downgrading a database upgraded from V6 to V5, refer to: Downgrading to V5 or V4.

Stage 3: Replication Instance File Upgrade

V6.3-002 does not require new replication instance files if you are upgrading from V5.5-000. However, V6.3-002 requires new replication instance files if you are upgrading from any version prior to V5.5-000. Instructions for creating new replication instance files are in the Database Replication chapter of the GT.M Administration and Operations Guide. Shut down all Receiver Servers on other instances that are to receive updates from this instance, shut down this instance Source Server(s), recreate the

instance file, restart the Source Server(s) and then restart any Receiver Server for this instance with the -UPDATERESYNC qualifier.

Note

Without the -UPDATERESYNC qualifier, the replicating instance synchronizes with the originating instance using state information from both instances and potentially rolling back information on the replicating instance. The -UPDATERESYNC qualifier declares the replicating instance to be in a wholesome state matching some prior (or current) state of the originating instance; it causes MUPIP to update the information in the replication instance file of the originating instance and not modify information currently in the database on the replicating instance. After this command, the replicating instance catches up to the originating instance starting from its own current state. Use -UPDATERESYNC only when you are absolutely certain that the replicating instance database was shut down normally with no errors, or appropriately copied from another instance with no errors.

Important

You must always follow the steps described in the Database Replication chapter of the GT.M Administration and Operations Guide when migrating from a logical dual site (LDS) configuration to an LMS configuration, even if you are not changing GT.M releases.

Stage 4: Journal Files Upgrade

On every GT.M upgrade:

- Create a fresh backup of your database.
- Generate new journal files (without back-links).

Important

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

Stage 5: Trigger Definitions Upgrade

If you are upgrading from V5.4-002A/V5.4-002B/V5.5-000 to V6.3-002 and you have database triggers defined in V6.2-000 or earlier, you need to ensure that your trigger definitions are wholesome in the older version and then run MUPIP TRIGGER -UPGRADE. If you have doubts about the wholesomeness of the trigger definitions in the old version use the instructions below to capture the definitions delete

them in the old version (-*), run MUPIP TRIGGER -UPGRADE in V6.3-002 and then reload them as described below.

You need to extract and reload your trigger definitions only if you are upgrading from V5.4-000/V5.4-000A/V5.4-001 to V6.3-002 or if you find your prior version trigger definitions have problems. For versions V5.4-000/V5.4-000A/V5.4-001 this is necessary because multi-line XECUTEs for triggers require a different internal storage format for triggers which makes triggers created in V5.4-000/V5.4-000A/V5.4-001 incompatible with V5.4-002/V5.4-002A/V5.4-002B/V5.5-000/V6.0-000/V6.0-001/V6.3-002.

To extract and reapply the trigger definitions on V6.3-002 using MUPIP TRIGGER:

1. Using the old version, execute a command like **mupip trigger -select="*" trigger_defs.trg**. Now, the output file trigger_defs.trg contains all trigger definitions.
2. Place -* at the beginning of the trigger_defs.trg file to remove the old trigger definitions.
3. Using V6.3-002, run **mupip trigger -triggerfile=trigger_defs.trg** to reload your trigger definitions.

To extract and reload trigger definitions on a V6.3-002 replicating instance using $ZTRIGGER():

1. Shut down the instance using the old version of GT.M.
2. Execute a command like **mumps -run %XCMD 'i $ztrigger("select")' > trigger_defs.trg** . Now, the output file trigger_defs.trg contains all trigger definitions.
3. Turn off replication on all regions.
4. Run **mumps -run %XCMD 'i $ztrigger("item","-*")** to remove the old trigger definitions.
5. Perform the upgrade procedure applicable for V6.3-002.
6. Run **mumps -run %XCMD 'if $ztrigger("file","trigger_defs.trg")'** to reapply your trigger definitions.
7. Turn replication on.
8. Connect to the originating instance.

Note

Reloading triggers renumbers automatically generated trigger names.

Downgrading to V5 or V4

You can downgrade a GT.M V6 database to V5 or V4 format using MUPIP DOWNGRADE.

Starting with V6.0-000, MUPIP DOWNGRADE supports the -VERSION qualifier with the following format:

```
MUPIP DOWNGRADE -VERSION=[V5|V4]
```

-VERSION specifies the desired version for the database header.

To qualify for a downgrade from V6 to V5, your database must meet the following requirements:

1. The database was created with a major version no greater than the target version.
2. The database does not contain any records that exceed the block size (spanning nodes).
3. The sizes of all the keys in database are less than 256 bytes.
4. There are no keys present in database with size greater than the Maximum-Key-Size specification in the database header, that is, Maximum-Key-Size is assured.
5. The maximum Record size is small enough to accommodate key, overhead, and value within a block.

To verify that your database meets all of the above requirements, execute MUPIP INTEG -NOONLINE. Note that the integrity check requires the use of -NOONLINE to ensure no concurrent updates invalidate the above requirements. Once assured that your database meets all the above requirements, MUPIP DOWNGRADE -VERSION=V5 resets the database header to V5 elements which makes it compatible with V5 versions.

To qualify for a downgrade from V6 to V4, your database must meet the same downgrade requirements that are there for downgrading from V6 to V5.

If your database meets the downgrade requirements, perform the following steps to downgrade to V4:

1. In a GT.M V6.3-002 environment:
   a. Execute MUPIP SET -VERSION=v4 so that GT.M writes updates blocks in V4 format.
   b. Execute MUPIP REORG -DOWNGRADE to convert all blocks from V6 format to V4 format.
2. Bring down all V6 GT.M processes and execute MUPIP RUNDOWN -FILE on each database file to ensure that there are no processes accessing the database files.
3. Execute MUPIP DOWNGRADE -VERSION=V4 to change the database file header from V6 to V4.
4. Restore or recreate all the V4 global directory files.
5. Your database is now successfully downgraded to V4.

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. Note that on 64-bit versions of GT.M, the object code is in shared libraries, rather than 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_V6.3-002_i686, then gtmprofile sets $gtm_dist to /usr/lib/fis-gtm/gtm_V6.3-002_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(smkx), 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.

This page is intentionally left blank.

Change History

V6.3-002

Fixes and enhancements specific to V6.3-002:

Id	Prior Id	Category	Summary
GTM-4851	S9D02-002298	Language	Uniform limit of 8192 on source lines
GTM-5150	-	Language	Evaluate timeout values as numeric to millisecond precision
GTM-5178	S9D11-002392	Language	Make BLKTOODEEP errors warnings, and catch all known cases
GTM-5250	S9E01-002413	Language	GT.M treats timeout values as having millisecond precision
GTM-5754	C9F08-002749	Language	When possible, compile XECUTE literal in stream
GTM-6657	C9K07-003302	Admin	MUPIP BACKUP -BKUPDBJNL=OFF leaves journal disabled regions disabled
GTM-7638	-	Admin	Please see GTM-8694
GTM-8165	-	Language	TPNOTACID checking more comprehensive and finer grained gtm_tpnotacidtime
GTM-8281	-	Language	Please see GTM-4851
GTM-8415	-	Other	Protect UNIX user information against possible getpwuid() invocation from an eternal call
GTM-8436	-	DB	Move some journaling logic from under a broad resource to a more granular one
GTM-8543	-	Admin	All journal file creation, renaming and deletion operations sent to the syslog
GTM-8616	-	Admin	MUPIP RUNDOWN logs additional information on its actions
GTM-8644	-	Language	More straightforward shell management and quoting for ZSYSTEM arguments

Id	Prior Id	Category	Summary
GTM-8663	-	Other	Protect MUPIP JOURNAL -PARALLEL better against a MUPIP STOP
GTM-8690	-	DB	Potentially speed up process startup by deferring encryption initialization
GTM-8694	-	Admin	Optional operational Restrictions on certain GT.M facilities
GTM-8698	-	DB	Better error handling for autoDB and statistics databases
GTM-8704	-	Admin	GT.M installation script accepts both new (49.1) and old (4.9.1) format of ICU version
GTM-8708	-	DB	Avoid unnecessary Epochs
GTM-8711	-	Admin	GDE sets the shell exit status when invoked from the SHELL
GTM-8712	-	Admin	Journal recovery clears semaphore and shared memory IDs
GTM-8713	-	Other	DSE CACHE -VERIFY fix to avoid inaccuate report
GTM-8714	-	Admin	Installation script guards against systemd removing shared IPC resources
GTM-8717	-	Language	Prevent $SELECT() compilation issues from terminating the process
GTM-8718	-	Language	Appropriate handling of an attempt to SET an invalid value into $ZROUTINES
GTM-8720	-	Language	Fix odd case of SET @ (indirect) involving $INCREMENT()
GTM-8724	-	Other	Prevent a rare SIGBUS when using auto-relink
GTM-8726	-	Admin	gtmpcat release kit
GTM-8727	-	Language	Better behavior in NOUNDEF mode for naked references
GTM-8730	-	Other	Handle replication startup after full ROLLBACK
GTM-8733	-	Language	$ZCONVERT() does not give BADCHAR errors in NOBADCHAR mode

Id	Prior Id	Category	Summary
GTM-8736	-	Language	Make default $ZOUTINES more explicit
GTM-8737	-	Admin	MUPIP JOURNAL -SHOW prints the characters representing the first 12 bytes of the userid and 16 bytes of the node name.
GTM-8738	-	Language	Limits on nesting of LOCK actions using extrinsic functions
GTM-8740	-	Admin	MUPIP display of whether Instance Freeze is enabled; allow a Source Server start to load the custom errors file if not already loaded
GTM-8742	-	Admin	Prevent misdirected journal updates by MUPIP JOURNAL
GTM-8743	-	Language	Fix to optimization of nested literals in Boolean expressions
GTM-8745	-	Other	Distribution build platform Updates
GTM-8753	-	Admin	Support for full database block writes of newly allocated blocks
GTM-8755	-	DB	Additional DBIOERR error message diagnostic information
GTM-8756	-	Admin	GT.M installation script is compatible with the AIX default ICU libraries
GTM-8759	-	DB	MUPIP BACKUP ONLINE produces an integrity error free comprehensive backup
GTM-8760	-	Language	Improved support for large environment variable values
GTM-8763	-	Admin	Multi-Process Forward Recovery Fix
GTM-8766	-	Admin	GDE and MUPIP SET produce appropriate errors when value being set is out of range
GTM-8771	-	Admin	Database file extensions with -NODEFER_ALLOCATE modified to accommodate xfs peculiarities

This page is intentionally left blank.

Database

- As a potential performance enhancement for applications that are limited by journal writes, with the BG access method, GT.M writes journal and replication information while holding a more granular resource, that is: after releasing control of the region, but while it still has control of the block associated with the update. This change does not apply to MM access method processing . Previously GT.M updated the journal and replication streams while holding the resource associated with the region. (GTM-8436)

- When it can, GT.M takes a lightweight approach to initializing encryption resources. Previously many processes concurrently attaching to an encrypted database could cause some of them to exit with a SEMWT2LONG error. (GTM-8690)

- When GT.M encounters an error creating an autoDB database (including those used to store statistics), it issues a MUCREFILERR which identifies the application code entryref where it encountered the issue followed by the underlying error. Previously the it only supplied the underlying error with no context as to where it occurred. In addition, if GT.M encounters an error running down a statistics database, it reports the error to the syslog; previously reported to the user's device. (GTM-8698)

- GT.M avoids premature epochs after a period of no updates. In particular, the update process writer helper avoids retrying the epoch repeatedly. (GTM-8708)

- A DBIOERR error message displays additional diagnostic information when it is available. Previously, it did not display some available error context. (GTM-8755)

- MUPIP BACKUP -ONLINE produces an integrity error free backup even after inappropriate killing of MUMPS processes. Previously under rare circumstances, killing MUMPS processes while an online backup was in progress could produce a database file with DBTNTOOLONG integrity errors. This issue was only observed in the GT.M development environment, and was never reported by a user. (GTM-8759)

This page is intentionally left blank.

Language

- GT.M source lines, XECUTE strings and Direct Mode input all accept up to 8192 byte values. In the case of source, the length includes the terminator(s). In the case of Direct Mode, the WIDTH and/or WRAP deviceparameter characteristics may cause a lower limit. Previously the limit for source lines was 8191, and 1024 or less for the other two. (GTM-8281)(GTM-4851)
- GT.M evaluates timeout values as numeric to millisecond precision; previously it evaluated them as integers, truncating any decimal fraction. (GTM-5150).
- GT.M reports BLKTOODEEP errors as warnings, including FOR lines with multiple leading dots (levels) underneath an undotted line (current level of zero). These new reports may identify latent issues in existing code. Previously GT.M reported some BLKTOODEEP cases as errors but did not report lines with multiple dots when the current level was zero, but rather silently ignored such lines. (GTM-5178)
- For JOB, LOCK, OPEN and READ commands, and $gtm_tpnotacidtime, GT.M supports fractional timeout values to millisecond precision with timeout values capped at 999,999.999 seconds (about 11.5 days); higher timeout values execute with the capped value. Previously GT.M supported timeouts as integers (GTM-5250)
- GT.M compiles XECUTE <literal> at compile time when the literal is valid GT.M code that has minimal impact on the M virtual machine. An XECUTE literal containing GOTO, NEW, QUIT, (nested) XECUTE and indirection cannot be precompiled because of the interaction of those features with the stack architecture of the M virtual machine. Precompiled XECUTE literals do not show up in $STACK() as having a separate stack level, but rather "disappear" into the stack level of the original XECUTE. Please observe the following cautions: ensure you compile with the same GT.M version, $gtm_chset, $gtm_local_collate, $gtm_patnumeric, $gtm_pattern_file and $gtm_pattern_table values (or lack thereof) as those used to run your application, and, if the application changes the run time values controlled by those environment variables, use variable operands or indirection, rather than literals for operands with pattern match (?) or sorts-after (]]). While we do not expect much impact on existing code, it is possible that an application that makes unusual use of XECUTE might require some rework. Note that indirection almost always performs better than an XECUTE that cannot be precompiled. Note also that adding a QUIT at the end of an XECUTE that does not contain a FOR will leave it for run time compilation. Previously an XECUTE that had not recently been used was always compiled at run-time. (GTM-5754)
- GT.M releases the database critical section and reports TPNOTACID for timed WRITE /* operations subject to timeouts that exceed gtm_tpnotacidtime when $TRESTART>2. Also, gtm_tpnotacidtime recognizes millisecond fractions of seconds; previously the value was truncated to whole seconds. FIS strongly recommends avoiding coding that violates the ACID transaction properties, particularly Isolation. Previously non-Isolated WRITE /* operations could allow a process to hold the database critical section for an indefinite period. In addition, TPNOTACID errors report the timeout; previously they did not. (GTM-8165)
- Addressed by GTM-4851 (GTM-8281)

- The ZSYSTEM command always invokes shell specified by the SHELL environment variable; previously it was possible to construct arguments that did not. In addition, ZSYSTEM arguments handle quoting (both single and double) in a more straightforward way. This may require changes to existing code to simplify quoting. An example where a change may be needed:

```
GTM> zsystem "echo '""hello world""'"
hello world
```

  will now produce:

```
GTM> zsystem "echo '""hello world""'"
"hello world"
```

  and should be changed to:

```
GTM> zsystem "echo 'hello world'"
hello world
```

  to get the same output. (GTM-8644)

- $SELECT() appropriately handles an argument where a truthvalue made up of literals precedes a syntax error, as well as arguments where a literal FALSE truthvalue conditionalizes a global reference which a later argument also references. Due to a compiler optimization in V6.3-001[A], such an error produced a segmentation violation (SIG-11), and the other case produced a GTMASSERT2. (GTM-8717)

- Attempting to set an invalid string into $ZROUTINES leaves the previous value of $ZROUTINES as it was. Previously, this could cause issues if auto-relink was in use, resulting in a segmentation violation (SIG-11). This issue was only observed in the GT.M development environment, and was never reported by a user. (GTM-8718)

- Using indirection in a setleft which uses $INCREMENT() to create an indirect source value on the right works appropriately. While this worked as expected with gtm_side_effects defined to a non-zero value, with a default or zero (0) setting of that compiler characteristic, due to a regression introduced in V6.0-001, an right hand expression with a side effect could cause an inappropriate order of evaluation, which typically manifested as misbehavior around undefined variables, either reporting UNDEF when it should not or not reporting UNDEF when it should. (GTM-8720)

- In NOUNDEF mode a naked reference using an undefined value as a subscript behaves appropriately; previously it could cause a GTMASSERT2 fatal error, or, in case of a name-level $ORDER(), return the first global name in the global directory. Note that, at least, syntax errors or UNDEF errors leave the naked reference at the point of the last valid subscript. While this is non-standard, it can be useful for debugging, would introduce inefficiencies to change, and has not, to date, generated any concerns from users. (GTM-8727)

- $ZCONVERT() operates appropriately in UTF-8 NOBADCHAR mode; previously it issued an error. As a consequence the pinentry script operates correctly. The pinentry issue was only observed in the GT.M development environment, and was never reported by a user. Note that defining the gtm_ztrap_form environment variable to the empty (null) string sets ZTRAP handling to adaptive,

while leaving it undefined sets the handling to "code" which can cause entryref device exception or ZTRAP handlers to fail with the error INDEXTRACHARs. (GTM-8733)

- If the $gtmroutines environment variable is not defined, GT.M defaults $ZROUTINES to "." - meaning the current working directory; previously in this case, GT.M defaulted $ZROUTINES to "" which had the same effect, but was not as clear. (GTM-8736)

- LOCK of an argument within a parenthetical list where the argument includes an extrinsic function that performs LOCK actions produces a BADLOCKNEST error except where there is only one such argument, it is the first argument in the list and the LOCK'ng as a consequence of the extrinsic function(s) is simple. Previously this construct could produce an unintended LOCK state or a segmentation violation (SIG-11). Note that this pattern may still produce some unintended outcomes, so FIS recommends against its use. This change also slightly modified a handful of LOCK-related error messages with the intention of improving their clarity. (GTM-8738)

- The GT.M compiler handles nested Boolean operations that include literal operands appropriately; this corrects a flaw in the Boolean optimizations introduced in V6.3-001[A] that, given certain patterns of multi-level nesting could produce an inappropriate UNIMPLOP error, and, in some cases, incorrect results. (GTM-8743)

- GT.M now properly handles environment variables whose contents is over 32K in size. GT.M reports an error if the content is larger than can fit in a local or global. Previously, environment variables whose contents was over 32K in size resulted in a SIG-11. This issue was only observed in the GT.M development environment,and was never reported by a user. (GTM-8760)

This page is intentionally left blank.

System Administration

- MUPIP BACKUP -BKUPDBJNL=OFF does not adjust the journaling state for a database with journaling disabled. Previously, -BKUPDBJNL=OFF turned off journaling in the backup database even when journaling was disabled, which required an extra step to explicitly reestablish the disabled journaling state characteristic for regions in the restored database that originally had journaling disabled. (GTM-6657)

- Addressed by GTM-8694 (GTM-7638)

- When creating, renaming and deleting journal files, GT.M processes send all status results to syslog correctly naming the files involved. Previously, only MUMPS processes sent this status information to syslog while MUPIP processes wrote the information to STDOUT. Additionally during a rename, GT.M creates an intermediate temporary file; previously, some messages did not include the intermediate file name. (GTM-8543)

- MUPIP RUNDOWN with no arguments logs a message in the syslog facility with information about its actions including the process ID, user ID, and its current working directory. This may be useful for investigating any case where RUNDOWN inappropriately removes a GT.M shared memory segment or semaphore (IPC). Previously RUNDOWN did not provide such information. (GTM-8616)

- Post installation, a system administrator can optionally add a restrict.txt file in $gtm_dist to restrict the use of certain GT.M facilities to a group-name. The owner and group for $gtm_dist/restrict.txt can be different from those used to install GT.M. The file may contain zero or more of the following case-insensitive lines in any order:

```
BREAK[:<group-name>]
ZBREAK[:<group-name>]
ZCMDLINE[:<group-name>]
ZEDIT[:<group-name>]
ZSYSTEM[:<group-name>]
CENABLE[:<group-name>]
PIPE_OPEN[:<group-name>]
DIRECT_MODE[:<group-name>]
DSE[:<group-name>]
TRIGGER_MOD[:<group-name>]
```

  If the file $gtm_dist/restrict.txt does not exist, GT.M does not restrict any facilities.

  Any non-empty lines that do not match the above format cause processes with read-only permission to behave as if they could not read the file, and GT.M enforces all restrictions.

  Restrictions apply as follows:

GT.M facility	Behavior
BREAK	GT.M ignores any break command

GT.M facility	Behavior
ZBREAK	any ZBREAK produces a RESTRICTEDOP error
ZCMDLINE	GT.M returns an empty string for all references to $ZCMDLINE
ZEDIT	any ZEDIT produces a RESTRICTEDOP error
ZSYSTEM	any ZSYSTEM produces a RESTRICTEDOP error
CENABLE	the process acts like $gtm_nocenable is TRUE and ignores any CENABLE deviceparameter
PIPE_OPEN	any OPEN of a PIPE device produces a RESTRICTEDOP error
DIRECT_MODE	**mumps -direct** terminates immediately with a RESTRICTEDOP error
DSE	terminates immediately with a RESTRICTEDOP error
TRIGGER_MOD	any $ZTRIGGER() or MUPIP TRIGGER that attempts a change or delete produces a RESTRICTEDOP error

If the file exists, a process:

- that has write authorization to restrict.txt has no restrictions
- that has no read access to restrict.txt is restricted from all facilities for which GT.M supports a restriction (currently the above list)
- that has read-only access to restrict.txt is restricted from any listed facility unless it is a member of the group specified in the optional group-id following the facility name

Note that restricting $ZCMDLINE prevents things like: **mumps -run %XCMD 'for read x xecute x'** which can act as substitutes for Direct Mode. (GTM-7638),(GTM-8694)

- GT.M installation script, gtminstall, accepts icu version input in the form of <major><minor>.<milli>.<micro>, which matches the convention and output of icu-config –-version, for ICU versions 49(4.9) and later. Previously gtminstall only accepted <major>.<minor>. (GTM-8704)
- When invoked from the shell, GDE appropriately maintains the return status; previously it did not. Also, the GT.M help facility handles an EOF on $PRINCIPAL; previously an EOF produced an internal error and generated an associated context dump file. (GTM-8711)
- MUPIP JOURNAL RECOVER clears semaphore and shared memory IDs from the DB fileheader as appropriate. Previously there was a small window between the MUPIP JOURNAL RECOVER running down the database and another GT.M process accessing the database which could result in a REQRUNDOWN error. This issue was only observed in the GT.M development environment, and was never reported by a user. (GTM-8712)
- If gtminstall finds the environment is a Linux installation using systemd, it prompts the user for permission to insert addRemoveIPC=no into /etc/systemd/logind.conf and restart logind; if the

user refuses, gtminstall stops the installation after issuing instructions on how to perform the task independently, and information on why it is necessary. (GTM-8714)

- GT.M releases include a gtmpcat_for_{release} kit which contains the files required to run gtmpcat against the released GT.M version. The install_gtmpcat.sh script included in the kit may be used to install the files in $gtm_dist/tools/gtmpcat. After installation, $gtm_dist/gtmpcat may be used to run gtmpcat. Note that the gtmpcat_for_{release} kit does not contain files necessary for analyzing other GT.M releases. Full gtmpcat releases contain these files for all supported production GT.M releases as of the gtmpcat release date. (GTM-8726)

- MUPIP JOURNAL -SHOW displays the characters representing the first 12 bytes of the userid and the first 16 bytes of the node name. Also, it does not print trailing spaces after the time-stamp. Previously it truncated userid to eight bytes, the node name to 12 bytes, and put 19 trailing spaces after the time-stamp. (GTM-8737)

- The command MUPIP REPLIC -SOURCE -JNLPOOL -SHOW displays whether the custom errors file is loaded. Previously the only mechanism to check this was via ^%PEEKBYNAME (available since V6.3-000) or $ZPEEK(). The following returns 0 when Instance Freeze is not enabled and 1 when it is:

```
$$^%PEEKBYNAME("jnlpool_ctl_struct.instfreeze_environ_inited")
```

  When no prior custom errors file has been previously loaded, a Source Server starting with the gtm_custom_errors environment variable pointing to a file loads the contents of the file and thus enables Instance Freeze without performing a complete shutdown. Changing a previously loaded set of custom errors requires starting from a complete shutdown state. Previously if the Source Server initializing the replication journal pool was started without the gtm_custom_errors environment variable defined, enabling Instance Freeze by loading custom errors required a restart from a complete shutdown state. (GTM-8740)

- GT.M does not update journals associated with an original database when working with a backup copy on the same system. V6.3-001A introduced an unintended change such that a MUPIP JOURNAL -{RECOVER|ROLLBACK} -FORWARD of the backup database marked the current journal of the original database as not current. This issue was only observed in the GT.M development environment, and was never reported by a user. (GTM-8742)

- When the environment variable gtm_fullblockwrites is set to two (2), a process writes all newly allocated database blocks in their entirety regardless of their actual valid contents. This relieves some file systems from tracking potentially significant fragments of unallocated space, and thus reduces file system metadata maintenance. When this value is set to one (1) a process writes entire file system blocks in their entirety regardless of their actual valid contents, on some file systems, this avoids reading in advance of most writes and thus reduces file system load and increases response time. Note that when the file system block size and the database block size are the same there is no difference between the settings of 1 and 2. If the environment variable is undefined or zero (0), a process writes only valid data. FIS advises using the same value for all processes. Previously GT.M only recognized values of FALSE (0) or TRUE (non-zero) and equivalent to 1. (GTM-8753)

- The GT.M installation script, configure, works appropriately on AIX configured to rely solely on the default ICU libraries. Previously, attempting to install GT.M with ICU support on AIX failed when it had only the default ICU libraries. (GTM-8756)

- MUPIP JOURNAL -{ROLLBACK|RECOVER} -FORWARD handles interrupts to worker processes correctly. Previously, a worker process which was interrupted immediately after being started attempted to perform cleanup operations on the database which should only be performed by the main process, which could lead to undefined behavior. This issue was only observed in the GT.M development environment, and was never reported by a user. (GTM-8763)

- GDE and MUPIP produce errors in response to an attempt to set global buffers higher than the maximum supported. Previously GDE accepted unsupported specifications and MUPIP could give a segmentation violation (SIG-11) error. This issue was only observed in the GT.M development environment, and was never reported by a user. (GTM-8766)

- GT.M deals with an apparent deviation from documented POSIX behavior when using xfs with -NODEFER_ALLOCATE. Doing a -NODEFER_ALLOCATE extension, GT.M requests only new space. The exception is that an explicit MUPIP EXTEND -BLOCK=0 extension does request full allocation of existing space, which might be of interest when first selecting -NODEFER_ALLOCATE on an existing file. It's not clear whether the choice of file system or its tuning affect the response to this request. Previously in -NODEFER_ALLOCATE mode, xfs could fail to extend database files with a GBLOFLOW error even though there was sufficient space available. (GTM-8771) 

Other

- GT.M protects the UNIX user information it maintains; previously, external calls could damage the information by using the getpwuid() service, which could result in unintended behavior. (GTM-8415)
- MUPIP JOURNAL -PARALLEL correctly handles a MUPIP STOP when using multiple process to work in parallel. Previously, in the rare case the stop arrived while MUPIP was creating such processes, the processes could write their FORCEDHALT messages concurrently, resulting in garbled and/or lost output. This was a cosmetic issue that did not affect the correctness of the MUPIP JOURNAL command , was only observed in the GT.M development environment, and was never reported by a user. (GTM-8663)
- DSE CACHE -VERIFY avoids certain spurious DBBSIZMN errors; previously, under rare circumstances, it produced such an inappropriate error. This issue was only observed in the GT.M development environment, and was never reported by a user. (GTM-8713)
- GT.M correctly handles multiple processes opening and closing a relinkctl file (the file name indicated by ZSHOW "A") as part of process startup and shutdown respectively. Previously, it was possible in rare cases for a process to terminate with a SIGBUS error (kill -7) while trying to open an relinkctl file. This issue was only observed in the GT.M development environment, and was never reported by a user. (GTM-8724)
- Starting replication on a supplementary instance works appropriately after a prior fetchresync rollback removed all updates from the non-supplementary originating instance. Previously, a receiver startup in this case incorrectly resulted in a REPLINSTNOHIST error. This issue was only observed in the GT.M development environment, and was never reported by a user. (GTM-8730)
- GT.M distributions are built on RedHat Enterprise Linux 7 x86_64, Debian Linux 9 x86_32, and AIX 7/POWER 7. Previously, GT.M was built on RedHat Enterprise Linux 6 x86_64/x86_32 and AIX 6/POWER 5. (GTM-8745)

This page is intentionally left blank.

Error and Other Messages

BADLOCKNEST

BADLOCKNEST, Unsupported nesting of LOCK commands

Run Time Error: GT.M detected a LOCK (or ZALLOCATE) argument using an extrinsic function that performed other LOCK (or ZALLOCATE) operations, which it could not safely nest.

Action: Revise the code to avoid such a construct. Note that FIS recommends avoiding this code pattern as it can produce unintended results that GT.M does not detect.

CLISTRTOOLONG

CLISTRTOOLONG, SSSS specified is BBBB bytes long which is greater than the allowed maximum of MMMM bytes

All GT.M Components Error: A command string SSSS of BBBB bytes exceeds the maximum supported command line length of MMMM

Action: Reduce the line length, by using unambiguous abbreviations or shortening names.

JNLBUFFPHS2SALVAGE

JNLBUFFPHS2SALVAGE, Salvaged journal records from process PPPP for database file DDDD at transaction number NNNN and journal-sequence-number/unique-token JJJJ with journal file starting offset OOOO and length LLLL

Run Time Information: Operator log message indicating clean up of journaling information abandoned by an abnormally terminated process.

Action: Investigate the cause of the process termination; report to your GT.M support channel when coincident with other issues.

JNLPOOLPHS2SALVAGE

JNLPOOLPHS2SALVAGE, Salvaged journal records from process PPPP for replication instance file iiii at journal sequence number JJJJ with journal pool starting offset OOOO and length LLLL

Run Time Information: Operator log message indicating clean up of replication information abandoned by an abnormally terminated process.

Action: Investigate the cause of the process termination; report to your GT.M support channel when coincident with other issues.

MUCREFILERR

MUCREFILERR, Error in/at EEEE creating database DDDD (region RRRR)

Run Time Error: Message accompanying another message indicating the failure to create an autodb database file. EEEE indicates the $ZPOSITION of the application which made the global reference that attempted to bring the database file into existence.

Action: Use the preceding message to diagnose and correct the problem, which may include missing environment variables and/or insufficient space or user privileges. Consider whether autodb creation is appropriate for this database file.

MURNDWNARGLESS

MURNDWNARGLESS, Argumentless MUPIP RUNDOWN started with process id PPPP by userid UUUU from directory DDDD

MUPIP Information: Operator log message indicating an argumentless MUPIP RUNDOWN, which uses IPC resources on the node to clean up inactive GT.M shared resources (memory and semaphores).

Action: None typically required; may be useful in diagnosing operational issues.

REC2BIG

REC2BIG, Record size (xxxx) is greater than maximum (yyyy) for region: zzzz

Run Time Error: This indicates that a SET attempted to create a database node with a combined length of keys and data (xxxx) that exceeds the maximum length yyyy permitted for region zzzz.

Action: Use smaller data records or keys in the program. If you want to enlarge the record size for the region, use GDE to change the Global Directory and recreate the database with MUPIP CREATE. If it is necessary to permit the data without allowing time to rebuild the database, use MUPIP CHANGE -RECORD_SIZE. Be careful when you increase the size for existing databases; use GDE to ensure that they have proper characteristics the next time they are CREATEd.

REQRLNKCTLRNDWN

REQRLNKCTLRNDWN, Error accessing relinkctl file rrrr for $ZROUTINES directory dddd. Must be rundown

Run Time Error: A process initiated an auto-relink action with ZLINK or ZRUPDATE, or an auto-relink check with DO, GOTO, ZBREAK, ZGOTO, ZPRINT or $TEXT() which required adding information for the routine in question to the Relinkctl file rrrr for directory dddd, but the shared memory associated with that Relinkctl file had been removed, presumably by an operator using a ipcrm.

Action: Run MUPIP RUNDOWN -RELINKCTL dddd to clear the state of the Relinkctl file. Determine the cause for the improper close and take action to prevent additional occurrences.

RESTRICTEDOP

RESTRICTEDOP, Attempt to perform a restricted operation: xxxx

All GT.M Components Error: The attempted operation, xxxx, was prevented based on the policy specified by the $gtm_dist/restrict.txt file.

Action: Check the permissions and contents of the restrict.txt file against the permissions of the user performing the operation.

RESTRICTSYNTAX

RESTRICTSYNTAX, Syntax error in file ffff at line number nnnn. All facilities restricted for process.

All GT.M Components Error: The file ffff, or $gtm_dist/restrict.txt, contains a syntax error at line nnnn. All facilities which may be specified in a restrict.txt file will be considered restricted, and restricted operations will result in RESTRICTEDOP errors or operations being ignored.

Action: Edit the restrict.txt file to remove the syntax error, and verify the permissions of the file reflect the desired access.

This page is intentionally left blank.