GT.M

Release Notes

V6.3-005

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 1.1	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 1.0	29 June 2018	V6.3-005

Table of Contents

V6.3-005

Overview

V6.3-005 provides a new form of restriction that allows users to filter ZSYSTEM and PIPE commands to ensure appropriate actions as an alternative to completely disabling them. The release brings 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.	-updok (recommended)

Pre V5.5-000 terminology	Pre V5.5-000 qualifier	Current terminology	Current qualifiers
		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.	-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.

Δ 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. 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

Platform	Supported Versions	Notes
		later), glibc (version 2.12 or later) and ncurses (version 5.7 or later).
		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.
		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 *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

Platform	Supported Versions	Notes
		NODEFER_ALLOCATE, do not use any filesystem other than ext4 and a kernel with the fix, or xfs.
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.

Parameter type	32-Bit	64-bit	Remarks
gtm_ulong_t	4-byte	8-byte	gtm_ulong_t is much the same as the C language unsigned long type.
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 V6.3-005 in a Filesystem Hierarchy Standard compliant location such as /usr/lib/fis-gtm/V6.3-005_arch (for example, /usr/lib/fis-gtm/V6.3-005_x86 on 32-bit Linux systems). A location such as /opt/fis-gtm/V6.3-005_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 V6.3-005

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 V6.3-005 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 V6.3-005 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 V6.3-005.

* 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 V6.3-005.

* 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 V6*:

There is no explicit procedure to upgrade a V6 database file when upgrading to a newer V6 version. After upgrading the Global Directory, opening a V6 database with a newer V6 GT.M process automatically upgrades fields in the database fileheader.

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 V6.3-005 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

V6.3-005 does not require new replication instance files if you are upgrading from V5.5-000. However, V6.3-005 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 V6.3-005 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 V6.3-005 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 V6.3-005 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/V6.3-005.

To extract and reapply the trigger definitions on V6.3-005 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 V6.3-005, run **mupip trigger -triggerfile=trigger_defs.trg** to reload your trigger definitions.

To extract and reload trigger definitions on a V6.3-005 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 V6.3-005.

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 V6.3-005 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_V6.3-005_i686, then gtmprofile sets $gtm_dist to /usr/lib/fis-gtm/gtm_V6.3-005_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.

Change History

V6.3-005

Fixes and enhancements specific to V6.3-005:

Id	Prior Id	Category	Summary
GTM-3659	S9B04-001840	Admin	Improved MUPIP LOAD error messaging
GTM-4647	S9C09-002219	Admin	The gtm_mstack_size environment variable specifies M stack size in KiB ✅
GTM-5059	S9D08-002355	Admin	⛔ gtm_mstack_crit optionaly specifies the percentage of the stack at which GT.M should produce a STACKCRIT ✅
GTM-5574	C9E11-002664	Other	Percent conversion routines handle larger numbers
GTM-7960	-	Admin	Warnings of approach to maximum database file size
GTM-8836	-	Admin	Account for MUPIP journal rollback multi-threading in hash table condition handler
GTM-8875	-	Other	More complete detection of STACKOFLOW
GTM-8877	-	Admin	Allow user to create M filters for the commands passed to zsystem and PIPE ✅
GTM-8910	-	DB	Prevent disruption of $ORDER(gvn,1) by MUPIP REORG
GTM-8930	-	Language	Improvement to return values from $VIEW("JNLPOOL")
GTM-8940	-	DB	Performance improvement for ftok semaphores
GTM-8941	-	Admin	LKE recognizes the full keyword for the -CRITICAL qualifier
GTM-8942	-	Other	Reduce impact of signal management
GTM-8943	-	Language	⛔ ZGOTO 0 in a call-in returns to the invoking C code ✅
GTM-8945	-	Admin	Prevent Receiver hang after killing an Update Process

Id	Prior Id	Category	Summary
GTM-8949	-	Other	Prevent recursive calls to system memory allocator
GTM-8951	-	Language	$TEXT() and ZPRINT use auto-relink when their argument includes a routinename
GTM-8952	-	Other	On 32-bit Linux, prevent ASSERTPRO when compiling a routine with many, many local variables
GTM-8953	-	Admin	ROLLBACK FORWARD with -VERIFY properly applies all valid updates
GTM-8954	-	Admin	Source Server sends all errors to the log file
GTM-8955	-	Other	Relink Locking Speedup
GTM-8956	-	Other	-NOWARNING compiler qualifier suppresses more messages
GTM-8957	-	Admin	MUPIP SET -NOREADONLY -ACC=BG works in one command
GTM-8958	-	Admin	⬤ TLS reference implementation plug-in disables SSLv3 by default
GTM-8959	-	Language	For 64-bit Linux, fix ZGOTO 0 within a call-in
GTM-8961	-	Language	Clean path reported by ZSHOW "D", $KEY, and $ZSOCKET("","LOCALADDRESS",) on AIX
GTM-8962	-	Language	⬤ Clean up ZSHOW "i" output ⬤
GTM-8964	-	Admin	MUPIP respects -READONLY (MM) database state
GTM-8965	-	Language	Alternation pattern with a large match produces a PATALTER2LARGE error
GTM-8967	-	DB	Correct reporting of globals reported by TPRESTART
GTM-8969	-	DB	When encrypting for the first time, prevent spurious CRYPTOPFAILED errors
GTM-8973	-	Other	External call table loading more careful about M labelrefs
GTM-8974	-	DB	Improved handling of significant IO pauses in Database or Journal I/O

Id	Prior Id	Category	Summary
GTM-8980	-	Language	VIEW and $VIEW() fixes, particularly related to region arguments ✅
GTM-8981	-	Language	Fix to IF @<literal>
GTM-8985	-	Language	Prevent possible incorrect result from a function with all literal arguments in a Boolean expression
GTM-8988	-	Admin	MUPIP RESTORE handling of out-of-space
GTM-8989	-	Admin	Prevent unusual MUPIP RUNDOWN hang
GTM-8990	-	Language	ZRUPDATE treats a cycle of symbolic links like a simple missing routine
GTM-8992	-	DB	GT.M never decrements database statistics for $DATA(), $GET(), $ORDER(), or $QUERY()
GTM-8996	-	Admin	GT.M issues an fsync on the database file after a region freeze goes into effect
GTM-9001	-	Other	Improved bounds checking in DSE and MUPIP

Database

* $ORDER(gvn,-1) and $ZPREVIOUS(gvn) appropriately handle the concurrent move of the global variable tree root block of the gvn by MUPIP REORG; previously, this concurrence could occasionally produce a GVORDERFAIL with a status code of tSSS. (GTM-8910)

* GT.M avoids superfluous public (ftok) semaphore control operations on database startup or rundown. (GTM-8940)

* TPRESTART messages correctly identify the global causing the conflict; previously they could report an incorrect global name. (GTM-8967)

* GT.M processes work correctly when a concurrent MUPIP REORG -ENCRYPT encrypts a previously unencrypted database file. Previously GT.M processes could fail with CRYPTOPFAILED errors unaccompanied by other errors explaining the reason for the failure. This issue was only observed in the GT.M development environment, and was never reported by a user. (GTM-8969)

* GT.M processes attempting to acquire a shared database resource and encountering long delays continue to wait if the process holding the resource is still active. Previously they intervened prematurely which could cause DBDANGER errors and accompanying database damage. GT.M processes attempting to acquire a shared resource required to perform journal writes and encountering an extended delay in acquiring the resource, issue JNLSENDOPER/JNLFLUSH/ JNLPROCSTUCK messages to the system log, and, if so configured, freeze the instance as a result, sending REPLINSTFROZEN/REPLINSTFREEZECOMMENT messages to the system log. They continue to attempt the journal writes, removing the instance freeze if they eventually succeed. Previously, such processes received an error, and the instance required a operator action to release the freeze. (GTM-8974)

* GT.M does not decrement database statistics for $DATA(), $GET(), $ORDER(), or $QUERY(). Note that, to minimize performance impact, GT.M updates most database statistics without concurrency controls, so the statistics are approximate and may have brief fluctuations. Previously GT.M made explicit adjustments which reporting was more likely to detect. MERGE operations increment DATA and GET database statistics to more accurately reflect database activity. Previously, Merge suppressed increments to those statistics under certain conditions.(GTM-8992)

Language

* If the process has access to an instance file designation, $VIEW("JNLPOOL") returns its name; if the process has not yet opened the pool, the return contains an asterisk after the name. If the process does not have sufficient information to determine a replication journal instance file, the function returns "No replication instance defined." Previously, the function with this argument returned an empty string if the process had not opened the journal pool. (GTM-8930)

* A ZGOTO 0 in case of call-in, unwinds all the M stack frames and returns to the invoking C routine. In case of a non-CI invocation, this terminates the process. A gtm_ci invocation without calling gtm_init() first, proceeds with the CI invocation. Previously, this produced 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-8943)

* With auto-relink enabled, $TEXT() and ZPRINT use auto-relink when their argument includes a routinename. Note that ZBREAK does not use auto-relink because its intended purpose is as a debugging facility and if the user intends the ZBREAK for a new version they can explicitly request it with a ZLINK. Previously, $TEXT() and ZPRINT used the version most recently linked by the process even if a new version had become available in an auto-relink enabled environment. (GTM-8951)

* ZGOTO 0 in a call-in returns to the invoking program. Previously on x86-64 Linux systems with glibc 2.24 or later, this generated a segmentation violation (SIG-11). [x86-64 Linux](GTM-8959)

* The path reported by ZSHOW "D", $KEY, and $ZSOCKET("","LOCALADDRESS",) for LOCAL sockets passed via JOB or WRITE /PASS is correct. Previously on AIX 7.2, there could have been extra bytes at the end. This issue was only observed in the GT.M development environment, and was never reported by a user.[AIX 7.2] (GTM-8961)

* ZSHOW "I" outputs $ZPIN and $ZPOUT even if they are the same as $PRINCIPAL and no longer displays $ZPROCESS, as it was only meaningful on OpenVMS; also, $ZTNAME appears in proper alphabetic sequence. Previously if $ZPIN or $ZPOUT matched $PRINCIPAL, ZSHOW "I" omitted them, $ZPROCESS appeared as an empty string and $ZTNAME was not ordered appropriately. (GTM-8962)

* Pattern match of a string with an alternation match exceeding what GT.M can handle produces a PATALTER2LARGE error; previously this condition produced a segmentation violation (SIG-11). (GTM-8965)

* VIEW and $VIEW() with a empty string or inappropriate region-list works appropriately; in V6.3-004 these could cause inappropriate results, including a segmentation violation (SIG-11). $VIEW("statshare") returns a 0 when the process has sharing disabled, a 1 when it has sharing enabled and a 2 when sharing is selectively enabled. In V6.3-004, it did not differentiate between the all and selective cases and returned 1 when sharing was disabled and selective disabling was also specified. $VIEW("statshare","<region>") works appropriately even if the region had been selectively disabled when full sharing is disabled and the region had not been opened. In V6.3-004, this set of conditions produced a segmentation violation (SIG-11). The error messages when invalid parameters

are passed to VIEW/$VIEW() print the name of the parameter; previously such error messages did not have the name of the parameter. (GTM-8980)

* IF @<literal> works correctly when the literal evaluates to FALSE; in versions V6.3-001 to V6.3-004, it tended to fail with an inappropriate INDEXTRACHARS error or a segmentation violation (SIG-11). (GTM-8981)

* The GT.M compiler appropriately handles possible string returns into Boolean expressions from functions with all literal arguments; in V6.3-000 to V6.3-004 it could produce an incorrect result. The workaround was to avoid all literal arguments for $CHAR(), $EXTRACT(), $PIECE(), their $Z*() variants and $SELECT() when they appeared in Boolean expressions. (GTM-8985)

* When ZRUPDATE encounters a cycle of symbolic links without finding a specified routine, it treats the same as a simple routine not found and ignores the missing routine. Previously, the command issued an error with text about "too many levels of symbolic links". This issue was observed in the GT.M development environment after upgrading to newer Linux distributions. (GTM-8990)

System Administration

*   MUPIP LOAD reports ranges of records not loaded due to missing database files; previously it reported an error for every such record. (GTM-3659)

*   GT.M supports specifying the M stack size in KiB with the gtm_mstack_size environment variable. No setting or a setting of 0 uses the default (272KiB). The minimum supported size is 25 KiB; GT.M reverts values smaller than this to 25 KiB. The maximum supported size is 10000 KiB; GT.M reverts values larger than this to 10000 KiB. (GTM-4647)

*   GT.M recognizes setting the environment variable gtm_mstack_crit_threshold to specify an integer between 15 and 95 defining the percentage of the stack which should be used before GT.M emits a STACKCRIT warning. If the value is below the minimum or above the maximum GT.M uses the minimum or maximum respectively. The default is 90. (GTM-5059)

*   An automatic database file extension, MUPIP EXTEND, INTEG and SIZE all put a message in the system logs when the database reaches 88% of its maximum size. Beyond this 88% threshold, manual and automatic extends only report at 1% intervals, but INTEG and SIZE report at every subsequent invocation while the condition persists. All but the automatic extension also produce the message for the operator. Previously GT.M gave no such warnings, which made it necessary to proactively check on database size. (GTM-7960)

*   When a multi-thread instance of MUPIP journal -rollback runs out of memory during a rehashing operation, child threads transfer error-handling to the parent thread and terminate themselves. Previously, child threads occasionally failed to report the error, causing MUPIP to halt without a descriptive error message. This issue was only observed in the GT.M development environment, and was never reported by a user. (GTM-8836)

*   The GT.M restriction mechanism recognizes the following lines:

```
ZSYSTEM_FILTER[:M labelref]
PIPE_FILTER[:M labelref]
```

The labelref must include a routine name. If a process is restricted by a ZSYSTEM or PIPE_OPEN line in the restrictions file that restriction takes precedence over the corresponding filter restriction. Otherwise when a process is subject to these restrictions, GT.M inserts an invocation of the labelref prior to the restricted command, passing a string containing the argument to the ZSYSTEM command or the command deviceparameter of the PIPE OPEN. The path to the filter routine must be included in $zroutines. FIS recommends that the filter routine is placed in a location with restricted access such as $gtm_dist. If the filter invocation return is -1,GT.M produces a RESTRICTEDOP error, otherwise it executes the command using the returned string via output parameters as a, possibly identical, replacement for the original string. Since GT.M uses the call-ins mechanism to execute the filters, a filter invocation inside a TP transaction in call-ins produces a CITPNESTED error. Note that because ZSYSTEM and OPEN are not Isolated actions FIS recommends against their use within a TP transaction. Filters also increment the nested level of call-ins. A recursive filter invocation

produces a NOFILTERNEST error. GT.M reports all filter errors to the operator log accompanied by a COMMFILTERERR.

An example restrict file for this:

```
cat $gtm_dist/restrict.txt

ZSYSTEM_FILTER:^filterzsy
PIPE_FILTER:^filterzsy
```

The actual filter routine:

```
filterzsy(inarg,outarg);
 if ""=inarg set outarg="-1;must provide a command" quit
 for i=1:1 set arg=$piece(inarg,";",i) quit:""=arg  do  quit:$data(outarg)
 . for  quit:$zchar(9,32)'[$extract(arg)  set arg=$extract(arg,2,9999)
 . set cmd=$piece(arg," ")
 . for restrict="sudo","cd" if cmd=restrict set outarg="-1;command "_restrict_" not
permitted" quit
 . quit:$data(outarg)
 . if "echo"=cmd set $piece(arg," ")="echo #",$piece(inarg,";",i)=arg    ;example of
modification
 set:'$data(outarg) outarg=inarg
 quit +outarg
```

Filter execution starts with $STACK=1 ($ZLEVEL=2).

Following are the GT.M commands, Intrinsic Special Variables, and functions whose behavior changes in the context of a filter invocation.

ZGOTO 0 (zero) returns to the processing of the restricted command as does ZGOTO 1 (one) with no entryref, while ZGOTO 1:entryref replaces the originally invoked filter and continues filter execution.

$ZTRAP/$ETRAP NEW'd at level 1.

$ZLEVEL initializes to one (1) in GTM$CI, and increments for every new stack level.

$STACK initializes to zero (0) in GTM$CI frame, and increments for every new stack level.

$ESTACK NEW'd at level one (1) in GTM$CI frame.

$ECODE/$STACK() initialized to the empty string at level one (1) in GTM$CI frame.

After the filter completes, GT.M restores the above to their values at the invocation of the filter. (GTM-8877) ●

* LKE recognizes the full keyword for the -CRITICAL qualifier; previously it only accepted -CRIT. (GTM-8941)

* The receiver process recovers after its update process was terminated with a signal while idle. Previously, on Linux systems with glibc 2.25 or newer, the receiver process could hang indefinitely, requiring manual cleanup of the process and shared memory. This issue was only observed in the GT.M development environment, and was never reported by a user. (GTM-8945)

* MUPIP JOURNAL -ROLLBACK -FORWARD when executed with -VERIFY properly applies all updates. Previously updates that should have been applied to the database were instead sent to the lost transaction file. The workaround was to execute MUPIP JOURNAL -VERIFY independent of the FOWARD ROLLBACK. (GTM-8953)

* The Source Server directs errors to the Source Server log file. A previous fix with GTM-8576 was incomplete and could result in TLS initialization error messages not logged to the server log file. (GTM-8954)

* Executing MUPIP SET -NOREAD_ONLY -ACC=BG on a read-only database sets the access mode to BG and turns off read-only. Previously, this action would result in a READONLYNOBG error message and no changes to the file. (GTM-8957)

* The TLS reference implementation plug-in disables SSLv3 by default. Previously, customers wishing to disable SSLv3 needed to add the configuration option 'ssl-options: "SSL_OP_NO_SSLv3";' to the "tls" namespace in the $gtmcrypt_config configuration file. (GTM-8958)

* MUPIP does not modify -READONLY (MM) database files; previously various MUPIP commands could inappropriately update state information in such database files, causing errors when subsequently using the database. (GTM-8964)

* MUPIP RESTORE exits with an error when it encounters an out-of-space condition. Previously, if MUPIP RESTORE encountered an out-of-space condition, if crashed with a segmentation violation (SIG-11). This issue was only observed in the GT.M development environment, and was never reported by a user. (GTM-8988)

* MUPIP RUNDOWN works as expected. In rare situations, processes killed during transaction commits could leave the transaction information in an inconsistent state that caused MUPIP RUNDOWN to hang. Note FIS strongly recommends against kill -9 of a process accessing the database as it can induce database damage. This issue was only observed in the GT.M development environment, and was never reported by a user. (GTM-8989)

* GT.M issues an fsync on the database file after a region freeze goes into effect, which forces the underlying file system to harden changes to secondary storage. (GTM-8996)

Other

* The %DH,%HD,%OH,%HO,%OD, and %DO routines now handle conversions with numbers up to GT.M's maximum string length in size. Additionally, %DH and %DO routines now handle the conversion of negative decimal numbers properly, even when the specified length is not long enough to represent that fully converted number. Previously, the %DH,%HD,%OH,%HO,%OD, and %DO routines could only deal with numbers up to the equivalent of the maximum 18-digit decimal number. Attempting to convert a number larger than this would cause roundoff in the final result, which still occurs if they are used for arithmetic. Moreover, specifying a length for the %DH and %DO routines that was not long enough to hold the fully converted result would cause the routines to produce incorrect values. The workaround for both these issues was to avoid using these percent routines with numbers greater than the maximum 18-digit decimal number and to always specify an appropriate length for the %DH and %DO routines. (GTM-5574)

* When a stack frame exceeds the M virtual machine stack pointer, GT.M issues a STACKOFLOW error and produces a GTM_FATAL_* context dump, but no core file, as this an application issue, rather than a GT.M problem. A module with a very large number of variables and/or dynamic literals can cause this problem. Note that the $gtm_mstack environment variable can, within limits, set the size of the M virtual machine stack size. Previously if the overflow was sufficient to make the stack pointer negative, the process exited with a segmentation violation (SIG-11). (GTM-8875)

* GT.M avoids superfluous signal management operations. Previously system traces showed more sigprocmask system calls than desirable, particularly on Linux systems with Meltdown/Spectre mitigation in place. (GTM-8942)

* GT.M prevents recursive calls to the system memory allocator. Previously, the system memory allocator could be called recursively which resulted in a process hanging. This issue was only observed in the GT.M development environment, and was never reported by a user. (GTM-8949)

* GT.M appropriately compiles programs with large numbers of local variable names on 32-bit Linux; previously, compiling a routine with enough local variable names produced an ASSERTPRO and no object file. This issue was only observed in the GT.M development environment, and was never reported by a user. [x86 Linux](GTM-8952)

* GT.M uses a lighter weight locking mechanism to protect the relink control file. Previously, a large number of concurrent relink control operations such as those done on process shutdown could cause the system to use an excessive amount of CPU. (GTM-8955)

* The GT.M compiler -NOWARNING qualifier for the MUMPS command and $ZCOMPILE suppresses warning messages for BADCHAR, BLKTOODEEP, and LITNONGRAPH; previously it did not. (GTM-8956)

* GT.M external call loading is more discriminating about M labelrefs; previously it did not detect all invalid labelrefs, which deferred error detection and made diagnosis more challenging. (GTM-8973)

* GT.M appropriately bounds checks variable length input parameters to MUPIP and DSE. Previously it did not detect an off-by-one buffer overrun. Also, MUPIP EXTRACT issues the ICUNOTENABLED

warning message when used with the -OCHSET qualifier while in M mode. Previously, attempting to use -OCHSET while in M mode resulted in a SIG-11. These issues were only observed in the GT.M development environment, and was never reported by a user. (GTM-9001)

Error and Other Messages

COLLDATAEXISTS ⚠

COLLDATAEXISTS, Collation type cannot be changed while xxxx data exists

Run Time Error: This indicates that an attempt was made to change the collation type while xxxx was either a subscripted local for a process collation change or a gvn name for global variable collation.

Action: KILL or NEW the local variables before you change the local collation type, or KILL a gvn before changing its collation.

COMMFILTERERR ⊕

COMMFILTERERR, Error executing the command filter for FFFF DDDD

Run Time Error: Reports a problem in filter code where FFFF describes the nature of the filter and DDDD some thing about the nature of the issue. There may be associated / related messages. Because filters are a potential security tool, these errors tend are generally reported to the operator log.

Action: Analyze the filter code in light of the messages and revise accordingly.

EXTRINTEGRITY ⚠

EXTRINTEGRITY, Database ffff potentially contains spanning nodes or data encrypted with two different keys

MUPIP Error: MUPIP EXTRACT cannot run because the database file ffff might contain spanning nodes or be partially encrypted with a particular key. Proceeding on a live database in such situation could result in data corruption.

Action: If you encounter this error while running MUPIP EXTRACT with -FORMAT="BINARY", re-run the command with the -FREEZE qualifier. MUPIP EXTRACT requires -FREEZE to acquire stand-alone access to produce a consistent copy of the data. However, not using -FREEZE when you request a MUPIP EXTRACT may produce a loadable, if inconsistent output. If you encountered this error while running MUPIP EXTRACT with ZWR or GO format, it is likely that your database is encrypted with more than one key; with BINARY output it may be multiple keys or spanning node data. If the issue is a key change, run MUPIP REORG -ENCRYPT to complete the encryption of the database. As a final resort, you may use an -OVERRIDE qualifier to proceed on a live database that either contains spanning nodes or is undergoing (re)encryption. Although EXTRACT -OVERRIDE may produce text for analysis, the result is not suitable as input for MUPIP LOAD and FIS highly discourages using -OVERRIDE.

FAILEDRECCOUNT ⊕

FAILEDRECCOUNT, LOAD unable to process MMMM records

MUPIP Error: MUPIP LOAD was unable to load MMMM records from the specified input extract.

Action: Examine prior RECLOAD error messages for causes for the failed records and address them.

ICUNOTENABLED ⊕

ICUNOTENABLED, ICU libraries not loaded

Run Time Warning: The operation required the library containing support for International Components for Unicode (ICU) but GT.M could not find libicu. There may be other messages.

Action: If you require UTF-8 support, install an appropriate ICU library - see the GT.M Adinistration and Operations Guide for information on ICU setup.

LOADRECCNT ⊕

LOADRECCNT, Last EXTRACT record processed by LOAD: RRRR

MUPIP Information: This message indicates number of records (RRRR) MUPIP LOAD processed. The number of records represents the sum of header records, successfully loaded data records, and failed records. Note LOAD may have stopped processing due to a record limit in the command or a <CTRL-C>.

Action: Ensure the identified stopping point corresponds with your intentions.

MSTACKCRIT ⊕

MSTACKCRIT, User-specified M stack size critical threshold of xxxx not appropriate; must be between mmmm and nnnn; reverting to kkkk

Run Time Error: The environment variable gtm_mstack_crit_threshold was set to an invalid value, either too large, in which case GT.M uses the largest acceptable value or too low, in which case GT.M uses the smallest acceptable value.

Action: If the adjusted value is unacceptable, revise or unset the environment variable.

NOFILTERNEST ⊕

NOFILTERNEST, Filter nesting not allowed

Run Time Error: Filter code must not invoke other code that requires a filter.

Action: Revise the filter code to adhere to the requirement.

PATALTER2LARGE ⊕

PATALTER2LARGE, Pattern match alternation exceeded the LLLL repetition limit on prospective matches

Run Time Error: An alternation pattern applied to a long occurrence of that pattern reached a GT.M limit (LLLL) on tracking the match.

Action: Revise the logic to reduce the size of the string being matched or to otherwise break up the match into smaller parts.

REGFILENOTFOUND △

REGFILENOTFOUND, Database file DDDD corresponding to region RRRR cannot be found

MUPIP Error: This indicates MUPIP cannot locate the database file DDDD mapped to region RRRR.

Action: Ensure that the current global directory is the one intended and that it maps the file intended. If the path is relative or includes environment variables, ensure that the current working directory and any enironment variable are appropriate. Also ensure the file exists and has authorizations, including its path, that make it available to the user attempting to access it. If the MUPIP command involves a statsDB (for example MUPIP INTEG -STATS), ensure that the appropriate regions have STATS enabled, that the $gtm_statsdir environment variable has been properly defined, and that other processes are using shared statistics, as MUPIP by itself does not create new statsDB databases. Note that MUPIP INTEG does not create statsDB and reports any that it skips with an informational message, but exits with a normal status after such skips.

VIEWCMD △

VIEWCMD, View parameter pppp is not valid with the VIEW command

Run Time Error: This indicates that the VIEW command has an argument pppp that is only valid with the $VIEW() function.

Action: Modify the argument.

VIEWFN △

VIEWFN, View parameter pppp is not valid with the VIEW command

Run Time Error: This indicates that the $VIEW() function has an argument pppp that is only valid with the VIEW command.

Action: Modify the argument.