

GT.M

Release Notes

V6.3-013

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

Copyright ©2020, 2022 Fidelity National Information Services, Inc. and/or its subsidiaries. All Rights Reserved.

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

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

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

Revision History		
Revision 1.1	04 July 2022	Added GTM-9276.
Revision 1.0	30 June 2020	V6.3-013

Table of Contents

V6.3-013	1
Overview	1
Conventions	1
Platforms	2
Platform support lifecycle	4
32- vs. 64-bit platforms	4
Call-ins and External Calls	4
Internationalization (Collation)	5
Environment Translation	5
Additional Installation Instructions	5
Recompile	6
Rebuild Shared Libraries or Images	6
Compiling the Reference Implementation Plugin	6
Upgrading to GT.M V6.3-013	7
Stage 1: Global Directory Upgrade	7
Stage 2: Database Files Upgrade	8
Stage 3: Replication Instance File Upgrade	10
Stage 4: Journal Files Upgrade	10
Stage 5: Trigger Definitions Upgrade	10
Downgrading to V5 or V4	11
Managing M mode and UTF-8 mode	12
Setting the environment variable TERM	13
Installing Compression Libraries	13
Change History	15
V6.3-013	15
Database	17
Language	18
System Administration	20
Other	21
Error and Other Messages	22
DBFILERONLY 	22
EXITSTATUS 	22
EXTRFMT 	22


V6.3-013

Overview

V6.3-013 adds NOISOLATION for non-TP mini-transactions and logging for access denials. It includes expanded maximums for Receiver Server and Source Server buffers. Also, it addresses some language processing issues that show up with unusual syntax, and typically cause errors, but in some cases incorrect results.

V6.3-013 also includes other fixes and enhancements. For more information, refer to the Change History section.

Items marked with  document new or different capabilities.

Please pay special attention to the items marked with the symbols  as those document items that have a possible impact on existing code, practice or process. Please be sure to recompile all objects to ensure all the updates are in place - we apologize that this release does not force the recompile.



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.	-updnok
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.

V6.3-013

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	<p>Only 64-bit versions of AIX with POWER7 as the minimum required CPU architecture level are Supported.</p> <p>While GT.M supports both UTF-8 mode and M mode on this platform, there are problems with the AIX ICU utilities that prevent FIS from testing 4-byte UTF-8 characters as comprehensively on this platform as we do on others.</p> <p>Running GT.M on AIX 7.1 requires APAR IZ87564, a fix for the POW() function, to be applied. To verify that this fix has been installed, execute instfix -ik IZ87564.</p> <p>AIX 7.1 TL 5 is Supportable.</p> <p>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.</p>
x86_64 GNU/Linux	Red Hat Enterprise Linux 7.8; Ubuntu 16.04 LTS, 18.04 LTS, and 20.04 LTS	<p>To run 64-bit GT.M processes requires both a 64-bit kernel as well as 64-bit hardware.</p> <p>GT.M should also run on recent releases of other major Linux distributions with a contemporary Linux kernel (2.6.32 or later), glibc (version 2.12 or later) and ncurses (version 5.7 or later).</p> <p>Due to build optimization and library incompatibilities, GT.M versions older than V6.2-000 are incompatible with glibc 2.24 and up. This incompatibility has not been reported by a customer, but was observed on internal test systems that use the latest Linux software distributions from Fedora (26), Debian (unstable), and Ubuntu (17.10). In internal testing, processes either hung or encountered a segmentation violation (SIG-11) during operation. Customers upgrading to Linux distributions that utilize glibc 2.24+ must upgrade their GT.M version at the same time as or before the OS upgrade.</p> <p>GT.M requires a compatible version of the libtinfo library. On Red Hat, the ncurses-libs and ncurses-compat-libs packages contain the libtinfo library. On Debian/Ubuntu, libtinfo5 and libncurses5 packages contain the libtinfo library. If any of these packages is not already installed on your system, please install using an appropriate package manager.</p> <p>To support the optional WRITE /TLS fifth argument (the ability to provide / override options in the tlsid section of the encryption configuration file), the reference implementation of the encryption plugin requires libconfig 1.4.x.</p> <p>Only the ext4 and xfs filesystems are Supported. Other filesystems are Supportable, but not Supported. Furthermore, if you use the NODEFER_ALLOCATE feature, FIS strongly recommends that you use xfs. If you must use NODEFER_ALLOCATE with ext4, you XXXX ensure that your kernel includes commit d2dc317d564a46dfc683978a2e5a4f91434e9711 (search for d2dc317d564a46dfc683978a2e5a4f91434e9711 at https://www.kernel.org/</p>

Platform	Supported Versions	Notes
		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. Ubuntu 19.10 is Supportable.
x86 GNU/Linux	Debian 10 (buster)	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.
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
<code>gtm_descriptor</code> in <code>gtm_descript.h</code>	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
<code>gtm_string_t</code> type in <code>gtmxc_types.h</code>	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 V6.3-013 in a Filesystem Hierarchy Standard compliant location such as `/usr/lib/fis-gtm/V6.3-013_arch` (for example, `/usr/lib/fis-gtm/V6.3-013_x86` on 32-bit Linux systems). A location such as `/opt/fis-gtm/`

V6.3-013_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, TLS replication, or TLS sockets, 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, libpgpme, 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 libpgpme11-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

These 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 are still 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 the above directions.
- Remove /usr/local/ssl/lib.donotuse to avoid future complications.

Upgrading to GT.M V6.3-013

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 V6.3-013 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 V6.3-013 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 V6.3-013.
- 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 V6.3-013.
- 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

Before starting the database file upgrade, use the prior GT.M version to perform an appropriate MUIP action to removes abandoned GT.M database semaphores and releases any IPC resources.

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 the fields in the database file header.

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 MUIP 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 V6.3-013 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

V6.3-013 does not require new replication instance files if you are upgrading from V5.5-000. However, V6.3-013 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 V6.3-013 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 wholeness 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 V6.3-013 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 V6.3-013 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/V6.3-013.

To extract and reapply the trigger definitions on V6.3-013 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 V6.3-013, run **mupip trigger -triggerfile=trigger_defs.trg** to reload your trigger definitions.

To extract and reload trigger definitions on a V6.3-013 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 V6.3-013.
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 rennumbers 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 V6.3-013 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_V6.3-013_i686, then gtmprofile sets \$gtm_dist to /usr/lib/fis-gtm/gtm_V6.3-013_i686/utf8).
 - On platforms where the object files have not been placed in a libgtmutil.so shared library, the last element of \$gtmroutines is \$gtm_dist(\$gtm_dist/..) so that the source files in the parent directory for utility programs are matched with object files in the utf8 subdirectory. On platforms where the object files are in libgtmutil.so, that shared library is the one with the object files compiled in the mode for the process.

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

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

Setting the environment variable TERM

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

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

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

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

Installing Compression Libraries

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

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

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

V6.3-013

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

V6.3-013

Fixes and enhancements specific to V6.3-013:

Id	Prior Id	Category	Summary
GTM-7759	-	Admin	Logging of access denials
GTM-7987	-	Admin	Replication servers retry on some potentially transient network errors
GTM-8772	-	Admin	MUPIP SET accepts actions that would not end FREEZE -ONLINE
GTM-8784	-	Admin	MUPIP SET rejects actions that would end FREEZE -ONLINE
GTM-8793	-	Admin	More suitable error on return from an internally driven script
GTM-8838	-	Language	Additional statistics
GTM-8878	-	Other	better ordering of journal record time stamps under a particular circumstance
GTM-9147	-	Admin	Larger maximum journal buffer size
GTM-9230	-	DB	VIEW "NOIOSLATION" has the described impact on no-TP updates as on TP updates
GTM-9252	-	Admin	Prevent inappropriate IPC_RMID syslog messages caused by prior use of %PEEKBYNAME or ZHELP
GTM-9259	-	Language	More careful keyword parsing
GTM-9275	-	Admin	MUPIP LOAD provides appropriate error message for some poorly formatted extract input
GTM-9276	-	Language	Improved bounds check for M strings
GTM-9277	-	Language	In gtm_side_effects mode, fix handling of integer function arguments involving side effects
GTM-9278	-	Other	Better errors for unusual conditions
GTM-9287	-	Language	Fix syntax error message line number reporting for large M files
GTM-9288	-	Other	Fix issue with over-long M filenames
GTM-9289	-	Admin	Appropriate behavior for MUPIP REPLIC -SOURCE -SHUTDOWN in the face of network issues

Change History

Id	Prior Id	Category	Summary
GTM-9291	-	Language	Fix handling of \$TEST as the first term of a Boolean that includes side effects
GTM-9292	-	Language	Fix handling of side effect processing turned off during Boolean processing for \$SELECT()
GTM-9293	-	Language	Where appropriate, accept an empty string result from argument indirection
GTM-9294	-	Language	Fix handling of \$QUERY(lvn) where the lvn has no subscripts or does not exist
GTM-9295	-	Language	🔴 Various \${Z}TRANSLATE() fixes
GTM-9296	-	Language	Continue compilation after a BADCHAR error
GTM-9311	-	Other	%YGBLSTAT NEWs local variables d and x to protect the user environment
GTM-9313	-	Language	\$ORDER() issue with 1st argument subscripts containing both a gvn and a Boolean expression, and 2nd argument is a literal

Database

- GT.M mini-transaction (non-TP) database SETs recognize the NOISOLATION characteristic; previously they did not. (GTM-9230) ✓

Language

- GT.M reports execution statistics WFR, BUS and BTS in the result from \$VIEW("GVSTAT",<region>). These statistics are described in the "ZSHOW Information Codes" section of the "GT.M Programmer's Guide". MUPIP -DUMPFHEAD reports these statistics as "n_wait_for_read", "n_buffer_scarce" and "nt_bt_scarce" respectively. Previously, MUPIP -DUMPFHEAD reported these statistics with the names "t_qread_ripsleep_cnt_cntr", "db_csh_get_too_many_loops_cntr" and "bt_putflush_dirty_cntr" along with the last transaction associated with the statistic with the same prefixes, but with the suffix "_tn" replacing "_cntr"; DUMPFHEAD no longer reports the transaction information. \$VIEW() did not report these counters at all. These statistics are also available through the %PEEKBYNAME() interface under same names as used by DUMPFHEAD, and on a systemwide basis as WFR, BUS and BTS through the utility program ^ %YGBLSTAT for those processes which have opted to share statistics. (GTM-8838) 🚫👍
- The GT.M compiler checks Z* keywords except for deviceparameters for the length provided by the source code and accepts a leading subset, with a small set of legacy abbreviations. This means that if you use a short form that's not unique, GT.M evaluates it as the alphabetically first keyword or legacy abbreviation. FIS recommends using at least four characters in stored code, but ZBIT* functions and deviceparameters with a leading "NO" may need more. For standard keywords, the compiler requires correct spelling for the full keyword through up to eight characters and recognizes standard abbreviations. Previously the compiler ignored characters following matches in its internal tables which caused surprise if there were similar keywords or typos - in particular it may reject misspellings it accepted in prior releases. (GTM-9259) 🚫
- The Call-in interface uses the length from supplied size of gtm_string_t types to effectively bounds check the returned string. Programs using GT.M call-ins need to reset the length of the gtm_string_t before invoking call-ins. Previously GT.M wrote whatever the MUMPS routine returned into the caller's buffers. Users of gtm_char_t type should allocate at least 1 MiB of space to ensure that the MUMPS routine, which can potentially write up to 1MiB, does not overflow the callers buffer. (GTM-9276)
- The GT.M compiler, when using gtm_side_effects, deals appropriately cases when the last argument in a Boolean expression has a side effect and the result of the entire expression is evaluated as an integer. Most integer evaluations are for intrinsic function arguments. The side effects include extrinsic expressions, \$INCREMENT() and indirect evaluations. Subsequent to the introduction of the gtm_side_effects mode, such evaluations could fail with an ASSERTPRO fatal error, segmentation violation (SIG-11), or rarely an incorrect result. (GTM-9277)
- GT.M correctly reports line numbers in syntax errors for M files longer than 64K-1 lines. Previously, it reported error messages with line numbers in excess of that size incorrectly (modulo 64Ki). (GTM-9287)
- The GT.M compiler appropriately handles cases where \$TEXT appears as the first term in a Boolean expression that includes side effects. Previously this pattern could cause fatal compilation errors or, infrequently, incorrect results. (GTM-9291)
- The GT.M compiler appropriately handles cases where the compiler detects a side effect within a \$SELECT() argument, but subsequently determines that the side effect would not come into play at runtime. Starting in V6.3-011 \$SELECT() changes created the possibility of this causing fatal compilation failures. (GTM-9292)
- GT.M accepts empty string results from argument indirection when the command would accept such an argument; previously it rejected such a result as an invalid expression. (GTM-9293)
- GT.M appropriately handles \$QUERY() of local variables that have no subscripts; previously such a pattern could cause a segmentation violation (SIG-11). (GTM-9294)
- The \$(Z)TRANSLATE() functions appropriately handle the case where the function appears more than once in a line and the function that evaluates first is within an arithmetic or Boolean expression. Subsequent to the optimization of \$[Z]TRANSLATE() in V6.3-005 (GTM-8947) such a pattern could give an incorrect result. Also, \$[Z]TRANSLATE() appropriately handles cases where UTF8 mode lengths exceeded expectations; previously these situations typically caused a segmentation violation (SIG-11). In addition, UTF-8 mode \$TRANSLATE() evaluations with NOBADCHAR checking give uniform results, typically matching the \$ZTRANSLATE() result with the same arguments. Previously the results depended on the code path followed to create the string containing the bad character(s). Note that FIS recommends avoiding the use of M standard functions without BADCHAR checking. (GTM-9295) 🚫👍

Language

- The GT.M compiler appropriately handles a BADCHAR error in a UTF-8 source file as a syntax error; previously it stopped the compilation and did not produce an object file. (GTM-9296)
- Prevent possible incorrect result when the first \$ORDER() argument contains subscripts with both a Boolean expression and a global reference, and the second argument is a literal. (GTM-9313)

System Administration

- By default GT.M uses the the syslog() facility to log a number of errors related to permissions and access. Previously, it only noted these messages on the console or, if output was redirected, in a GT.M process's error output. The GT.M restriction LOGDENIALS provides a facility for disabling this logging on a Unix group basis. If the restriction mechanism is not used, the logging takes place for all GT.M processes. If the restriction is used logging takes place for specified groups only. To support restriction based configuration of LOGDENIALS, the GT.M restriction handling code now supports group names using the POSIX Portable Filename Character Set - 3.282 Portable Filename Character Set. That set of characters from which portable filenames are constructed is: lower- and upper-case ASCII letters (a-z, A-Z), ASCII numbers (0-9), and punctuation characters period (.), underscore (_) and dash (-). (GTM-7759) ✓
- Replication servers try to reconnect for some errors from the TLS/SSL layers; previously, these errors caused the servers to terminate with a TLSIOERROR. (GTM-7987)
- When FREEZE -ONLINE is in place, MUPIP SET fails for any command that requires a write to a frozen database file, and does so with the OFRZACTIVE warning message. In such circumstances, the operator must release the freeze before performing a standalone operation. The purpose of the FREEZE -ONLINE is to provide a consistent state for a storage-based backup without impacting on-line users. (GTM-8772)
- MUPIP SET flushes only the database file header for the following non-standalone operations: Epoch taper, Flush time, Hard spin count, Inst freeze on error, Sleep spin count and Spin sleep mask. During FREEZE -ONLINE, MUPIP SET rejects non-standalone operations and issues GTM-W-OFRZACTIVE. In such circumstances, the operator must release the freeze before performing the operation. Previously, MUPIP SET flushed all buffers for non-standalone operations and implicitly released the freeze during FREEZE -ONLINE. (GTM-8784) ❌
- GT.M provides an appropriate error message which indicates a non-zero exit status returned from the script specified by gtm_prodstuckexec (or other GT.M driven script). Previously, GT.M issued a system call error with an inappropriate errno. (GTM-8793)
- MUPIP SET -JOURNAL -BUFFSIZE accepts values up to 1Mi 512-byte blocks, corresponding to 16GiB; previously the maximum 32Ki, corresponding to 16 Mib. (GTM-9147) ✓
- Processes exit appropriately after using a -READ_ONLY database such as \$gtm_dist/gtmhelp.dat; previously they could produce a harmless, but concerning, SYSCALL syslog entry mentioning semctl (IPC_RMID, ...) (GTM-9252)
- MUPIP LOAD provides appropriate error messages when the record after the header of a FORMAT=GO or ZWR file is missing or invalid; previously, in such cases MUPIP issued GTM-E-MAXSTRLEN. (GTM-9275)
- MUPIP REPLIC -SOURCE -SHUTDOWN performs a shutdown for a Source Server that keeps retrying to establish a connection with the Receiver Server, but keeps failing due to a network address resolution error. Also, the Source Server provides more timely recognition and recovery from transient network problems. Previously, MUPIP REPLIC -SOURCE -SHUTDOWN timed out in such a situation. (GTM-9289)

Other

- GT.M attempts to correctly time stamp journal files in cases where system load delays the exiting logic of an exiting process. Previously such cases could cause slightly out-of-order time stamps. (GTM-8878)
- GT.M Utility programs (DSE, LKE, MUPIP) output more details when there is a failure to communicate with gtmsecshr and when an unexpected error causes GT.M to open a database file as read-only. (GTM-9278)
- GT.M handles long M filenames appropriately; previously filenames longer than 32 bytes (not including the path or the .m extension) caused a buffer overflow, possibly resulting in improper timer operation or a segmentation violation (SIG-11). (GTM-9288)
- %YGBLSTAT NEWs local variables d and x to protect the user environment; previously it did not. (GTM-9311)

Error and Other Messages

DBFILERONLY

DBFILERONLY, The database file ffff was opened as read-only (perms pppp)

All GT.M Components Error: Database file ffff was opened read-only with permissions pppp, but the read-only status is inconsistent with application expectations.

Action: Use the error and any follow-on messages to assess whether or not the read-only status is correct or the rejection is appropriate.

EXITSTATUS

EXITSTATUS, Unexpected process exit (xxxx), exit status aaaa -- called from module yyyy at line zzzz

Run Time Error: Indicates a non-zero exit status aaaa returned from a process started in the context of xxxx. The following are common values (other values are possible depending on the script called) and descriptions for the exit status: 1-"Catchall for general errors", 2-"Misuse of shell builtins", 126-"Command invoked cannot execute", 127-"Command not found", 128-"Invalid argument to exit" and 130-"Script terminated by Control-C".

Action: Use the exit status aaaa to adjust the script causing the unexpected exit.

EXTRFMT

EXTRFMT, Extract error: invalid record format - no records found.

MUPIP Error: This indicates that LOAD could not process the sequential output file because the record after the header is invalid.

Action: Verify the file has a valid format and actually contains records.
