GT.M

Release Notes

V7.1-011

Empowering the Financial World

FIS

Contact Information

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

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

Legal Notice

Revision History		
Revision 1.0	09 April 2026	V7.1-011

Table of Contents

GT.M Release Notes

V7.1-011

Overview

V7.1-011 includes a performance improvement for some types of indirection better memory efficiency for compilations and a number of other fixes and minor enhancements. This release is intended for use on more contemporary versions of Linux - please see the Platform section for specifics.

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

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

Note

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

Conventions

This document uses the following conventions:

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

Note

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

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

✅ denotes a new feature that requires updating the manuals.

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

⛔ denotes deprecated messages.

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

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

Platform	Supported Versions	Notes
IBM Power Systems AIX	7.2 TL 5, 7.3 TL 4	Only 64-bit versions of AIX with POWER9 as the minimum required CPU architecture level are Supported. AIX 7.1 and POWER7/8 support was dropped in V7.1-008. 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. 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. Beginning with V7.1-008, users must install the libc++.rte package to use UTF-8 mode support on AIX.
x86_64 GNU/Linux	Red Hat Enterprise Linux 9.7 and 10.1; Ubuntu 22.04 LTS and 24.04 LTS; Amazon Linux 2023	GT.M should also run on recent releases of other major Linux distributions with a contemporary Linux kernel (5.14 or later), glibc (version 2.34 or later) and ncurses (version 6.2 or later). As of V7.1-001, GT.M on x86-64 requires hardware/virtualized support for AVX instructions. **glibc 2.41+** GT.M versions V7.1-000 and older are incompatible with executable stack restrictions introduced in glibc 2.41. GT.M versions V7.1-001 and newer are compatible. For example, this affects Ubuntu 25.04 and up. RHEL 9, RHEL 10 and Amazon Linux 2023 are unaffected. **glibc 2.36+** GT.M versions V6.2-001 and older are incompatible with glibc 2.36 on Linux/x86_64 systems without AVX2 support. GT.M versions V6.2-002 and newer are compatible. This interaction was discovered during

Platform	Supported Versions	Notes
		routine testing. This can cause segmentation violations (SIG-11) in processes performing concurrent updates to the same database block, which terminate the process, but do not damage the database. The issue is due to the way glibc performs certain memory operations when using SSE2 instructions. The glibc behavior was subsequently modified to avoid this issue, and the change was included in glibc 2.37. Linux/x86_64 systems with support for AVX2 instructions are not vulnerable, as glibc chooses its AVX2 implementation, when available, over its SSE2 implementation, and the problematic behavior is specific to SSE2. Note, depending on how CPU virtualization is configured, that virtual environments may not support AVX2 even if the underlying hardware does. **glibc 2.24+** GT.M versions V6.1-000 and older are incompatible with glibc 2.24 and up due to build optimization and library incompatibilities. 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, Debian, and Ubuntu. 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. **libconfig 1.4+ is required** To support the optional WRITE /TLS fifth argument (the ability to provide / override options in the tlsid section of the encryption configuration file), the reference implementation of the encryption plugin requires libconfig 1.4.x or later. **File Systems:** Only the ext4 and xfs filesystems are Supported. Other filesystems are Supportable, but not Supported. Furthermore, if you use the NODEFER_ALLOCATE feature, FIS strongly recommends that you use xfs. If you must use NODEFER_ALLOCATE with ext4, you must ensure that your kernel includes commit d2dc317d564a46dfc683978a2e5a4f91434e9711 (search for d2dc317d564a46dfc683978a2e5a4f91434e9711 at https://www.kernel.org/pub/linux/kernel/v4.x/ChangeLog-4.0.3). The Red Hat Bugzilla identifier for the bug is 1213487. With NODEFER_ALLOCATE, do not use any filesystem other than ext4 and a kernel with the fix, or xfs.  **Note** FIS recommends recompiling the reference encryption plugins to match the target platform. See Compiling the Reference Implementation Plugin section for instructions. OpenSSL 3.0 by default does not allow client-side initiated TLSv1.2 renegotiation requests due to potential DoS attacks. Because of this, the reference TLS implementation in GT.M versions before V7.0-004 do not use the appropriate OpenSSL 3.0 API to enable support for client-side initiated TLSv1.2 renegotiation. Customers needing to replicate to/from GT.M versions before V7.0-004 with OpenSSL 3.0 must use -RENEGOTIATE_INTERVAL=0 in the Source Server

Platform	Supported Versions	Notes
		startup. This limitation only affects database replication and not SOCKET devices.

Important

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

Platform support lifecycle

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

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

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

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

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

GT.M as Open Source Software (OSS)

FIS maintains and releases GT.M on Linux as OSS. GT.M does not include any OSS libraries.

However, using some GT.M capabilities activates APIs that require the user make some OSS software available:

- Compression: zlib
- Encryption: libconfig and openssl (or equivalent as determined by the encryption plugin); key management is the user's responsibility
- UTF-8 mode: libicuio

while those are what FIS tests with, as long as the API is compatible, substitutions should work.

Note

Linux distributions include various OSS components some of which GT.M relies on.

Additional Installation Instructions

To install GT.M, see the "Installing GT.M" section in the GT.M Administration and Operations Guide. For minimal down time, upgrade a current replicating instance and restart replication. Once that replicating instance is current, switch it to become the originating instance.

Upgrade the prior originating instance to become a replicating instance, and perform a switchover when you want it to resume an originating primary role.

Caution

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

- FIS strongly recommends installing each version of GT.M in a separate (new) directory, rather than overwriting a previously installed version. If you have a legitimate need to overwrite an existing GT.M installation with a new version, you must first shut down all processes using the old version. FIS suggests installing GT.M V7.1-011 in a Filesystem Hierarchy Standard compliant location such as /usr/lib/fis-gtm/V7.1-011_arch (for example, /usr/lib/fis-gtm/V7.1-011_x86_64 on Linux systems). A location such as /opt/fis-gtm/V7.1-011_arch would also be appropriate.
- Use the appropriate MUPIP command (e.g. ROLLBACK, RECOVER, RUNDOWN) of the old GT.M version to ensure all database files are cleanly closed.
- Make sure gtmsecshr is not running. If gtmsecshr is running, first stop all GT.M processes including the DSE, LKE and MUPIP utilities and then perform a **MUPIP STOP*pid_of_gtmsecshr***.
- Starting with V6.2-000, GT.M no longer supports the use of the deprecated $gtm_dbkeys and the master key file it points to for database encryption. To convert master files to the libconfig format, please click to download the CONVDBKEYS.m program and follow instructions in the comments near the top of the program file. If you are using $gtm_dbkeys for database encryption, please convert master key files to libconfig format immediately after upgrading to V6.2-000 or later. Also, modify your environment scripts to include the use of gtmcrypt_config environment variable.

Recompile

- Recompile all M and C source files.

Rebuild Shared Libraries or Images

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

Compiling the Reference Implementation Plugin

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

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

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

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

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

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

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

Re-evaluate TLS configuration options

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

Upgrading to V7.1-011

Before you begin

GT.M supports upgrade from V5*, V6.* and V7.* versions to V7.1-011.

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

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

GT.M upgrade procedure for V7.1-011 consists of 5 stages:

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

Before starting, read the upgrade instructions of all stages carefully. Your upgrade procedure for GT.M V7.1-011 depends on your GT.M upgrade history and your current version.

Stage 1: Global Directory Upgrade

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

To upgrade from any previous version of GT.M:

- Open your Global Directory with the GDE utility program of GT.M V7.1-011.
- Execute the EXIT command. This command automatically upgrades the Global Directory.
- If you inadvertently open a Global Directory of an old format with no intention of upgrading it, execute the QUIT command rather than the EXIT command.

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

- Open the global directory with the GDE utility program of V7.1-011.
- Execute the SHOW -COMMAND -FILE=file-name command. This command stores the current Global Directory settings in the file-name command file. If the old version is significantly out of date, edit the command file to remove the commands that do not apply to the old format. Alternatively, you can use the output from SHOW -ALL or SHOW -COMMAND as a guide to manually enter equivalent GDE commands for the old version.

An analogous procedure applies in the reverse direction.

Stage 2: Database Files Upgrade

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

There are three upgrade paths available when you upgrade to V7.1-011.

V7 Upgrade Path 1: In-place Upgrade

To upgrade from GT.M V7*:

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

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

There are two phases to upgrade from V6 to V7:

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

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

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

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

Phase 1: Standalone MUPIP UPGRADE

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

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

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

UPGRADE:

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

At this point, after a successful MUPIP UPGRADE:

- All DT blocks are in V7m format and all GVT index blocks remain in V6/V6p format
- Subsequent activity that updates index blocks for existing GVTs implicitly converts any V6 index blocks to V6p format after applying the offset
- No process other than MUPIP REORG -UPGRADE converts GVT index blocks from V6p format to V7m format; in other words, adding new nodes does not create GVT index blocks with V7 format - adding new nodes splits existing index blocks and such block splits retain the pre-existing block format
- Newly created GVTs, storing new global names, have V7m format
- Data blocks, at level zero (0), and local bit map blocks have the same format in V6 and V7, so, for consistency, normal updates also give those blocks a V7m format designation

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

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

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

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

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

MUPIP REORG -UPGRADE:

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

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

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

- MUPIP REORG

  Because this operation may not visit every block in the database it may fail to upgrade static/unchanging blocks
- MUPIP REORG -ENCRYPT
- MUPIP INTEG -TN_RESET

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

Failure to perform Phase 3 has **NO** implications for V7.1-011 but might be an issue for any as-yet unplanned further enhancement.

Important

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

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

1. Offline Upgrade instance to use new GT.M V7.1-002+ version - at this point, customers can use the upgraded the GT.M version without any DB changes

2. Online MUPIP SET -INDEX_RESERVED_BYTES=n - where n is 1/3 the block size

3. Online MUPIP REORG -MIN_LEVEL=1 -NOSWAP - free up space in all index blocks to ease the block reference change from 32bits (4bytes) to 64bits (8bytes); this operation alters only index blocks (-MIN_LEVEL=1), and so generates a much lower volume of before image journal records.

4. Offline MUPIP UPGRADE -move blocks around to make space for the expanded master bitmap and upgrade the index blocks in the directory tree (tree of Global names).

5. Online MUPIP REORG -UPGRADE - upgrade the remaining index blocks

6. Online MUPIP SET -INDEX_RESERVED_BYTES=0 - remove the previously applied reservation as it is no longer needed; some application may find it produces a continuing performance benefit.

7. (optional) Online REORG -MIN_LEVEL=1 -NOSWAP -NOSPLIT - coalesce the index blocks to leave index blocks in a less fragmented state

V7 Upgrade Path 2: EXTRACT and LOAD

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

- MUPIP EXTRACT -FREEZE followed by a MUPIP LOAD

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

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

- MERGE command with two global directories and Extended References

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

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

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

V7 Upgrade Path 3: No change

Continue using your V6 databases with GT.M V7.1-011. In case you do not wish to operate with files of differing format, specify the -V6 qualifier when invoking MUPIP CREATE.

Choosing the right upgrade path

Choose V7 Upgrade Path 1 or 2 if you anticipate a database file to grow to over 994Mi blocks or require trees of over 7 levels as V7.1-011 supports 16Gi blocks and 11 levels. Note that the maximum size of a V7 database file having 8KiB block size is 114TiB (8KiB*16Gi).

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

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

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

Database Compatibility Notes

- Changes to the database file header may occur in any release. GT.M automatically upgrades database file headers as needed. Any changes to database file headers are upward and downward compatible within a major database release number, that is, although processes from only one GT.M release can access a database file at any given time, processes running different GT.M releases with the same major release number can access a database file at different times.
- Databases created with V5.3-004 through V5.5-000 can grow to a maximum size of 224Mi (234,881,024) blocks. This means, for example, that with an 8KiB block size, the maximum database file size is 1,792GiB; this is effectively the size of a single global variable that has a region to itself and does not itself span regions; a database consists of any number of global variables. A database created with GT.M versions V5.0-000 through V5.3-003 can be upgraded with the V5 version of MUPIP UPGRADE to increase the limit on database file size from 128Mi to 224Mi blocks.
- Databases created with V5.0-000 through V5.3-003 have a maximum size of 128Mi (134, 217,728) blocks. GT.M versions V5.0-000 through V5.3-003 can access databases created with V5.3-004 and later as long as they remain within a 128Mi block limit.
- Database created with V6.0-000 through V6.3-014 have a maximum size of 1,040,187,392 (992Mi) blocks.
- Database created with V7.0-000 and up have a maximum size of 17,112,825,856 (~16Gi) blocks.

Stage 3: Replication Instance File Upgrade

GT.M V7.1-011 does not require new replication instance files when upgrading from any version after V6.0-000.

Stage 4: Journal Files Upgrade

On every GT.M upgrade:

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

Important

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

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

Stage 5: Trigger Definitions Upgrade

GT.M V7.1-011 does not require trigger definition upgrade when upgrading GT.M from any version after V6.3-000. If upgrading from a prior GT.M release, please see the instructions in the release notes for V6.3-014.

Managing M mode and UTF-8 mode

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

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

and an object file, and the object predates the source file and was generated with the same setting of $gtm_chset/$ZCHset. A GT.M process generates an error if it encounters an object file generated with a different setting of $gtm_chset/$ZCHset than that processes' current value.

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

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

- Actual files for GT.M executable programs (mumps, mupip, dse, lke, and so on) are in the parent directory, that is, the location specified for installation.
- Object files for programs written in M (GDE, utilities) have two versions - one compiled with support for UTF-8 mode in the utf8 subdirectory, and one compiled without support for UTF-8 mode in the parent directory. Installing GT.M generates both versions of object files, as long as ICU 3.6 or greater is installed and visible to GT.M when GT.M is installed, and you choose the option to install UTF-8 mode support. During installation, GT.M provides an option that allows placing the object code in shared libraries in addition to individual files in the directory.
- The utf8 subdirectory has files called mumps, mupip, dse, lke, and so on, which are relative symbolic links to the executables in the parent directory (for example, mumps is the symbolic link ../mumps).
- When a shell process sources the file gtmprofile, the behavior is as follows:
  - If $gtm_chset is "m", "M" or undefined, there is no change from the previous GT.M versions to the value of the environment variable $gtmroutines.
  - If $gtm_chset is "UTF-8" (the check is case-insensitive),
    - $gtm_dist is set to the utf8 subdirectory (that is, if GT.M is installed in /usr/lib/fis-gtm/gtm_V7.1-011_i686, then gtmprofile sets $gtm_dist to /usr/lib/fis-gtm/gtm_V7.1-011_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 sect1 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

V7.1-011

Fixes and enhancements specific to V7.1-011:

Id	Prior Id	Category	Summary
GTM-1499	GTM-F132851	Admin	DSE dump -fileheader command displays the "Starting VBN" value in hexadecimal format
GTM-7245	GTM-DE200522	Admin	MUPIP INTEG -TN_RESET now avoids reporting DBTNTOOLG errors on freshly created database.
GTM-8251	GTM-F134792	Language	Support for 36 arguments for external call-outs and call-in, and 32 for GTMJI
GTM-8623	GTM-F135001	Admin	MUPIP REPLIC -SOURCE -FREEZE=OFF flushes all buffers when there are dirty buffers and no flush timers,
GTM-9139	GTM-F135287	Language	Improve performance for some cases of indirection
GTM-10925		Other	Prevent possible segmentation violation (SIG-11) at process startup
GTM-11236		DB	Prevent updates between MUPIP UPGRADE and MUPIP REORG -UPGRADE from potentially damaging an MM database
GTM-11433		Admin	Revise MUPIP SET -TRIGGER_FLUSH_LIMIT boundries and default

Id	Prior Id	Category	Summary
GTM-11467		Language	Detect explicitly invalid $ORDER() direction argument at compile time
GTM-11488		DB	Avoid incorrect permission on AUTODB files, e.g. statsDB
GTM-11501		Admin	MUPIP INTEG -FILE fixes incorrect blocks to upgrade counter in file header and logs DBBTUFIXED
GTM-11503		Admin	Ensure system installer does not mishandle ownership and permissions for GT.M configure
GTM-11506		Language	Improve performance for some cases of indirection - see GTM-9139
GTM-11518		Language	Fix handling of literal 524.286 in odd circumstances
GTM-11544		Language	Ensure concatenation of all integers can reach maximum string length
GTM-11549		DB	Improve parsing of trigger options strings
GTM-11602		Other	GT.M uses less memory when compiling routines
GTM-11611		Language	On AIX, GT.M appropriately compiles unusually long jumps in Boolean expressions

Database

- GT.M appropriately handles MM databases in the period between a MUPIP UPGRADE and the completion of a MUPIP REORG -UPGRADE. Previously, concurrent activity could cause GT.M to adjust the version of an MM index block without upgrading the pointers inside the block. This issue did not affect BG databases. The workaround was to prevent update activity on any MM database between the UPGRADE and the REORG -UPGRADE. (GTM-11236)

- GT.M processes appropriately manage permissions on AUTODB databases, such as statsDBs. Starting with V7.1-006, when two processes sharing the same user id tried to concurrently create and initialize the AUTODB file, and set up its shared memory, the process was subject to a race condition which could leave the associated shared memory with inappropriate (0600) permissions, rather than the permissions matching the database file. Processes of other user ids attempting to access the database could encounter errors. The workaround was to start a single process to perform the creation and initialization of the database before permitting access at scale and ensuring no interval where there all processes accessing the database have exited. The most common case involved statsDB and led to STATSDBERRs in the syslog and disabling of stats for affected processes. For statsDBs, the workaround was to ensure that one process initialized statsDBs by performing a view "STATSHARE":"*" before there is competition from other processes and by keeping this process alive for the life of the instance or by "passing the "baton" to another process which survives it. Users should ensure that at least one process with access to the statsDBs remains alive to prevent statsDB deletion and a recurrence of the race condition when the next process with interest in stats starts. (GTM-11488)

- GT.M parses database trigger "options" string correctly. Previously, options string parsing could fail due to improper buffer termination. This problem was only seen in development and not reported by a customer. The workaround was to repeat the trigger load. (GTM-11549)

Language

- GT.M now supports 36 parameters in call-outs to C and call-ins from C. In addition, the GTMJI plugin supports 32 parameters to Java external calls in addition to the required "class" and method parameters. Previously GT.M supported 32 arguments to external call-outs and call-ins, but GTMJI could only use 29 of these as general purpose arguments to Java methods (GTM-8251)

- GT.M Provides fast path handling of argument indirection for local and global variables, $[Z]DATA(), $INCREMENT(), single-argument $ORDER(), $QUERY(), and $ZPREVIOUS() intrinsic functions, and the HANG, KILL, READ, and WRITE operations. This enhancement

streamlines the internal evaluation of GLVN (Global and Local Variable Name) indirection, resulting in performance improvements of more than twofold. Previously, GT.M incurred additional runtime overhead when resolving indirection references for these intrinsic functions and operations. (GTM-9139)

- $ORDER() compilation detects an explicitly invalid second, direction specifying, argument during compilation. A change in V6.3-014 caused such a case to only show up at run time. (GTM-11467)
- Please see GTM-9139 (GTM-11506)
- The GT.M compiler appropriately handles a literal 524.286; previously, if a potentially timed operation appeared on a line without a timeout before the use of literal 524.286 and there were limited intervening literals, the compiler failed to transfer this unique value to the object file, which caused an invalid address (possibly a segmentation violation / SIG-11) at run-time. the workaround was to present the value as (say) ""_524.286. (GTM-11518)
- Concatenation appropriately detects when its arguments would exceed the maximum string length (currently 1MiB). Note that UTF-8 characters may require multiple bytes. Previously constructing a string consisting only of concatenated integers terminated with a premature (64 bytes early) MAXSTRLEN. The workaround was any action to ensure the expression underwent at least one operation other than concatenation that treated the accumulation as a string prior to reaching a length of 2*20-64. (GTM-11544)
- On AIX, GT.M appropriately handles conditional jumps in generated code using highly complex Boolean expressions which transfer more than approximately 8Ki native machine instructions forward or backward. Previously, the GT.M compiler generated incorrect code for such jumps and could cause unintended transfers of control; for example, entry into an argumentless DO block with an extremely long FALSE postconditional. This was discovered in our test environment, and we have no record of this reported by any customer and no examples of naturally generated code which trigger the issue. [AIX] (GTM-11611)

System Administration

- To maintain consistency, DSE DUMP -FILEHEADER command displays the "Starting VBN" value in hexadecimal format; previously, the DSE DUMP -FILEHEADER command displayed the "Starting VBN" value in decimal format. (GTM-1499)
- MUPIP INTEG -TN_RESET appropriately handles a freshly created database; previously, in such a case, it incorrectly reported DBTNTOOLG errors. The workaround was to avoid the MUPIP INTEG -TN_RESET as a newly created database has no transactions to reset. (GTM-7245)
- MUPIP REPLIC -SOURCE -FREEZE=OFF, when there are dirty buffers and no flush timers, flushes all dirty buffers to ensure durability. Previously, after removing the instance freeze under those unusual circumstances, GT.M could allow dirty buffers to remain unflushed indefinitely when no flush timers were present and no further updates were performed on the region. (GTM-8623)
- MUPIP SET -TRIGGER_FLUSH_LIMIT accepts a minimum of the greater of one-thirty-second, 64 and one-eighth of the global buffers, and a maximum and default of the lesser of fifteen-sixteenths and 2**18 (.25 Mi) dirty global buffers. Previously, the minimum was one-eighth, and the maximum and default were fifteen-sixteenth of the number of global buffers. (GTM-11433)
- If MUPIP INTEG -FILE fixes incorrectly high blocks to upgrade counter in the file header, it prints an informational DBBTUFIXED message; Previously it fixed blocks to upgrade counter quietly without printing the informational message. (GTM-11501)
- The configure script does not use the system provided install executable to set ownership and permissions. This is a workaround for an issue in uutils/coreutils/install 0.2.2, which does not set the right permissions. This was only seen in our development environment and not reported by any clients. (GTM-11503)

Other

- GT.M prevents M-stack initialization from being interrupted. Previously, it was not protected which could result in a segmentation violation (SIG-11). This was only seen in development and not reported by a customer. (GTM-10925)
- GT.M uses less memory when compiling routines and optimizes redundant jumps. Users can expect a memory savings of more than 5% in compilation and a reduction in jump chains in some compiled code. (GTM-11602)

Error and Other Messages

CIMAXPARAM Δ

CIMAXPARAM, Exceeded maximum number of parameters in the call-in table entry. An M routine cannot accept more than 36 parameters.

Call in/Run Time Error: This indicates that the call-in table specified by $GTMCI contains more than 36 parameters. Since an M formallist can only accept up-to 36 parameters, user cannot pass more than 36 arguments to gtm_ci(), excluding <c-call-name> and <ret-type>.

Action: Reduce the number of parameters to be less than 36, in the call-in table as well in the M routine.