

GT.M

Release Notes

V7.2-000

Empowering
the Financial World

FIS

Contact Information

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

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

Legal Notice

Copyright ©2026 Fidelity National Information Services, Inc. and/or its subsidiaries. All Rights Reserved.

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





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

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

Revision History

Revision 1.0	18 June 2026	V7.2-000
--------------	--------------	----------

Table of Contents


GT.M Release Notes	1
V7.2-000	1
Overview	1
Conventions	1
Platforms	2
Additional Installation Instructions	4
Upgrading to V7.2-000	6
Managing M mode and UTF-8 mode	11
Setting the environment variable TERM	12
Installing Compression Libraries	13
Change History	13
V7.2-000	13
Database	14
Language	14
System Administration	16
Other	16
Error and Other Messages	17
FALLINTOFLST 	17
TPTIMEOUT 	17
TRIGNOXPEL 	17
XPELNOTP 	17


GT.M Release Notes

V7.2-000

Overview

V7.2-000 includes performance improvements in heap management and capturing database statistics as well as a number of operational improvements and other fixes. This release is intended for use on 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.

⚠️ 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	<p>Only 64-bit versions of AIX with POWER9 as the minimum required CPU architecture level are Supported.</p> <p>AIX 7.1 and POWER7/8 support was dropped in V7.1-008.</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>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> <p>Beginning with V7.1-008, users must install the libc++.rte package to use UTF-8 mode support on AIX.</p>
x86_64 GNU/Linux	<p>Red Hat Enterprise Linux 9.7 and 10.1;</p> <p>Ubuntu 22.04 LTS, 24.04 LTS and 26.04 LTS;</p> <p>Amazon Linux 2023</p>	<p>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.</p> <p>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.</p> <p>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</p>

Platform	Supported Versions	Notes
		<p>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.</p> <p>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.</p> <p>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.</p> <p>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.</p> <div data-bbox="753 1356 837 1440" style="text-align: center;"> </div> <p>Note</p> <p>FIS recommends recompiling the reference encryption plugins to match the target platform. See Compiling the Reference Implementation Plugin section for instructions.</p> <p>OpenSSL 3.0 by default does not allow client-side initiated TLSv1.2 renegotiation requests due to potential DoS attacks. Because of this, the reference TLS implementation in GT.M versions before V7.0-004 do not use the appropriate OpenSSL 3.0 API to enable support for client-side initiated TLSv1.2 renegotiation. Customers needing to replicate to/from GT.M versions before V7.0-004 with OpenSSL 3.0 must use <code>-RENEGOTIATE_INTERVAL=0</code> in the Source Server</p>

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 V7.2-000 in a Filesystem Hierarchy Standard compliant location such as `/usr/lib/fis-gtm/V7.2-000_arch` (for example, `/usr/lib/fis-gtm/V7.2-000_x86_64` on Linux systems). A location such as `/opt/fis-gtm/V7.2-000_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 `gtmsechr` is not running. If `gtmsechr` is running, first stop all GT.M processes including the DSE, LKE and MUPIP utilities and then perform a **MUPIP STOPpid_of_gtmsechr**.
- 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 `gtmencrypt_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 `libcrypt`, `libpgme`, `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 libcrypt11-dev libpgme11-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 `libcrypt-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 V7.2-000



Before you begin

GT.M supports upgrade from V5*, V6.* and V7.* versions to V7.2-000.

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 V7.2-000 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 V7.2-000 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 V7.2-000.
- 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 V7.2-000.
- 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 V7.2-000.

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>}
```

GT.M Release Notes

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 V7.2-000 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.

GT.M Release Notes

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 V7.2-000. 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 V7.2-000 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

GT.M Release Notes

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 V7.2-000 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 V7.2-000 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

GT.M Release Notes

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_V7.2-000_i686, then gtmprofile sets \$gtm_dist to /usr/lib/fis-gtm/gtm_V7.2-000_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(xen1), key_backspace(kbs),
key_dc(kdch1),key_down(kcud1), key_left(kcub1), key_right(kcuf1), key_up(kcuu1), key_insert(kich1),
keypad_local(rmkx),keypad_xmit(smkn), lines(lines).
```

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

Installing Compression Libraries

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

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

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

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

Change History

V7.2-000

Fixes and enhancements specific to V7.2-000:

Id	Prior Id	Category	Summary
GTM-5632	GTM-DE195950	Language	Option to preserve terminal settings with external calls and call-ins 🟢
GTM-5647	GTM-DE195950	Admin	The source server reports "Msg Total" in its log to include all types of messages sent to the receiver
GTM-6587	GTM-DE196294	Admin	MUPIP INTEG better detects nondatabase files
GTM-7264	GTM-F134235	Admin	🔴 MUPIP FTOK prints the file name as the last field 🟢
GTM-9154	GTM-DE201258	Admin	The Source Server correctly displays the journal filename and records journal file switches in its log file
GTM-9664	GTM-DE270934	Admin	Correct error details reported with JNLEXTEND and JNLACCESS messages
GTM-9763	GTM-DE519651	Admin	MUPIP REPLIC -SOURCE/-RECEIVER -CHANGELOG issues REPLLOGOPN when the server process cannot open the new log file
GTM-10734		Language	Handle errors during SOCKET CLOSE
GTM-10996		Language	Improved heap management
GTM-11238		Other	gtm_icu_version only required to select a particular ICU library
GTM-11275		Admin	DSE dump -block=<n> command displays block or bitmap status while dumping blocks 🟢

GT.M Release Notes

Id	Prior Id	Category	Summary
GTM-11299		Admin	MUPIP INTEG, when waiting for kill-in-progress to clear, proceeds as soon as the kill-in-progress clears
GTM-11346		Language	TRESTART recognizes an optional argument XPEL=label 🟢
GTM-11517		Language	More appropriate handling of ZSTEP actions
GTM-11571		Language	OPEN or USE for a SOCKET device with a CONNECT deviceparameter completes at connection
GTM-11584		Other	Eliminate a memory leak associated with local variable subscripts in triggers
GTM-11600		Admin	Correct supplementary replication stream history handling after consecutive online rollbacks on the root primary
GTM-11631		Language	Handling of NOEXTRINSIC error prevents inappropriate access to stacked local variables
GTM-11635		Language	Prevent inappropriate NUMOFLOW errors associated with deviceparameters
GTM-11639		Admin	Running down a read-only database with argumentless MUPIP RUNDOWN removes associated semaphores.
GTM-11646		Language	Protect encrypted-split-principal encryption strings against garbage collection
GTM-11670		Language	Prevent inappropriate FALLINTOFLST error after an embedded DO block
GTM-11677		DB	Efficient handling of heavyweight database shared statistics
GTM-11687		Language	\$QUERY(lvn) indirection improvement
GTM-11700		Language	Appropriate enforcement of the INDRMAXLEN limit for certain indirections
GTM-11709		Language	ACIDTP on an external call respects \$ZMAXTPTIME
GTM-11719		DB	Better handling of concurrent attempts to initialize autodb files

Database

- At the cost of temporary counter approximation / variance, GT.M efficiently manages heavyweight database shared statistics, such as the number of gets, block reads, and block writes. Internally, GT.M updates these heavyweight statistics every 32 increments by a given process, meaning the statistic should now be considered accurate within a range of $\pm(32 * n_processes)$ or in most cases having a precision of around ± 32 . All updates are flushed on ordinary process exit. (GTM-11677)
- GT.M processes competing against each other to initialize an autodb correctly handle errors in that process and more reliably attach to the eventually created database after encountering it in the process of being created. Previously, there were certain concurrency windows in which processes could fail to attach to the autodb or encounter a segmentation violation (SIG-11). (GTM-11719)

Language

- A new keyword (TERMIO) may be added to the external call table entries to request the terminal attributes be reset whenever control is given to that external routine. This is only needed if the external routine does terminal input or output. A new variant of the Call-In initialization routine gtm_init (gtm_init_extra) is available to enable similar control of terminal attributes. In addition to the existing

GT.M Release Notes

function of `gtm_init`, `gtm_init_extra` takes a flag argument to enable the resetting of terminal attributes before GT.M returns to the calling program from `gtm_init_extra` and `gtm_exit`. The value of the flag controls when and how the attributes are reset on return from `gtm_ci`, `gtm_cip`, and `gtm_zstatus`. Previously, GT.M would set the terminal attributes it needs at startup (the `gtm_init()` call for the Call-In case) and reset the attributes on exit (the `gtm_exit()` call for the Call-In case.) Among other settings, GT.M would disable echo and `icrnl` (convert carriage return on input to line feed) which could appear to have disabled terminal input. (GTM-5632) 🟢

- When CLOSE of a SOCKET device encounters an error, say due to buffered write failures, `$DEVICE` reflects the error, and with `IOERROR="TRAP"` GT.M invokes the application's error handler. The application error handling code should repeat the CLOSE to avoid leaving the socket in a CLOSE-WAIT state. Previously, SOCKET CLOSE errors could retain stale information from prior I/O operations in `$DEVICE` rather than reflecting the CLOSE outcome, and with `IOERROR="TRAP"` the process could hang. (GTM-10734)
- GT.M includes a redesigned mechanism for managing the heap (string pool) efficiently. The new mechanism persists string state in-between each heap compaction ("garbage collection") to avoid the need to re-sort persistent data each time. Users can expect less costly garbage collection, especially in workloads with many persistent entries on the m-stack, in the indirection cache, or in local variable trees. (GTM-10996)
- TRESTART recognizes an option argument, `XPEL=label`, specifying GT.M perform a RESTART of the entire transaction, a ROLLBACK leaving `$TLEVEL=0`, a DO using the LABELREF as the DO argument, and, when the DO completes, reinitiate the transaction starting with the original (outer-most) TSTART as if for the first time with `$TRESTART=0`. Attempting to activate this feature on a transaction that is not restartable produces a TRESTNOT error. Attempting to use XPEL while in a trigger produces a TRIGNOXPEL error. GT.M evaluates a literal XPEL value at compile time to ensure its validity. If DO of the specified code fails, the failure invokes the current error handling logic. A transfer of control by a (non-FOR terminating) QUIT, including an implicit QUIT due to error handling, or a ZGOTO that abandons the stack level associated with the TSTART within which the XPEL originated terminates the XPEL (and the already rolled back transaction). This feature includes the read-only `$ZINXPEL` ISV which provides an integer counter analogous to `$TRESTART`. A non-zero `$ZINXPEL` in combination with a zero (0) `$TLEVEL` indicates operation within the scope of the XPEL DO. Attempting a TSTART within the scope of XPEL logic produces a XPELNOTP error. (GTM-11346) 🟢
- ZSTEP preserves actions under most circumstances; previously it sometimes lost actions, for example on routine replacement (ZLINK or ZRUPDATE) or when handling a `$ZINTERRUPT`, which could cause a segmentation violation (SIG-11). (GTM-11517)
- OPEN or USE for a SOCKET device with a CONNECT deviceparameter completes at the time of connection. Previously, socket connection incurred a fixed 100 millisecond wait even when the connection completed immediately. (GTM-11571)
- GT.M delivers the NOEXTRINSIC error appropriately at the M stack level of the problematic QUIT; previously it mismanaged the error delivery so a lower level could retain inappropriate access to some local variables. The workaround was to code correctly and avoid the error. This was only seen in development and not reported by a customer. (GTM-11631)
- The GT.M compiler appropriately parses deviceparameters. Previously, due to a regression in V7.1-007, in some cases GT.M could issue spurious NUMOFLOW errors while parsing deviceparameters. This could occur when a literal string which would produce a NUMOFLOW if coerced to a number was passed to a deviceparameter expecting a string (and not necessarily a number), or without a literal deviceparameter argument in certain cases where the most recently encountered string literal would have produced a NUMOFLOW under the same conditions. The workaround is to ensure the most recently encountered literal at compile or indirection time would not produce a NUMOFLOW. (GTM-11635)
- GT.M correctly handles encrypted output to a split \$PRINCIPAL device where stdout and stdin are different. Previously, GT.M did not protect certain encryption strings against garbage collection in this case, which could lead incorrectly encrypted output or a segmentation violation (SIG-11). This was only seen in development and not reported by a customer. (GTM-11646)
- The GT.M compiler avoids embedding a FALLINTOFLST error after inserting an (implicit) QUIT after an embedded DO block. Note: the QUIT insertion occurs even if the line containing the associated argumentless DO includes an effective terminating QUIT or HALT, and the insertion produces a warning (if enabled). Previously in such a pattern, the compiler inappropriately disabled the formalist label by also embedding a run-time FALLINTOFLST error. The workaround was to explicitly terminate the embedded subroutine with a QUIT. (GTM-11670)
- GT.M avoids compiling \$QUERY() indirection for a local variable under certain conditions. Previously, iterating with indirection across 1Mi local array/variable with \$QUERY, was approximately 15% times slower. (GTM-11687)

GT.M Release Notes

- GT.M reports INDRMAXLEN errors for keys exceeding the 8192-byte maximum indirection length for \$[Z]DATA, \$INCREMENT(), single-argument \$ORDER(), \$QUERY(), and \$ZPREVIOUS intrinsic functions, and the HANG, KILL, READ, and WRITE operations. In V7.1-011, GT.M did not issue INDRMAXLEN error messages for these operations in edge cases, allowing processing of overlength keys to continue without the expected error only producing a MAXSTRLEN error if a key exceeded that MiB limit. (GTM-11700)
- External call returns coded with ACIDTP do not check for TPTIMEOUT; in recent versions they did, causing premature TPTIMEOUT errors. (GTM-11709)

System Administration

- The source server correctly reports "Msg Total" in its log to include all types of messages sent to the receiver, including heartbeat messages and protocol handshake messages. Previously, "Msg Total" only counted transaction data and per-transaction message headers, omitting other message types. This made "Msg Total" inconsistent with the receiver server, which correctly counted all received bytes. (GTM-5647)
- MUPIP INTEG reports DBNOTGDS for non database files. Previously, such files could produce misleading encryption or file read errors. (GTM-6587)
- MUPIP FTOK command prints the variable length file name as the last field keeping the other fields always aligned irrespective of the length of the file name. Previously, MUPIP FTOK displayed the file name as the first field. (GTM-7264) 🍏 🍏
- When reading from a journal file, the Source Server correctly displays the journal filename and records journal file switches in its log file; previously it could display garbage characters after the journal filename and did not record the switch. (GTM-9154)
- GT.M correctly reports secondary error details for JNLEXTEND and JNLACCESS messages. Previously, these messages could report inaccurate system error codes. (GTM-9664)
- MUPIP REPLIC -SOURCE -CHANGELOG and -RECEIVER -CHANGELOG appropriately issue a REPLLOGOPN error and continue with the existing log file when the server process cannot open the new log file. Previously, if the new log file became inaccessible between the time the MUPIP CHANGELOG command validated it and the server's subsequent attempt to open it, the server terminated with a segmentation violation (SIG-11). The workaround was to ensure file and directory permissions remained accessible to the server process before issuing the CHANGELOG command. (GTM-9763)
- The DSE DUMP -BLOCK=<n> command, by default, or when used with the -STATUS qualifier (dse dump -block=<n> -status), displays the block or bitmap status while dumping blocks. For normal blocks, the bitmap status displays as free, busy, recycled, or corrupt, whereas for bitmap blocks, the block status displays as freespace or full. When the command includes the -NOSTATUS qualifier (dse dump -block=<n> -nostatus), DSE does not display the block or bitmap status information. While the block or bitmap status could be accessed with the MAP command, previously, the DSE DUMP -BLOCK=<n> command did not display block or bitmap status. (GTM-11275) 🍏
- MUPIP INTEG, when waiting for kill-in-progress to clear, proceeds as soon as the kill-in-progress condition clears. Previously, MUPIP INTEG continued to wait approximately one minute even if the kill-in-progress had cleared. (GTM-11299)
- GT.M appropriately handles supplementary replication after consecutive online rollbacks on the root primary. Previously, back-to-back online rollbacks could cause the receiver server's stream history cache to become stale for streams whose history records were not re-sent on the latest connection, leading the source server on the supplementary propagating primary to fatally exit with a STRMNUMMISMATCH2 error. The workaround was to restart the receiver server on the downstream supplementary instance after consecutive online rollbacks. This was only seen in development and not reported by a customer. (GTM-11600)
- Running down a read-only database with argumentless MUPIP RUNDOWN removes associated semaphores. A change in V7.1-011 aimed at eliminating a spurious error introduced this inappropriate semaphore leakage. (GTM-11639)

Other

- GT.M uses the ICU (international Components for Unicode) library without requiring the gtm_icu_version environment variable set regardless of whether the library is compiled with or without symbol renaming for all supported ICU versions (ie: versions 3.6 and higher). GT.M also directs the ICU library to use GT.M's memory manager. The gtm_icu_version environment variable is still

supported to choose a particular ICU library, and does not have to be removed from working GT.M setups. Previously, GT.M required the `gtm_icu_version` environment variable if the ICU library was compiled with symbol renaming and did not require ICU to use GT.M's memory manager. (GTM-11238)

- GT.M does not leak memory associated with subscripted local variables and aliases in triggered code. Previously, the memory backing any use of subscripted local variables was not freed after the trigger completed, leading to a memory leak for the process performing the trigger. (GTM-11584)

Error and Other Messages

FALLINTOFLST

FALLINTOFLST, Fall-through to a label with formallist is not allowed

Run Time/Compile Time Error: This error indicates that M code reached a label with a formallist by falling through from the previous label. When issued as a warning, it indicates the compiler determined such an error could happen. If the preceding line is the end of an embedded DO block, the compiler issues this error as a warning (if enabled) and ensures the block cannot fall through by explicitly inserting the implicit QUIT. The warning and insert occur even if the line containing the argumentless DO associated with the block contains a terminating QUIT or HALT other than at the end of the block.

Action: Revisit your code to ensure that all invocations of labels with a formallist occur using a DO command or extrinsic function (\$\$).

TPTIMEOUT

TPTIMEOUT, Transaction timeout after rrrr restart(s) and gggg grace period(s) [from zzzz dddd]

Run Time Error: This message indicates that the transaction did not complete within \$ZMAXTPTIME. rrrr is the number of restarts, if any. GT.M divides the time from the initial (outermost) TSTART to the \$ZMAXTPTIME limit into five (gggg) equal grace-period intervals, each ending in a SIGALRM checkpoint, so that a single suppressed SIGALRM costs only one interval rather than terminating the transaction prematurely or letting it hang. The timeout occurs when the cumulative time reaches \$ZMAXTPTIME. An external call that manipulates GT.M's SIGALRM handler can suppress one or more of these signals and cause a premature TPTIMEOUT, so review external calls and external interactions carefully to avoid premature TPTIMEOUT. During external calls, GT.M sends a warning to the syslog as each grace period expires, and after all five expire it delivers an error to both the syslog and the process. The error reports \$ZPOSITION zzzz and \$ZDIRECTORY dddd where GT.M detected the expiration. Timeouts prevent runaway processes, damage from software bugs, and transactions so large as to be hostile to other users.

Action: Review the message(s) for any information about the cause of the transaction timeout, and/or research what the process was doing at the time of the warning or error. If the transaction uses custom programs, routines, or functions, they may need to be debugged.

TRIGNOXPEL

TRIGNOXPEL, Triggers are not allowed in an XPEL routine

Run Time Error: GT.M does not support use of a TP transaction when executing code invoked using the XPEL modifier on the TRESTART command and triggers use implicit TP to wrap application code.

Action: [Re]architect the application observing this limitation, which exists to prevent possible circular recursion.

XPELNOTP

XPELNOTP, TP not allowed in an XPEL routine

undefined Error: GT.M does not support a TSTART while executing code invoked using an XPEL modifier on a TRESTART command.

GT.M Release Notes

Action: [Re]architect the application observing this limitation, which exists to prevent possible circular recursion.