



V6.1-000



# GT.M

## Administration and Operations Guide

**UNIX Edition**

# GT.M Administration and Operations Guide

Publication date September 04, 2014

Copyright © 2011-2014 Fidelity Information Services, Inc. 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 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 V6.1-000/1	04 September 2014	<ul style="list-style-type: none"><li>• In Chapter 2: “Installing GT.M” (page 4), added a new section called “Reference Implementation Plugin - Shared Library Dependencies” (page 8).</li><li>• In Chapter 7: “Database Replication” (page 173) and under the “Procedures ” (page 203) section, corrected the downloadable scripts example (msr_proc2.tar.gz) for setting up an A-&gt;P replication configuration.</li></ul>
Revision V6.1-000	01 August 2014	<ul style="list-style-type: none"><li>• Updated Chapter 3: “Basic Operations” (page 12), Chapter 4: “Global Directory Editor” (page 32), Chapter 5: “General Database Management” (page 67), and Chapter 7: “Database Replication” (page 173) for V6.1-000. For chapter specific changes, refer to the chapter-level revision history.</li></ul>
Revision V6.0-003/1	19 February 2014	<ul style="list-style-type: none"><li>• Added a new appendix called Appendix G: “Packaging GT.M Applications” (page 441).</li><li>• In “REORG ” (page 102), added a caution note on running successive REORGs.</li><li>• In “Environment Variables” (page 17), improved the description of the gtm_nocenable environment variable.</li><li>• In “File Header Data Elements ” [266], updated the descriptions for V6.0-003.</li></ul>
Revision V6.0-003	27 January 2014	<ul style="list-style-type: none"><li>• In Chapter 3: “Basic Operations” (page 12), added the -help and -? command line options for the gtm script and the descriptions of gtm_ipv4_only and gtm_etrp environment variables.</li><li>• In Chapter 4: “Global Directory Editor” (page 32), corrected the syntax of [NO]STDNULLCOLL and -COLLATION_DEFAULT and added a</li></ul>

		<p>paragraph about what happens when database file with automatic extension disabled (EXTENSION_COUNT=0) starts to get full.</p> <ul style="list-style-type: none"> <li>• In Chapter 5: “General Database Management” (page 67), added the -MUTEX_SLOTS qualifier for MUPIP SET and -OVERRIDE qualifier for MUPIP RUNDOWN.</li> <li>• In Chapter 7: “Database Replication” (page 173), added downloadable script examples for A-&gt;B and A-&gt;P replication scenarios, information about the ACTIVE_REQUESTED mode, PASSIVE_REQUESTED mode, and IPv6 support.</li> <li>• In Chapter 9: “GT.M Database Structure(GDS)” (page 265), corrected the description of block header fields.</li> </ul>
Revision V6.0-001/2	10 April 2013	<ul style="list-style-type: none"> <li>• In Chapter 5: “General Database Management” (page 67), added a planning/preparation checklist that can be used before starting a MUPIP BACKUP.</li> <li>• In Appendix F: “GT.MPCAT - GT.M Process/Core Analysis Tool ” (page 437), added information about the --cmdfile option.</li> </ul>
Revision V6.0-001/1	22 March 2013	<ul style="list-style-type: none"> <li>• In Chapter 5: “General Database Management” (page 67), Chapter 6: “GT.M Journaling” (page 130), and Chapter 7: “Database Replication” (page 173), improved the formatting all command syntaxes.</li> <li>• In “SET Object Identifying Qualifiers ” (page 141), added the descriptions of -repl_state={on off} and -dbfilename=filename qualifiers of -jnfile.</li> </ul>
Revision V6.0-001	27 February 2013	<ul style="list-style-type: none"> <li>• Added Appendix F: “GT.MPCAT - GT.M Process/Core Analysis Tool ” (page 437). gtmpcat is a diagnostic tool for FIS to provide GT.M support.</li> <li>• Updated for V6.0-001. For chapter-specific revisions, refer to Chapter 3: “Basic Operations” [12], Chapter 4: “Global Directory Editor” (page 32), Chapter 5: “General Database Management” (page 67), Chapter 6: “GT.M Journaling” (page 130), Chapter 7: “Database Replication” (page 173), and Chapter 10: “Database Structure Editor” (page 279).</li> </ul>
Revision V6.0-000/1	21 November 2012	<ul style="list-style-type: none"> <li>• Updated for V6.0-000. For chapter-specific revisions, refer to Chapter 3: “Basic Operations” [12], Chapter 4: “Global</li> </ul>

		<p>Directory Editor” (page 32), Chapter 6: “GT.M Journaling” (page 130), Chapter 10: “Database Structure Editor” (page 279), and Appendix B: “Monitoring GT.M Messages” (page 411).</p>
Revision V6.0-000	19 October 2012	<ul style="list-style-type: none"> <li>• In Chapter 4: “Global Directory Editor” (page 32), added the description of the -INST_FREE_ON_ERROR region qualifier.</li> <li>• In Chapter 5: “General Database Management” (page 67), added the description of the -INST_FREE_ON_ERROR qualifier of MUPIP SET.</li> <li>• In Chapter 7: “Database Replication” (page 173), added the description of the -FREEZE qualifier of MUPIP REPLICATE -SOURCE.</li> <li>• In Chapter 3: “Basic Operations” (page 12) added gtm_custom_errors to the list of “Environment Variables” [17].</li> </ul>
Revision 11	05 October 2012	<ul style="list-style-type: none"> <li>• In Chapter 12: “Database Encryption” (page 363), added information about setting the environment variable GTMXC_gpgagent in “Special note - Gnu Privacy Guard version 2” [389].</li> <li>• In Chapter 5: “General Database Management” (page 67), improved the description of MUPIP INTEG and the -ADJACENCY qualifier.</li> <li>• In Chapter 3: “Basic Operations” (page 12) added GTMXC_gpgagent to the list of “Environment Variables” [17].</li> </ul>
Revision 10	28 September 2012	<ul style="list-style-type: none"> <li>• In Chapter 11: “Maintaining Database Integrity” [324], added the “O5-Salvage of a damaged spanning node” (page 354) and “Recovering data from damaged binary extracts” (page 331) sections.</li> <li>• In Chapter 9: “GT.M Database Structure(GDS)” [265], added a new section called “Using GDS records to hold spanning nodes ” [274] and corrected the first sentence of the "Use of Keys" section as follows:  GT.M locates records by finding the first key in a block lexically greater than, or equal to, the current key.</li> <li>• In “Database Structure Editor” (page 279), improved the description of -DATA.</li> </ul>

		<ul style="list-style-type: none"> <li>• In “Starting the Receiver Server ” (page 243), improved the description of - UPDATERESYNC.</li> <li>• In Appendix D: “Building Encryption Libraries” (page 425), removed a redundant step in the “Solaris 9 and 10 (SPARC) ” (page 428) section.</li> <li>• In Chapter 4: “Global Directory Editor” (page 32), added the description of the STDNULLCOLL qualifier and corrected the description of the NULL_SUBSCRIPTS qualifier.</li> <li>• In Chapter 4: “Global Directory Editor” (page 32) and Chapter 5: “General Database Management” (page 67), corrected the maximum value for - LOCK_SPACE and improved the description of -FLUSH_TIME.</li> </ul>
Revision 9	14 August 2012	<ul style="list-style-type: none"> <li>• In Chapter 7: “Database Replication” (page 173), improved the description of the replication WAS_ON state and added the “Recovering from the replication WAS_ON state ” (page 219) section.</li> </ul>
Revision 8	03 August 2012	<ul style="list-style-type: none"> <li>• In Chapter 4: “Global Directory Editor” (page 32) and Chapter 5: “General Database Management” (page 67), improved the description of - LOCK_SPACE.</li> <li>• In Chapter 4: “Global Directory Editor” (page 32), improved the overview description of GDE.</li> <li>• In Chapter 5: “General Database Management” (page 67), added a note on MUPIP BACKUP supporting only one concurrent -online backup.</li> <li>• In Chapter 10: “Database Structure Editor” (page 279) added the description of the GVSTATS qualifier.</li> </ul>
Revision 7	26 July 2012	<ul style="list-style-type: none"> <li>• In Chapter 1: “About GT.M” (page 1), added a new section called “Command Qualifiers”.</li> </ul>
Revision 6	19 July 2012	<ul style="list-style-type: none"> <li>• In Chapter 13: “GT.CM Client/Server” (page 400), corrected the example of starting a GT.CM server.</li> <li>• In screen and print pdfs, removed excessive spacing around SVG diagrams in Chapter 7: Database Replication (page 173).</li> </ul>
Revision 5	17 July 2012	<ul style="list-style-type: none"> <li>• Updated Chapter 7: “Database Replication” (page 173) for V5.5-000.</li> </ul>

		<ul style="list-style-type: none"> <li>• In Chapter 3: Basic Operations (page 12), corrected the description of the environment variable <code>gtm_non_blocked_write_retries</code>.</li> <li>• Fixed a typo (stand-alone -&gt; stand-alone) in "Recommendations" (page 432).</li> </ul>
Revision 4	6 June 2012	<ul style="list-style-type: none"> <li>• Added Chapter 13: "GT.CM Client/Server" (page 400).</li> <li>• In Chapter 5: "General Database Management" (page 67), added the description of MUPIP REORG -TRUNCATE and MUPIP LOAD -STDIN.</li> <li>• In Chapter 6: "GT.M Journaling" (page 130), added the description of -ROLLBACK -ONLINE, updated the description of -EXTRACT to include the -stdout value, and added the definitions of unam, clntunam, and ALIGN records.</li> <li>• In Chapter 8: "M Lock Utility (LKE)" (page 252), added the description of the LOCKSPACEUSE message.</li> <li>• In Chapter 10: "Database Structure Editor" (page 279), updated the description of -SIBLING.</li> <li>• In Chapter 4: "Global Directory Editor" (page 32), added AUTOSWITCHLIMIT as an option for the -JOURNAL region qualifier.</li> <li>• In Chapter 7: "Database Replication" (page 173), removed -LOG as an option for -STATSLOG.</li> </ul>
Revision 3	02 May 2012	<ul style="list-style-type: none"> <li>• Added Appendix E: "GT.M Security Philosophy" (page 430).</li> <li>• In "Rolling Back the Database After System Failures" (page 249), corrected the command syntax and the example.</li> <li>• In "Starting the Source Server" (page 230), corrected the quoting of -filter examples.</li> <li>• In "Journal Extract Formats" (page 168), added the definition of tid.</li> </ul>
Revision 2	19 March 2012	<ul style="list-style-type: none"> <li>• In "O4-Salvage of Data Blocks with Lost Indices" (page 352), fixed syntax errors in the example.</li> <li>• Updated the section called "Journal Extract Formats" [168] for V5.5-000.</li> </ul>

		<ul style="list-style-type: none"> <li>• In “Installation Procedure” (page 5), removed an incorrect reference to libgtmshr.so.</li> </ul>
Revision 1	5 March 2012	<ul style="list-style-type: none"> <li>• Updated “Basic Operations” (page 12) for V5.5-000.</li> <li>• In “Database Replication” [173], improved the description of the -instsecondary qualifier and added the “Stopping a Source Server” [241] section and the “Shutting Down an instance” [209] procedure.</li> </ul>
Revision V5.5-000	27 February 2012	<ul style="list-style-type: none"> <li>• Updated “Installing GT.M” (page 4) for V5.5-000.</li> <li>• Updated “REORG ” [102] for V5.5-000. Also, improved the description of -EXCLUDE and -SELECT qualifiers.</li> <li>• In “CHANGE -Fileheader Qualifiers” [290], added a caution note about changing the value of the -CORRUPT_FILE fileheader element.</li> <li>• In “Starting the Source Server” [230], added information about using external filters for pre-V5.5-000 versions.</li> </ul>
Revision 4	13 January 2012	<ul style="list-style-type: none"> <li>• In “Starting the Source Server” [230], added an example of a replication filter.</li> <li>• In “Before you begin” [4], added a note to caution users about installing and operating GT.M from an NFS mounted directory.</li> <li>• In “Operating in DSE” [279], improved the description of the Change command.</li> </ul>
Revision 3	26 December 2011	In “GT.M Journaling” [130], added the Journal Extract format for the ZTRIG record.
Revision 2	2 December 2011	<ul style="list-style-type: none"> <li>• In “GT.M Journaling” [130], improved the description of -EXTRACT and corrected the heading levels of some sections under MUPIP JOURNAL.</li> <li>• In “Database Replication” [173], corrected the usage of the term propagating instance, improved the description of -stopsourcefilter, and changed -nopropagatingprimary to -propagatingprimary.</li> <li>• Added Index entries for gtmsecshr.</li> <li>• In “Compiling ICU” [414], changed the instructions for Compiling ICU 4.4 on AIX 6.1.</li> </ul>
Revision 1	10 November 2011	<ul style="list-style-type: none"> <li>• In the “Conventions Used in This Manual” [xi] section, added a brief description about wrapping long-line examples in Print and Screen PDFs.</li> </ul>

		<ul style="list-style-type: none"><li>• In the “SET ” [109] section, changed "MUPIP SET" to "MUPIP SET".</li><li>• In the “Startup” [204] section, changed "Restore the replication instance file" to "Recreate the replication instance file".</li></ul>
Revision V5.4-002B	24 October 2011	Conversion to documentation revision history reflecting GT.M releases with revision history for each chapter.

# Table of Contents

About This Manual .....	x
1. About GT.M .....	1
2. Installing GT.M .....	4
3. Basic Operations .....	12
4. Global Directory Editor .....	32
5. General Database Management .....	67
6. GT.M Journaling .....	130
7. Database Replication .....	173
8. M Lock Utility (LKE) .....	252
9. GT.M Database Structure(GDS) .....	265
10. Database Structure Editor .....	279
11. Maintaining Database Integrity .....	324
12. Database Encryption .....	363
13. GT.CM Client/Server .....	400
A. GT.M's IPC Resource Usage .....	405
B. Monitoring GT.M Messages .....	411
C. Compiling ICU on GT.M supported platforms .....	414
D. Building Encryption Libraries .....	425
E. GT.M Security Philosophy .....	430
F. GTMPCAT - GT.M Process/Core Analysis Tool .....	437
G. Packaging GT.M Applications .....	441
Glossary .....	446
Index .....	447

---

## About This Manual

GT.M is a high-end database application development platform offered by Fidelity National Information Services (FIS). GT.M provides an M (also known as MUMPS) language environment that largely complies with ISO/IEC 11756:1999. GT.M's compiler for the standard M scripting language implements full support for ACID (Atomic, Consistent, Isolated, Durable) transactions, using optimistic concurrency control and software transactional memory (STM) that resolves the common mismatch between databases and programming languages. The GT.M data model is a hierarchical associative memory (that is, multi-dimensional array) that imposes no restrictions on the data types of the indexes and the content - the application logic can impose any schema, dictionary or data organization suited to its problem domain.

GT.M's unique ability to create and deploy logical multi-site configurations of applications provides unrivaled continuity of business in the face of not just unplanned events, but also planned events, including planned events that include changes to application logic and schema.

You can install and manage GT.M using the utilities described in this manual and standard operating system tools. The first three chapters provide an overview of GT.M, installation procedures, and GT.M system environment. The remaining chapters describe GT.M operational management.

---

## Intended Audience

This manual is intended for users who install GT.M and manage the GT.M user environment. The presentation of information assumes a working knowledge of UNIX, but no prior knowledge of GT.M.

---

## Purpose of the Manual

This GT.M Administration and Operations Guide explains how to install and manage GT.M.

---

## How to Use This Manual

First, read Chapter 1: “*About GT.M*” (page 1) for an overview of GT.M system management. Then, proceed to the chapter that discusses the area of interest.

The presentation of information in each chapter is designed to be useful for even a first-time user. Each chapter starts with an overview of the topic at hand and then moves on to related GT.M utility program commands and qualifiers. This list is organized alphabetically by command and then by the qualifiers for each command. Then, the chapter provides recommendations from FIS to implement and operate important aspects of the topic. It ends with an exercise that reinforces the concepts introduced in the chapter.

FIS recommends users read the chapters in a top-down manner. After becoming familiar with GT.M, use the “Commands and Qualifiers” section of each chapter as a reference manual.

---

## Overview

This manual contains twelve chapters and an Appendix. Here is a brief overview of each chapter:

Chapter 1: “*About GT.M*” (page 1) introduces GT.M administration and operations.

## About This Manual

Chapter 2: “*Installing GT.M*” (page 4) provides procedures for installing GT.M.

Chapter 3: “*Basic Operations*” (page 12) describes operations required to start GT.M and keep it running.

Chapter 4: “*Global Directory Editor*” (page 32) describes global directories, which control placement and access of global variables, and explains how to use the Global Directory Editor (GDE) to create and maintain global directories.

Chapter 5: “*General Database Management*” (page 67) describes how to use a GT.M utility called MUPIP to perform database and non-database operations.

Chapter 6: “*GT.M Journaling*” (page 130) describes the journaling capabilities of GT.M.

Chapter 7: “*Database Replication*” (page 173) describes how to implement continuous application availability using multiple systems.

Chapter 8: “*M Lock Utility (LKE)*” (page 252) describes how to use a GT.M utility called M Lock Utility (LKE) to examine and adjust M locks.

Chapter 9: “*GT.M Database Structure(GDS)*” (page 265) provides an overview of GT.M Database Structure (GDS).

Chapter 10: “*Database Structure Editor*” (page 279) describes how to use the Database Structure Editor (DSE) to examine and modify sections of the database, should that ever be necessary.

Chapter 11: “*Maintaining Database Integrity*” (page 324) describes procedures for verifying and maintaining the integrity of GDS databases.

Chapter 12: “*Database Encryption*” (page 363) describes procedures for encrypting data in the database and journal files.

Appendix A: “*GT.M's IPC Resource Usage*” (page 405) describes how GT.M processes use UNIX Interprocess Communication (IPC).

Appendix C: “*Compiling ICU on GT.M supported platforms*” (page 414) contains sample ICU installation instructions. Although GT.M uses ICU to enable the support for Unicode™, ICU is not FIS software and FIS does not support ICU. The sample instructions for installing and configuring ICU are merely provided as a convenience to you.

"Appendix C" describes security considerations for deploying applications on GT.M.

---

## Conventions Used in This Manual

References to other GT.M documents are implemented as absolute hypertext links.

Use GT.M with any UNIX shell as long as environment variables and scripts are consistently created for that shell. In this manual, UNIX examples are validated on Ubuntu Linux (Ubuntu Linux uses **dash** for **/bin/sh**). Examples in later chapters assume that an environment has been set up as described in the chapters Chapter 2: “*Installing GT.M*” (page 4) and Chapter 3: “*Basic Operations*” (page 12).

FIS made a conscientious effort to present intuitive examples and related error messages that appear if a user tries those examples. However, due to environment and shell differences, you may occasionally obtain different results (although the differences should be relatively minor). Therefore, FIS suggests that you try the examples in a database environment that does not contain any valued information.

Examples follow the descriptions of commands. For readability purposes, long shell command lines are wrapped into multiple lines. The ► icon denotes the starting point of wrap. Users must always assume a single line is correct and that the ► icon means

## About This Manual

continuation. To facilitate copy/paste of examples from Screen PDF, clicking on  icon below a wrapped example opens a window or tab where that example is available with unwrapped command lines. For Print PDF, an unwrapped command lines example can be downloaded from the URL below the example.

In M examples, an effort was made to construct examples where command lines did not wrap, in many cases using the argumentless DO.

The examples make frequent use of literals in an attempt to focus attention on particular points. In normal usage arguments are far more frequently variables.

---

## Chapter 1. About GT.M

Revision History		
Revision V5.5-000/7	26 July 2012	Added a new section called “Command Qualifiers” (page 3).
Revision V5.5-000/4	10 May 2012	Updated for V5.5-000.
Revision V5.4-002B	24 October 2011	Conversion to documentation revision history reflecting GT.M releases with revision history for each chapter.

GT.M is a high-end performance database application development and deployment platform from Fidelity National Information Services (FIS). GT.M provides an M language subsystem, a database engine, and a complete set of utilities for administration and operation. One can install and manage GT.M using the utilities described in this manual and standard operating system tools. This chapter provides an overview of the GT.M system environment and utilities.

---

### Hardware/Operating System Environment

GT.M runs on a wide variety of UNIX/Linux implementations as well as OpenVMS. Consult FIS for currently supported versions. Each GT.M release is extensively tested by FIS on a set of specific versions of operating systems on specific hardware architectures (the combination of operating system and hardware architecture is referred to as a platform). This set of specific versions is considered **Supported**. There will 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 support team knows of no reason why GT.M would not work. This larger set of versions is considered **Supportable**. There is an even larger set of platforms on which GT.M may well run satisfactorily, but where the FIS GT.M team lacks the knowledge to determine whether GT.M is **Supportable**. These are considered **Unsupported**. Contact FIS GT.M Support with inquiries about your preferred platform.

System requirements vary widely depending on application needs, and should be empirically determined for each situation.

---

### Installation

GT.M installation is semi-automatic, using the system administration scripts provided with GT.M. The installation procedure is described in Chapter 2: “*Installing GT.M*” (page 4).

---

### Security

Users require no special privileges to run GT.M beyond standard system access and suitable access to relevant application and database files. All the standard UNIX/OpenVMS security features are available to protect GT.M resources.

FIS strongly recommends a security design where each user has no more authorizations than they require to do their assigned roles and each user's actions are distinguishable in an audit.



## Note

Root or superuser access is required to install GT.M, but not to use it. FIS recommends against routine use of the root userid to run GT.M.

---

## Program Development Environment

GT.M provides a compiler and run-time environment for the M language. Program development uses standard editors and utility programs.

GT.M requires minimal administrative effort.

M routines are stored as text files and object files in the native file system. Compilation of source text into object code is typically automatic, but may be invoked explicitly. A user may store routines in multiple libraries and/or directories organized into search hierarchies, and change the search paths as needed.

For more information on the GT.M language and programming environment, see *GT.M Programmer's Guide*.

---

## Database Subsystem

The GT.M database subsystem consists of a run-time library and a set of utilities which operate on one or more user-specified Global Directories (GD) and database files. GT.M stores M global variables in database files, which are ordinary UNIX files. Internally, the UNIX files are organized as balanced trees (B-trees) in GT.M Data Structures (GDS). See "GT.M Database Structure" chapter for more information on B-trees and the GDS file structure.

A directory maps global names to a database file. GT.M processes use this mapping when storing and retrieving globals from the database. Multiple global directories can reference a single database file, and a database file can be referenced in multiple global directories, with some exceptions, as discussed in Chapter 4: "*Global Directory Editor*" (page 32). Use the Global Directory Editor (GDE) to create and maintain global directories.

In addition to mapping global variables to database files, global directories also store initial parameters used by the MUPIP CREATE command when creating new database files. GT.M uses environment variables to locate the global directory or, optionally database files.

---

## GT.M Utility Programs

GT.M provides utility programs to administer the system. Each utility is summarized below, and described later in this manual.

### GDE

The Global Directory Editor (GDE) is a GT.M utility program that creates and maintains global directories. GDE provides commands for operating on the global directory.

### MUPIP

MUPIP (M Peripheral Interchange Program) is the GT.M utility program for general database operations, GT.M Journaling, Multi-site Database Replication, and some non-database operations.

## LKE

The M Lock Utility (LKE) is the GT.M utility program that examines and modifies the lock space where GT.M maintains the current M LOCK state. LKE can monitor the locking mechanism and remove locks. See Chapter 8: “*M Lock Utility (LKE)*” (page 252) for more information.

## DSE

The Database Structure Editor (DSE) is the GT.M utility program to examine and alter the internal database structures. DSE edits GT.M Database Structure (GDS) files. It provides an extensive database “patch” facility (including block integrity checks), searches for block numbers and nodes, and provides symbolic examination and manipulation facilities. See Chapter 10: “*Database Structure Editor*” (page 279) for more information.

## Command Qualifiers

Each utility program has its own set of commands. Qualifiers are used as arguments for a command. A qualifier is always prefixed with a hyphen (-). Some qualifier allow assigning values with an equal (=) sign where as some allow the use of sub-qualifiers as their arguments. If you specify the same qualifier more than once, MUPIP, DSE, and LKE acts upon the qualifier that appears latest. However, you cannot specify qualifiers that have sub-qualifiers more than once. With GDE, specifying the same qualifier more than once produces an error.

---

## Database Integrity

GT.M tools verify and maintain database integrity. As described in Chapter 11: “*Maintaining Database Integrity*” (page 324), database integrity refers to a state of logical and physical consistency in the database when all of the globals and pointers are correct, thereby making all data accessible. Chapter 11 [324] describes how to use the MUPIP INTEG command and the DSE utility to detect and repair integrity problems, and supplies procedures for avoiding such problems.

---

## Interprocess Communication

GT.M uses UNIX Interprocess Communication (IPC) resources to coordinate access to the database. Additionally, GT.M includes a daemon process gtmsecshr that implements process wake-up for M locks and clean-up of IPC resources after certain types of abnormal process termination. See Appendix A: “*GT.M's IPC Resource Usage*” (page 405) for more information.

---

## Chapter 2. Installing GT.M

Revision History		
Revision V6.1-000/1	04 September 2014	Added a new section called “Reference Implementation Plugin - Shared Library Dependencies” (page 8).
Revision V5.5-000/2	19 March 2012	Removed an incorrect reference to libgtmshr.so.
Revision V5.5-000	27 February 2012	Updated for V5.5-000.
Revision 4	13 January 2012	In “Before you begin” [4], added a note to caution users about installing and operating GT.M from an NFS mounted directory.
Revision V5.4-002B	24 October 2011	Conversion to documentation revision history reflecting GT.M releases with revision history for each chapter.

This chapter describes the installation procedure for GT.M. Always see the release notes for special instructions before installing GT.M.

---

### Obtaining GT.M Distribution Media

FIS prefers to distribute GT.M online via the Internet. GT.M for selected platforms, including GNU/Linux on the popular x86 architecture, can be downloaded under the terms of the Affero GNU General Public License (AGPL) version 3, from Source Forge (<http://sourceforge.net/projects/fis-gtm>). Contact GT.M Support ([gtmsupport@fisglobal.com](mailto:gtmsupport@fisglobal.com)) to obtain a copy of a GT.M distribution for other platforms or on physical media.

---

### Before you begin

Before you begin installing GT.M, perform the following tasks:

- Read the GT.M Release Notes documentation. The release documents contain the latest information that may be critical for installing and configuring GT.M. They are located at the User Documentation tab on the GT.M website ([www.fis-gtm.com](http://www.fis-gtm.com)).
- Determine whether or not, GT.M access is restricted to a group. Keep the group name handy as you will have to enter it during the installation process.
- Set the environment variable `gtm_log` to a directory where GT.M should create log files. In conformance with the Filesystem Hierarchy Standard, FIS recommends `/var/log/fis-gtm/$gtmver` as the value for `$gtm_log` unless you are installing the same version of GT.M in multiple directories. Note that `$gtmver` can be in the form of `V6.1-000_x86` which represents the current GT.M release and platform information.

If you do not set `gtm_log`, GT.M creates log files in a directory in `/tmp` (AIX, GNU/Linux, Tru64 UNIX) or `/var/tmp` (HP-UX, Solaris). However, this is not recommended because it makes GT.M log files vulnerable to the retention policy of a temporary directory.



## Important

Starting with V6.0-000, gtmsecshr logs its messages in the system log and the environment variable gtm\_log is ignored.

- If you need to perform Unicode™-related operations in GT.M, you must have at least ICU version 3.6 installed. GT.M uses ICU 3.6 (or above) to provide the support for Unicode. GT.M generates the distribution for Unicode only if ICU 3.6 (or above) is installed on your system. By default, GT.M uses the most current version of ICU. GT.M expects ICU to have been built with symbol renaming disabled and issues an error at startup if the currently installed version of ICU has been built with symbol renaming enabled. If you intend to use a version of ICU built with symbol renaming enabled or any version other than the default, keep the MAJOR VERSION and MINOR VERSION numbers ready as you will have to enter it as MajorVersion.MinorVersion (for example "3.6" to denote ICU-3.6) during the installation process.



## Caution

Installing GT.M on an NFS mounted directory is not recommended. Several NFS characteristics violate GT.M database design assumptions which may manifest themselves as hard to diagnose problems. If you still choose to install and operate GT.M from an NFS mounted directory, there are chances that at some point you will face significant problems with performance and response time.

While you should never operate GT.M database and journal files from an NFS mounted directory you can safely, except on Linux, use an NFS mounted directory for keeping source and object code modules and performing sequential file IO. While NFS mounted files may work for you, historically they have not provided sufficient support for file locking over NFS to prevent intermittent errors when you have a significant concurrent file activity.

## Installation Procedure

Create a backup copy of any existing version of GT.M before running the installation procedure. The installation procedure overwrites any existing version of GT.M in the installation target directory.



## Important

Never install a GT.M release into an existing directory overwriting another release that might be needed for research or recovery and **never** on the one currently in use. FIS recommends installing each version of GT.M in a separate directory using the naming convention `/usr/lib/fis-gtm/version_platform`, where version is the GT.M version being installed and platform is the hardware architecture. For example, `/usr/lib/fis-gtm/V6.1-000_x86_64`.

Run the installation procedure as root for everything to install correctly.

Find or create a temporary directory to hold your GT.M distribution files and change to that directory.

Example:

```
$ cd /tmp
```

or

```
$ cd /usr/tmp
```



## Note

When choosing a temporary directory, keep in mind the following points:

- Whether it is safe to use a directory (like /tmp) subject to concurrent use by the operating system, applications, and so on.
- Whether you can protect any sensitive information in temporary files with appropriate access permissions.

Unpack the distribution to the current directory with a command such as the following (your UNIX version may require you to first use `gzip` to decompress the archive and then `tar` to unpack it; the `tar` options may also vary):

```
$ tar zxvf /Distrib/GT.M/gtm_V55000_linux_x8664_pro.tar.gz
```



## Note

The name of the device and the `tar` options may vary from system to system.

The GT.M distribution contains various GT.M system administration scripts. These include:

<code>configure</code>	to install GT.M
<code>gtm</code>	sets a default user environment and starts GT.M.
<code>gtmbase</code>	An example to set up a default user environment.

Note that `gtmbase` is simply an example of the type of script you can develop. Do not use it as is.

For more information on using `gtmbase` and other scripts, refer to Chapter 3: “*Basic Operations*” (page 12).

To start the GT.M installation procedure, execute the following script:

```
#!/configure
```

The `configure` script displays a message like the following:

```
GT.M Configuration Script
GT.M Configuration Script Copyright 2009, 2012 Fidelity Information Services, Inc. Use of this
software is restricted by the provisions of your license agreement.
```

```
What user account should own the files? (bin)
```

Enter the name of the user who should own GT.M distribution files. The default is **bin**. If there is no user with the name `bin`, the `configure` script asks for an alternate user who should own GT.M distribution files.

```
What group should own the files? (bin)
```

Enter the name of the group who should own GT.M distribution files. The default is `bin`. If there is no group with the name `bin`, the `configure` script asks for an alternate group who should own GT.M distribution files.

```
Should execution of GT.M be restricted to this group? (y or n)
```

Choose **y** to restrict the ownership of your GT.M distribution to the specified group. This is a security-related option. By default, GT.M does not restrict the ownership to a group.

```
In what directory should GT.M be installed?
```

## Installing GT.M

Enter a directory name, such as `/usr/lib/fis-gtm/V6.1-000_x86_64`. If the directory does not exist, the configure script displays a message like the following:

```
Directory /usr/lib/fis-gtm/V6.1-000_x86_64 does not exist.  
Do you wish to create it as part of this installation? (y or n)
```

Choose **y** to create the directory or **n** to cancel the installation.

```
Installing GT.M...
```

This is followed by a confirmation message for installing UTF-8 support.

```
Should UTF-8 support be installed? (y or n)
```

Choose **y** to confirm. If you choose **n**, Unicode functionality is not installed.



### Note

GT.M requires at least ICU Version 3.6 to install the functionality related to Unicode.

If you choose **y**, the configure script displays a message like the following:

```
Should an ICU version other than the default be used?
```

If you choose **n**, the configure script looks for the symbolic link of `libicuuc.so` to determine the default ICU version.

If you choose **y**, the configure script displays a message like the following:

```
Enter ICU version (at least ICU version 3.6 is required.  
Enter as <major-ver>.<minor-ver>):
```

Enter the ICU version number in the *major-ver.minor-ver* format. The configure script displays the following message:

```
All of the GT.M MUMPS routines are distributed with uppercase  
names. You can create lowercase copies of these routines  
if you wish, but to avoid problems with compatibility in  
the future, consider keeping only the uppercase versions  
of the files.
```

```
Do you want uppercase and lowercase versions of the MUMPS routines? (y or n).
```

Choose **y** to confirm. The configure script then displays the list of MUMPS routines while creating their lowercase versions. Then the script compiles those MUMPS routines. On platforms where GT.M supports placing object code in shared library, it creates a library `libgtmutil.so` with the object files of the utility programs and GDE distributed with GT.M. On these platforms, it asks the question:

```
Object files of M routines placed in shared library /usr/lib/fis-gtm/V6.1-000_x86_64/libgtmutil.so.  
Keep original .o object files (y or n)?
```

Unless your scripts require separate object files for these routines, you can safely delete the `.o` object files, and place `$gtm_dist/libgtmutil.so` as the last element of `gtmroutines` in lieu of `$gtm_dist`.

The configure script then asks you:

```
Installation completed. Would you like all the temporary files removed from this directory? (y or n)
```

Choose **y** to confirm. FIS recommends deleting all temporary files.

## Installing GT.M

Congratulations! GT.M is now installed on the system. Proceed to “Reference Implementation Plugin - Shared Library Dependencies” (page 8) section to identify and resolve shared library dependencies on your system and then Chapter 3: “Basic Operations” (page 12) to set up a default user environment.

## Reference Implementation Plugin - Shared Library Dependencies

The database encryption and TLS reference implementation plugin requires libcrypto11, libpgpme11, libconfig 1.4.x, and libssl1.0.0 runtime libraries. Immediately after installing GT.M, install these libraries using the package manager of your operating system. If these runtime libraries are not available, the plugin may produce errors like the following (for example on Ubuntu x86\_64):

```
%GTM-I-TEXT, libxxxxxx.so.<ver>: cannot open shared object file: No such file or directory
```

..where libxxxxxx is either libcrypto, libpgpme, libconfig, libcrypto, or libssl and <ver> is the version number of the libxxxxxx shared library. To locate missing share library dependencies in your reference implementation plugin, execute the following command:

```
$ ldd $gtm_dist/plugin/libgtm* | awk '/^usr/{libname=$0}/not found/{print libname;print}'
```

This command produces an output like:

```
/usr/lib/fis-gtm/V6.1-000_x86_64/plugin/libgtmcrypto_openssl_AES256CFB.so:
    libcrypto.so.10 => not found
/usr/lib/fis-gtm/V6.1-000_x86_64/plugin/libgtmcrypto_openssl_BLOWFISHCFB.so:
    libcrypto.so.10 => not found
/usr/lib/fis-gtm/V6.1-000_x86_64/plugin/libgtmtls.so:
    libssl.so.10 => not found
```

...which means that libcrypto.so.10 and libssl.so.10 shared libraries are missing from the system and the reference implementation plugin will produce a libxxxxxx.so.<ver>: cannot open shared object file: No such file or directory error during runtime. Note down the missing shared libraries and proceed as follows for each missing library:

- **If libxxxxxx.so.<ver> is already installed:** Set the environment variable LD\_LIBRARY\_PATH to include its location.
- **If libxxxxxx is not installed:** Install the runtime libxxxxxx library using the package manager; check for pre-packaged distributions for your operating system and ensure that it contains the library having version specified with <ver>.
- **If libxxxxxx is installed but its version number is different from what GT.M expects (<ver>):** You can either symlink the unversioned libxxxxxx.so library to libxxxxxx.so.<ver> (recommended) or recompile the reference implementation plugin. The instructions are as follows:

### Creating a symbolic link from the unversioned .so (libxxxxxx.so) to libxxxxxx.so.<ver>

1. Install the development libraries for libcrypto11, libpgpme11, libconfig 1.4.x, libssl1.0.0. The package name of development libraries usually have the **-dev** suffix 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 libcrypto11-dev libpgpme11-dev libconfig-dev libssl-dev
```

These packages contain the unversioned .so library (libxxxxxx.so).

2. Find the directory where the unversioned .so library is installed on your system. Note down the name of the directory.

## Installing GT.M

3. Login as root and change to the `$gtm_dist/plugin` directory.
4. Create the symlink from `libxxxxxx.so` to `libxxxxxx.so.<ver>`. For example, if `libcrypto.so.10` and `libssl.so.10` shared libraries are missing, create their symbolic links in `$gtm_dist/plugin` from the unversioned `.so` library location that you noted in step 2:

```
ln -s /usr/lib/x86_64-linux-gnu/libcrypto.so libcrypto.so.10
ln -s /usr/lib/x86_64-linux-gnu/libssl.so libssl.so.10
```



### Note

The reference implementation plugin libraries are linked using the RPATH directive `$ORIGIN` which means that these libraries always start looking for library dependencies from the current directory. Therefore, it is safe to create symlinks for the missing libraries in `$gtm_dist/plugin` directory. Do not create symlinks to shared libraries in directories managed by package managers and never symlink incompatible library versions as it may not always lead to predictable results.

## Recompiling the Reference Implementation Plugin

1. Install the development libraries for `libgcrypt11`, `libgpgme11`, `libconfig 1.4.x`, `libssl1.0.0`. The package name of development libraries usually have the `-dev` suffix and are available through the package manager. For example, on `Ubuntu_x86_64` a command like the following installs the required development libraries:
2. Login as root and set the `gtm_dist` environment variable to point to the location of the GT.M distribution.
3. Note down the file permissions of the `*.so` files in `$gtm_dist/plugin` directory.
4. Unpack `$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
```

5. Recompile the reference implementation plugin and install it in the `$gtm_dist/plugin` directory.

```
make uninstall
make
make install
make clean
```

6. Assign permissions noted in step 3 to the newly installed `.so` files in `$gtm_dist/plugin`.



### Important

If you are running the V6.1-000 reference implementation plugin (database encryption and TLS replication) on a UNIX platform other than RedHat, please follow instructions from “Creating a symbolic link from the unversioned `.so` (`libxxxxxx.so`) to `libxxxxxx.so.<ver>`” (page 8) to create symlinks for the missing `libcrypto.so.10` and `libssl.so.10` shared libraries. GT.M UNIX distributions are compiled and linked on a RedHat platform where `libcrypto.so.10` and `libssl.so.10` version libraries are available from package managers. On UNIX platforms other than RedHat, these library versions may not be available from package managers and need to be linked to their unversioned `.so` libraries.

## gtminstall script

gtminstall is an experimental GT.M installation facility that when used stand-alone attempts to download the latest / current production GT.M distribution from sourceforge.net and installs GT.M using reasonable defaults. It is included with the GT.M binary distribution and you can also use it to install GT.M from the temporary directory in which you unpack the GT.M distribution. It allows considerable customization using the following command line switches:

Command line switches	*	Description
--build-type buildtype	*	Type of GT.M build, default is pro
--copyenv dirname		Copy gtmprofile and gtmcshrc files to dirname; incompatible with linkenv
--copyexec dirname		Copy gtm script to dirname; incompatible with linkexec
--debug -	*	Turn on debugging with set -x
--distrib dirname or URL		Source directory for GT.M distribution tarball, local or remote
--dry-run		Do everything short of installing GT.M, including downloading the distribution
--group group		Group that should own the GT.M installation
--group-restriction		Limit execution to a group; defaults to unlimited if not specified
--help		Print this usage information
--installdir dirname		Directory where GT.M is to be installed; defaults to /usr/lib/fis-gtm/version_platform
--keep-obj		Keep .o files of M routines (normally deleted on platforms with GT.M support for routines in shared libraries). By default, this option is disabled.
--linkenv dirname		Create link in dirname to gtmprofile and gtmcshrc files; incompatible with copyenv
--linkexec dirname		Create link in dirname to gtm script; incompatible with copyexec
--overwrite-existing		Install into an existing directory, overwriting contents; defaults to requiring new directory
--prompt-for-group	*	GT.M installation script will prompt for group; default is yes for production releases V5.4-002 or later, no for all others
--ucaseonly-utils		Install only upper case utility program names; defaults to both if not specified
--user username		User who should own GT.M installation; default is root
--utf8 ICU_version		Install UTF-8 support using specified major.minor ICU version; specify default to use default version
--verbose -	*	Output diagnostic information as the script executes; default is to run quietly

- options that take a value (e.g. --group) can be specified as either --option=value or --option value
- options marked with \* are likely to be of interest primarily to GT.M developers
- version is defaulted from the mumps file if one exists in the same directory as the installer
- This version must run as root.

To run the gtminstall script, unpack and run it a root. For example, on an x86\_64 GNU/Linux platform, ./gtminstall installs GT.M with M mode support in /usr/lib/fis-gtm/V6.1-000\_x86\_64.

## Installing GT.M

Example:

```
$ sudo ./gtminstall # installs latest version in M mode only
$ sudo ./gtminstall --utf8 default # install latest version with UTF-8 mode support
$ sudo ./gtminstall --distrib /Distrib/GT.M V6.0-003 # install V6.0-0030 from a local
▶ directory
```



## Chapter 3. Basic Operations

Revision History		
Revision V6.1-000	01 August 2014	In “Environment Variables” (page 17), improved the descriptions of <code>gtm_tptime</code> and <code>gtm_zquit_anyway</code> and added the descriptions of <code>gtm_crypt_config</code> , <code>gtmtls_passwd_&lt;tlslabel&gt;</code> , and <code>gtm_crypt_FIPS</code> .
Revision V6.0-003/1	19 February 2014	In “Environment Variables” (page 17), improved the description of <code>gtm_nocenable</code> .
Revision V6.0-003	27 January 2014	<ul style="list-style-type: none"><li>• In “gtm” (page 17), added the <code>-help</code> and <code>-?</code> command line options.</li><li>• In “Environment Variables” (page 17), added the description of <code>gtm_ipv4_only</code> and <code>gtm_etrap</code>.</li></ul>
Revision V6.0-001	27 February 2013	<ul style="list-style-type: none"><li>• Added the “Configuring huge pages for GT.M x86[-64] on Linux” (page 29) section.</li><li>• In “Environment Variables” (page 17), added the description of <code>gtm_side_effects</code>, <code>gtm_extract_nocol</code>, and <code>gtm_crypt_plugin</code> environment variables.</li></ul>
Revision V6.0-000/1	21 November 2012	<ul style="list-style-type: none"><li>• In “Environment Variables” (page 17), added the description of the environment variables <code>gtm_tptime_delta</code>, <code>gtm_tptime_first</code>, <code>gtm_tptime</code>, and <code>gtm_zmaxptime</code>.</li><li>• Added a note which states that the environment variable <code>gtm_log</code> is ignored from V6.0-000 onwards.</li></ul>
Revision V6.0-000	19 October 2012	In “Environment Variables” (page 17), added the description of the environment variable <code>gtm_custom_errors</code> .
Revision V5.5-000/11	05 October 2012	In “Environment Variables” (page 17), added the description of the environment variable <code>GTMXC_gpgagent</code> .
Revision V5.5-000/5	17 July 2012	Corrected the description of the environment variable <code>gtm_non_block_write_retries</code> to include the PIPE write behavior.
Revision V5.5-000/1	5 March 2012	Updated for V5.5-000.
Revision V5.4-002B	24 October 2011	Conversion to documentation revision history reflecting GT.M releases with revision history for each chapter.

## GT.M Environment Setup

Several environment variables control the operation of GT.M. Some of them must be set up for normal operation, where as for others GT.M assumes a default value if they are not set.

Your GT.M distribution comes with many scripts that set up a default GT.M environment for the shell of your choice. These scripts are as follows:

1. **gtmprofile**: uses reasonable defaults to set up a system and GT.M application development environment for POSIX shells. The gtmprofile script sets default values for environment variables gtm\_dist, gtmgbldir, gtm\_icu\_version, gtm\_log, gtm\_principal\_editing, gtm\_prompt, gtm\_retention, gtmroutines, gtm\_tmp, and gtmver. When you source the gtmprofile script, it creates a default execution environment (global directory and a default database file with BEFORE\_IMAGE journaling) if none exists.
2. **gtmchsrc**: sets up a default GT.M environment for C-shell compatible shells. It sets up default values for gtm\_dist, gtmgbldir, gtm\_chset and gtmroutines. It also creates aliases so you can execute GT.M and its utilities without typing the full path. While gtmprofile is current with GT.M releases, gtmchsrc is at the same level of sophistication as gtmprofile\_preV54000. It is not as actively maintained as the gtmprofile script.
3. **gtmprofile\_preV54000**: This script was provided as gtmprofile in GT.M distributions prior to V5.4-000. This script is a POSIX shell equivalent of gtmchsrc.
4. **gtmbase**: detects the shell type and adds gtmprofile to .profile or gtmchsrc to .cshrc so the shell automatically sources gtmprofile or gtmchsrc on a subsequent login operation. FIS does not recommend using gtmbase as is - use it as an example of a script for you to develop suitable for your systems. It is not as actively maintained as the gtmprofile script.
5. **gtm**: starts GT.M in direct mode on POSIX shells. The gtm script sources gtmprofile. It also deletes prior generation journal and temporary files older than the number of days specified by the environment variable gtm\_retention. It attempts to automatically recover the database when it runs and as such is suitable for "out of the box" usage of GT.M. Although it will work for large multi-user environments, you may want to modify or replace it with more efficient scripting.
6. **gdedefaults**: a GDE command file that specifies the default values for database characteristics defined by GDE.

These scripts are designed to give you a friendly out-of-the-box GT.M experience. Even though you can set up an environment for normal GT.M operation without using these scripts, it is important to go through these scripts to understand the how to manage environment configuration.

### gtmprofile

gtmprofile helps you set an environment for POSIX shells. It uses reasonable defaults to set up all environment variables for normal GT.M operation. gtmprofile sets the following environment variables:

gtmprofile sets the following environment variables:

gtm\_dist, gtmgbldir, gtm\_icu\_version, gtm\_log, gtm\_principal\_editing, gtm\_prompt, gtm\_retention, gtmroutines, gtm\_tmp, gtmver.

If \$gtm\_chset is set to UTF-8, gtmprofile also attempts to set LC\_CTYPE to the first usable utf8 locale.

The following shell variables are used by the script and left unset at its completion:

old\_gtm\_dist, old\_gtmroutines, old\_gtmver, tmp\_gtm\_tmp, tmp\_passwd.

## Basic Operations

The gtmascript (and gtm) scripts, by design, are idempotent so that calling them repeatedly is safe. The GT.M installation process ensures that gtmprofile always sets gtm\_dist correctly. Idempotency is implemented by checking the value of \$gtm\_dist, and skipping all changes to environment variables of \$gtm\_dist.

When gtm sources gtmprofile, it provides a default execution environment (global directory and a default database with BEFORE\_IMAGE journaling) if none exists. It also creates a directory .fis-gtm with a default structure like the following:

```
.fis-gtm
|-- V6.1-000_x86
|   |-- g
|   |   |-- gtm.dat
|   |   |-- gtm.gld
|   |   |-- gtm.mjl
|   |-- o
|   |   '-- utf8
|   '-- r
'-- r
```

where V6.1-000\_x86 represents current release and platform information. Note that this directory structure has different locations for GT.M routines (r), object files (o), and database-related files (g). gtmprofile sets gtmroutines to something like /usr/gtmnode1/.fis-gtm/V6.0-003\_x86/o(/usr/gtmnode1/.fis-gtm/V6.0-003\_x86/r /usr/gtmnode1/.fis-gtm/r) <path to the GT.M distribution>. This specifies that GT.M searches for routines in /usr/gtmnode1/.fis-gtm/V6.0-003\_x86/r, then /usr/gtmnode1/.fis-gtm/r, and finally in your GT.M distribution directory. gtmroutines also specifies that all object files should be generated in /usr/gtmnode1/.fis-gtm/V6.0-003\_x86/o, which also specifies the GT.M lookup location for finding object files.

Suppose you have a routine MyRoutine.m in /usr/gtmnode1/.fis-gtm/V6.1-000\_x86/r. On compilation, GT.M stores MyRoutine.o in /usr/gtmnode1/.fis-gtm/V6.0-003\_x86/o. Everytime a process first runs this routine, GT.M looks in the /fis-gtm/V6.0-003\_x86/o directory and links it prior to its first execution. For a comprehensive discussion of GT.M source and object file management, refer to the \$ZROUTINES section in the *GT.M Programmer's Guide*.

When \$gtm\_chset is set to UTF-8, gtmprofile sets gtmroutines to something like /usr/gtmnode1/.fis-gtm/V6.1-000\_x86/o/utf8(/usr/gtmnode1/.fis-gtm/V6.0-003\_x86/r /usr/gtmnode1/.fis-gtm/r) <path to the GT.M distribution>/utf8. So, on compilation, GT.M stores MyRoutine.o in /usr/gtmnode1/.fis-gtm/V6.0-003\_x86/o/utf8. Note that GT.M does not store object files in /usr/gtmnode1/.fis-gtm/V6.0-003\_x86/o which was the case in M mode. Everytime you run this routine, GT.M looks for MyRoutine.o in the .../o/utf8 directory and executes it. gtmprofile automatically modifies environment variables based on the required settings. In the above example, notice how gtmprofile modifies gtmroutines based on the character mode set for gtm\_chset.

If a GT.M environment exists, gtmprofile looks for and replaces substrings in environment variables with the GT.M version (gtmver and gtm\_dist). This allow you to to switch between different GT.M versions and character set modes (M and UTF-8). This way you can use the same source code files to generate different object code files, depending on the GT.M version and the setting of \$gtm\_chset.

On platforms where GT.M supports object code in shared libraries, if there are any shared libraries in object directories such as /usr/gtmnode1/.fis-gtm/V6.1-000\_x86\_64/o, the gtmprofile script automatically includes them in the gtmroutines environment variable.

The gtmprofile script also provides a framework to extend GT.M with plug-ins. Place the M code in \$gtm\_dist/plugin/r, the M mode object code in the \$gtm\_dist/plugin/o directory and UTF-8 mode object modules in the \$gtm\_dist/plugin/o/utf8 directory. On platforms where GT.M supports object code in shared libraries, you can create shared libraries instead of .o files. You can place external call tables in the plugin subdirectory, as well as shared libraries of calls to C code. The gtmprofile script automatically includes these in the gtmroutines environment variable.

## Basic Operations

gtmprofile automatically changes the environment variable values based on the values of gtmver and gtmkdir set by the last sourced gtmprofile. Consider the following example:

```
$ env | grep gtm
$ source /usr/lib/fis-gtm/V6.1-000_x86_64/gtmprofile
$ env | grep gtm
gtm_repl_instance=/home/jdoe/.fis-gtm/V6.1-000_x86_64/g/gtm.repl
gtm_log=/tmp/fis-gtm/V6.1-000_x86_64
gtm_prompt=GTM>
gtm_retention=42
gtmver=V6.1-000_x86_64
gtm_icu_version=4.2
gtmgbldir=/home/jdoe/.fis-gtm/V6.1-000_x86_64/g/gtm.gld
PATH=/home/jdoe/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin
:/bin:/usr/games:/usr/lib/fis-gtm/V6.1-000_x86_64
gtmroutines=/home/jdoe/.fis-gtm/V6.1-000_x86_64/o(/home/jdoe/.fis-gtm/V6.0-003_x86_64/r
/home/jdoe/.fis-gtm/r) /usr/lib/fis-gtm/V6.1-000_x86_64/plugin/o
(/usr/lib/fis-gtm/V6.1-000_x86_64/plugin/r)
/usr/lib/fis-gtm/V6.1-000_x86_64/libgtmutil.so /usr/lib/fis-gtm/V6.0-003_x86_64/
gtmdir=/home/jdoe/.fis-gtm
gtm_principal_editing=EDITING
gtm_tmp=/tmp/fis-gtm/V6.1-000_x86_64
gtm_dist=/usr/lib/fis-gtm/V6.1-000_x86_64
$
```

Notice how gtmprofile sets up \$gtmroutines to allow for a single source directory /home/jdoe/.fis-gtm/r to generate object files used by multiple GT.M versions, in this case in /home/jdoe/.fis-gtm/V6.1-000\_x86\_64/o. Now, let's consider the following example for a different version of GT.M.

Now, let us source gtmprofile for a different GT.M:

```
$ env | grep "gtm"
$ source /usr/lib/fis-gtm/V6.1-000_x86/gtmprofile
$ env | grep "gtm"
gtm_repl_instance=/home/jdoe/.fis-gtm/V6.1-000_x86/g/gtm.repl
gtm_log=/tmp/fis-gtm/V6.1-000_x86
gtm_prompt=GTM>
gtm_retention=42
gtmver=V6.1-000_x86
gtm_icu_version=4.4
gtmgbldir=/home/jdoe/.fis-gtm/V6.1-000_x86/g/gtm.gld
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:
/usr/lib/fis-gtm/V6.1-000_x86
gtmroutines=/home/jdoe/.fis-gtm/V6.1-000_x86/o(/home/jdoe/.fis-gtm/V6.0-003_x86/r
/home/jdoe/.fis-gtm/r)
/usr/lib/fis-gtm/V6.1-000_x86/plugin/o(/usr/lib/fis-gtm/V6.0-003_x86/plugin/r) /usr/lib/fis-gtm/V6.0-003_x86
gtmdir=/home/jdoe/.fis-gtm
gtm_principal_editing=EDITING
gtm_tmp=/tmp/fis-gtm/V6.1-000_x86
gtm_dist=/usr/lib/fis-gtm/V6.1-000_x86
$
```

Notice how gtmprofile automatically changes the environment variables to the new GT.M version. Notice that the same routines in /home/jdoe/.fis-gtm/r now put their object files in /home/jdoe/.fis-gtm/V6.1-000\_x86/o.

Now, let us edit gtmroutines and source the original gtmprofile:

```
$ export gtmroutines="/tmp/$gtmver $gtm_dist"
$ source /usr/lib/fis-gtm/V6.1-000_x86_64/gtmprofile
$ echo $gtmroutines /tmp/V6.1-000_x86_64 /usr/lib/fis-gtm/V6.0-003_x86_64/libgtmutil.so
$
```

Notice how gtmprofile again automatically modifies gtmroutines.



## Note

This scenario of sourcing gtmprofile is only for the sake of example.

gtmprofile creates the following aliases:

```
alias dse="$gtm_dist/dse"
alias gde="$gtm_dist/mumps -run GDE"
alias gtm="$gtm_dist/gtm"
alias lke="$gtm_dist/lke"
alias mupip="$gtm_dist/mupip"
```

If /var/log/fis-gtm/\$gtmver directory exists, gtmprofile sets it as the value for \$gtm\_log. If gtmprofile does not find /var/log/fis-gtm/\$gtmver, it uses \$gtm\_tmp to set the value of \$gtm\_log, which may not meet your file retention needs.

gtmprofile creates the following aliases:

```
alias dse="$gtm_dist/dse"
alias gde="$gtm_dist/mumps -run GDE"
alias gtm="$gtm_dist/gtm"
alias lke="$gtm_dist/lke"
alias mupip="$gtm_dist/mupip"
```

If /var/log/fis-gtm/\$gtmver directory exists, the gtmprofile script sets it as the value for \$gtm\_log. If gtmprofile does not find /var/log/fis-gtm/\$gtmver, it uses \$gtm\_tmp to set the value of \$gtm\_log, which may not meet your file retention needs.

## gtmcshrc

Sets a default GT.M environment for C type shell. It sets the \$gtm\_dist, \$gtmgbldir, \$gtm\_chset, \$gtmroutines, and adds \$gtm\_dist to the system environment variable PATH.

To source the gtmcshrc script, type:

```
$ source <path_to_GT.M_installation_directory>/gtmcshrc
```

You can also run the gtmbase script which places the above command in the .cshrc file so the script will get automatically sourced the next time you log in.

gtmcshrc also creates the following aliases.

```
alias gtm '$gtm_dist/mumps -direct'
alias mupip '$gtm_dist/mupip'
alias lke '$gtm_dist/lke'
alias gde '$gtm_dist/mumps -r ^GDE'
alias dse '$gtm_dist/dse'
```

Now you can run GT.M and its utilities without specifying a full path to the directory in which GT.M was installed.

## gtmbase

Adds the following line to `.profile` or `.cshrc` file depending on the shell.

In the POSIX shell, `gtmbase` adds the following line to `.profile`:

```
. <gtm_dist pathname>/gtmprofile
```

In the C shell, adds the following line to `.cshrc`:

```
source <gtm_dist pathname>/gtmcshrc
```

## gdedefaults

Specifies default or template values for database characteristics defined by GDE.

## gtm

The `gtm` script starts with `#!/bin/sh` so it can run with any shell. Also, you can use it to both run a program and run in direct mode. It sources `gtmprofile` and sets up default GT.M database files with `BEFORE_IMAGE` journaling. It automatically recovers the database on startup. This script sets up everything you need to run GT.M for a simple out-of-box experience.

For multi-user multi-environment systems, you should modify or replace the `gtm` script for your configuration.

The `gtm` script deletes all prior generation journal files (`*_<time and date stamp>` files) older than `$gtm_retention` days from the directory that contains the global directory (as pointed to by `$gtmgbldir`) and any subdirectories of that directory. By default, `$gtm_retention` is 42. However, you might want to align it with the backup frequency of your database.

Note that the removal of prior generation journal files is not specific to the database/journal files indicated by the current `$gtmgbldir` but the directory from where you run the `gtm` script.

If you plan to use GT.M in UTF-8 mode, set `$gtm_chset` to UTF-8 and then run the `gtm` script.

If you intend to use Database Encryption, set `$gtm_passwd` and `$gtm_dbkeys` first and then run the `gtm` script.

### To run the gtm script type:

```
$ <path to your GT.M Distribution>/gtm
```

### To invoke the help to assist first-time users, type:

```
$ <path to your GT.M Distribution>/gtm -help
gtm -dir[ect] to enter direct mode (halt returns to shell)
gtm -run <entryref> to start executing at an entryref
gtm -help / gtm -h / gtm -? to display this text
```

## Environment Variables

A comprehensive list of environment variables that are directly or indirectly used by GT.M follows:

**EDITOR** is a standard system environment variable that specifies the full path to the editor to be invoked by GT.M in response to the `ZEDit` command (defaults to `vi`, if `$EDITOR` is not set).

## Basic Operations

**gtm\_badchar** specifies the initial setting of the VIEW command that determines whether GT.M should raise an error when it encounters an illegal UTF-8 character sequence. It is ignored in M mode.

**gtm\_baktmpdir** specifies the directory where mupip backup creates temporary files. If \$gtm\_baktmpdir is not defined, GT.M uses the \$GTM\_BAKTMPDIR environment variable if defined, and otherwise uses the current working directory.

**gtm\_chset** determines the mode in which GT.M operates. If it has a value of "UTF-8" GT.M assumes that strings are encoded in UTF-8. In response to a value of "M" (on indeed anything other than "UTF-8"), GT.M treats all 256 combinations of the 8 bits in a byte as a single character.

**gtm\_chset\_locale** (z/OS only) specifies the locale for UTF-8 operations on z/OS.

**gtm\_collate\_n** specifies the shared library holding an alternative sequencing routine when collation is used. The syntax is gtm\_collate\_n=pathname where n is an integer from 1 to 255 that identifies the collation sequence, and pathname identifies the shared library containing the routines for that collation sequence.

**gtm\_db\_startup\_max\_wait** specifies how long to wait for a resolution of any resource conflict when they first access a database file. GT.M uses semaphores maintained using UNIX Inter-Process Communication (IPC) services to ensure orderly initialization and shutdown of database files and associated shared memory. Normally the IPC resources are held in an exclusive state only for very brief intervals. However, under unusual circumstances that might include extreme large numbers of simultaneous database initializations, a long-running MUPIP operation involving standalone access (like INTEG -FILE or RESTORE), an OS overload or an unpredicted process failure the resources might remain unavailable for an unanticipated length of time. \$gtm\_db\_startup\_max\_wait specifies how long to wait for the resources to become available:

- -1 - Indefinite wait until the resource becomes available; the waiting process uses the gtm\_proctuckexec mechanism at 48 and 96 seconds.
- 0 - No wait - if the resource is not immediately available, give a DBFILERR error with an associated SEMWT2LONG
- > 0 - Seconds to wait - rounded to the nearest multiple of eight (8); if the specification is 96 or more seconds, the waiting process uses the gtm\_proctuckexec mechanism at one half the wait and at the end of the wait; if the resource remains unavailable, the process issues DBFILERR error with an associated SEMWT2LONG

The default value for the wait if \$gtm\_db\_startup\_max\_wait is not defined is 96 seconds.

**gtm\_crypt\_plugin:** If you wish to continue using Blowfish CFB with V6.0-001 via the reference implementation of the plugin, you need to change a symbolic link post-installation, or define the environment variable gtm\_crypt\_plugin. If the environment variable gtm\_crypt\_plugin is defined and provides the path to a shared library relative to \$gtm\_dist/plugin, GT.M uses \$gtm\_dist/plugin/\$gtm\_crypt\_plugin as the shared library providing the plugin. If \$gtm\_crypt\_plugin is not defined, GT.M expects \$gtm\_dist/plugin/libgtmcrpt.so to be a symbolic link to a shared library providing the plugin. The expected name of the actual shared library is libgtmcrpt\_cryptlib\_CIPHER.so (depending on your platform, the actual extension may differ from .so), for example, libgtmcrpt\_openssl\_AESCFB. GT.M cannot and does not ensure that the cipher is actually AES CFB as implemented by OpenSSL - GT.M uses CIPHER as salt for the hashed key in the database file header, and cryptlib is for your convenience, for example, for troubleshooting. Installing the GT.M distribution creates a default symbolic link.

**gtm\_custom\_errors** specifies the complete path to the file that contains a list of errors that should automatically stop all updates on those region(s) of an instance which have the Instance Freeze mechanism enabled.

**gtm\_dbkeys** is used by the encryption reference plugin (not used by GT.M directly) for the name of a file providing a list of database files and their corresponding key files.

**gtm\_dist** specifies the path to the directory containing the GT.M system distribution. gtm\_dist must be defined for each user. If you are not using the **gtm** script or sourcing **gtmprofile**, consider defining gtm\_dist in the login file or as part of the default system environment. In UTF-8 mode, the gtm\_dist environment variable specifies the path to the directory containing the

## Basic Operations

GT.M system distribution for Unicode. The distribution for Unicode is located in subdirectory utf8 under the GT.M distribution directory. For example, if the GT.M distribution is in /usr/lib/fis-gtm/V6.1-000\_x86, set gtm\_dist to point to /usr/lib/fis-gtm/V6.0-003\_x86/utf8 for UTF-8 mode. Correct operation of GT.M executable programs requires gtm\_dist to be set correctly.

**gtm\_env\_translate** specifies the path to a shared library to implement the optional GT.M environment translation facility.

**gtm\_etrp** specifies an initial value of \$ETRAP to override the default value of "B" for \$ZTRAP as the base level error handler. The gtmprofile script sets gtm\_etrp to "**Write:(0=\$STACK) ""Error occurred: "",\$ZStatus,!"**" which you can customize to suit your needs.

**gtm\_extract\_nocol** specifies whether a MUPIP JOURNAL -EXTRACT (when used without -RECOVER or -ROLLBACK) on a database with custom collation should use the default collation if it is not able to read the database file. In a situation where the database file is inaccessible or the replication instance is frozen with a critical section required for the access held by another process and the environment variable gtm\_extract\_nocol is defined and evaluates to a non-zero integer or any case-independent string or leading substrings of "TRUE" or "YES", MUPIP JOURNAL -EXTRACT issues the DBCOLLREQ warning and proceeds with the extract using the default collation. If gtm\_extract\_nocol is not set or evaluates to a value other than a positive integer or any case-independent string or leading substrings of "FALSE" or "NO", MUPIP JOURNAL -EXTRACT exits with the SETEXTRENV error.



### Warning

Note that if default collation is used for a database with custom collation, the subscripts reported by MUPIP JOURNAL -EXTRACT are those stored in the database, which may differ from those read and written by application programs.

**gtm\_fullblockwrites** specifies whether a GT.M process should write a full database block worth of bytes when writing a database block that is not full. Depending on your IO subsystem, writing a full block worth of bytes (even when there are unused garbage bytes at the end) may result in better database IO performance by replacing a read-modify-read low level IO operation with a single write operation.

**gtm\_gdscert** specifies the initial value of the VIEW command that controls whether GT.M processes should test updated database blocks for structural damage. By default, GT.M does not check database blocks for structural damage, because the impact on performance is usually unwarranted.

**gtm\_gvdupsetnoop** specifies the initial value of the VIEW command that controls whether a GT.M process should enable duplicate SET optimization. If it is defined, and evaluates to a non-zero integer or any case-independent string or leading substrings of "TRUE" or "YES", a SET command does not change the value of an existing node, GT.M does not perform the update or execute any trigger code specified for the node.

**gtm\_icu\_version** specifies the MAJOR VERSION and MINOR VERSION numbers of the desired ICU. For example "3.6" denotes ICU-3.6. If \$gtm\_chset has the value "UTF-8", GT.M requires libicu with version 3.6 or higher. If you must chose between multiple versions of libicu or if libicu has been compiled with symbol renaming enabled, GT.M requires gtm\_icu\_version to be explicitly set.

**gtm\_ipv4\_only** specifies whether a Source Server should establish only IPv4 connections with a Receiver Server. If it is defined, and evaluates to a non-zero integer, or any case-independent string or leading substring of "TRUE" or "YES", the Source Server establishes only IPv4 connections with the Receiver Server. Set gtm\_ipv4\_only for environments where different server names are not used for IPv4 and IPv6 addresses and the Source Server connects to a Receiver Server running a GT.M version prior to V6.0-003.

**gtm\_jnl\_release\_timeout** specifies the number of seconds that a replicating Source Server waits when there is no activity on an open journal file before closing it. The default wait period is 300 seconds (5 minutes). If \$gtm\_jnl\_release\_timeout specifies

0, the Source Server keeps the current journal files open until shutdown. The maximum value for `$gtm_jnl_release_timeout` is 2147483 seconds.

**gtm\_keep\_obj** specifies whether the `gtminstall` script should delete the object files from the GT.M installation directory. If `gtm_keep_obj` is set to "Y", the `gtminstall` script enables the `--keep-obj` option. By default, `--keep-obj` is disabled.

**gtm\_lct\_stdnull** specifies whether a GT.M process should use standard collation for local variables with null subscripts or legacy GT.M collation.

**gtm\_local\_collate** specifies an alternative collation sequence for local variables.

**gtm\_log** specifies a directory where the `gtm_secshr_log` file is stored. The `gtm_secshr_log` file stores information gathered in the `gtmsecshr` process. FIS recommends that a system-wide default be established for `gtm_log` so that `gtmsecshr` always logs its information in the same directory, regardless of which user's GT.M process invokes `gtmsecshr`. In conformance with the Filesystem Hierarchy Standard, FIS recommends `/var/log/fis-gtm/$gtmver` as the value for `$gtm_log` unless you are installing the same version of GT.M in multiple directories. Note that `$gtmver` can be in the form of `V6.1-000_x86` which represents the current GT.M release and platform information. If you do not set `$gtm_log`, GT.M creates log files in a directory in `/tmp` (AIX, GNU/Linux, Tru64 UNIX) or `/var/tmp` (HP-UX, Solaris). However, this is not recommended because it makes GT.M log files vulnerable to the retention policy of a temporary directory.



### Important

Starting with V6.0-000, `gtmsecshr` logs its messages in the system log and the environment variable `gtm_log` is ignored.

**gtm\_tprestart\_log\_delta** specifies the number of transaction restarts for which GT.M should wait before reporting a transaction restart to the operator logging facility. If `gtm_tprestart_log_delta` is not defined, GT.M initializes `gtm_tp_restart_log_delta` to 0.

**gtm\_tprestart\_log\_first** specifies the initial number of transaction restarts for which GT.M should wait before reporting any transaction restart to the operator logging facility. If `gtm_tprestart_log_delta` is defined and `gtm_tprestart_log_first` is not defined, GT.M initializes `gtm_tprestart_log_first` to 1.

**gtm\_max\_sockets** specifies the maximum number of client connections for socket devices. The default is 64.

**gtm\_memory\_reserve** specifies the size in kilobytes of the reserve memory that GT.M should use in handling and reporting an out-of-memory condition. The default is 64 (KB).

**gtm\_nocenable** specifies whether the `$principal` terminal device should ignore `<CTRL-C>` or use `<CTRL-C>` as a signal to place the process into direct mode; a `USE` command can modify this device characteristic. If `gtm_nocenable` is defined and evaluates to a non-zero integer or any case-independent string or leading substrings of "TRUE" or "YES", `$principal` ignores `<CTRL-C>`. If `gtm_nocenable` is not set or evaluates to a value other than a positive integer or any case-independent string or leading substrings of "FALSE" or "NO", `<CTRL-C>` on `$principal` places the process into direct mode at the next opportunity (usually at a point corresponding to the beginning of the next source line).

**gtm\_non\_blocked\_write\_retries** modifies FIFO or PIPE write behavior. A `WRITE` which would block is retried up to the number specified with a 100 milliseconds delay between each retry. The default value is 10 times.

**gtm\_noundef** specifies the initial value of the `VIEW` command that controls whether a GT.M process should treat undefined variables as having an implicit value of an empty string. If it is defined, and evaluates to a non-zero integer or any case-independent string or leading substring of "TRUE" or "YES", then GT.M treats undefined variables as having an implicit value of an empty string. By default, GT.M signals an error on an attempt to use the value of an undefined variable.

## Basic Operations

**gtm\_passwd** used by the encryption reference plugin (not used by GT.M directly) for the obfuscated (not encrypted) password to the GNU Privacy Guard key ring.

**gtm\_patnumeric** specifies the initial value of the ISV \$ZPATNUMERIC in UTF-8 mode.

**gtm\_pattern\_file** and **gtm\_pattern\_table** specify alternative patterns for the ? (pattern) syntax.

**gtm\_principal** specifies the value for \$principal, which designates an alternative name (synonym) for the principal \$IO device.

**gtm\_principal\_editing** determines whether the previous input to a READ command can be recalled and edited before pressing ENTER to submit it.



### Note

The GT.M direct mode commands have a more extensive capability in this regard, independent of the value of this environment variable.

**gtm\_prompt** specifies the initial value of the ISV \$ZPROMPT, which controls the GT.M direct mode prompt. By default, the direct mode prompt is "GTM>".

**gtm\_procstuckexec** specifies a shell command or a script to execute when any of the following conditions occur:

- An explicit MUPIP FREEZE or an implicit freeze, such as BACKUP, INTEG -ONLINE, and so on, after a one minute wait on a region.
- MUPIP actions find kill\_in\_prog (KILLS in progress) to be non-zero after a one minute wait on a region. Note that GT.M internally maintains a list of PIDs (up to a maximum of 8 PIDs) currently doing a KILL operation.
- The BUFOWNERSTUCK, INTERLOCK\_FAIL, JNLPROCSTUCK, SHUTDOWN, WRITERSTUCK, MAXJNLQIOLOCKWAIT, MUTEXLCKALERT, SEMWT2LONG, and COMMITWAITPID operator messages are being logged.

You can use this as a monitoring facility for processes holding a resource for an unexpected amount of time. Typically, for the shell script or command pointed to by gtm\_procstuckexec, you would write corrective actions or obtain the stack trace of the troublesome processes (using their PIDs). GT.M passes arguments to the shell command / script in the order specified as follows:

1. *condition* is the name of the condition. For example, BUFOWNERSTUCK, INTERLOCK\_FAIL, and so on.
2. *waiting\_pid* is the PID of the process reporting the condition.
3. *blocking\_pid* is the PID of the process holding a resource.
4. *count* is the number of times the script has been invoked for the current condition (1 for the first occurrence).

Each invocation generates an operator log message and if the invocation fails an error message to the operator log. The shell script should start with a line beginning with #! that designates the shell.



### Note

Make sure user processes have sufficient space and permissions to run the shell command / script.

**gtm\_obfuscation\_key**: If \$gtm\_obfuscation\_key specifies the name of file readable by the process, the encryption reference plug-in uses an SHA-512 hash of the file's contents as the XOR mask for the obfuscated password in the environment variable

## Basic Operations

**gtm\_passwd.** When `gtm_obfuscation_key` does not point to a readable file, the plug-in creates an XOR mask based on the `userid` and `inode` of the `mumps` executable and then computes an SHA-512 hash of the XOR mask to use as a mask.

`gtm_obfuscation_key` can be used as a mechanism to pass an obfuscated password between unrelated processes (for example, a child process with a different `userid` invoked via a `sudo` mechanism), or even from one system to another (for example, over an `ssh` connection).

**gtm\_quiet\_halt** specifies whether GT.M should disable the `FORCEDHALT` message when the process is stopped via `MUPIP STOP` or by a `SIGTERM` signal (as sent by some web servers).

**gtm\_repl\_instance** specifies the location of the replication instance file when database replication is in use.

**gtm\_repl\_instsecondary** specifies the name of the replicating instance in the current environment. GT.M uses `$gtm_repl_instsecondary` if the `-instsecondary` qualifier is not specified.

**gtm\_retention** (not used by GT.M directly) - used by the `gtm` script to delete old journal files and old temporary files it creates.

**gtm\_side\_effects:** When the environment variable `gtm_side_effects` is set to one (1) at process startup, GT.M generates code that performs left to right evaluation of actual arguments, function arguments, operands for non-Boolean binary operators, `SET` arguments where the target destination is an indirect subscripted `glvn`, and variable subscripts. When the environment variable is not set, or set to zero (0), GT.M retains its traditional behavior, which re-orders the evaluation of operands using rules intended to improve computational efficiency. This reordering assumes that functions have no side effects, and may generate unexpected behavior (`x+$increment(x)` is a pathological example). When `gtm_side_effects` is set to two (2), GT.M generates code with the left-to-right behavior, and also generates `SIDEFFECTEVAL` warning messages for each construct that potentially generates different results depending on the order of evaluation. As extrinsic functions and external calls are opaque to the compiler at the point of their invocation, it cannot statically determine whether there is a real interaction. Therefore `SIDEFFECTEVAL` warnings may be much more frequent than actual side effect interactions and the warning mode may be most useful as a diagnostic tool to investigate problematic or unexpected behavior in targeted code rather than for an audit of an entire application. Note that a string of concatenations in the same expression may generate more warnings than the code warrants. Other values of the environment variable are reserved for potential future use by FIS. It is important to note that `gtm_side_effects` affects the generated code, and must be in effect when code is compiled - the value when that compiled code is executed is irrelevant. Note also that `XECUTE` and `auto-ZLINK`, `explicit ZLINK` and `ZCOMPILE` all perform run-time compilation subject to the characteristic selected when the process started. Please be aware that programming style where one term of an expression changes a prior term in the same expression is an unsafe programming practice. The existing environment variable `gtm_boolean` may separately control short-circuit evaluation of Boolean expressions but a setting of 1 (or 2) for `gtm_side_effects` causes the same boolean evaluations as setting `gtm_boolean` to 1 (or 2). The differences in the compilation modes may include not only differences in results, but differences in flow of control.

**gtm\_snaptmpdir** specifies the location to place the temporary "snapshot" file created by facilities such as on-line `mupip integ`. If `$gtm_snaptmpdir` is not defined, GT.M uses the `$GTM_BAKTMPDIR` environment variable if defined, and otherwise uses the current working directory.

**gtm\_stdckill** enables the standard-compliant behavior to kill local variables in the exclusion list if they had an alias that is not in the exclusion list. By default, this behavior is disabled.

**gtm\_sysid** specifies the value for the second piece of the `$$System ISV`. `$$SYSTEM` contains a string that identifies the executing M instance. The value of `$$SYSTEM` is a string that starts with a unique numeric code that identifies the manufacturer. Codes were originally assigned by the MDC (MUMPS Development Committee). `$$SYSTEM` in GT.M starts with "47" followed by a comma and `$gtm_sysid`.

**gtm\_tmp** specifies a directory where socket files used for communication between `gtmsecshr` and GT.M processes are stored. All processes using the same GT.M should have the same `$gtm_tmp`.

## Basic Operations

**gtm\_tpnnotacidtime** specifies the maximum time that a GT.M process waits for non-Isolated timed command (JOB, LOCK, OPEN, READ, or ZALLOCATE) running within a transaction to complete before it releases all critical sections it owns and sends a TPNOTACID information message to the system log. A GT.M process owns critical sections on all or some of the regions participating in a transactions only during final retry attempts (when \$TRETRY>3). **gtm\_tpnnotacidtime** specifies time in seconds; the default is 2 seconds. The maximum value of **gtm\_tpnnotacidtime** is 30 and the minimum is 0. If **gtm\_tpnnotacidtime** specifies a time outside of this range, GT.M uses the default value. The GT.M behavior of releasing critical sections in final retry attempt to provide protection from certain risky coding patterns which, because they are not Isolated, can cause deadlocks (in the worst case) and long hangs (in the best case). As ZSYSTEM and BREAK are neither isolated nor timed, GT.M initiates TPNOTACID behavior for them immediately as it encounters them during execution in a final retry attempt (independent of **gtm\_tpnnotacidtime**). Rapidly repeating TPNOTACID messages are likely associated with live-lock, which means that a process is consuming critical resources repeatedly within a transaction, and is unable to commit because the transaction duration is too long to commit while maintaining ACID transaction properties.

**gtm\_trace\_gbl\_name** enables GT.M tracing at process startup. Setting **gtm\_trace\_gbl\_name** to a valid global variable name instructs GT.M to report the data in the specified global when a VIEW command disables the tracing, or implicitly at process termination. This setting behaves as if the process issued a VIEW "TRACE" command at process startup. However, **gtm\_trace\_gbl\_name** has a capability not available with the VIEW command, such that if the environment variable is defined but evaluates to zero (0) or, only on UNIX, to the empty string, GT.M collects the M-profiling data in memory and discards it when the process terminates (this feature is mainly used for in-house testing). Note that having this feature activated for process that otherwise don't open a database file (such as GDE) can cause them to encounter an error.

**gtm\_trigger\_etrp** provides the initial value for \$ETRAP in trigger context; can be used to set trigger error traps for trigger operations in both mumps and MUPIP processes.

**gtm\_zdate\_form** specifies the initial value for the \$ZDATE ISV.

**gtm\_zinterrupt** specifies the initial value of the ISV \$ZINTERRUPT which holds the code that GT.M executes (as if it is the argument for an XECUTE command) when a process receives a signal from a MUPIP INTRPT command.

**gtm\_zlib\_cmp\_level** specifies the zlib compression level used in the replication stream by the source and receiver servers. By default, replication does not use compression.

**gtm\_zmaxtptime** specifies the initial value of the \$ZMAXTPTIME Intrinsic Special Variable, which controls whether and when GT.M issues a TPTIMEOUT error for a TP transaction that runs too long. **gtm\_zmaxtptime** specifies time in seconds and the default is 0, which indicates "no timeout" (unlimited time). The maximum value of **gtm\_zmaxtptime** is 60 seconds and the minimum is 0; this range check does not apply to SET \$ZMAXTPTIME. GT.M ignores **gtm\_zmaxtptime** if it contains a values outside of this recognized range.

**gtm\_zquit\_anyway** specifies whether the code of the form QUIT <expr> execute as if it were SET <tmp>=<expr> QUIT:\$QUIT tmp QUIT, where <tmp> is a temporary local variable in the GT.M runtime system that is not visible to application code. This setting is not a compiler-time setting, but rather a run-time one. If **gtm\_zquit\_anyway** is defined and evaluates to 1 or any case-independent string or leading substrings of "TRUE" or "YES", code of the form QUIT <expr> executes as if it were SET <tmp>=<expr> QUIT:\$QUIT tmp QUIT. If **gtm\_zquit\_anyway** is not defined or evaluates to 0 or any case-independent string or leading substrings of "FALSE" or "NO", there is no change in the execution of code of the form QUIT <expr>.

**gtm\_ztrap\_form** and **gtm\_zyerror specify** the behavior of error handling specified by \$ZTRAP as described in the Error Processing chapter of the *GT.M Programmer's Guide*.

**gtm\_ztrap\_new** specifies whether a Set of \$ZTRAP also implicitly News it.

**GTMC**I specifies the call-in table for function calls from C code to M code.

**gtmcompile** specifies the initial value of the \$ZCompile ISV.

## Basic Operations

**gtmconfig** specifies the location of the configuration file required for database encryption and/or TLS support. A configuration file is divided into two sections—database encryption section and TLS section. The database encryption section contains a list of database files and their corresponding key files. Do not add a database encryption section if you are not using an encrypted database. The TLS section contains a TLSID label that identifies the location of root certification authority certificate in PEM format and leaf-level certificate with its corresponding private key file. Note that the use of the `gtmconfig` environment variable require the `libconfig` library to be installed.

**gtmFIPS** specifies whether the plugin reference implementation should attempt to use either OpenSSL or Libcrypt to provide database encryption that complies with FIPS 140-2. When the environment variable `$gtmFIPS` is set to 1 (or evaluates to a non-zero integer, or any case-independent string or leading substring of "TRUE" or "YES"), the plugin reference implementation attempts to use `libcrypt` (from GnuPG) and `libcrypto` (OpenSSL) in "FIPS mode".

**gtmblidir** specifies the initial value of the `$ZGBLDIR` ISV. `$ZGBLDIR` identifies the global directory. A global directory maps global variables to physical database files, and is required to access M global variables. Users who maintain multiple global directories must choose one to use. To automate this definition, define `gtmblidir` in the user's login file.

**gtmroutines** specifies the initial value of the `$ZROUTINES` ISV.

**gtmtls\_passwd\_<label>** `$gtmtls_passwd_<label>` specifies the obfuscated password of the encrypted private key pair. You can obfuscate passwords using the 'maskpass' utility provided along with the encryption plugin. If you choose to use unencrypted private keys, set the `gtmtls_passwd_<label>` environment variable to a non-null dummy value; this prevents inappropriate prompting for a password.

**gtmver** (not used by GT.M directly) - The current GT.M version. The `gtmprofile` script uses `$gtmver` to set other environment variables.

**GTMXC\_gpgagent** specifies the location of `gpgagent.tab`. By default, GT.M places `gpgagent.tab` in the `$gtm_dist/plugin/` directory. `GTMXC_gpgagent` is used by `pinentry-gtm.sh` and is meaningful only if you are using Gnu Privacy Guard version 2.

**LC\_CTYPE** is a standard system environment variable used to specify a locale. When `$gtm_chset` has the value "UTF-8", `$LC_CTYPE` must specify a UTF-8 locale (e.g., "en\_US.utf8").

**LC\_ALL** is a standard system environment variable used to select a locale with UTF-8 support. `LC_ALL` is an alternative to `LC_TYPE`, which overrides `LC_TYPE` and has a more pervasive effect on other aspects of the environment beyond GT.M.

**LD\_LIBRARY\_PATH** is a standard system environment variable (`LIBPATH` in AIX) to point to the location of ICU. GT.M requires ICU 3.6 (or above) to perform Unicode related functionality.

**old\_gtm\_dist** (not used by GT.M directly) - The path of the prior GT.M distribution. The `gtmprofile` script uses this value to set other environment variables.

**old\_gtmroutines** (not used by GT.M directly) - The prior routine search path. The `gtmprofile` script uses this value to set other environment variables.

**old\_gtmver** (not used by GT.M directly) - The value of `gtmver` that was set when the `gtmprofile` script was last sourced. The `gtmprofile` script uses this value to set other environment variables.

**tmp\_gtm\_tmp** (not used by GT.M directly) - It is used by the `gtmprofile` script in maintaining `gtm_tmp`.

**tmp\_passwd** (not used by GT.M directly) - It is used by the `gtmprofile` script in maintaining `gtm_passwd`.

**TZ** is a standard system environment variable that specifies the timezone to be used by GT.M processes, if they are not to use the default system timezone (GT.M assumes the system clock is set to UTC).

## Basic Operations

The `gtmprofile` and `gtmschrc` scripts sets the following environment variables. FIS recommends using the `gtmprofile` script (or the `gtm` script which sources `gtmprofile`) to set up an environment for GT.M.

Environment Variables	Set up by GT.M shell scripts
<code>gtm_dist*</code>	<code>gtmprofile</code> , <code>gtmschrc</code>
<code>gtmgbldir*</code>	<code>gtmprofile</code> , <code>gtmcshrc</code>
<code>gtm_icu_version</code>	<code>gtmprofile</code>
<code>gtm_log*</code>	<code>gtmprofile</code>
<code>gtm_principal_editing</code>	<code>gtmprofile</code>
<code>gtm_prompt</code>	<code>gtmprofile</code>
<code>gtm_retention</code>	<code>gtmprofile</code>
<code>gtmroutines*</code>	<code>gtmprofile</code> , <code>gtmcshrc</code>
<code>gtm_tmp</code>	<code>gtmprofile</code>
<code>gtmver</code>	<code>gtmprofile</code>
<code>gtm_repl_instance</code>	<code>gtmprofile</code>
<code>LC_CTYPE</code>	<code>gtmprofile</code>
<code>old_gtm_dist</code>	<code>gtmprofile</code>
<code>old_gtmroutines</code>	<code>gtmprofile</code>
<code>old_gtmver</code>	<code>gtmprofile</code>
<code>tmp_gtm_tmp</code>	<code>gtmpropfile</code>
<code>tmp_passw</code>	<code>gtmprofile</code>
* denotes environment variables that must be defined for normal GT.M operation.	

While creating an environment for multiple processes accessing the same version of GT.M, bear in mind the following important points:

1. A GT.M version has an associated `gtmsecshr` (located by `$gtm_dist`). If multiple processes are accessing the same GT.M version, each process must use the same combination of `$gtm_tmp` and `$gtm_log`.
2. In conformance with the Filesystem Hierarchy Standard, FIS recommends `/var/log/fis-gtm/$gtmver` as the value for `$gtm_log`. Note that `$gtmver` can be in the form of `V5.4-001_x86` which represents the current GT.M release and platform information.
3. FIS recommends setting `$gtm_tmp` to a temporary directory `/tmp` (*AIX, GNU/Linux, Tru64 UNIX*) or `/var/tmp` (*HP-UX, Solaris*). The `gtmprofile` script sets `$gtm_tmp` to `/tmp/fis-gtm/$gtmver`.
4. If you do not set `$gtm_log`, GT.M creates log files in a directory in `/tmp` (*AIX, GNU/Linux, Tru64 UNIX*) or `/var/tmp` (*HP-UX, Solaris*). However, this is not recommended because it makes GT.M log files vulnerable to the retention policy of a temporary directory.

Always set the same value of `$gtm_tmp` for all processes using the same GT.M version. Having different `$gtm_tmp` for multiple processes accessing the same GT.M version may prevent processes from being able to communicate with `gtmsecshr` and cause performance issues.

---

### Configuring and operating GT.M with Unicode™ support (optional)

The configure script provides the option to install GT.M with or without Unicode™ support for encoding international character sets. This section describes the system environment required to install and operate GT.M with Unicode™ support. Users who handle data in ASCII or other single-byte character sets such as one of the ISO-8859 representations and do not foresee any use of character sets beyond single byte character sets, may proceed to the next section.

### M mode and UTF-8 mode

A GT.M process can operate in either M mode or UTF-8 mode. In certain circumstances, both M mode and UTF-8 mode may concurrently access the same database.

`$gtm_chset` determines the mode in which a process operates. If it has a value of M, GT.M treats all 256 combinations of the 8 bits in a byte as a character, which is suitable for many single-language applications.

If `$gtm_chset` has a value of UTF-8, GT.M (at process startup) interprets strings as being encoded in UTF-8. In this mode, all functionality related to Unicode™ becomes available and standard string-oriented operations operate with UTF-8 encoding. In this mode, GT.M detects character boundaries (since the size of a character is variable length), calculates glyph display width, and performs string conversion between UTF-8 and UTF-16.

If you install GT.M with Unicode support, all GT.M components related to M mode reside in your GT.M distribution directory and Unicode-related components reside in the `utf8` subdirectory of your GT.M distribution. For processes in UTF-8 mode, in addition to `gtm_chset`, ensure that `$gtm_dist` points to the `utf8` subdirectory, that `$gtmroutines` includes the `utf8` subdirectory (or the `libtmutil.so` therein) rather than its parent directory.

### Compiling ICU

GT.M uses ICU 3.6 (or above) to perform Unicode™-related operations. GT.M generates the distribution for Unicode only if ICU 3.6 (or above) is installed on the system. Therefore, install an appropriate ICU version before installing GT.M to perform functionality related to Unicode.

Note that the ICU installation instructions may not be the same for every platform. If `libicu` has been compiled with **symbol renaming enabled**, GT.M requires `$gtm_icu_version` be explicitly set. By default, GT.M uses the most current installed version of ICU. GT.M expects ICU to have been built with **symbol renaming disabled** and issues an error at startup if the currently installed version of ICU has been built with **symbol renaming enabled**. To use a different version of ICU (not the currently installed) or a version of ICU built with **symbol renaming enabled**, use `$gtm_icu_version` to specify the **MAJOR VERSION** and **MINOR VERSION** numbers of the desired ICU formatted as *MajorVersion.MinorVersion* (for example "3.6" to denote ICU-3.6). When `$gtm_icu_version` is so defined, GT.M attempts to open the specific version of ICU. In this case, GT.M works regardless of whether or not symbols in this ICU have been renamed. A missing or ill-formed value for this environment variable causes GT.M to only look for non-renamed ICU symbols. Note that display widths for a few characters are different starting in ICU 4.0.



#### Note

If you are using `gtmprofile`, you do not have to set `$gtm_icu_version`.

After installing ICU 3.6 (or above), you also need to set the following environment variables to an appropriate value.

1. LC\_CTYPE
2. LC\_ALL
3. LD\_LIBRARY\_PATH
4. TERM

---

## Starting GT.M

### To start GT.M from a POSIX shell:

1. Execute **gtm** from your shell prompt:

```
$ <path_to_gtm_installation_directory>/gtm
```

### To start GT.M in UTF-8 mode from a POSIX shell:

1. First, set **\$gtm\_chset** to *UTF-8*.

```
$ export gtm_chset="UTF-8"
```

2. Execute the **gtm** script.

```
$ <path_to_gtm_installation_directory>/gtm
```

### To start GT.M from a C-type shell:

1. First source the **gtmschrc** script to set up a default GT.M environment. At your shell prompt, type:

2. 

```
$ source <path_to_gtm_installation_directory>/gtmschrc
```

3. Run the **gtm** alias to start GT.M in direct mode.

```
$ gtm
```

### To start GT.M in UTF-8 mode from a C-type shell:

1. Set the environment variable **gtm\_chset** to UTF-8.

```
$ setenv gtm_chset UTF-8
```

2. Source the **gtmschrc** script to set up default GT.M unicode environment.

```
$ source <path_to_gtm_installation_directory>/gtmschrc
```

3. Run the **gtm** alias to start GT.M in direct mode.

```
$ gtm
```

### To start GT.M without using any script:

1. Define **gtm\_dist**, **gtm\_log**, **gtm\_tmp**, **gtmgbldir**, and **gtmroutines**. Ensure that **gtm\_dist** points to the location of your GT.M distribution.
2. Add **gtm\_dist** to the system environment variable **PATH**.

## Basic Operations

3. Ensure that you have set an appropriate value for TERM.
4. If possible, add these environment variables in your login file so you do not have to create them again the next time you start your shell.
5. Set the following aliases to run GT.M and its utilities.

```
alias dse="$gtm_dist/dse"
alias gde="$gtm_dist/mumps -run ^GDE"
alias gtm="$gtm_dist/mumps -direct"
alias lke="$gtm_dist/lke"
alias mupip="$gtm_dist/mupip"
```

6. Run the **gtm** alias to start GT.M in direct mode.

```
$ gtm
```

### To start GT.M in UTF-8 mode without using any script:

1. Define `gtm_dist`, `gtm_log`, `gtmgbldir`, and `gtmroutines`. Ensure that `gtm_dist` points to the `utf8` subdirectory of your GT.M distribution.
2. Set `gtm_routines` to include the `utf8` subdirectory of your GT.M distribution. Note that the `utf8` subdirectory includes all Unicode-related GT.M functionality. Type a command like the following:

```
$ gtmroutines=". $gtm_dist `echo $gtm_dist | sed 's/utf8(\\/\)*//'\`"
```

3. Ensure that you have installed ICU 3.6 (or above) and have set appropriate values for `LC_CTYPE`, `LC_ALL`, `LD_LIBRARY_PATH`, and `TERM`.

If you have built ICU with symbol renaming enabled, set `gtm_icu_version` to an appropriate ICU version.

4. Add `gtm_dist` to the system environment variable `PATH`.
5. Set `gtm_chset` to UTF-8.
6. If possible, add these environment variables in your login file so you do not have to create them again the next time you start your shell.
7. Set the following aliases to run GT.M and its utilities.

```
alias dse="$gtm_dist/dse"
alias gde="$gtm_dist/mumps -run ^GDE"
alias gtm="$gtm_dist/mumps -direct"
alias lke="$gtm_dist/lke"
alias mupip="$gtm_dist/mupip"
```

8. Type the following command to start GT.M in direct mode.

```
$ gtm
```

9. At the GT.M prompt, type the following command.

```
GTM>w $ZCHSET
UTF-8 ; the output confirms UTF-8 mode.
```



## Note

If you are configuring a GT.M environment without using the **gtmprofile** script (or the **gtm** script which sources **gtmprofile**), bear in mind the following recommendation from FIS:

- All GT.M processes should use the same settings for **gtm\_log** and **gtm\_tmp**, especially for production environments. This is because **gtmsecshr** inherits these values from whichever GT.M process first uses its services.
- If there are multiple GT.M versions active on a system, FIS recommends different sets of **gtm\_log** and **gtm\_tmp** values for each version as using the same values for different distributions can cause significant performance issues.

GT.M has three invocation modes: compiler, direct, and auto-start. To invoke GT.M in these modes, provide the following arguments to the **gtm** script or the **mumps** command.

1. **-direct**: Invokes GT.M in direct mode where you can enter M commands interactively.
2. **<list of M source files>**: Invokes GT.M in compiler mode, invoke GT.M by entering a list of file names to compile as a argument. GT.M then compiles the specified programs into .o files. UNIX wildcards (\* and ?) are acceptable within the file names.
3. **-run ^routine\_name**: -r invokes GT.M in auto-start mode. The second argument is taken to be an M entryref, and that routine is automatically executed, bypassing direct mode. Depending on the shell being used, you may need to put the entryref in quotes.

When executing M programs, GT.M incrementally links any called programs. For example, the command **GTM> d ^TEST** links the object file TEST.o and executes it; if the **TESTM** program calls other M routines, those are automatically compiled and linked.

---

## Configuring huge pages for GT.M x86[-64] on Linux

GT.M on Linux supports using huge pages (depending on the CPU architecture, typically 2MiB or 4MiB rather than the default 4KiB) for shared memory segments for BG access and for the heap of mumps processes. Using huge pages reduces the need for page tables at the cost of a slight increase in total system virtual memory usage. While your mileage may vary, FIS believes that many large applications will benefit from configuring and using huge pages. If you do not configure your system for huge pages, GT.M continues to use the default page size. Note that using huge pages requires no change whatsoever at the application code level, or even to database management operations such as replication, backup, integ, reorg, etc. - the changes discussed here pertain entirely to configuring system memory and the virtual memory manager to improve application performance.

To use huge pages:

- You must have a 32- or 64-bit x86 CPU, running a Linux kernel with huge pages enabled. All currently Supported Linux distributions appear to support huge pages; to confirm, use the command: `grep hugetlbfs /proc/filesystems` which should report: `nODEV hugetlbfs`
- You must have `libhugetlbfs.so` installed in a standard location for system libraries. Installing the library using your Linux system's package manager should place it in a standard location. (Note that `libhugetlbfs` is not in Debian repositories and must be manually installed; GT.M on Debian releases is Supportable, not Supported.)

## Basic Operations

- You must have a sufficient number of huge pages available:
  - To reserve Huge Pages boot Linux with the `hugepages=num_pages` kernel boot parameter; or, shortly after bootup when unfragmented memory is still available, with the command: `hugeadm --pool-pages-min DEFAULT:num_pages` command
  - For subsequent on-demand allocation of Huge Pages, use: `hugeadm --pool-pages-max DEFAULT:num_pages` or set the value of `/proc/sys/vm/nr_overcommit_hugepages`. These delayed (from boot) actions do not guarantee availability of the requested number of huge pages; however, they are safe as, if a sufficient number of huge pages is not available, Linux simply uses traditional sized pages.
- To use huge pages for shared memory segments for the BG database access mode, both of the following are required:
  - Permit GT.M processes to use huge pages for shared memory segments (where available, FIS recommends 1, below; however not all file systems support extended attributes). Either:
    1. set the `CAP_IPC_LOCK` capability needs for your mumps, mupip and dse processes with a command such as `setcap 'cap_ipc_lock+ep' $gtm_dist/mumps`, or
    2. permit the group used by GT.M processes needs to use huge pages with, as root, `echo gid >/proc/sys/vm/hugetlb_shm_group`.
  - Set the environment variable `HUGETLB_SHM` for each process to yes.
- To use huge pages for process working space and dynamically linked code:
  - Set the environment variable `HUGETLB_MORECORE` for each process to yes.
- Although not required to use huge pages, your application is also likely to benefit from including the path to `libhugetlbf.so` in the `LD_PRELOAD` environment variable.
- If you enable huge pages for all applications (by setting `HUGETLB_MORECORE`, `HUGETLB_SHM`, and `LD_PRELOAD` as discussed above in `/etc/profile` and/or `/etc/csh.login`), you may find it convenient to suppress warning messages from common applications that are not configured to take advantage of huge pages by also setting the environment variable `HUGETLB_VERBOSE` to zero (0).

At this time, huge pages cannot be used for MM databases; the text, data, or bss segments for each process; or for process stack.

Refer to the documentation of your Linux distribution for details. Other sources of information are:

- <http://www.kernel.org/doc/Documentation/vm/hugetlbpage.txt>
- <http://lwn.net/Articles/374424/>
- <http://www.ibm.com/developerworks/wikis/display/LinuxP/libhuge+short+and+simple>
- the HOWTO guide that comes with `libhugetlbf` (<http://sourceforge.net/projects/libhugetlbf/files/>)



### Note

- In order to assure predictable performance, FIS strongly recommends that you invest the effort to appropriately tune your system before using huge pages in production. For example, use of on demand huge pages (over-committing) may result in delays while the kernel performs memory management activities to satisfy requests for memory - but this may or may not be material in your situation.

## Basic Operations

- Since the memory allocated by Linux for shared memory segments mapped with huge pages is rounded up to the next multiple of huge pages, there is potentially unused memory in each such shared memory segment. You can therefore increase any or all of the number of global buffers, journal buffers, and lock space to make use of this otherwise unused space. You can make this determination by looking at the size of shared memory segments using `ipcs`. Contact FIS GT.M support for a sample program to help you automate the estimate.
- Transparent huge pages may further improve virtual memory page table efficiency. Some Supported releases automatically set `transparent_hugepages` to "always"; others may require it to be set at or shortly after boot-up. Consult your Linux distribution's documentation.



## Important

A fatal SIGBUS error occurs when a GT.M process using huge pages tries to spawn another process and Linux does not have sufficient available huge pages to provide the required memory for the spawned process. The following involve spawning additional processes:

- JOB
- OPEN of a PIPE device
- ZSYSTEM
- interprocess signaling that requires the services of `gtmsecshr` when `gtmsecshr` is not already running
- SPAWN commands in DSE, GDE, and LKE
- Argumentless MUPIP RUNDOWN
- Replication-related MUPIP commands that start server processes and/or helper processes

Depending on when huge pages become unavailable, the SIGBUS error may go to the initiating process or may only prevent the new process from starting. The solution is to tune Linux to ensure that the number of huge pages required by your application are always available to your application. Because such tuning may require a reboot, an interim workaround is to unset the environment variable `HUGETLB_MORECORE` for GT.M processes until you are able to reboot or otherwise make available an adequate supply of huge pages.

---

## Chapter 4. Global Directory Editor

Revision History		
Revision V6.1-000	01 August 2014	<ul style="list-style-type: none"><li>• In “Global Directory” (page 32), explained the concepts of mapping different subscripts of the same global to different regions.</li><li>• In “Global Director Editor Commands” (page 42), added example of globals spanning regions, the description of the new - GBLNAME, -COLLATION, MUTEX_LOCK qualifiers.</li></ul>
Revision V6.0-003	27 January 2014	<ul style="list-style-type: none"><li>• In “Region Qualifiers” (page 54), corrected the syntax of [NO]STDNULLCOLL and - COLLATION_DEFAULT .</li><li>• In “Segment Qualifiers” (page 58), added a paragraph about what happens when database file with automatic extension disabled (EXTENSION_COUNT=0) starts to get full.</li></ul>
Revision V6.0-001	27 February 2013	In “Segment Qualifiers” (page 58) and under the -ALLOCATION section, specified that GT.M always reserves 32 global buffers for BG access method out of the requested allocation.
Revision V6.0-000/1	21 November 2012	Updated for V6.0-000.
Revision V6.0-000	19 October 2012	In “Region Qualifiers” (page 54), added the description of - [NO]INST[_FREEZE_ON_ERROR].
Revision V5.5-000/10	28 September 2012	Added the description of -[NO]STDNULLCOLL, corrected the description of - NULL_SUBSCRIPTS, and corrected the maximum value for -LOCK_SPACE.
Revision V5.5-000/8	03 August 2012	Improved the description of the LOCK_SPACE qualifier and GDE Overview.
Revision V5.5-000/4	6 June 2012	Added AUTOSWITCHLIMIT as an option for - JOURNAL region qualifier.
Revision V5.4-002B	24 October 2011	Conversion to documentation revision history reflecting GT.M releases with revision history for each chapter.

---

## Global Directory

A global directory is analogous to a telephone directory. Just as a telephone directory helps you find the phone number (and the address) given a person's name, a global directory helps GT.M processes find the database file of an M global variable node. But because its life is independent of the databases it maps, a global directory has a second purpose in addition to holding key mappings, which is to hold database characteristics for MUPIP CREATE. While changes to the mappings take effect as soon as

A process loads a new global directory, MUPIP CREATE transfers the other characteristics to the database file, but other GT.M processes never use the global directory defined characteristics, they always use those in the database file.

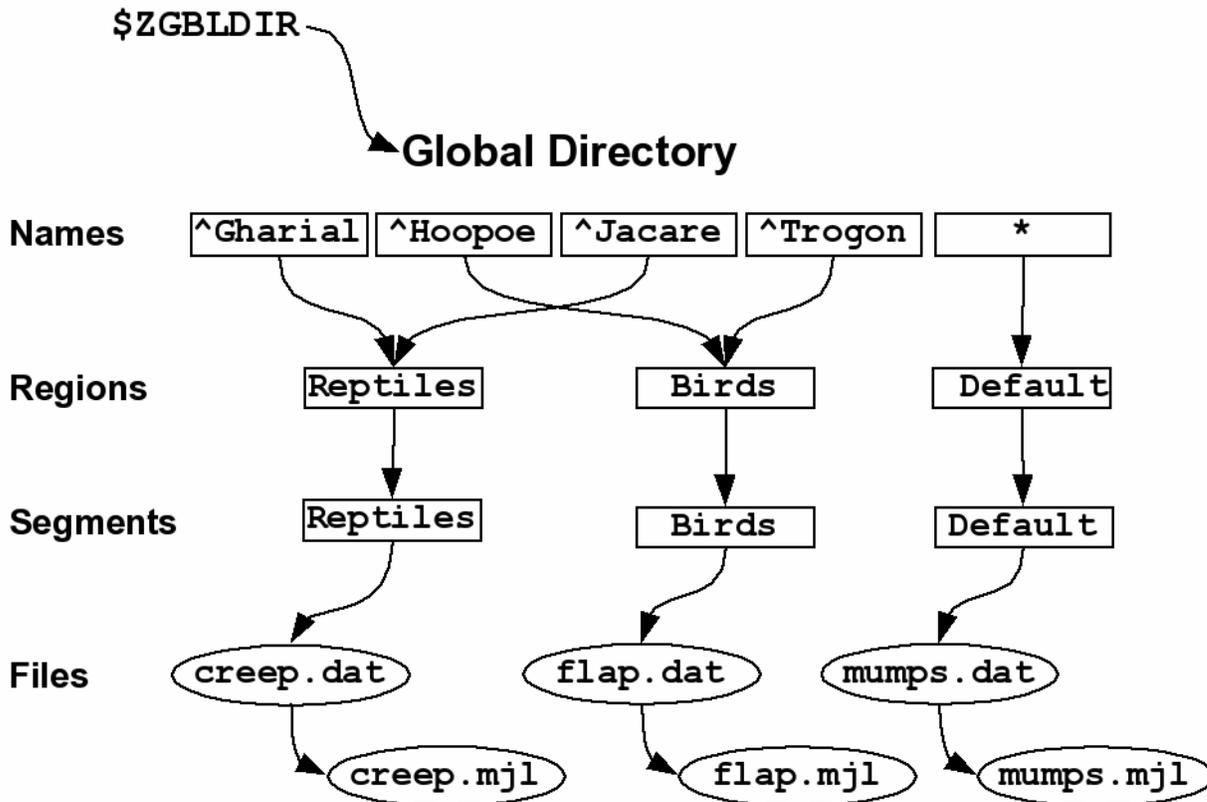
GT.M manages routines in files and libraries separately from globals. For more information on routine management, refer to the Development Cycle chapter in *GT.M Programmer's Guide*.

A set of M global variables (Names or Name spaces) and / or their subscripts map to Regions that define common sets of properties such as the maximum record length and whether null subscripts collate in conformance to the M standard. Each Region maps to a Segment that defines the properties relating to the file system such as the file name, the initial allocation, and number of global buffers. These properties and mapping rules are stored in a binary file called global directory. By default, a global directory file has an extension of **.gld**. You can specify any filename and extension of your choice for a global directory as long as it is valid on your operating system; GT.M documentation always uses the default extension.

The location of the global directory is pointed to by the Intrinsic Special Variable \$ZGBLDIR. GT.M processes initialize \$ZGBLDIR at process startup from the environment variable gtmgldir and can modify it during execution. For example, with a simple **SET \$ZGBLDIR** command, a process can switch back and forth between development and testing databases.

Consider a global variable ^TMP that holds only temporary data that is no longer meaningful when a system is rebooted. A global directory can map ^TMP to region TEMP that maps to a database file called scratch.dat, with all other globals mapped to gtm.dat. A global directory allows the separation of persistent data (gtm.dat) from non-persistent data (scratch.dat), so that each database file may get appropriately configured for operations—for example, the database administrator may choose to exclude scratch.dat from backup/archival procedures or periodically delete and recreate scratch.dat using MUPIP CREATE.

Consider the following illustration:



There are four M global variables--^Gharial, ^Hoopoe, ^Jacare, and ^Trogon. ^Gharial and ^Jacare map to region REPTILES that maps to database file creep.dat and ^Hoopoe and ^Trogon map to region BIRDS that maps to database file flap.dat. The

## Global Directory Editor

default namespace \* maps to a region called DEFAULT that maps to database file gtm.dat. \* denotes all globals other than the explicitly named ^Gharial, ^Hoope, ^Jacare, and ^Trogon. All globals store data in their respective database files. Each database file has a single active journal file. To enforce access restrictions on globals so that herpetologists have access to ^Gharial and ^Jacare and only ornithologists have access to ^Hoopoe and ^Trogon, one just needs to assign appropriate read / write permissions to creep.dat and flap.dat.



### Note

Each database file can have a single active journal file. A journal can be linked to its predecessor journal file to form a chain of journal files.

You can also map different subscripts of the same global to different regions when subscripts have logically separable data. Consider the following global directory example:

Global Directory							
Global	^EURWest	^EURWest	^EURCentral	^US	^US	^Australia	*
Collation	0	0	1	0	0	0	0
Subscript	^EURWest ("UK")	^EURWest ("France")	^EURCentral ("Poland")	^US ("South", "a":"m")	^US (South, "m":{"})	N/A	N/A
Region	UKREG	FRREG	POREG	USSALREG	USSMZREG	AUSREG	DEFAULT
Segment	UKSEG	FRSEG	POSEG	USSALSEG	USSMZSEG	AUSSEG	DEFAULT
File	UK.dat	France.dat	Poland.dat	USSAL.dat	USSMZ.dat	AUS.dat	gtm.dat

Mapping ↓

^US and ^EURWest have logically separable subscripts that map to different regions. ^EURCentral holds data that has a different collation order than others so it maps to a different region. Such mapping improves operational administration and permits a larger total size. It may also improve performance if the access patterns of the distinct parts allow accesses to all or some of them to use optimizations in the GT.M database engine, for example, to optimize serial accesses.

In a nutshell, the database attributes and mapping rules defined in a global directory allow you to:

- **Finer-grained Access Control-** To block access, or updates, to a portion of the data.
- **Improve Operational Administration-** When a global becomes so big that that breaking it up improves operational administration or permit a larger total size.
- **Compliment Application Design-** To separate global and / or their subscripts in a way that achieves a design goal without writing addition code. For example, mapping globals to regions that are not replicated.
- **Manage Volatility-** some data is static, or relatively so, and you wish to leverage that to tailor your backup and integrity verification patterns, or to use MM access.
- **Improve Manageability and Performance-** When a global variable is overloaded with logically separate data, distributing the logically separate components each to its own database region improves manageability and performance when access patterns use optimization in the GT.M database engine.

## GDE Overview

The GT.M Global Directory Editor (GDE) is a utility for creating, examining, and modifying a global directory. GDE is a program written in M and you can invoke it from the shell with `$gtm_dist/mumps -run ^GDE`.

Because GDE is an M program, you can also invoke GDE from a GT.M process with **DO ^GDE**. If you invoke GDE with a DO and modify the map of global directly currently opened by that process, you must **HALT** and restart the process for the process to pick up the revised mapping. FIS expects users normally run GDE from the shell **--\$gtm\_dist/mumps -run GDE**.

The input to GDE can be a command file. In a production environment, FIS recommends using command files to define database configurations and putting them under version control.



### Caution

A global directory stores database attributes and mapping rules. Processes use mapping rules to determine which database file contains a global variable node. MUPIP CREATE uses database attributes to create new database file(s). Once MUPIP CREATE applies the database attributes to create a database file, GT.M does not use the attributes until the next MUPIP CREATE. If you use MUPIP SET (or DSE) to change the attributes of a database file, always perform an equivalent change to any global directory used for a subsequent MUPIP CREATE. Conversely, if you change attributes with GDE, existing database files must be explicitly changed with MUPIP SET or DSE.

## Identifying the Current Global Directory

At process startup, the environment variable `gtmgbldir` identifies the global directory to the process. M application code can access and change the global directory through the `$ZGBLDIR` intrinsic special variable, which is initialized from `$gtmgbldir` at process startup. M application code can also use extended global references with the `||` or `{}`  syntax.

Note that `$gtmgbldir` / `$ZGBLDIR` are pathnames. If they do not start with a `"/`, then the pathname is relative and GT.M searches for the global directory starting in the current working directory.

To change the Global Directory used by processes, specify a new value for `gtmgbldir`.

Example:

```
$ export gtmgbldir=/home/jdoe/node1/prod.gld
```

When you invoke GDE and no Global Directory exists for `gtmgbldir`, GDE creates a minimal default Global Directory that is a starting point or template for building global directories for your specific needs.

To retain the default Global Directory, exit GDE without making any changes.

Example:

```
$ export gtmgbldir=/home/jdoe/node1/prod.gld
```

## Creating a Default Global Directory

When you invoke GDE and no Global Directory exists for `gtmgbldir`, GDE produces a default Global Directory that contains a minimal set of required components and values for database characteristics. It can be used for purposes such as development and testing work. A default Global Directory also serves as a starting point or template for building custom global directories.

To retain the default Global Directory, quit GDE without making any changes.

Example:

```
$ gtmgbldir=/usr/accntg/jones/mumps.gld
```

```
$ export gtmgblmdir
$ $gtm_dist/mumps -dir
GTM>do ^GDE
%GDE-I-GDUSEDEFS, Using defaults for Global Directory
/usr/accntg/jones/mumps.gld
GDE> EXIT
%GDE-I-VERIFY, Verification OK
%GDE-I-GDCREATE, Creating Global Directory file
/usr/accntg/jones/mumps.gld
```

## Mapping Global Variables in a Global Directory

Mapping is the process of connecting a global variable name or a subtree or a subscript range to a database file.

A complete mapping has the following four components:

- NAME
- REGION
- SEGMENT
- FILE

These components may be defined in any order, but the final result must be a complete logical path from name to file:

```
NAME(s) ---> REGION ---> SEGMENT ---> FILE
```

The default Global Directory contains one complete mapping that comprises these entries for name, region, segment, and file.

```
* ---> DEFAULT ---> DEFAULT ---> mumps.dat
(NAME) (REGION) (SEGMENT) (FILE)
```

The \* wildcard identifies all possible global names. Subsequent edits create entries for individual global names or name prefixes.

Regions and segments store information used to control the creation of the file. The characteristics stored with the region and segment are passed to MUPIP only when creating the database file using the CREATE command, so subsequent changes to these characteristics in the Global Directory have no effect on an existing database.

On EXIT, GDE validates the global directory to ensure that every legal global variable node maps to exactly one region; that every region has at least one global variable node mapping to it and that it maps to exactly one segment; that every segment has exactly one region mapping to it; and that the attributes for each region and segment are internally consistent. GDE will not create a structurally unsound global directory, and will not exit until it validates the global directory. Informational messages advise you of structural inconsistencies.

## Examining the Default Global Directory

A Global Directory looks like this:

*** TEMPLATES ***								
Region	Def Coll	Rec Size	Key Size	Null Subs	Std Null Coll	Inst Freeze on Error	Qdb Rndwn	
<default>	0	4080	255	NEVER	Y Y	DISABLED	DISABLED	

## Global Directory Editor

```

-----
      Jnl File (def ext: .mjl)  Before  Buff      Alloc      Exten      AutoSwitch
-----
<default>      <based on DB file-spec>  Y      2308      2048      2048      8386560

Segment        Active          Acc Typ Block      Alloc Exten Options
-----
<default>      *              BG DYN 4096      5000 10000 GLOB =1000
                                LOCK = 40
                                RES = 0
                                ENCR = OFF
                                MSLT =1024
<default>      MM DYN 4096      5000 10000 DEFER
                                LOCK = 40
                                MSLT =1024

      *** NAMES ***
Global          Region
-----
*              DEFAULT

      *** REGIONS ***
Region          Dynamic          Def      Rec      Key Null      Std      Inst
Segment        Segment          Coll     Size   Size Subs     Null     Freeze  Qdb
-----
DEFAULT        DEFAULT          0        4080   255 NEVER     Y        Y      DISABLED DISABLED

      *** JOURNALING INFORMATION ***
Region          Jnl File (def ext: .mjl)  Before Buff      Alloc      Exten      AutoSwitch
-----
DEFAULT        $gtmdir/$gtmver/g/gtm.mjl
                                Y      2308      2048      2048      8386560

      *** SEGMENTS ***
Segment        File (def ext: .dat)Acc Typ Block      Alloc Exten Options
-----
DEFAULT        $gtmdir/$gtmver/g/gtm.dat
                                BG DYN 4096      5000 10000 GLOB=1000
                                LOCK= 40
                                RES = 0
                                ENCR=OFF
                                MSLT=1024

      *** MAP ***
- - - - - Names - - - - -
From          Up to          Region / Segment / File(def ext: .dat)
-----
%              ...              REG = DEFAULT
              SEG = DEFAULT
              FILE = $gtmdir/$gtmver/g/gtm.dat
LOCAL LOCKS   REG = DEFAULT
              SEG = DEFAULT
              FILE = $gtmdir/$gtmver/g/gtm.dat

```

There are five primary sections in a Global Directory:

- TEMPLATES
- NAMES
- REGIONS
- SEGMENTS
- MAP

## Global Directory Editor

The function of each section in the Global Directory is described as follows:

### *TEMPLATES*

This section of the Global Directory provides a default value for every database or file parameter passed to GT.M as part of a region or segment definition. GDE uses templates to complete a region or segment definition where one of these necessary values is not explicitly defined.

GDE provides initial default values when creating a new Global Directory. You can then change any of the values using the appropriate `-REGION` or `-SEGMENT` qualifiers with the `TEMPLATE` command.

### *NAMES*

An M program sees a monolithic global variable namespace. The `NAMES` section of the Global Directory partitions the namespace so that a global name or a global name with a subscript range reside in different database files. An M global can reside in one more database file, each database file can store many M globals.

### *REGIONS*

The `REGIONS` section lists all of the regions in the Global Directory. Each region defines common properties for a set of M global variables or nodes; therefore, multiple sets of names from the `NAMES` section map onto a single region.

You assign these values by specifying the appropriate qualifier when you create or modify individual regions. If you do not specify a value for a particular parameter, GDE assigns the default value from the `TEMPLATES` section.

### *SEGMENTS*

This section of the Global Directory lists currently defined segments. While regions specify properties of global variables, segments specify the properties of files. There is a one-to-one mapping between regions and segments. You assign these values by specifying the appropriate qualifier when you create or modify individual segments. If you do not specify a value for a particular parameter, GDE assigns the default value from the `TEMPLATES` section.

### *MAP*

This section of the Global Directory lists the current mapping of names to region to segment to file. In the default Global Directory, there are two lines in this section: one specifies the destination for all globals, the other one is for M LOCK resources with local variable names. If you add any new mapping component definitions (that is, any new names, regions, or segments), this section displays the current status of that mapping. Any components of the mapping not currently defined display "NONE". Because GDE requires all elements of a mapping to be defined, you will not be able to EXIT (and save) your Global Directory until you complete all mappings.

## Global Directory Abbreviations

GDE uses the following abbreviations to display the output of a global directory. The following list show global directory abbreviations with the associated qualifiers. For a description of the function of individual qualifiers, see "GDE Command Summary".

Abbreviation	Full Form
Acc	ACCESS_METHOD
Alloc	ALLOCATION
AutoSwitch	AUTOSWITCHLIMIT
Block	BLOCK_SIZE
Buff	BUFFER_SIZE
Def Coll	COLLATION_DEFAULT

Exten	EXTENSION_COUNT
File	FILE_NAME
GLOB	GLOBAL_BUFFER_COUNT
Inst Freeze On Error	INST_FREEZE_ON_ERROR
JNL	JOURNAL
KeySize	KEY_SIZE
LOCK	LOCK_SPACE
MSLT	MUTEX_SLOTS
Null Subs	NULL_SUBSCRIPTS
Qdb Rndwn	QDBRUNDOWN
Std Null Coll	STDNULLCOLL
Rec Size	RECORD_SIZE
RES	RESERVED_BYTES
Region	REGION
Typ	DYNAMIC_SEGMENT

## Customizing a Global Directory

Once you have installed GT.M and verified its operation, create Global Directories based on your needs. To create customized Global Directories, use the appropriate GDE commands and qualifiers to build each desired Global Directory. The GDE commands are described later in this chapter.

You can also create a text file of GDE commands with a standard text editor and process this file with GDE. In a production environment, this gives better configuration management than interactive usage with GDE.

## Adding a Journaling Information Section

If you select the -JOURNAL option when you ADD or CHANGE a region in a Global Directory, the following section is added to your Global Directory and displays when you invoke SHOW. The columns provided display the values you selected with the journal options, or defaults provided by FIS for any options not explicitly defined.

*** JOURNALING INFORMATION ***						
Region	Jnl File (def ext: .mjl)	Before	Buff	Alloc	Exten	AutoSwitch
-----						
DEFAULT	\$gtmdir/\$gtmver/g/gtm.mjl	Y	2308	2048	2048	8386560

For more information about journaling, see the section on the JOURNAL qualifier in this chapter and Chapter 6: “GT.M Journaling” (page 130).

## Using GDE

The default installation procedure places the GDE utility into a directory assigned to the environment variable gtm\_dist.

To invoke GDE:

from within GTM, use the command:

```
GTM>do ^GDE
```

from the shell, enter:

```
$ mumps -r GDE
```

## Global Directory Editor

GDE displays informational messages like the following, and then the GDE> prompt:

```
%GDE-I-LOADGD, loading Global Directory file /prod/mumps.gld
%GDE-I-VERIFY, Verification OK
GDE>
```

If this does not work, contact your system manager to investigate setup and file access issues.

To leave GDE:

1. Use the GDE EXIT command to save all changes and return to the caller.

```
GDE> EXIT
```

2. Use the GDE QUIT command to discard all changes and return to the caller. This will not save any changes.

```
GDE> QUIT
```

## Guidelines for Mapping

This section lists the parameters that apply to defining each component of a mapping.

### NAMES

The NAMES section contains mappings of M global name spaces. More than one name space can map to a single region but a single name space can only map to one region.

A name space:

- Is case sensitive.
- Must begin with an alphabetic character or a percent sign (%).
- Can be a discrete "global" name, for example, aaa corresponds to the global variable ^aaa.
- Can be a global name ending with a wild card ("\*"), for example, abc\* represents the set of global nodes which have abc as the starting prefix.
- Can be a subtree of a global name, for example, abc(1) represents a subtree of the global ^abc.
- Can be a subscript range, for example, abc(1:10) represents all nodes starting from ^abc(1) up to (but not including) to ^abc(10).
- A global name can be one to 31 alphanumeric characters. However, the combined length of a global and its subscripts is limited to 1,019 bytes (the maximum key size supported by GT.M). Note that the byte length of the subscripted global specification can exceed the maximum KeySize specified for its region.
- Maps to only one region in the Global Directory.

### REGIONS

The REGIONS section contain mappings of database region. A region is a logical structure that holds information about a portion of a database, such as key-size and record-size. A key is the internal representation of a global variable name. In this chapter the terms global variable name and key are used interchangeably. A record refers to a key and its data.

A Global Directory must have at least one region. A region only maps to a single segment. More than one name may map to a region.

A region name:

- Can include alphanumeric, dollar signs (\$), and underscores ( \_ ).
- Can have from 1 to 31 characters.

GDE automatically converts region names to uppercase, and uses DEFAULT for the default region name.

### SEGMENTS

The SEGMENTS section contains mappings for segments. A segment defines file-related database storage characteristics. A segment must map to a single file. A segment can be mapped by only one region.

GT.M uses a segment to define a physical file and access method for the database stored in that file.

A segment-name:

- Can include alphanumeric, dollar signs (\$), and underscores ( \_ )
- Can have from one to 31 characters

GDE automatically converts segment names to uppercase. GDE uses DEFAULT for the default segment name.

### FILE

Files are the structures provided by UNIX for the storage and retrieval of information. Files used by GT.M must be random-access files resident on disk.

By default, GDE uses the file-name mumps.dat for the DEFAULT segment. GDE adds the .dat to the file name when you do not specify an extension. Avoid non-graphic and punctuation characters with potential semantic significance to the file system in file names as they will produce operational difficulties.

## Example of a Basic Mapping

To complete this procedure, you must have already opened a Global Directory.

- ADD a new global variable name.

```
GDE> add -name cus -region=cusreg
```

This maps the global name cus to the region cusreg.

- ADD region cusreg, if it does not exist.

```
GDE> add -region cusreg -dynamic=cusseg
```

This creates the region cusreg and connects it to the segment cusseg. -d[ynamic] is a required qualifier that takes the associated segment-name as a value.

- ADD segment cusreg, if it does not exist, and link it to a file.

```
GDE> add -segment cusseg -file=cus.dat
```

This creates the segment cusseg and connects it to the file cus.dat.

## Global Directory Editor

To review the information you have added to the Global Directory, use the SHOW command.

To perform a consistency check of the configuration, use the VERIFY command.

To exit the Global Directory and save your changes, use the EXIT command. GDE performs an automatic verification. If successful, the mappings and database specifications become part of the Global Directory, available for access by processes, utilities, and the run-time system.

Only MUIP CREATE uses the database specifications; run-time processes and other utility functions use only the map.

---

## Global Director Editor Commands

This section describes GDE commands. GDE allows abbreviations of commands. The section describing each command provides the minimum abbreviation for that command and a description of any qualifiers that are not object-related. The section discussing the object-type describes all the associated object-related qualifiers.

Command Syntax:

The general format of GDE commands is:

```
command [-object-type] [object-name] [-qualifier]
```

where:

- object-type** Indicates whether the command operates on a -N[AME] space, -R[EGION], or -S[EGMENT].
- object-name** Specifies the name of the N[AME] space, R[EGION], or S[EGMENT]. Objects of different types may have the same name. Name spaces may include the wildcard operator (\*) as a suffix.
- qualifier** Indicates an object qualifier.

The format description for each individual command specifies required qualifiers for that command.

The @, EXIT, HELP, LOG, QUIT, SETGD, and SPAWN commands do not use this general format. For the applicable format, refer to the section explaining each of these commands.

Comments on command lines start with an exclamation mark (!) and run to the end of line.



### Caution

An exclamation mark not enclosed in quotation marks (") (for example in a subscript) causes GDE to ignore the rest of that input line.

## Specifying File Names in Command Lines

file-names must either appear as the last item on the command line or be surrounded by quotation marks. Because UNIX file naming conventions permit the use of virtually any character in a file-name, once a qualifier such as -FILE\_NAME or -LOG introduces a file-name and the first character after the equal sign is not a quotation mark, GT.M treats the entire remainder of the line as the file-name. When using quotation marks around file-names, GDE interprets a pair of embedded quotation marks as a single quotation mark within the file-name. Note that the use of Ctrl or punctuation characters such as exclamation mark (!), asterisk (\*), or comma (,) in a file-name is likely to create significant operational file management challenges. FIS strongly recommends against such practices.

## Font/Capitalization Conventions Used in this Chapter

All GT.M and GDE commands and qualifiers may be entered in either upper or lower case at the command prompt. However, when you SHOW your current Global Directory, GDE uses the following case conventions:

- Region and segment names always display in uppercase
- Name space object names always appear in case in which they are entered.
- File-names always appear in case in which they are entered.



### Note

The .dat extension is appended to the file-name when the database file is created, but does not appear in the Global Directory listing, unless you enter it that way.

The descriptions of these commands and qualifiers appear in various cases and fonts throughout this documentation. This section describes the conventions used in describing these commands and qualifiers.

- In text: all GT.M commands and qualifiers appear in uppercase.
- In examples: the entire command line is shown in lower case, and appears in bold typewriter font.

## @

The @ command executes a GDE command file. Use the @ command to execute GDE commands stored in a text file.

The format of the @ command is:

```
@file-name
```

The file-name specifies the command file to execute. Use the file-name alone for a file in the current working directory or specify the relative path or the full path.

GDE executes each line of the command file as if it were entered at the terminal.

Example:

```
GDE> @standard
```

This command executes the GDE commands in the file to standard in the current working directory. standard should contain GDE commands; comments should start with an exclamation mark (!).

## Add

The ADD command inserts a new name, region, or segment into the Global Directory.

The format of the ADD command is one of the following:

```
A[DD] -N[AME] namespace -R[EGION]=region-name
A[DD] -R[EGION] region-name -D[YNAMIC]=segment-name [-REGION-qualifier...]
A[DD] -S[EGMENT] segment-name [-SEGMENT-qualifier...] -F[ILE_NAME]=file-name
A[DD] -G[BLNAME] global-name [-GBLNAME-qualifier ...]
```

The ADD command requires specification of an object-type and object-name. GDE supplies default values from the templates for qualifiers not explicitly supplied in the command.

**namespace** specifies a global name or a global name with subscript(s) or a global name with a subscript range in the form of **global[[*\**]][(*from-subscript*:*to-subscript*)]**.

Name spaces and file-names are case-sensitive; other objects are not case-sensitive.

## -Name

Maps a namespace to a region in the global directory. The format of the ADD -NAME command is:

```
A[DD]-N[AME] namespace -R[EGION]=region-name
```

- You can map a global and its subtrees to different regions.
- You can also use colon (:) to map ranges of subscripted names and their subtrees to a region. Ranges are closed on the left and open on the right side of the colon. For example, **add -name PRODAGE(0:10) -region DECADE0** maps **^PRODAGE(0)** to **^PRODAGE(9)**, **assuming the application always uses integer subscripts**, to region **DECADE0**.
- You can also use \$CHAR() and \$ZCHAR() to specify unprintable characters as subscripts. "" (an empty string) or no value (e.g. 20: or :20 or :) specify open-ended ranges, which span, on the left, from the first subscript ("" to, on the right, the last possible string.
- Regions that contain global variables sharing the same unsubscripted name that span regions must use standard null collation; attempting to use the deprecated original null collation produces an error.

Example:

```
GDE> add -name IMPL -region=OTHERMUMPS ! Map MUMPS implementations to OTHERMUMPS
GDE> add -name IMPL("GT.M") -region=MYMUMPS ! While mapping GT.M to
> MYMUMPS
```



These examples map an entire subtree of a global to a region.

Example:

```
GDE> add -name PRODAGE(0:10) -region=DECADE0 ! Ranges are closed on the left and open
> on the right
GDE> add -name PRODAGE(10:20) -region=DECADE1 ! PRODAGE(10) maps to DECADE1
GDE> add -name PRODAGE(20:30) -region=DECADE2
```



This example uses a colon (:) to map ranges of subscripted names and their subtrees to a region. Note that ranges are specific numbers or strings - GDE does not support wildcards (using **"\*\*"**) in ranges.

Example:

```
GDE> add -name=PRODAGE(:10) -region=DECADE0 ! This line and
> the next are equivalent
GDE> add -name PRODAGE("":10) -region=DECADE0 ! numbers up to, but not including, 10
GDE> add -name PRODAGE(20:) -region=DECADE2 ! 20 thru all numbers (> 20) + strings
GDE> add -name PRODAGE(20:"") -region=DECADE2 ! same as the add just above
```



## Global Directory Editor

These examples demonstrate the use of \$CHAR() and \$ZCHAR() to specify unprintable characters; Notice that the arguments are positive integers (exponential - E syntax not allowed), and valid code points for \$CHAR() or in range for \$ZCHAR(), both with respect to the current \$ZCHSET. Also, "" (an empty string) or no value (e.g. 20: or :20 or :) specify open-ended ranges, which span, on the left, from the first subscript ("") to, on the right, the last possible string.

Example:

```
GDE> add -name MODELNUM -region=NUMERIC
GDE> add -name MODELNUM($char(0):) -region=STRING
```

This example map numeric subscripts and strings to separate regions.

Example:

```
GDE> add -name DIVISION("Europe","a":"m") -region EUROPEAL
GDE> add -name DIVISION("Europe","m":"z") -region EUROPEM
GDE> add -name DIVISION("Australia") -region AUSTRALIA
GDE> add -name DIVISION("USA","South","a":"m") -region USSAL
GDE> add -name DIVISION("USA","South","m":"{") -region USSMZ
GDE> add -name DIVISION("USA","WestCoast") -region USWC
```

This example maps global variables with the same unsubscripted name at multiple subscript levels.

Example:

```
GDE> add -name x -region=REG1
GDE> add -name x(5) -region=REG1
GDE> add -name x(5,10:) -region=REG2
GDE> add -name x(5:20) -region=REG2
GDE> add -name x(20) -region=REG2
GDE> add -name x(20,40) -region=REG2
GDE> add -name x(20,40,50:) -region=REG3
GDE> add -name x(20,40:) -region=REG3
GDE> add -name x(20:) -region=REG3
```

This example performs the following mapping:

- from ^x, upto but not including ^x(5,10), maps to REG1
- from ^x(5,10), upto but not including ^x(20,40,50), maps to to REG2
- from ^x(20,40,50) through the last subscript in ^x maps to REG 3

## -Segment

Maps a segment to a database file. The syntax of the ADD -SEGMENT command is:

```
A[DD]-S[EGMENT] segment-name [-SEGMENT-qualifier...]
► -F[ILE_NAME]=file-name
```



Example:

```
GDE> add -segment temp -file_name=scratch
```

## Global Directory Editor

This command creates a segment-name TEMP and maps it to the file scratch.dat in the current working directory. However, if you were to specify scratch as the file-name, in other words, an environment variable, each process uses the file using the translation of that environment variable at run-time.

### -Region

Maps a region to a segment. The syntax of the ADD -REGION command is:

```
A[DD]-R[EGION] region-name -D[YNAMIC]=segment-name [-REGION-qualifier...]
```

### -Gblname

Provides a mechanism to specify the collation for global variables sharing the same unsubscripted name. Specifying a collation is necessary for globals that span multiple regions and use an alternate collation. Because the global name EURCentral (described in the Introduction section) uses an alternate collation, it requires an entry in the GBLNAME section. The format of the ADD -GBLNAME command is:

```
A[DD] -G[BLNAME] -C[OLLATION]=collation_number
```

- Because string subscripts are subject to collation (the unsubscripted portion of a global variable name and numeric subscripts are not), GDE needs to know the collation sequence number associated with each unsubscripted global variable name. Most standard collation (the default) has a collation number of zero (0). As a consequence, when you use alternative collation(s) (other than 0), the collation transforms must be available to GDE in the same way as they are to other GT.M components. All of a global (all nodes sharing the same unsubscripted global name) must have a single collation, which is implicitly the case for globals that do not span multiple regions.
- Globals that do not span multiple regions and do not have any collation characteristics defined in the GBLNAME section of the global directory take on the default collation characteristics defined in the database region to which they map. On the other hand, globals that span multiple regions have their collation implicitly (collation 0), or explicitly, established by the GBLNAME section of the global directory and cannot adopt a differing collation based on the region collation characteristic. Because GT.M determines collation for globals spanning multiple regions by the GBLNAME characteristic, which cannot change once the database files are created, GDE reports collation on many error messages.

Example:

```
GDE> add -gblname EURCentral -collation=1
GDE> show -gblname
```

```
*** GBLNAMES ***
Global                               Coll Ver
-----
EURCentral                           1    0
```

## Change

The CHANGE command alters the name-to-region or region-to-segment mapping and /or the environment for a region or segment.

The format of the CHANGE command is:

```
C[HANGE]-N[AME] namespace -R[EGION]=new-region
C[HANGE]-R[EGION] region-name [-REGION-qualifier...]
C[HANGE]-S[EGMENT] segment-name [-SEGMENT-qualifier...]
```

## Global Directory Editor

```
C[HANGE] -G[BLNAME] -C[OLLATION]=collation_number
```

The CHANGE command requires specification of an object-type and object-name.

Once you exit GDE, mapping changes take effect for any subsequent image activation (for example, the next RUN or the mumps -direct command). Changes to database parameters only take effect for new database files created with subsequent MUPIP CREATE commands that use the modified Global Directory. Use the MUPIP SET command (or in some cases DSE) to change characteristics of existing database files.

Example:

```
GDE> change -region master -dynamic=temp -key=100
```

This command changes the region master to use the segment temp and establishes a maximum KEY\_SIZE of 100 characters for the next creation of a file for this region. The segment change takes effect the first time the system uses the Global Directory after the GDE session EXITS, while the KEY\_SIZE change takes effect after the next MUPIP CREATE that creates a new database file for segment temp.

## Delete

The DELETE command removes a name, region, or segment from the Global Directory. The DELETE command does not delete any actual data. However, GT.M does not access database files that do not have mapped global variables except through extended references using an alternative global directory that does not map to them. Note that GT.M replication does not support global updates made with extended references, unless they actually map to a database file that is a part of the replicated instance.

The format of the DELETE command is:

```
D[ELETE] -N[AME] namespace  
D[ELETE] -R[EGION] region-name  
D[ELETE] -S[EGMENT] segment-name  
D[ELETE] -G[BLNAME] global-name
```

The DELETE command requires specification of an object-type and object-name.

Deleting a name removes the namespace-to-region mapping. Deleting a region unmaps all names mapped to the region. Deleting a segment unmaps the region mapped to the segment.

You may map the deleted names to another region or the deleted region to another segment using the CHANGE command.

The default namespace (\*) cannot be deleted.

Example:

```
GDE> del -name T*
```

This command deletes the explicit mapping of all global names starting with the letter "T." This command does not delete any global variables. However, it may make preexisting globals starting with the letter "T" invisible, at least while using this global directory, because the T\* global names map to the default namespace going forward.

## Exit

The EXIT command writes all changes made in the current GDE editing session to the Global Directory and terminates the current editing session.

The format of the EXIT command is:

```
E[EXIT]
```

GDE performs a full verification test (VERIFY) on the data. If the verification succeeds, GDE writes the new Global Directory to file system and issues a verification message.

If the verification fails, GDE displays a listing of all unverifiable mappings and waits for corrections. Make appropriate corrections, or leave the Global Directory in its original, unedited state by using the QUIT command.

If you have not made any changes to the Global Directory, GDE does not save a new Global Directory unless the original global directory had an older format which GDE has automatically upgraded. Note that while GDE upgrades older global directories to the current version, there is no facility to downgrade global directories to prior versions, so you should always save copies of any global directories that might be needed to retrieve archival data.

## Help

The HELP command displays online information about GDE commands and qualifiers.

The format of the HELP command is:

```
H[ELP] [topic...]
```

where **topic** specifies the GDE command for which you want information. If you omit the topic, GDE prompts you for it.

## LOCKS

The LOCKS command specifies the region into which GT.M maps "local" locks(those with resource names not starting with a caret symbol ^). GDE maps locks on resource names, starting with a caret symbol, to the database region mapped for the global variable name matching the resource name.

The format of the LOCKS command is:

```
LOC[KS] -R[EGION]=region-name
```

The LOCKS -REGION= qualifier allows specification of a region for local locks. By default, GDE maps local locks to the DEFAULT region .

Example:

```
GDE> lock -region=main
```

This command maps all locks on resource names that don't start with the caret symbol, "^" to the region main.



### Caution

GT.M associates LOCKs for global names with the database region holding the corresponding unsubscripted global name. Suppose a global called ^EURWest spans multiple regions in multiple global directories, a command like LOCK ^EURWest may not work in the same way as it would do if ^EURWest did not span multiple regions. Before using a command like LOCK ^EURWest where ^EURWest spans multiple regions in multiple directories, ensure that the corresponding unsubscripted ^EURWest map to the same region in all the global directories. Alternatively, you can use LOCK globalname (with no leading up-arrow) and control LOCK interactions with the LOCKS global directory characteristic or use transaction processing to eliminate the use of LOCKs to protect global access.

## LOG

The LOG command creates a log file of all GDE commands and displays for the current editing session. Because the system places an exclamation point (!) (i.e., the comment symbol) before all display lines that are not entered by the user. In the log, the log can be used with the @ symbol as a command procedure.

The format of the LOG command is:

```
LOG
LOG -ON[=file-name]
LOG -OF[F]
```

The LOG command, without a qualifier, reports the current status of GDE logging. The LOG command displays a message showing whether logging is in effect and the specification of the current log file for the GDE session.

The log facility can be turned on and off using the -ON or -OFF qualifiers any time during a GDE session. However, GDE closes the log files only when the GDE session ends.

The -ON qualifier has an optional argument of a file, which must identify a legal UNIX file. If LOG -ON has no file-argument, GDE uses the previous log file for the editing session. If no log file has previously been specified during this editing session, GDE uses the default log file GDELOG.LOG.

Example:

```
GDE> log -on="standard.log"
```

This command turns on logging of the session and directs the output to standard.log.

## Quit

The QUIT command ends the current editing session without saving any changes to the Global Directory. GDE does not update the Global Directory file.

The format of the QUIT command is:

```
Q[UIT]
```

If the session made changes to the Global Directory, GDE issues a message warning that the Global Directory has not been updated.

## Rename

The RENAME command allows you to change a namespace, the name of a region, or the name of a segment.

The format of the RENAME command is:

```
R[ENAME] -N[AME] old-name new-name
R[ENAME] -R[EGION] old-region-name new-region-name
R[ENAME] -S[EGMENT] old-segment-name new-segment-name
R[ENAME] -G[BLNAME] old-global-name new-global-name
```

The RENAME command requires specification of an object-type and two object-names.

## Global Directory Editor

When renaming a region, GDE transfers all name mappings to the new region. When renaming a segment, GDE transfers the region mapping to the new segment.

Example:

```
GDE> rename -segment stable table
```

This command renames segment stable to table and shifts any region mapped to stable so it is mapped to table.

## SEtgd

The SETGD command closes out edits on one Global Directory and opens edits on another.

The format of the SETGD command is:

```
SE[TDG] -F[ILE]=file-name [-Q[UIT]]
```

The -FILE=file-name specifies a different Global Directory file. When you provide a file-name without a full or relative pathname GDE uses the current working directory; if the file is missing an extension, then GDE defaults the type to .gld.

The -QUIT qualifier specifies that any changes made to the current Global Directory are not written and are lost when you change Global Directories.

SETGD changes the Global Directory that GDE is editing. If the current Global Directory has not been modified, or the -QUIT qualifier appears in the command, the change simply occurs. However, if the current Global Directory has been modified, GDE verifies the Global Directory, and if the verification is successful, writes that Global Directory. If the verification is not successful, the SETGD fails.

Example:

```
GDE> SETGD -f="temp"
```

This changes the Global Directory being edited to temp. The quotation marks around the file name identifies the name of the file unequivocally to UNIX. If the -f is the final qualifier on the line, then the quotation marks are unnecessary.

## SHow

The SHOW command displays information contained in the Global Directory about names, regions, and segments.

The format of the SHOW command is:

```
SH[OW] -C[OMMAND] -F[ILE]=[gde-command-file]
SH[OW] -N[AME] [namespace]
SH[OW] -R[EGION] [region-name]
SH[OW] -S[EGMENT] [segment-name]
SH[OW] -M[AP] [-R[EGION]=region-name]
SH[OW] -T[EMPLATE]
SH[OW] -G[BLNAME]
SH[OW] -A[LL]
```

-COMMAND: Displays GDE commands that recreate the current Global Directory state.

-F[ILE]=gde-command-file: Optionally specifies a file to hold the GDE commands produced by -COMMAND. -FILE must always appear after -COMMAND.

## Global Directory Editor

Please consider using command files produced with the SHOW -COMMAND -FILE for creating new regions and segments in a global directory as the defaults come from the templates. If you inadvertently upgrade a global directory, you can use SHOW -COMMAND to create a file of commands that you can input to GDE with the prior GT.M release to recreate the prior global directory file.



### Note

When GDE encounters an error while executing the **@command-file** command, it stops processing the command file and returns to the operator prompt, which gives the operator the option of compensating for the error. If you subsequently issue **@command-file** command again in the same session for the same command-file, GDE resumes processing it at the line after the last error.

-NAME, -REGION, -SEGMENT, -GBLNAME, -MAP, -TEMPLATE, and -ALL are qualifiers that cause GDE to display selected portions of the Global Directory as follows:

-MAP: Displays the current mapping of all names, regions, segments, and files. This qualifier corresponds to the section of the SHOW report titled **\*\*\*MAP\*\*\***. The output of a SHOW -MAP may be restricted to a particular region by specifying a -REGION qualifier with a region name argument.

-TEMPLATE: Displays the current region and segment templates. This qualifier corresponds to the section of the SHOW report titled:

```
***TEMPLATES***
```

-ALL: Displays the entire Global Directory. This qualifier corresponds to displaying "all" sections of the SHOW report:

```
***TEMPLATES***, ***NAMES***, ***REGIONS***, ***SEGMENTS***, ***MAP***.
```

By default, SHOW displays -ALL.

If you want to print the Global Directory, create a log file by executing LOG -ON= before executing the SHOW command. The LOG command captures all the commands entered and output. You can print the log file if you want a hard copy record.

If you want to export the current Global Directory state, create a GDE command file with the SHOW -COMMAND -FILE=gde-command-file and run it in the target environment.

### Example:

```
GDE>show -template
```

```
*** TEMPLATES ***
```

Region	Def Coll	Rec Size	Key Size	Null Subs	Std Null Coll	Inst Jnl	Freeze on Error	Qdb Rndwn	
<default>		0	4080	255	NEVER	Y	Y	DISABLED	DISABLED
	Jnl File (def ext: .mjl)	Before	Buff	Alloc	Exten	AutoSwitch			
<default>	<based on DB file-spec>	Y	2308	2048	2048	8386560			

Segment	Active	Acc Typ	Block	Alloc	Exten	Options
<default>	*	BG DYN	4096	5000	10000	GLOB =1000 LOCK = 40 RES = 0 ENCR = OFF MSLT =1024
<default>		MM DYN	4096	5000	10000	DEFER LOCK = 40

This displays only the TEMPLATES section of the Global Directory.

```
GDE>SHOW -command
TEMPLATE -REGION -COLLATION_DEFAULT=0
TEMPLATE -REGION -NOINST_FREEZE_ON_ERROR
TEMPLATE -REGION -JOURNAL=(ALLOCATION=2048,AUTOSWITCHLIMIT=8386560,BEFORE_IMAGE,BUFFER_SIZE=2308,EXTENSION=2048)
TEMPLATE -REGION -KEY_SIZE=64
TEMPLATE -REGION -NULL_SUBSCRIPTS=NEVER
TEMPLATE -REGION -NOQDBRUNDOWN
TEMPLATE -REGION -RECORD_SIZE=256
TEMPLATE -REGION -STDNULLCOLL
!
TEMPLATE -REGION -NOJOURNAL
!
TEMPLATE -SEGMENT -ACCESS_METHOD=BG
TEMPLATE -SEGMENT -ALLOCATION=100
TEMPLATE -SEGMENT -BLOCK_SIZE=1024
TEMPLATE -SEGMENT -NOENCRYPTION_FLAG
TEMPLATE -SEGMENT -EXTENSION_COUNT=100
TEMPLATE -SEGMENT -GLOBAL_BUFFER_COUNT=1024
TEMPLATE -SEGMENT -LOCK_SPACE=40
TEMPLATE -SEGMENT -MUTEX_SLOTS=1024
TEMPLATE -SEGMENT -RESERVED_BYTES=0
!
TEMPLATE -SEGMENT -ACCESS_METHOD=MM
TEMPLATE -SEGMENT -ALLOCATION=100
TEMPLATE -SEGMENT -BLOCK_SIZE=1024
TEMPLATE -SEGMENT -DEFER
TEMPLATE -SEGMENT -NOENCRYPTION_FLAG
TEMPLATE -SEGMENT -EXTENSION_COUNT=100
TEMPLATE -SEGMENT -GLOBAL_BUFFER_COUNT=1024
TEMPLATE -SEGMENT -LOCK_SPACE=40
TEMPLATE -SEGMENT -MUTEX_SLOTS=1024
TEMPLATE -SEGMENT -RESERVED_BYTES=0
!
TEMPLATE -SEGMENT -ACCESS_METHOD=BG
!
DELETE -REGION DEFAULT
DELETE -SEGMENT DEFAULT
ADD -REGION AUSREG -DYNAMIC_SEGMENT=AUSSEG
ADD -REGION DEFAULT -DYNAMIC_SEGMENT=DEFAULT
ADD -REGION FRREG -DYNAMIC_SEGMENT=FRSEG
ADD -REGION POREG -DYNAMIC_SEGMENT=POSEG
ADD -REGION UKREG -DYNAMIC_SEGMENT=UKSEG
ADD -REGION USSALREG -DYNAMIC_SEGMENT=USSALSEG
ADD -REGION USSMZREG -DYNAMIC_SEGMENT=USSMZSEG
!
ADD -SEGMENT AUSSEG -FILE_NAME="AUS.dat"
ADD -SEGMENT DEFAULT -FILE_NAME="gtm.dat"
ADD -SEGMENT FRSEG -FILE_NAME="France.dat"
ADD -SEGMENT POSEG -FILE_NAME="Poland.dat"
ADD -SEGMENT UKSEG -FILE_NAME="UK.dat"
ADD -SEGMENT USSALSEG -FILE_NAME="USSAL.dat"
ADD -SEGMENT USSMZSEG -FILE_NAME="USSMZ.dat"
!
ADD -GBLNAME EURCentral -COLLATION=1
!
LOCKS -REGION=DEFAULT
ADD -NAME Australia -REGION=AUSREG
ADD -NAME EURCentral("Poland") -REGION=POREG
ADD -NAME EURWest("France") -REGION=FRREG
ADD -NAME EURWest("UK") -REGION=UKREG
ADD -NAME US("South","a":"m") -REGION=USSALREG
ADD -NAME US("South","m": "{") -REGION=USSMZREG
!
```

This command displays the GDE commands to recreate the spanning region example described in the Introduction section.

## Template

The `TEMPLATE` command maintains a set of `-REGION` and `-SEGMENT` qualifier values for use as templates when `ADDING` regions and segments. When an `ADD` command omits qualifiers, GDE uses the template values as defaults.

GDE maintains a separate set of `-SEGMENT` qualifier values for each `ACCESS_METHOD`. When GDE modifies the `ACCESS_METHOD`, it activates the appropriate set of `TEMPLATES` and sets all unspecified qualifiers to the template defaults for the new `ACCESS_METHOD`. Use the `GDE SHOW` command to display qualifier values for all `ACCESS_METHODS`.

The format of the `TEMPLATE` command is:

```
T[EMPLATE] -R[EGION] [-REGION-qualifier...]
T[EMPLATE] -S[EGMENT] [-SEGMENT-qualifier...]
```

The `TEMPLATE` command requires specification of an object-type.

Example:

```
GDE> template -segment -allocation=200000
```

This command modifies the segment template so that any segments `ADDED` after this time produce database files with an `ALLOCATION` of 200,000 GDS blocks.

## Verify

The `VERIFY` command validates information entered into the current Global Directory. It checks the name-to-region mappings to ensure all names map to a region. The `VERIFY` command checks region-to-segment mappings to ensure each region maps to a segment, each segment maps to only one region, and the segment maps to a UNIX file. The `EXIT` command implicitly performs a `VERIFY -ALL`.

The format of the `VERIFY` command is:

```
V[ERIFY]
V[ERIFY] -N[AME] [namespace]
V[ERIFY] -R[EGION] [region-name]
V[ERIFY] -S[EGMENT] [segment-name]
V[ERIFY] -M[AP]
V[ERIFY] -G[BLNAME]
V[ERIFY] -T[EMPLATE]
V[ERIFY] -A[LL]
```

The object-type is optional. `-MAP`, `-TEMPLATE`, and `-ALL` are special qualifiers used as follows:

- MAP** Checks that all names map to a region, all regions map to a segment, and all segments map to a file.
- TEMPLATE** Checks that all templates currently are consistent and useable.
- ALL** Checks all map and template data.

`VERIFY` with no qualifier, `VERIFY -MAP`, and `VERIFY -ALL` each check all current information.

Example:

```
GDE> verify -region regis
```

This command verifies the region `regis`.

## Name, Region, and Segment Qualifiers

The `-NAME`, `-REGION`, and `-SEGMENT` qualifiers each have additional qualifiers used to further define or specify characteristics of a name, region, or segment. The following sections describe these additional qualifiers.

### Name Qualifiers

The following `-NAME` qualifier can be used with the `ADD` or `CHANGE` commands.

```
-REGION=region-name
```

Specifies the name of a region. Region names are not case-sensitive, but are represented as uppercase by GDE.

The minimum length is one alphabetic character.

The maximum length is 31 alphanumeric characters.

Example:

```
GDE> add -name a* -region=areg
```

This command creates the namespace `a*`, if it does not exist, and maps it to the region `areg`.

*Summary*

GDE NAME Qualifiers			
QUALIFIER	DEFAULT	MINIMUM	MAXIMUM
<code>-R[EGION]=region-name (characters)</code>	(none)	1A	16A/N

### Region Qualifiers

The following `-REGION` qualifiers can be used with the `ADD`, `CHANGE`, or `TEMPLATE` commands.

```
-C[OLLATION_DEFAULT]=number
```

Specifies the number of the collation sequence definition to be used as the default for this database file. The number can be any integer from 0 to 255. The number you assign as a value must match the number of a defined collation sequence that resides in the shared library pointed to by the environment variable `gtm_collate_n`. For information on defining this environment variable and creating an alternate collation sequence, refer to the "Internationalization" chapter in the *GT.M Programmer's Guide*.

The minimum `COLLATION_DEFAULT` number is zero, which is the standard M collation sequence.

The maximum `COLLATION_DEFAULT` number is 255.

By default, GDE uses zero (0) as the `COLLATION_DEFAULT`.

```
-D[YNAMIC_SEGMENT]=segment-name
```

Specifies the name of the segment to which the region is mapped. Segment-names are not case-sensitive, but are displayed as uppercase by GDE.

The minimum length is one alphabetic character.

The maximum length is 31 alphanumeric characters.

*-K[KEY\_SIZE]=size in bytes*

Specifies the maximum size of keys, in bytes, which can be stored in the region. The KEY\_SIZE must be less than the RECORD\_SIZE. GDE rejects the command if the KEY\_SIZE is inappropriate for the RECORD\_SIZE.

The minimum KEY\_SIZE is three bytes.

The maximum KEY\_SIZE is 1,019 bytes.

When determining the maximum key size, applications should consider the following:

- GT.M uses packed decimal representation for numeric subscripts which may be larger or smaller than the original representation.
- GT.M substitutes an element terminator for the caret (^), any comma (,), and any right parenthesis ()).
- GT.M adds an extra byte for every string element, including the global name.

For example, the key ^ACN ("Name", "Type") internally occupies 17 bytes.

By default, GDE uses a KEY\_SIZE of 64 bytes.

*-R[RECORD\_SIZE]=size in bytes*

Specifies the maximum size (in bytes) of a global variable node's value that can be stored in a region.

If the size of a global exceeds one database block, GT.M implicitly spans that global across multiple database blocks. In the event a global variable node spans multiple blocks, and the process is not already within a TP transaction, the GT.M run-time system automatically and transparently performs the entire operation within an implicit TP transaction (as it does for Triggers).

The minimum RECORD\_SIZE is zero. A RECORD\_SIZE of zero only allows a global variable node that does not have a value. A typical use of a global variable node with RECORD\_SIZE of zero is for creating indices (where the presence of a node is all that is required).

The maximum RECORD\_SIZE is 1,048,576 bytes (1MiB).

By default, GDE uses a RECORD\_SIZE of 256 bytes.

*-[NO]N[ULL\_SUBSCRIPTS]=[ALWAYS|NEVER|EXISTING]*

Indicates whether GT.M allows null subscripts for global variables stored in the region (that is, whether GT.M permits references such as ^aaa("",1)).

ALWAYS indicates that the null subscripts for global variables are allowed.

NEVER indicates that null subscripts for global variables are not allowed.

EXISTING indicates that null subscripts for global variable can be accessed and updated, but not created anew.

By default, regions have -NULL\_SUBSCRIPTS=NEVER.

*-[NO]STDNULLCOLL*

Determines whether GT.M null subscripts collate in conformance to the M standard.

If STDNULLCOLL is specified, subscripts of globals in the database follow the M standard where the null subscript collates before all other subscripts.

If NOSTDNULLCOLL is specified, null subscripts collate between numeric and string subscripts. FIS strongly recommends that you use STDNULL and against using this non-standard null collation, which is the default for historical reasons.

*-[NO]INST[\_FREEZE\_ON\_ERROR]*

Controls whether custom errors in a region should automatically cause an Instance Freeze. This qualifier modifies the value of "Inst Freeze on Error" file header element.

For more information on setting up a list of custom errors that automatically invoke an Instance Freeze, refer to "Instance Freeze" (page 193).

For more information on setting or clearing an Instance Freeze on an instance irrespective of whether any region is enabled for Instance, refer to "Starting the Source Server" (page 230).

*-[NO]Q[QDBRUNDOWN]*

Quickens normal process shutdown where a large number of processes accessing a database file are required to shutdown almost simultaneously, for example, in benchmarking scenarios. When a terminating GT.M process observes that a large number of processes are attached to a database file and QDBRUNDOWN is enabled, it bypasses checking whether it is the last process accessing the database. Such a check occurs in a critical section and bypassing it also bypasses the usual RUNDOWN actions which accelerates process shutdown removing a possible impediment to process startup. By default, QDBRUNDOWN is disabled.

Note that with QDBRUNDOWN there is a possibility of race condition that might leave the database fileheader and IPC resources in need of cleanup. Although QDBRUNDOWN minimizes the probability of such a race condition, it cannot eliminate it. FIS recommends restricting QDBRUNDOWN usage to special circumstances such as benchmarking and recommends NOQDBRUNDOWN for normal GT.M usage. When using QDBRUNDOWN, FIS recommends an explicit MUPIP RUNDOWN of the database file after the last process exits, to ensure the cleanup of database fileheader and IPC resources.

*-[NO]J[JOURNAL][=journal-option-list]*

This qualifier establishes characteristics for the journal file on newly created databases.

-NOJOURNAL specifies that updates to the database file are not journaled. -NOJOURNAL does not accept an argument assignment.

-JOURNAL specifies that journaling is allowed. -JOURNAL takes one or more arguments in a journal-option-list. The journal-option-list contains keywords separated with commas (,) enclosed in parentheses ( ) with file-names quoted (for example, change -region test -journal=(before,file="foo") . If the list contains only one keyword, the parentheses and quotes are optional.

Although you do not have to establish the criteria for your journaling process at this point, it is efficient to do so, even if you are not entirely sure you will use journaling. The options available for -JOURNAL set up the environment, so it is ready for you to enable with MUPIP SET -JOURNAL. You can also change or add any of the established options at that time.

For more information about journaling, see Chapter 6: "GT.M Journaling" (page 130).

The journal-option-list includes:

- [NO]BE[FORE\_IMAGE]
- F[ILE\_NAME]=file-specification-name
- AUTOSWITCHLIMIT=blocks
- A[LLOCATION]=blocks
- E[XTENSION]=blocks
- BU[FFER\_SIZE]=pages

The following section describes some -JOURNAL options.

*-AU[TOSWITCHLIMIT]=blocks*

Specifies the limit on the size of a journal file. When the journal file size reaches the limit, GT.M automatically switches to a new journal file with a back-pointer to the prior journal file.

*-[NO]BE[FORE\_IMAGE]*

[NO]BEFORE\_IMAGE controls whether the journal should include before-image records.

The BEFORE\_IMAGE option is required if you plan to consider "roll-back" (Backward) recovery of the associated database file or if you plan to use certain database replication options. For a description of this type of recovery, refer to Chapter 6: "GT.M Journaling" (page 130).

*-F[ILE\_NAME]="file-name"*

Specifies the name of the journal file.

Unless the name is the sole journaling option, and is the last parameter on the line, it should always be enclosed in quotation marks in this context.

Journal file-specifications-names are limited to 255 characters.

By default, GDE derives the file-specification-name from the database "file-name".

By default, GDE uses a journal file extension of .mjl.

### *Journal Options Summary*

With GDE, you can create the journal files and define the journal parameters; however, you must use MUIP SET to explicitly turn it ON, and you must specify BEFORE/NOBEFORE at that time.

For information on all Journal options and their allowable minimum and maximum values, see "SET -JOURNAL Options" (page 143) in the "GT.M Journaling" chapter.

### *Summary*

The following table summarizes GDE region qualifiers. It provides their abbreviations, defaults (as provided by FIS), and allowable minimum and maximum values.

GDE REGION Qualifiers			
QUALIFIER	DEFAULT	MINIMUM	MAXIMUM
-C[OLLATION_DEFAULT]=number (integer)	0	0	255
-D[YNAMIC_SEGMENT] =segment-name (char)	-	1	16
-K[EY_SIZE]=size in bytes (integer)	64	3	1,019
-R[ECORD_SIZE]=size in bytes (integer)	256	7	1,048,576 (1 MiB)
-N[ULL_SUBSCRIPTS]=[ALWAYS NEVER EXISTING]	NEVER	-	-
-[NO]STDNULLCOLL	N	-	-
-[NO]INST[_FREEZE_ON_ERROR]	DISABLED	-	-
-[NO]Q[DBRUNDOWN]	DISABLED	-	-
-[NO]J[OURNALS] [=journal-option-list]	-NOJ	-	-

## Segment Qualifiers

The following -SEGMENT qualifiers can be used with the ADD, CHANGE, or TEMPLATE commands.

`-AC[CESS_METHOD]=code`

Specifies the access method or the GT.M buffering strategy for storing and retrieving data from the global database file.

- code can have 2 values - Buffered Global (BG) or Memory Mapped (MM). The default value is BG.
- With BG, the global buffer pool manages the buffers (the OS/file system may also provide additional buffering). You get the choice of using BEFORE\_IMAGE or NOBEFORE\_IMAGE journaling for your database. For details on the implications of these forms of Journaling, see Chapter 6: “GT.M Journaling” (page 130).
  - BG supports both forward and backward recovery and rollback to recover a database without a restore. For more information forward and backward recovery and rollback, see Chapter 5: “General Database Management” (page 67).
  - BG is a likely choice when you need faster recovery times from system failures.
- With MM, GT.M bypasses the global buffer pool and relies entirely on the OS/file system to manage the data traffic between memory and disk. GT.M has no control over the timing of disk updates, therefore there is a greater reliance on the OS/file system for database performance.
  - MM supports NOBEFORE\_IMAGE journaling only. GT.M issues an error if you use MM with BEFORE\_IMAGE Journaling. MM also supports MUPIP FORWARD -RECOVER and MUPIP JOURNAL -ROLLBACK with the -RESYNC or -FETCHRESYNC qualifiers to generate lost and broken transaction files. For more information, see the Journaling and Replication chapters.
  - MM does not support backward recovery/rollback.
  - Depending on your file system, MM may be an option when you need performance advantage in situations where the above restrictions are acceptable.

## Global Directory Editor

- GDE maintains a separate set of segment qualifier values for each ACCESS\_METHOD.
- When GDE modifies the ACCESS\_METHOD, it activates the appropriate set of TEMPLATES and sets all unspecified qualifiers to the default values of the new ACCESS\_METHOD.

Example:

```
GDE> change -segment DEFAULT -access_method=MM
```

This command sets MM as the access method or the GT.M buffering strategy for storing and retrieving database for segment DEFAULT.

*-AL[LOCATION]=blocks*

Specifies the number of blocks GT.M allocates to a disk file when MUPIP creates the file. For GDS files, the number of bytes allocated is the size of the database file header plus the ALLOCATION size times the BLOCK\_SIZE.

- The minimum ALLOCATION is 10 blocks.
- The maximum ALLOCATION is 1,040,187,392 blocks.
- By default, GDE uses an ALLOCATION of 100 blocks.
- The maximum size of a database file is 1,040,187,392(992Mi) blocks.
- Out of the requested allocation, GT.M always reserves 32 global buffers for BG access method for read-only use to ensure that non-dirty global buffers are always available.
- The default ALLOCATION was chosen for initial development and experimentation with GT.M. Because file fragmentation impairs performance, make the initial allocation for production files and large projects large enough to hold the anticipated contents of the file for a length of time consistent with your UNIX file reorganization schedule.

*-BL[BLOCK\_SIZE]=size*

Specifies the size, in bytes, of each database block in the file system. The BLOCK\_SIZE must be a multiple of 512. If the BLOCK\_SIZE is not a multiple of 512, GDE rounds off the BLOCK\_SIZE to the next highest multiple of 512 and issues a warning message.

If the specified BLOCK\_SIZE is less than the minimum, GDE uses the minimum BLOCK\_SIZE. If the specified BLOCK\_SIZE is greater than the maximum, GDE issues an error message.

A BLOCK\_SIZE that is equal to the page size used by your UNIX implementation serves well for most applications, and is a good starting point.

You should determine the block sizes for your application through performance timing and benchmarking. In general, larger block sizes are more efficient from the perspective of the input/output subsystem. However, larger block sizes use more system resources (CPU and shared memory) and may increase collision and retry rates for transaction processing.



### Note

Global nodes that span blocks incur some overhead and optimum application performance is likely to be obtained from a BLOCK\_SIZE that accommodates the majority of nodes within a single block. If you adjust the BLOCK\_SIZE, you should also adjust GLOBAL\_BUFFER\_COUNT.

GDE does not allow you to change the block size to an arbitrary number. It always rounds the block size to the next higher multiple of 512, because the database block size must always be a multiple of 512.

The minimum BLOCK\_SIZE is 512 bytes.

The maximum BLOCK\_SIZE is 65,024 bytes.



### Note

FIS recommends against using databases with block sizes larger than 16KiB. If a specific global variable has records that have large record sizes, FIS recommends placing that global variable in a file by itself with large block sizes and using more appropriate block sizes for other global variables. 4KiB and 8KiB are popular database block sizes.

By default, GDE uses a BLOCK\_SIZE of 1024 bytes.

### *-[NO]ENcryption*

Specifies whether or not the database file for a segment is flagged for encryption. Note that MUPIP CREATE acquires an encryption key for this file and puts a cryptographic hash of the key in the database file header.

### *-EX[TENSION\_COUNT]=blocks*

Specifies the number of extra GDS blocks of disk space by which the file should extend. The extend amount is interpreted as the number of usable GDS blocks to create with the extension. To calculate the number of host operating system blocks added with each extension, multiply the number of GDS blocks added by (GDS BLOCK\_SIZE/host BLOCK\_SIZE); add one local bitmap block for each 512 blocks added in each extension to the amount from step 1. If the extension is not a multiple of 512, remember to roundup when figuring the number of bitmap blocks.

When a MUPIP EXTEND command does not include a -BLOCKS= qualifier, EXTEND uses the extension size in the database header.

The extension amount may be changed with the MUPIP SET command.

The minimum EXTENSION is zero blocks.

When a database file with automatic extension disabled (EXTENSION\_COUNT=0) starts to get full, GT.M records the FREEBLSLOW warning in the system log. So as to not compromise performance, GT.M checks whenever the master bit map must be updated to show that a local bit map is full, and issues the warning if there are fewer than 512 free blocks or if the number of free blocks is less than total blocks/32. This means that for databases whose size is 512 blocks or less the warning comes at the last successful update before the database becomes full.

The maximum EXTENSION is 65,535 blocks.

By default, GDE uses an EXTENSION of 100 blocks.

Like allocation, the default extension amount was chosen for initial development and experimentation. Use larger extensions for larger actual applications. Because multiple file extensions adversely affect performance, set up extensions appropriate to the file allocation.

### *-F[ILE\_NAME]=file-name*

Specifies the file for a segment.

The maximum file name length is 255 characters.

By default, GDE uses a file-name of mumps followed by the default extension, which is .dat. You can specify any filename and extension of your choice for a database file as long as it is valid on your operating system.

*-G[GLOBAL\_BUFFER\_COUNT]=size*

Specifies the number of global buffers for a file. Global buffers reside in shared memory and are part of the database caching mechanisms. Global buffers do not apply to MM databases.

Choose the settings for this qualifier carefully. Small numbers of global buffers tend to throttle database performance. However, if your system has limited memory and the database file traffic is not heavy enough to hold the cache in RAM, increasing GLOBAL\_BUFFER\_COUNT may trigger paging.

If database global buffers are paged out, it will result in poor performance. Therefore, do not increase this factor to a large value without careful observation.

The proper number of GLOBAL\_BUFFERs depends on the application and the amount of primary memory available on the system. Most production databases exhibit a direct relationship between the number of GLOBAL\_BUFFERs and performance. However, the relationship is not linear, but asymptotic, so that increases past some point have progressively less benefit. This point of diminishing returns depends on the application. For most applications, FIS expects the optimum number of GLOBAL\_BUFFERs to be between 1K and 64K.

Because transaction processing can be involved in an update and a transaction is limited to half the GLOBAL\_BUFFER\_COUNT, the value for GLOBAL\_BUFFER\_COUNT should therefore be at least twenty-six plus twice the number of the blocks required by the largest global variable node in your application.

Generally, you should increase the number of GLOBAL\_BUFFERs for production GDS database files. This is because GT.M uses the shared memory database cache associated with each GDS file for the majority of caching.

The minimum GLOBAL\_BUFFER\_COUNT for BG is 64 blocks.

The maximum for GLOBAL\_BUFFER\_COUNT for BG is 2,147,483,647 blocks, but may vary depending on your platform.

By default, GDE uses a GLOBAL\_BUFFER\_COUNT that is appropriate for initial development use on each platform, but probably too small for production applications.



### Note

If global buffers are "paged out," improvements in system performance resulting from more global buffers will be more than offset by the dramatic slowdown that results from global buffers that are "paged out."

*-L[LOCK\_SPACE]=integer*

Specifies the number of pages of space to use for the lock database stored with this segment. The size of a page is always 512 bytes.

As GT.M runs out of space to store LOCK control information, LOCKs become progressively less efficient. If a single process consumes all the LOCK space, it cannot continue, and any other processes cannot proceed using LOCKs.

The minimum LOCK\_SPACE is 10 pages.

## Global Directory Editor

The maximum LOCK\_SPACE is 65,536 pages.

By default, GDE uses a LOCK\_SPACE of 40 pages.

LOCK\_SPACE usage depends on the number of locks and the number of processes waiting for locks. To estimate lock space needs, here is a rule of thumb:

- 1.5KiB overhead for the lock space, plus
- 640 bytes for each lock base name, plus
- 128 bytes for each subscript, plus
- 128 bytes for each waiting process.

Generally, you would limit LOCK\_SPACE only when memory is scarce or you want to be made aware of unexpected levels of LOCK usage. For most other cases, there is no reason to limit the LOCK\_SPACE. If you are introducing new code, FIS recommends using TSTART and TCOMMIT as a more efficient alternate for most LOCKs because it pushes the responsibility for Isolation onto GT.M, which internally manages them with optimistic algorithms.

`-M[UTEX_SLOTS]=integer`

Specifies the number of mutex slots for a database file. GT.M uses mutex slots to manage database contention. FIS recommends you configure the slots to cover the maximum number of processes you expect to concurrently access the database file, as an insufficient number of slots can lead to much steeper and more severe degradation of performance under heavy loads. The minimum is 1Ki and the maximum is 32Ki. The default of 1Ki should be appropriate for most situations.

`-R[RESERVED_BYTES]=size`

Specifies the size to be reserved in each database block. RESERVED\_BYTES is generally used to reserve room for compatibility with other implementations of M or to observe communications protocol restrictions. RESERVED\_BYTES may also be used as a user-managed fill factor.

The minimum RESERVED\_BYTES is zero bytes.

The maximum Reserved\_Bytes is the block size minus the size of the block header (which is 7 or 8 depending on your platform) minus the maximum record size.

By default, GDE uses a RESERVED\_BYTES size of zero bytes.

### Summary

The following table summarizes GDE segment qualifiers. It provides abbreviations, defaults (as provided by FIS), and allowable minimum and maximum values.

GDE SEGMENT Qualifiers			
QUALIFIER	DEFAULT	MIN	MAX
-AC[CESS_METHOD]=BG MM	BG	-	-
<b>** BLOCK_SIZE minus the size of the block header</b>			
<b>* May vary by platform</b>			

## Global Directory Editor

GDE SEGMENT Qualifiers			
QUALIFIER	DEFAULT	MIN	MAX
-AL[LOCATION]=size (blocks)	100	10	1,040,187,392(992Mi)
-BL[OCK_SIZE]=size (bytes)	1,024	512	65,024
-[NO]EN[CRYPTION]	0		
-EX[TENSION_COUNT]=size (blocks)	100	0	65,535
-F[ILE_NAME]=file-name (chars)	mumps.dat	-	255
-G[LOBAL_BUFFER_COUNT]=size (blocks)	1024*	64	2,147,483,647
-L[OCK_SPACE]=size (pages)	40	10	65536
-M[UTEX_SLOTS]=integer	1,024	1,024	32,768
-R[ESERVED_BYTES]=size (bytes)	0	0	block size-7
<b>** BLOCK_SIZE minus the size of the block header</b>			
<b>* May vary by platform</b>			

## Gblname Qualifiers

The following -GBLNAME qualifier can be used with the ADD, CHANGE, or TEMPLATE commands.

*-C[OLLATION]=collation\_number*

Specifies the collation number for a global name; a value of 0 specifies standard M collation. The first time that a GT.M processes accesses a global variable name in a database file, it determines the collation sequence as follows:

- If a Global Variable Tree (GVT) exists (that is, global variable nodes exist, or have previously existed, even if they have been KILL'd), use the existing collation:
  - If there is a collation specified in the Directory Tree (DT) for that variable, use it after confirming that this matches the collation in the global directory.
  - else (that is, there is no collation specified in the DT):
    - If there is collation specified for that global variable in the global directory use it
    - else if there is a default for that database file, use it
    - else (that is, neither exists), use standard M collation
- else (that is, a GVT does not exist, which in turn means there is no DT):
  - If there is collation specified for that global variable in the global directory use it
  - else, if there is a default for that database file, use it
  - else (that is, neither exists), use standard M collation

## GDE Command Summary

The following table summarizes GDE commands, abbreviations, object types, required object names, and optional qualifiers.

GDE Command Summary		
Command	Specified Object Type	Required Object Name/[Optional] Qualifier
@	N/A	file-name
A[DD]	-N[AME]	namespace -R[EGION]=region-name
-	-R[EGION]	region-name -D[YNAMIC]=segment-name [-REGION-qualifier...]
-	-S[EGMENT]	segment-name -F[ILE_NAME]=file-name [-SEGMENT-qualifier...]
-	-G[BLNAME]	global-name -C[OLLATION]=collation
C[HANGE]	-N[AME]	namespace -R[EGION]=new-region
-	-R[EGION]	region-name [-REGION-qualifier...]
-	-S[EGMENT]	segment-name [-SEGMENT-qualifier]
-	-G[BLNAME]	global-name -C[OLLATION]=collation
D[ELETE]	-N[AME]	namespace
-	-R[EGION]	region-name
-	-S[EGMENT]	segment-name
-	-G[BLNAME]	global-name -C[OLLATION]=collation
E[XIT]	N/A	N/A
HE[LP]	N/A	Keyword
LOC[KS]	N/A	-R[EGION]=region-name
LOG	N/A	[-ON][=file-name]
* -ALL is the default for the SHOW and VERIFY commands.		

Global Directory Editor

GDE Command Summary		
Command	Specified Object Type	Required Object Name/[Optional] Qualifier
		[-OF[F]]
Q[UIT]	N/A	N/A
R[ENAME]	-N[AME]	old-name new-name
-	-R[EGION]	old-reg-name new-reg-name
-	-S[EGMENT]	old-seg-name new-seg-name
-	-G[BLNAME]	global-name -C[OLLATION]=collation
SE[TGD]	N/A	-F[ILE]=file-name [-Q[UIT]]
SH[OW]	-N[AME]	[namespace]
-	-R[EGION]	[region-name]
-	-S[EGMENT]	[segment-name]
-	-G[BLNAME]	global-name -C[OLLATION]=collation
-	-M[AP]	[R[EGION]=region-name]
-	T[EMPLATE]	N/A
-	-A[LL]*	N/A
T[EMPLATE]	-R[EGION]	[-REGION-qualifier...]
-	-S[EGMENT]	[ -SEGMENT-qualifier...]
V[ERIFY]	-N[AME]	[namespace]
-	-R[EGION]	[region-name]
-	-S[EGMENT]	[segment-name]
-	-G[BLNAME]	global-name -C[OLLATION]=collation
-	-M[AP]	N/A
-	-T[EMPLATE]	N/A
-	-A[LL]*	N/A
* -ALL is the default for the SHOW and VERIFY commands.		

## GDE Command Qualifier Summary

The following table summarizes all qualifiers for the ADD, CHANGE, and TEMPLATE commands. The defaults are those supplied by FIS.

GDE Command Qualifiers						
QUALIFIER	DEF	MIN	MAX	NAM	REG	SEG
-AC[CESS_METHOD]=code	BG	-	-	-	-	X
-AL[LOCATION]=size(blocks)	100	10	1,040,187,392(992Mi)	-	-	X
-BL[OCK_SIZE]=size(bytes)	1024	512	65024	-	-	X
-C[OLLATION_DEFAULT]=id-number (integer)	0	0	255	-	X	-
-D[YNAMIC_SEGMENT]=segment-name (chars)	*	1A	16A/N	-	X	-
-EX[TENSION_COUNT]=size (blks)	100	0	65535	-	-	X
-F[ILE_NAME]=file-name (chars)	**	1A	255A/N	-	-	X
-G[LOBAL_BUFFER_COUNT]=size (blocks)	1,024 ***	64	2,147,483,647 ***	-	-	X
-K[EY_SIZE]=size (bytes)	64	3	1,019	-	X	-
-L[OCK_SPACE]=size (pages)	40	10	65,536	-	-	X
-M[UTEX_SLOTS]=integer	1,024	1,024	32,768	-	-	X
-[NO]INST[_FREEZE_ON_ERROR]	FALSE	-	-	-	X	-
-[NO]Q[DBRUNDOWN]	FALSE	-	-	-	X	-
-[NO]J[JOURNAL]=option-list	-NOJ	-	-	-	X	-
-N[ULL_SUBSCRIPTS]=[ALWAYS NEVER  EXISTING]	NEVER or ****	-	-	-	X	-
-[NO]STDNULLCOLL[=TRUE FALSE]	FALSE	-	-	-	X	-
-R[ECORD_SIZE]=size (bytes)	256	7	1,048,576	-	X	-
-R[EGION] region-name (chars)	*	1A	16A/N	X	-	-
-R[ESERVED_BYTES]=size (bytes)	0	0	blocksize	-	-	X
<p>* <b>DEFAULT is the default region- and segment-name</b></p> <p>** <b>MUMPS is the default file-name</b></p> <p>*** <b>May vary by platform</b></p> <p>**** <b>-NONULL_SUBSCRIPTS</b></p>						

## Chapter 5. General Database Management

Revision History		
Revision V6.1-000	01 August 2014	<ul style="list-style-type: none"> <li>In "Introduction" (page 68), added a note recommending against running any GT.M component as root.</li> </ul>
Revision V6.0-003	27 January 2014	<ul style="list-style-type: none"> <li>In "SET" (page 109), added the -Mutex_slots qualifier.</li> <li>In "RUNDOWN" (page 108), added the -Override qualifier.</li> </ul>
Revision V6.0-001/2	10 April 2013	In "BACKUP" (page 71), added notes for "Before starting a MUPIP BACKUP".
Revision V6.0-001/1	22 March 2013	Improved the formatting of all command syntaxes.
Revision V6.0-001	27 February 2013	<ul style="list-style-type: none"> <li>In "BACKUP" (page 71), added information about the BKUPRUNNING message and added information on the handling of KILLS in Progress and Abandoned Kills during a backup.</li> <li>In "EXTRACT" (page 82), added the description of -STDOUT.</li> <li>In MUPIP INTEG, specified the threshold limit of incorrectly marked busy errors in "-MAP" (page 92) and added information on clearing KILLS in Progress and Abandoned Kills.</li> </ul>
Revision V6.0-000/1	21 November 2012	<ul style="list-style-type: none"> <li>In MUPIP SET, added the description of "-PArtil_recov_bypass" (page 110).</li> <li>Updated "TRIGGER" (page 116) for V6.0-000.</li> </ul>
Revision V6.0-000	19 October 2012	In MUPIP SET, added the description of "-INST_freeze_on_error" (page 112).
Revision V5.5-000/11	05 October 2012	Corrected the description of the -ADJACENCY qualifier and added more information about "INTEG" (page 88).
Revision V5.5-000/10	28 September 2012	Corrected the maximum value of "-Lock_space" (page 111) and improved the description of "-Flush_time" (page 111).
Revision V5.5-000/8	03 August 2012	<ul style="list-style-type: none"> <li>In "SET" [109], improved the description of the LOCK_SPACE qualifier.</li> <li>In "BACKUP" [71], added a note about supporting only one concurrent backup.</li> </ul>
Revision V5.5-000/4	6 June 2012	<ul style="list-style-type: none"> <li>In "LOAD" [98], added the description of -STDIN and its example.</li> </ul>

## General Database Management

		<ul style="list-style-type: none"><li>• In “REORG ” [102], added the description of -TRUNCATE.</li><li>• In “-ROLLBACK [{-ON[LINE]}-NOO[NLINE]]” [155], added the description of -ONLINE.</li></ul>
Revision V5.5-000	27 February 2012	Updated “REORG ” [102] for V5.5-000. Also, improved the description of -EXCLUDE and -SELECT qualifiers.
Revision 1	10 November 2011	In the “SET ” [109] section, changed “MUPIP SET” to “MUPIP SET”.
Revision V5.4-002B	24 October 2011	Conversion to documentation revision history reflecting GT.M releases with revision history for each chapter.

## Introduction

This chapter describes common database management operations such as creating database files, modifying database characteristics, database backup and restore, routine integrity checks, extracting or loading data, and optimizing performance.

GT.M uses M Peripheral Interchange Program (MUPIP) for GT.M database management, database journaling, and logical multisite replication (LMS). This chapter summarizes the MUPIP commands pertaining to GT.M database management and serves as a foundation for more advanced GT.M functionality described for Journaling and LMS.

For MUPIP commands pertaining to database journaling, refer to Chapter 6: “*GT.M Journaling*” (page 130).

For MUPIP commands pertaining to multisite database replication, refer to Chapter 7: “*Database Replication*” (page 173).



### Note

Two MUPIP operations - INTRPT and STOP - perform process management functions. All other MUPIP operations relate to the operation of the database.

The GT.M installation procedure places the MUPIP utility program in a directory specified by `$gtm_dist`.

Invoke MUPIP by executing the `mupip` program at the shell prompt. If this does not work, consult your system manager (MUPIP requires that the `$gtm_dist` point to the directory containing the MUPIP executable image).

```
$gtm_dist/mupip
MUPIP>
```

MUPIP asks for commands, with the MUPIP> prompt. Enter the EXIT command at the MUPIP> prompt to stop the utility. MUPIP performs one operation at a time, and automatically terminates after most operations.

When additional information appears on the command line after the `mupip` program name, MUPIP processes the additional information as its command, for example:

```
$gtm_dist/mupip stop 1158
```

This starts MUPIP and stops the process with Process ID (PID) 1158.

Some MUPIP commands require information contained in the global directory. Therefore, a process must have access to a valid global directory before using any MUPIP commands other than EXIT, INTRPT, JOURNAL, RESTORE, STOP and the -file option of any command that has that option.

The environment variable `gtmgbldir` specifies the active global directory.

A `gtmgbldir` value of `mumps.gld` tells MUPIP to look for a global directory file `mumps.gld` in the current directory. For more information on the global directory, refer to “*Global Directory Editor*” (page 32).



### Important

FIS recommends against running GT.M components as root. When run as root, GT.M components use the owner and group of the database file as the owner and group of newly created journal files, backup files, snapshot files, shared memory, and semaphores. In addition, they set the permissions on the resulting files, shared memory, and semaphores, as if running as the owner of the database file and as a member of the database file group.

## Operations - Standalone and Concurrent Access

While most MUPIP operations can be performed when GT.M processes are actively accessing database files, some operations require stand-alone access. When using standalone access, no other process can access the database file(s). When using concurrent access, other processes can read or update the database file(s) while MUPIP accesses them. A few operations permit concurrent access to read database files, but not to update them. All MUPIP operations can be performed with stand-alone access - there is never a requirement for another process to be accessing database files when MUPIP operates on them.

Most MUPIP operations require write access to the database files with which they interact. The exceptions are `INTRPT` and `STOP`, which do not require database access, but may require other privileges; `EXTRACT`, which requires read access; and `INTEG`, which may require write access, depending on the circumstances it encounters and the qualifiers with which it is invoked. The following table displays some of the MUPIP operations and their database access requirements.

Operations	MUPIP command	Database Access Requirements
Backup database files	MUPIP BACKUP	Backup never requires standalone access and concurrent write access is controlled by <code>-[NO]ONLINE</code> .
Create and initialize database files	MUPIP CREATE	Standalone access
Converts a database file from one endian format to the other (BIG to LITTLE or LITTLE to BIG)	MUPIP ENDIANCVT	Standalone access
Recover database files (for example, after a system crash) and extract journal records	MUPIP JOURNAL	Standalone access
Restore databases from bytestream backup files	MUPIP RESTORE	Standalone access
Properly close database files when processes terminate abnormally.	MUPIP RUNDOWN	Standalone access
Modify database and/or journal file characteristics	MUPIP SET	Standalone access is required if the MUPIP SET command specifies <code>-ACCESS_METHOD</code> , <code>-GLOBAL_BUFFERS</code> , <code>-MUTEX_SLOTS</code> , -

## General Database Management

Operations	MUPIP command	Database Access Requirements
		LOCK_SPACE or -NOJOURNAL, or if any of the -JOURNAL options ENABLE, DISABLE, or BUFFER_SIZE are specified.
Backup database files	MUPIP BACKUP	Concurrent access.
Grow the size of BG database files	MUPIP EXTEND	Concurrent access.
Export data from database files into sequential (flat) or binary files	MUPIP EXTRACT	Although MUPIP EXTRACT command works with concurrent access, it implicitly freezes the database to prevent updates. Therefore, from an application standpoint, you might plan for a standalone access during a MUPIP EXTRACT operation.
Prevent updates to database files	MUPIP FREEZE	Standalone access.
Check the integrity of GDS databases	MUPIP INTEG	Concurrent access. However, standalone access is required if MUPIP INTEG specifies -FILE.
Import data into databases	MUPIP LOAD	Although MUPIP LOAD works with concurrent access, you should always assess the significance of performing a MUPIP LOAD operation when an application is running because it may result in an inconsistent application state for the database.
Defragment database files to improve performance	MUPIP REORG	Concurrent access.
Send an asynchronous signal to a GT.M process	MUPIP INTRPT	Non-database access.
Stop GT.M processes	MUPIP STOP	Non-database access.

## MUPIP

The general format of MUPIP commands is:

```
mupip command [-qualifier[...]] [object[,...]] [destination]
```

MUPIP allows the abbreviation of commands and qualifiers. In each section describing a command or qualifier, the abbreviation is also shown (for example, B[ACKUP]). The abbreviated version of the command you can use on the command line is B. To avoid future compatibility problems and improve the readability, specify at least four characters when using MUPIP commands in scripts.

Although you can enter commands in both upper and lower case (the mupip program name itself must be in lower case on UNIX/Linux), the typographical convention used in this chapter is all small letters for commands. Another convention is in the presentation of command syntax. If the full format of the command is too long for a single line of print, the presentation wraps around into additional lines.

```
$ mupip backup -bytestream -transaction=1 accounts,history,tables,miscellaneous
▶ /var/production/backup/
```



When you enter a MUPIP command, one of its variable arguments is the region-list. region-list identify the target of the command and may include the UNIX wildcards "?" and "\*". Region-lists containing UNIX wildcard characters must always

be quoted, for example, "\*" to prevent inappropriate expansion by the UNIX shell. Similarly, for file and directory names you might want to avoid non-graphic characters and most punctuations except underbars (\_), not because of GT.M conventions but because of inappropriate expansion by UNIX shells.

MUPIP qualifier values are restricted only by the maximum size of the command input line, which is 4KB on some systems and upto 64KB on others.

## Commands and Qualifiers

The MUPIP commands described in this section are used for common database operations and serves as the foundation for more advanced functionality like Journaling and Replication.

### BACKUP

Saves the contents of the database. It provides a consistent application snapshot across all database regions involved in the backup operation.

The format of the MUPIP BACKUP command is:

```
B[ACKUP]
[
  -BK[UPDBJNL]={DISABLE|OFF}]
  -B[YTESTREAM] [-NET[TIMEOUT]]
  -DA[TABASE]
  -DBG
  -[NO]NEWJNLFILES[=[NO]PREVLINK],[NO]S[YNC_IO]]
  -O[NLINE]
  -REC[ORD]
  -REPL[ACE]
  -REPLINSTANCE=target_location
  -S[INCE]={DATABASE|BYTESTREAM|RECORD}
  -T[RANSACTION]=hexadecimal_transaction_number
] region-list[,...] destination-list
```



#### Important

MUPIP BACKUP does a more comprehensive job of managing backup activities than other backup techniques such as a SAN backup, breaking a disk mirror, or a file system snapshot because it integrates journal management, instance file management, and records timestamps in the database file headers. To use other techniques, you must first freeze all regions concurrently with a command such as MUPIP FREEZE - ON "\*" in order to ensure a consistent copy of files with internal structural integrity. FIS neither endorses nor tests any third party products for backing up a GT.M database.

- MUPIP BACKUP supports two methods of database backup: -BYTESTREAM and -DATABASE. MUPIP BACKUP - BYTESTREAM directs the output to a broad range of devices, including disks, TCP sockets, and pipes. MUPIP BACKUP - DATABASE directs the output to random access devices (that is, disks).
- [NO]ONLINE qualifier determines whether MUPIP BACKUP should suspend updates to regions. For example, MUPIP BACKUP -NOONLINE suspends updates to all regions from the time it starts the first region until it finishes the last region. However, it does not suspend processes that only read from the database.
- By default, MUPIP BACKUP is -DATABASE -ONLINE.

## General Database Management

- If any region name does not map to an existing accessible file, or if any element of the destination list is invalid, BACKUP rejects the command with an error.
- region-list may specify more than one region of the current global directory in a list. Regions are separated by a comma, and wildcards can be used to specify them. Any region-name may include the wildcard characters \* and % (remember to escape them to protect them from inappropriate expansion by the shell). Any region name expansion occurs in M (ASCII) collation order.
- Depending on the type of backup, destination-list may be a single directory, or a comma separated list of destinations including files, piped commands, or a TCP socket address (a combination of IPv4 or IPV6 hostname and a port number).
- Region-list and destination-list items are matched in order - the first region is mapped to the first destination, the second to the second destination, and so on. If GT.M encounters a region mapped to a directory, GT.M treats that directory as the destination for all subsequent regions in the region-list.
- GT.M implicitly timestamps both BYTESTREAM and DATABASE backups using relative timestamps (transaction numbers). You can also explicitly specify a RECORD timestamp for custom-control (SANS or mirrored disk) backup protocol. You might want to use these timestamps as reference points for subsequent backups.
- It takes approximately one (1) minute (per region) for BACKUP -ONLINE to give up and bypass a KILLS in progress; backup does not wait for Abandoned Kills to clear.
- The environment variable gtm\_baktmpdir specifies the directory where mupip backup creates temporary files. If gtm\_baktmpdir is not defined, GT.M uses the deprecated GTM\_BAKTMPDIR environment variable if defined, and otherwise uses the current working directory.
- When you restrict access to a database file, GT.M propagates those restrictions to shared resources associated with the database file, such as semaphores, shared memory, journals and temporary files used in the course of MUPIP BACKUP.
- GT.M supports only one concurrent -ONLINE backup on a database. MUPIP BACKUP displays the BKUPRUNNING message if started when there is an already running BACKUP.
- MUPIP BACKUP protects against overwriting of existing destination files. However, it cannot protect other destinations, for example, if the destination is a pipe into a shell command that overwrites a file.



### Before starting a MUPIP BACKUP

Perform the following tasks before you begin a database backup.

- Ensure adequate disk space for target location and temporary files. Set the environment variable gtm\_baktmpdir to specify the directory where MUPIP BACKUP creates temporary files. If gtm\_baktmpdir is not defined, GT.M uses the deprecated GTM\_BAKTMPDIR environment variable if defined, and otherwise uses the current working directory. Do not place temporary files in the current directory for large databases in production environments.
- When using replication, ensure that the Source/Receiver process is alive (MUPIP REPLIC -SOURCE/-RECEIVER -CHECKHEALTH). Always backup the replicating instance file with the database (BACKUP -REPLINST).
- If you intend to use a -DATABASE backup at the same time in the same computer system as the source database, be sure to disable journaling in the backed up database with -BKUPDBJNL=DISABLE.

## General Database Management

- When doing a complete backup, switch journal files as part of the backup command using `-NEWJNLFILES=NOPREVLINK`. This aligns the journal files with the backup and simplifies journal file retention.
- If you follow separate procedures for backup and archive (moving to secondary storage), you can save time by starting archive as soon as MUPIP BACKUP completes the process of creating a backup database file for a region. You do not need to wait for MUPIP BACKUP to complete processing for all regions before starting archive. For example, a message like:

```
DB file /home/jdoe/.fis-gtm/V6.0-001_x86_64/g/gtm.dat backed up in file /backup/gtm.dat
Transactions up to 0x0000000000E92E04 are backed up.
```

confirms that gtm.dat is backed up correctly and is ready for archive.

- Determine an appropriate frequency, timing, and backup method (`-BYTESTREAM` or `-COMPREHENSIVE`) based on the situation.
- Ensure the user issuing backup commands has appropriate permissions before starting the backup. Backup files have the ownership of the user running MUPIP BACKUP.
- There is one circumstance under which a MUPIP BACKUP is not advised. When your operational procedures call for taking backups of unmodified databases and journal files on rebooting a system after a crash, then use an underlying operating system command (`cp`, `cpio`, `gzip`, `tar`, and so on) which will open the files read-only. Note that for ordinary system crashes where the system simply stops writing to open files at power down, you can use MUPIP JOURNAL to recover journaled database files, and taking backups on reboot should not be required. However, for system crashes with the possibility of damage to files already written to disk (for example, if the crash involved an IO controller with the potential for having written random data to disk immediately prior to power down), such backups on reboot are appropriate.

Example:

```
$ mupip backup "*" /gtm/bkup
```

This example creates ready-to-run database backup of all regions.

### **-BKupdbjnl**

A backup database shares the same journaling characteristics of the source database. However, with BKUPDBJNL you can disable or turn off journaling in the backup database. Use this qualifier if you intend to open your backup database at the same time in the same environment as the source database.

The format of the BKUPDBJNL qualifier is:

```
-BK[UPDBJNL]={DISABLE|OFF}
```

- Specify `DISABLE` to disable journaling in the backup database.
- Specify `OFF` to turn off journaling in the backup database.
- Only one of the qualifiers `DISABLE` or `OFF` can be specified at any given point.

## -Bytestream

Transfers MUPIP BACKUP output to a TCP connection, file (or a backup directory), or a pipe. If there are multiple .dat files, BYTESTREAM transfers output to a comma separated list of TCP connections, incremental backup files and/or directories, or pipes. When used with -SINCE or -TRANSACTION, MUPIP BACKUP allow incremental backup, that is, include database blocks that have changed since a prior point specified by the -SINCE or -TRANSACTION.



### Note

MUPIP BACKUP output to a TCP connection saves disk I/O bandwidth on the current system.

All bytestream backups needs to be restored to a random access file (with MUPIP RESTORE) before being used as a database file. -BYTESTREAM can also send the output directly to a listening MUPIP RESTORE process via a TCP/IP connection or a pipe.

The format of the BYTESTREAM qualifier is:

```
-B[YTESTREAM]
```

- -BYTESTREAM is compatible with -SINCE and -TRANSACTION.
- -INCREMENTAL is deprecated in favor of -BYTESTREAM. For upward compatibility, MUPIP temporarily continues to support the deprecated -INCREMENTAL.

## -Database

Creates a disk-to-disk backup copy of the files of all selected regions. DATABASE backup copy is a ready-to-use a GT.M database unlike BYTESREAM backup which is required to be restored to a random access file.



### Note

The DATABASE qualifier does not support backup to magnetic tape.

The format of the DATABASE qualifier is:

```
-D[ATABASE]
```

- By default, MUPIP BACKUP uses -DATABASE.
- The DATABASE qualifier is only compatible with the -[NO]NEW[JNLFILES], -ONLINE, and -RECORD qualifiers.
- -COMPREHENSIVE is depreciated in favor of -DATABASE. For upward compatibility, MUPIP temporarily continues to support the deprecated -COMPREHENSIVE.

## -NETtimeout

Specifies the timeout period when a bytestream BACKUP data is sent over a TCP/IP connection. The format of the NETTIMEOUT qualifier is:

```
NET[TIMEOUT]=seconds
```

- The default value is 30 seconds.
- Use only with: -BYTESTREAM.

## -NEWJNLFILES

Determines the journaling characteristics of the database files being backed-up. All the established journaling characteristics apply to new journal files. This qualifier is effective only for an ONLINE backup (the default), when the database has journaling enabled.

The format of the NEWJNLFILES qualifier is:

```
-[NO]NEWJNLFILES[=[NO]PREVLINK], [NO]S[YNC_IO]]
```

- -NEWJNLFILES can take the following three values:
  1. PREVLINK: Back links new journal files with the prior generation journal files. This is the default value.
  2. NOPREVLINK: Indicates that there should be no back link between the newly created journals and prior generation journal files.
  3. SYNC\_IO: Specifies that every WRITE to a journal file to be committed directly to disk. On high-end disk subsystems (for example, those that include non-volatile cache and that consider the data to be committed when it reaches this cache), this might result in better performance than the NOSYNC\_IO option. NOSYNC\_IO turn off this option.
- -NONEWJNLFILES causes journaling to continue with the current journal files. It does not accept any arguments.
- The default is -NEWJNLFILES=PREVLINK.

## -Online

Specifies that while a MUPIP BACKUP operation is active, other processes can update the database without affecting the result of the backup. The format of the ONLINE qualifier is:

```
-[NO]O[NLINE]
```

- MUPIP BACKUP -ONLINE creates a backup of the database as of the moment the backup starts. If the running processes subsequently update the database, the backup does not reflect those updates.
- MUPIP BACKUP -ONLINE on regions(s) waits for up to one minute so any concurrent KILL or MUPIP REORG operations can complete. If the KILL or MUPIP REORG operations do not complete within one minute, MUPIP BACKUP -ONLINE starts the backup with a warning that the backup may contain incorrectly marked busy blocks. Such blocks waste space and can desensitize operators to much more dangerous errors, but otherwise don't affect database integrity. If you get such an error, it may be better to stop the backup and restart it when KILL or MUPIP REORG operations are less likely to interfere. Performing MUPIP STOP on a process performing a KILL or MUPIP REORG operation may leave the database with incorrectly marked busy blocks. In this situation, GT.M converts the ongoing KILLS flag to abandoned KILLS flag. If MUPIP BACKUP -ONLINE encounters ADANDONED\_KILLS, it gives a message and then starts the backup. An ABANDONED\_KILLS error means both the original database and the backup database possibly have incorrectly busy blocks which should be corrected promptly.
- By default, MUPIP BACKUP is -ONLINE.

## -Record

Timestamps (in the form of a transaction number) a database file to mark a reference point for subsequent bytestream, database, or custom backup (SANS or disk mirror) protocols. Even though -DATABASE and -BYTESTREAM both mark their

own relative timestamps, -RECORD provides an additional timestamp option. MUPIP FREEZE also provides the -RECORD qualifier because a FREEZE may be used to set the database up for a SAN or disk-mirror based backup mechanism.

The format of the RECORD qualifier is:

```
-R[RECORD]
```

- Use -RECORD (with the hyphen) to timestamp a reference point and use RECORD as a keyword (as in -SINCE=RECORD) to specify the starting point for a MUPIP BACKUP operation.
- -RECORD replaces the previously RECORDED transaction identifier for the database file.

### -REPLace

Overwrites the existing destination files.

The format of the REPLACE qualifier is:

```
-[REPL]ACE
```

- By default, MUPIP BACKUP protect against overwriting the destination files. -REPLACE disables this default behavior.
- -REPLACE is compatible only with -DATABASE.

### -REPLInstance

Specifies the target location to place the backup of the replication instance file.



#### Note

The replication instance file should always be backed up with the database file.

The format of the REPLINSTANCE qualifier is:

```
-REPLI[NSTANCE]=<target_location>
```

### -Since

Includes blocks changed since the last specified backup. The format of the SINCE qualifier is:

```
-S[INCE]={DATABASE|BYTESTREAM|RECORD}
```

- D[ATABASE] - Backup all changes since the last MUPIP BACKUP -DATABASE.
- B[YTESTREAM] - Backup all changes since the last MUPIP BACKUP -BYTESTREAM.
- R[ECORD] - Backup all changes since the last MUPIP BACKUP -RECORD.

By default, MUPIP BACKUP -BYTESTREAM operates as -SINCE=DATABASE.

Incompatible with: -TRANSACTION.

## -Transaction

Specifies the transaction number of a starting transaction that causes BACKUP -BYTESTREAM to copy all blocks that have been changed by that transaction and all subsequent transactions. The format of the TRANSACTION qualifier is:

```
-T[TRANSACTION]=transaction-number
```

- A Transaction number is always 16 digit hexadecimal number. It appears in a DSE DUMP -FILEHEADER with the label "Current transaction".
- If the transaction number is invalid, MUPIP BACKUP reports an error and rejects the command.
- It may be faster than a DATABASE backup, if the database is mostly empty.
- Incompatible with: -DATABASE, -SINCE



### Note

A point in time that is consistent from an application perspective is unlikely to have the same transaction number in all database regions. Therefore, except for -TRANSACTION=1, this qualifier is not likely to be useful for any backup involving multiple regions.

## Examples for MUPIP BACKUP

Example:

```
$ mupip backup -bytestream REPTILES,BIRDS bkup
```

Suppose that the environment variable gtmgbldir has regions REPTILES and BIRDS that map to files called REPTILES.DAT and BIRDS.DAT (no matter which directory or directories the files reside in). Then the above example creates bytestream backup files REPTILES.DAT and BIRDS.DAT in the bkup directory since the last DATABASE backup.

Example:

```
$ mupip backup -bkupdbjnl="OFF" "*"
```

This command turns off journaling in the backup database.

Example:

```
$ mupip backup -bytestream "*" tcp://philadelphia:7883,tcp://tokyo:8892
```

Assuming a Global Directory with two regions pointing to ACN.DAT and HIST.DAT, this example creates a backup of ACN.DAT to a possible MUPIP RESTORE process listening at port 7883 on server philadelphia and HIST.DAT to a possible MUPIP RESTORE process listening at port 8893 on server tokyo.

Always specify the <machine name> and <port> even if both backup and restore are on the same system, and ensure that the MUPIP RESTORE process is started before the MUPIP BACKUP process.

Example:

```
$ mupip backup -database -noonline "*" bkup
DB file /home/gtmnode1/gtmuser1/mumps.dat backed up in file bkup/mumps.dat
Transactions up to 0x000000000000F42C3 are backed up.
```

```
BACKUP COMPLETED.
```

This command creates a disk-to-disk backup copy of all regions of the current database in directory bkup. GT.M freezes all the regions during the backup operation.

Example:

```
$ mupip backup -bytestream -nettimeout=420 DEFAULT tcp://{org_host}:6200
```

This command creates a backup copy of the DEFAULT region with timeout of 420 seconds.

Example:

```
$ mupip backup -bytestream DEFAULT "" | gzip -c > online5pipe.inc.gz''
```

This command sends (via a pipe) the backup of the DEFAULT region to a gzip command.

Example:

```
$ mupip backup -online DEFAULT bkup
DB file /gtmnode1/gtmuser1/mumps.dat backed up in file bkup/mumps.dat
Transactions up to 0x00000000483F807C are backed up.
```

```
BACKUP COMPLETED.
```

This command creates a backup copy of the DEFAULT region of the current database in directory bkup. During the backup operation, other processes can read and update the database.

Example:

```
$ mupip backup -record DEFAULT bkup
```

This command sets a reference point and creates a backup copy of the DEFAULT region of the current database in directory bkup.

Example:

```
$ mupip backup -online -record DEFAULT bkup1921
DB file /home/reptiles/mumps.dat backed up in file bkup1921/mumps.dat
Transactions up to 0x00000000000F4351 are backed up.
```

Example:

```
$ mupip backup -bytestream -since=record DEFAULT bkup1921onwards
MUIPI backup of database file /home/reptiles/mumps.dat to bkup1921onwards/mumps.dat
DB file /home/reptiles/mumps.dat incrementally backed up in file bkup1921onwards/mumps.dat
6 blocks saved.
Transactions from 0x00000000000F4351 to 0x00000000000F4352 are backed up.
```

```
BACKUP COMPLETED.
```

The first command sets a reference point and creates a backup copy of the DEFAULT region of the current database in directory bkup1921. The second command completes a bytestream backup starting from the reference point set by the first command.

Example:

```
$ mupip backup -bytestream -transaction=1 DEFAULT bkup_dir
MUPIP backup of database file /gtmnode1/gtmuser1/mumps.dat to bkup_dir/mumps.dat
DB file /gtmnode1/gtmuser1/mumps.dat incrementally backed up in file bkup/mumps.dat
5 blocks saved.
Transactions from 0x0000000000000001 to 0x0000000000000003 are backed up.
```

```
BACKUP COMPLETED.
```

This command copies all in-use blocks of the DEFAULT region of the current database to directory `bkup_dir`.

Example:

```
$ mupip backup -newjnlfiles=noprevlink, sync_io "*" backupdir
```

This example creates new journal files for the current regions, cuts the previous journal file link for all regions in the global directory, enables the SYNC\_IO option and takes a backup of all databases in the directory `backupdir`.

## CREATE

Creates and initializes database files using the information in a Global Directory file. If a file already exists for any segment, MUPIP CREATE takes no action for that segment.

The format of the CREATE command is:

```
CR[EATE] [-R[EGION]=region-name]
```

The single optional `-REGION` qualifier specifies a region for which to create a database file.

Note that one GT.M database file grows to a maximum size of 224M (234,881,024) blocks. This means, for example, that with an 8KB block size, the maximum single database file size is 1,792GB (8KB\*224M). Note that this is the size of one database file -- a logical database (an M global variable namespace) can consist of an arbitrary number of database files.

### -Region

Specifies a single region for creation of a database file. By default, MUPIP CREATE creates database files for all regions in the current Global Directory that do not already have a database file.

The format of the REGION qualifier is:

```
-R[EGION]=region-name
```

## Examples for MUPIP CREATE

Example:

```
$ mupip create -region=REPTILES
```

This command creates the database file specified by the Global Directory (named by the GT.M Global Directory environment variable) for region REPTILES.

## DOWNGRADE

The MUPIP DOWNGRADE command changes the file header format to V4 or V5. The format of the MUPIP DOWNGRADE command is:

```
D[OWNGRADE] -V[ERSION]={V4|V5} file-name
```

For V4:

- It reduces the size from 8 bytes to 4 bytes for fields like current transaction (CTN), maximum tn (MTN) and others that contain transaction numbers.
- It removes the results of any prior DBCERTIFY run on the database.
- You cannot downgrade a V5 database which has standard null collation. In such a case, perform a MUPIP EXTRACT - FORMAT=ZWR operation on the V5 database and then perform a MUPIP LOAD operation on a V4 database.

**-VERSION={V4|V5}**

Specifies file header format. For more information on the downgrade criteria for your database, refer to the release notes document of your current GT.M version.

### Examples for MUPIP DOWNGRADE

Example:

```
$ mupip downgrade mumps.dat
```

This command changes the file-header of mumps.dat to V4 format.

## ENDIANCVT

Converts a database file from one endian format to the other (BIG to LITTLE or LITTLE to BIG). The format of the MUPIP ENDIANCVT command is:

```
ENDIANCVT [-OUTDB=<outdb-file>] -OV[ERRIDE] <db-file>
```

- <db-file> is the source database for endian conversion. By default ENDIANCVT converts <db-file> in place.
- outdb writes the converted output to <outdb-file>. In this case, ENDIANCVT does not modify the source database <db-file>.
- ENDIANCVT produces a <outdb-file> of exactly the same size as <db-file>.



### Important

Ensure adequate storage for <outdb-file> to complete the endian conversion successfully.

- ENDIANCVT requires standalone access to the database.
- GT.M displays a confirmation request with the “from” and “to” endian formats to perform the conversion. Conversion begins only upon receiving positive confirmation, which is a case insensitive "yes".

- In a multi-site replication configuration, the receiver server automatically detects the endian format of an incoming replication stream and converts it into the native endian format. See Database Replication chapter for more information.
- Encrypted database files converted with ENDIANCVT require the same key and the same cipher that were used to encrypt them.



### Note

GT.M on a big endian platform can convert a little endian database into big endian and vice versa; as can GT.M on a little endian platform. GT.M (run-time and utilities other than MUPIP ENDIANCVT) on a given endian platform opens and processes only those databases that are in the same endian format. An attempt to open a database of a format other than the native endian format produces an error.

### -Override

Enables MUPIP ENDIANCVT to continue operations even if GT.M encounters the following errors:

- "minor database format is not the current version"
- "kills in progress"
- "a GT.CM server is accessing the database"

Note that the OVERRIDE qualifier does not override critical errors (database integrity errors, and so on) that prevent a successful endian format conversion.

### Examples for MUPIP ENDIANCVT

```
$ mupip endiancvt mumps.dat -outdb=mumps_cvt.dat
Converting database file mumps.dat from LITTLE endian to BIG endian on a LITTLE endian system

Converting to new file mumps_cvt.dat

Proceed [yes/no] ?
```

This command detects the endian format of mumps.dat and converts it to the other endian format if you type yes to confirm.

### EXIT

Stops a MUPIP process and return control to the process from which MUPIP was invoked.

The format of the MUPIP EXIT command is:

```
EXI[T]
```

The EXIT command does not accept any qualifiers.

### EXTEND

Increases the size of a database file. By default, GT.M automatically extends a database file when there is available space.

The format of the MUPIP EXTEND command is:

```
EXTE[ND] [-BLOCKS=<data-blocks-to-add>] region-name
```

- The only qualifier for MUPIP EXTEND is BLOCKS.
- The required region-name parameter specifies the name of the region to expand.
- EXTEND uses the Global Directory to map the region to the dynamic segment and the segment to the file.

## -Blocks

Specifies the number of GDS database blocks by which MUPIP should extend the file. GDS files use additional blocks for bitmaps. MUPIP EXTEND adds the specified number of blocks plus the bitmap blocks required as overhead. For more information about bitmaps, refer to Chapter 9: “*GT.M Database Structure(GDS)*” (page 265). The format of the BLOCK qualifier is:

```
-BLOCKS=data-blocks-to-add
```

By default, EXTEND uses the extension value in the file header as the number of GDS blocks by which to extend the database file. You can specify as many blocks as needed as long as you are within the maximum total blocks limit (which could be as high as 224 million GDS blocks).

## Examples for MUPIP EXTEND

```
$ mupip extend DEFAULT -blocks=400
```

This command adds 400 GDE database block to region DEFAULT.

Example:

```
$ mupip extend REPTILES -blocks=100
```

This command adds 100 GDE database blocks to the region REPTILES.

## EXTRACT

Backups certain globals or to extract data from the database for use by another system. The MUPIP EXTRACT command copies globals from the current database to a sequential output file in one of three formats-GO, BINARY, or ZWR. The format of the EXTRACT command is:

```
EXTR[ACT]
[
  -FO[RMAT]={GO|B[INARY]|Z[WR]}
  -FR[EEZE]
  -LA[BEL]=text
  -[NO]L[OG]
  -S[ELECT]=global-name-list]
[
  {-ST[DOUT]|file-name}
```

- By default, MUPIP EXTRACT uses -FORMAT=ZWR.
- MUPIP EXTRACT uses the Global Directory to determine which database files to use.

## General Database Management

- MUPIP EXTRACT supports user collation routines. When used without the -FREEZE qualifier, EXTRACT may operate concurrently with normal GT.M database access.
- To ensure that MUPIP EXTRACT reflects a consistent application state, suspend the database updates to all regions involved in the extract, typically with the FREEZE qualifier, or backup the database with the ONLINE qualifier and extract files from the backup.
- EXTRACT places its output in the file defined by the file- name. EXTRACT may output to a UNIX file on any device that supports such files, including magnetic tapes.
- In UTF-8 mode, MUPIP EXTRACT write sequential output file in the UTF-8 character encoding. Ensure that MUPIP EXTRACT commands and corresponding MUPIP LOAD commands execute with the same setting for the environment variable gtm\_chset.



### Note

Magnetic tapes may have a smaller maximum file size than disks.

For information on extracting globals with the %GO utility, refer to "M Utility Routines" chapter of the *GT.M Programmer's Guide*. MUPIP EXTRACT is typically faster, but %GO can be customized.

The following sections describe the qualifiers of MUPIP EXTRACT command.

### -Format

Specifies the format of the output file. The format of the FORMAT qualifier is:

```
-FO[RMAT]=format_code
```

The format code is any one of the following:

1. B[INARY] - Binary format, used for database reorganization or short term backups. MUPIP EXTRACT -FORMAT=BINARY works much faster than MUPIP EXTRACT -FORMAT=GO and MUPIP EXTRACT -FORMAT=ZWR. Note: There is no defined standard to transport binary data from one GT.M implementation to another. Further, FIS reserves the right to modify the binary format in new versions. The first record of a BINARY format data file contains the header label. The header label is 87 characters long. The following table illustrates the components of the header label.

BINARY Format Data File Header Label	
CHARACTERS	EXPLANATION
1-26	Fixed-length ASCII text: "GDS BINARY EXTRACT LEVEL 4".
27-40	Date and time of extract in the \$ZDATE() format: "YEARMMDD2460SS".
41-45	Decimal maximum block size of the union of each region from which data was extracted.
46-50	Decimal maximum record size of the union of each region from which data was extracted.
51-55	Decimal maximum key size of the union of each region from which data was extracted.
56-87	Space-padded label specified by the -LABEL qualifier. For extracts in UTF-8 mode, GT.M prefixes UTF-8 and a space to -LABEL. The default LABEL is "GT.M MUPIP EXTRACT"

## General Database Management

2. GO - Global Output format, used for files to transport or archive. -FORMAT=GO stores the data in record pairs. Each global node produces one record for the key and one for the data. MUPIP EXTRACT -FORMAT=GO has two header records - the first is a test label (refer to the LABEL qualifier) and the second contains a data, and time.
3. ZWR - ZWRITE format, used for files to transport or archive that may contain non-graphical information. Each global node produces one record with both key and data. MUPIP EXTRACT -FORMAT=ZWR has two header records, which are the same as for FORMAT=GO, except that the second record ends with the text " ZWR"

### -FReeze

Prevents database updates to all database files from which the MUPIP EXTRACT command is copying records. FREEZE ensures that a MUPIP EXTRACT operation captures a "sharp" image of the globals, rather than one "blurred" by updates occurring while the copy is in progress.

The format of the FREEZE qualifier is:

```
-FR[EEZE]
```

By default, MUPIP EXTRACT does not "freeze" regions during operation.

### -LAbel

Specifies the text string that becomes the first record in the output file. MUPIP EXTRACT -FORMAT=BINARY truncates the label text to 32 characters. The format of the LABEL qualifier is:

```
-LA[BEL]=text
```

- By default, EXTRACT uses the label "GT.M MUPIP EXTRACT."
- For more detailed information about the -FORMAT=BINARY header label, refer to the description of EXTRACT -FORMAT=BINARY.

### -LOg

Displays a message on stdout for each global extracted with the MUPIP EXTRACT command. The message displays the number of global nodes, the maximum subscript length and maximum data length for each global. The format of the LOG qualifier is:

```
-[NO]LO[G]
```

By default, EXTRACT operates -LOG.

### -Select

Specifies globals for a MUPIP EXTRACT operation. The format of the SELECT qualifier is:

```
-S[ELECT]= global-specification
```

- By default, EXTRACT selects all globals, as if it had the qualifier -SELECT=\*
- The caret symbol (^) in the specification of the global name is optional.

The global-specification can be:

## General Database Management

- A parenthetical list, such as (a,B,C). In this case, MUPIP EXTRACT selects all globals except ^a, ^B, and ^C.
- A global name, such as MEF. In this case, MUPIP EXTRACT selects only global ^MEF.
- A range of global names, such as A7:B6. In this case, MUPIP EXTRACT selects all global names between ^A7 and ^B6, inclusive.
- A list, such as A,B,C. In this case, MUPIP EXTRACT selects globals ^A, ^B, and ^C.
- Global names with the same prefix, such as PIGEON\*. In this case, EXTRACT selects all global names from ^PIGEON through ^PIGEONzzzzz.



### Note

If the rules for selection are complex, it may be easier to construct an ad hoc Global Directory that maps the global variables to be extracted to the database file. This may not be permissible if the database file is part of a replicated instance. If this is the case, work with a backup of the database.

## -STdout

Redirects database extract to the standard output stream. The format of the STDOUT qualifier is:

```
-ST[DOUT]
```

## Examples for MUPIP EXTRACT

Example:

```
$ mupip extract -format=go -freeze big.glo
```

This command prevents database updates during a MUPIP EXTRACT operation.

Example:

```
$ mupip extract -format=GO mumps_i.go
```

This command creates an extract file called mumps\_i.go in "Global Output" format. Use this format to transport or archive files. The first record of a GO format file contains the header label, "GT.M MUPIP EXTRACT," as text.

Example:

```
$ mupip extract -format=BINARY v5.bin
```

This command creates an extract file called v5.bin in Binary format. Use this format for reorganizing a database or for short-term backups.

Example:

```
$ mupip extract -format=ZWR -LABEL=My_Label My_Extract_File
```

This example extracts all globals from the current database to file My\_Extract\_File (in ZWRITE format) with label My\_Label.

Example:

```
$ mupip extract -nolog FL.GLO
```

## General Database Management

This command creates a global output file, FL.GLO, (which consists of all global variables in the database) without displaying statistics on a global-by-global basis. As there is no label specified, the first record in FL.GLO contains the text string "GT.M MUPIP EXTRACT."

Example:

```
$ mupip extract -select=Tyrannosaurus /dev/tty
```

This command instructs EXTRACT to dump the global ^Tyrannosaurus to the device (file-name) /dev/tty.

## FREEZE

Temporarily suspends (freezes) updates to the database. If you prefer a non-GT.M utility to perform a backup or reorganization, you might use this facility to provide standalone access to your GT.M database. You might use MUPIP FREEZE to suspend (and later resume) database updates for creating mirrored disk configuration or re-integrating a mirror.

GT.M BACKUP, INTEG, and REORG operations may implicitly freeze and unfreeze database regions. However, for most operations, this freeze/unfreeze happens internally and is transparent to the application.

The format of the MUPIP FREEZE command is:

```
F[REEZE] {-OF[F] [-OV[ERRIDE]]|-ON [-R[ECORD]]} region-list
```

- The region-list identifies the target of the FREEZE.
- MUPIP FREEZE waits for one minute so that concurrent KILL or MUPIP REORG operations can complete. If the KILL or MUPIP REORG commands do not complete within one minute, MUPIP FREEZE terminates operations and unfreezes any regions it had previously marked as frozen.
- To ensure that a copy or reorganized version of a database file contains a consistent set of records, concurrent MUPIP utilities, such as BACKUP (without the ONLINE qualifier) and EXTRACT, include mechanisms to ensure that the database does not change while the MUPIP utility is performing an action. FIS recommends the use of the -ONLINE qualifier with BACKUP.
- A MUPIP FREEZE can be removed only by the user who sets the FREEZE or by using -OVERRIDE.
- After MUPIP FREEZE -ON, processes that are attempting updates "hang" until the FREEZE is removed by the MUPIP FREEZE -OFF command or DSE. Make sure that procedures for using MUPIP FREEZE, whether manual or automated, include provisions for removing the FREEZE in all appropriate cases, including when errors disrupt the normal flow.
- A -RECOVER/-ROLLBACK for a database reverts to a prior database update state. Therefore, a -RECOVER/-ROLLBACK immediately after a MUPIP FREEZE -ON removes the freeze. However, -RECOVER/-ROLLBACK does not succeed if there are processes attached (for example when a process attempt a database update immediately after a MUPP FREEZE -ON) to the database.

FREEZE must include one of the qualifiers:

```
-OF[F]  
-ON
```

The optional qualifiers are:

```
-OV[ERRIDE]  
-R[ECORD]
```

## **-OFF**

Clears a freeze set by another process with the same userid.

The format of the OFF qualifier is:

```
OF[F]
```

- When used with `-OVERRIDE`, `-OFF` stops a freeze operation set by a process with a different userid.
- Incompatible with: `-ON`, `-RECORD`

## **-ON**

Specifies the start of a MUPIP FREEZE operation. The format of the ON qualifier is:

```
-ON
```

Incompatible with: `-OFF`, `-OVERRIDE`

## **-OVERRIDE**

Release a freeze set by a process with a different userid. GT.M provides `OVERRIDE` to allow error recovery in case a procedure with a freeze fails to release. The format of the `OVERRIDE` qualifier is:

```
-OV[ERRIDE]
```

- `OVERRIDE` should not be necessary (and may even be dangerous) in most schemes.
- Incompatible with: `-ON`, `-RECORD`

## **-Record**

Specifies that a MUPIP FREEZE operation should record an event as a reference point. You might use MUPIP FREEZE to set up your database for a custom-backup mechanism (SAN or mirror-based).

The format of the `RECORD` qualifier is:

```
-R[ECORD]
```

- You might use `-RECORD` to integrate MUPIP BACKUP `-BYTESTREAM` with an external backup mechanism.
- `-RECORD` replaces the previously `RECORDed` transaction identifier for the database file.
- Incompatible with: `-OFF` and `-OVERRIDE`.

## **Examples for MUPIP FREEZE**

Example:

```
$ mupip freeze -off DEFAULT
```

This command stops an ongoing MUPIP FREEZE operation on the region `DEFAULT`.

Example:

```
$ mupip freeze -on "*"
```

This command prevents updates to all regions in the current Global Directory.

Example:

```
$ set +e
$ mupip freeze -on -record "*"
$ tar cvf /dev/tape /prod/app/*.dat
$ mupip freeze -off
$ set -e
```

The set +e command instructs the shell to attempt all commands in the sequence, regardless of errors encountered by any command. This ensures that the freeze -off is processed even if the tar command fails. FREEZE prevents updates to all database files identified by the current Global Directory. The -record qualifier specifies that the current transaction in each database be stored in the RECORD portion of the database file header. The tar command creates a tape archive file on the device /dev/tape, containing all the files from /prod/app that have an extension of .dat. Presumably all database files in the current Global Directory are stored in that directory, with that extension. The second FREEZE command re-enables updates that were suspended by the first FREEZE. The set -e command re-enables normal error handling by the shell.

Example:

```
$ mupip freeze -override -off DEFAULT
```

This command unfreezes the DEFAULT region even if the freeze was set by a process with a different userid.

## FTOK

Produces the "public" (system generated) IPC Keys (essentially hash values) of a given file.

The format of the MUPIP FTOK command is:

```
FT[OK] [-DB] [-JNLPOOL] [-RECVPOOL] file-name
```

### -DB

Specifies that the file-name is a database file. By default, MUPIP FTOK uses -DB.

### -JNLPOOL

Specifies that the reported key is for the Journal Pool of the instance created by the current Global Directory.

### -RECVPOOL

Specifies that the reported key is for the Receive Pool of the instance created by the current Global Directory.

## INTEG

Performs an integrity check on a GT.M database file. You can perform structural integrity checks on one or more regions in the current Global Directory without bringing down (suspending database updates) your application. However, a MUPIP INTEG on

## General Database Management

a single file database requires standalone access but does not need a Global Directory. The order in which the MUPIP INTEG command selects database regions is a function of file system layout and may vary as files are moved or created. Execute a MUPIP INTEG operations one database file at a time to generate an report where the output always lists database files in a predictable sequence. For example, to compare output with a reference file, run INTEG on one file at a time.

Always use MUPIP INTEG in the following conditions:

- Periodically - to ensure ongoing integrity of the database(s); regular INTEGs help detect any integrity problems before they spread and extensively damage the database file.
- After a crash - to ensure the database was not corrupted. (Note: When using before-image journaling, when the database is recovered from the journal file after a crash, an integ is not required).
- When database errors are reported - to troubleshoot the problem.

Improving the logical and physical adjacency of global nodes may result in faster disk I/O. A global node is logically adjacent when it is stored within a span of contiguous serial block numbers. A global node is physically adjacent when it resides on adjacent hard disk sectors in a way that a single seek operation can access it. Database updates (SETs/KILLs) over time affect the logical adjacency of global nodes. A MUPIP INTEG reports the logical adjacency of your global nodes which may indicate whether a MUPIP REORG could improve the database performance. A native file system defragmentation improves physical adjacency.



### Note

Most modern SAN and I/O devices often mask the performance impact of the adjustments in logical and physical adjacency. If achieving a particular performance benchmark is your goal, increasing the logical and physical adjacency should be only one of many steps that you might undertake. While designing the database, try to ensure that the logical adjacency is close to the number of blocks that can physically reside on your hard disk's cylinder. You can also choose two or three cylinders, with the assumption that short seeks are fast.

The format of the INTEG command is:

```
I[INTEG]
[
  -A[DJACENCY]=integer
  -BL[OCK]=hexa;block-number
  -BR[IEF]
  -FA[ST]
  -FU[LL]
  -[NO]K[EYRANGES]
  -[NO]MAP[=integer]
  -[NO]MAXK[EYSIZE][=integer]
  -S[UBSCRIPT]=subscript]
  -TN[_RESET]
  -[NO]TR[ANSACTION][=integer]
  -[NO]O[NLINE]
]
{[-FILE] file-name|-REG[ION] region-name}
```

- MUPIP INTEG requires specification of either file(s) or region(s).
- Press <CTRL-C> to stop MUPIP INTEG before the process completes.

- The file-name identifies the database file for a MUPIP INTEG operation. The region-list identifies one or more regions that, in turn, identify database files through the current Global Directory.
- MUPIP INTEG operation keeps track of the number of blocks that do not have the current block version during a non-fast integ (default or full) and matches this value against the blocks to upgrade counter in the file-header. It issues an error if the values are unmatched and corrects the count in the file header if there are no other integrity errors.



### Important

Promptly analyze and fix all errors that MUPIP INTEG reports. Some errors may be benign while others may be a sign of corruption or compromised database integrity. If operations continue without fixes to serious errors, the following problems may occur:

- Invalid application operation due to missing or incorrect data.
- Process errors, including inappropriate indefinite looping, when a database access encounters an error.
- Degrading application level consistency as a result of incomplete update sequences caused by the prior symptoms.

FIS strongly recommends fixing the following errors as soon as they are discovered:

- Blocks incorrectly marked free - these may cause accelerating damage when processes make updates to any part of the database region.
- Integrity errors in an index block - these may cause accelerating damage when processes make updates to that area of the database region using the faulty index. For more information, refer to Chapter 11: “*Maintaining Database Integrity*” (page 324).

MUPIP INTEG -FAST and the “regular” INTEG both report these errors (These qualifiers are described later in this section). Other database errors do not pose the threat of rapidly spreading problems in GDS files. After the GT.M database repair, assess the type of damage, the risk of continued operations, and the disruption in normal operation caused by the time spent repairing the database. For information on analyzing and correcting database errors, refer to Chapter 11: “*Maintaining Database Integrity*” (page 324). Contact your GT.M support channel for help assessing INTEG errors.

The following sections describe the qualifiers of the INTEG command.

### -ADjacency

Specifies the logical adjacency of data blocks that MUPIP INTEG should assume while diagnosing the database. By default, MUPIP INTEG operates with -ADJACENCY=10 and reports the logical adjacency in the “Adjacent” column of the MUPIP INTEG report.

- The complexity of contemporary disk controllers and the native file system may render this report superfluous. But when it is meaningful, this report measures the logical adjacency of data.
- A MUPIP REORG improves logical adjacency and a native file system defragmentation improves physical adjacency.

The format of the ADJACENCY qualifier is:

```
-AD[JACENCY]=integer
```

## **-Block**

Specifies the block for MUPIP INTEG command to start checking a sub-tree of the database. MUPIP INTEG -BLOCK cannot detect "incorrectly marked busy errors"

The format of the BLOCK qualifier is:

```
-BL[OCK]=block-number
```

- Block numbers are displayed in an INTEG error report or by using DSE.
- Incompatible with: -SUBSCRIPT and -TN\_RESET

## **-Brief**

Displays a single summary report by database file of the total number of directory, index and data blocks. The format of the BRIEF qualifier is:

```
-BR[IEF]
```

- By default, MUPIP INTEG uses the BRIEF qualifier.
- Incompatible with: -FULL

## **-FAst**

Checks only index blocks. FAST does not check data blocks.

The format of the FAST qualifier is:

```
-FA[ST]
```

- -FAST produces results significantly faster than a full INTEG because the majority of blocks in a typical database are data blocks.
- While INTEG -FAST is not a replacement for a full INTEG, it very quickly detects those errors that must be repaired immediately to prevent accelerated database damage.
- By default, INTEG checks all active index and data blocks in the database.
- Incompatible with: -TN\_RESET.

## **-File**

Specifies the name of the database file for the MUPIP INTEG operation. FILE requires exclusive (stand-alone) access to a database file and does not require a Global Directory. The format of the FILE qualifier is:

```
-FI[LE]
```

- With stand-alone access to the file, MUPIP INTEG -FILE is able to check whether the reference count is zero. A non-zero reference count indicates prior abnormal termination of the database.
- The -FILE qualifier is incompatible with the -REGION qualifier.
- By default, INTEG operates on -FILE.

## -FULL

Displays an expanded report for a MUPIP INTEG operation. With -FULL specified, MUPIP INTEG displays the number of index and data blocks in the directory tree and in each global variable tree as well as the total number of directory, index and data blocks. The format of the FULL qualifier is:

```
-FU[LL]
```

- The -FULL qualifier is incompatible with the -BRIEF qualifier.
- By default, INTEG reports are -BRIEF.
- Use -FULL to have INTEG report all global names in a region or list of regions.

## -Keyranges

Specify whether the MUPIP INTEG report includes key ranges that identify the data suspected of problems it detects. The format of the KEYRANGES qualifier is:

```
-[NO]K[EYRANGES]
```

By default, INTEG displays -KEYRANGES.

## -MAP

Specifies the maximum number of "incorrectly marked busy errors" that MUPIP INTEG reports. The format of the MAP qualifier is:

```
-[NO]MAP[=max_imb_errors]
```

- <max\_imb\_errors> specifies the threshold limit for the number of incorrectly marked busy errors.
- -NOMAP automatically sets a high threshold limit of 1000000 (1 million) incorrectly marked busy errors (-MAP=1000000).
- By default, INTEG reports a maximum of 10 map errors (-MAP=10).



### Note

MUPIP INTEG reports "incorrectly marked free" errors as they require prompt action. MAP does not restrict them.

An error in an index block prevents INTEG from processing potentially large areas of the database. A single "primary" error may cause large numbers of "secondary" incorrectly marked busy errors, which are actually useful in identifying valid blocks that have no valid index pointer. Because "real" or primary incorrectly marked busy errors only make "empty" blocks unavailable to the system, they are low impact and do not require immediate repair.



### Note

After a database recovery with -RECOVER (for example, using -BEFORE\_TIME) or -ROLLBACK (for example, using -FETCHRESYNC), the database may contain incorrectly marked busy errors. Although these errors are benign, they consume available space. Schedule repairs on the next opportunity.

**-MAXkeysize**

Specifies the maximum number of "key size too large" errors that a MUPIP INTEG operation reports. The format of the MAXKEYSIZE qualifier is:

```
-[NO]MAX[KEYSIZE][=integer]
```

- By default, INTEG reports a maximum of 10 key size errors (-MAXKEYSIZE=10).
- -NOMAXKEYSIZE removes limits on key size reporting so that INTEG reports all key size too large errors.
- -NOMAXKEYSIZE does not accept assignment of an argument.
- "Key size too large" errors normally occur only if a DSE CHANGE -FILEHEADER -KEY\_MAX\_SIZE command reduces the maximum key size.

**-Online**

Specifies that while a MUPIP INTEG operation is active, other processes can update the database without affecting the result of the backup. Allows checking database structural integrity to run concurrently with database updates. The format of the ONLINE qualifier is:

```
-[NO]O[NLINE]
```

- -NOONLINE specifies that the database should be frozen during MUPIP INTEG.
- By default, MUPIP INTEG is online except for databases containing V4 blocks for which the default is -NOONLINE. Note that databases containing V4 blocks should exist only in databases that are in the process of being migrated from V4 to V5; please complete your migration to the V5 format before using MUPIP INTEG -ONLINE.
- Since MUPIP INTEG -ONLINE does not freeze database updates, it cannot safely correct errors in the "blocks to upgrade" and "free blocks" fields in the file header, while MUPIP INTEG -NOONLINE can correct these fields.
- As it checks each database file, MUPIP INTEG -ONLINE creates a sparse file of the same size as the database. As each GT.M process updates the database, it places a copy of the old block in the sparse file before updating the database. For any database blocks with a newer transaction number than the start of the INTEG, MUPIP uses the copy in the sparse file. Thus, analogous with MUPIP BACKUP -ONLINE, INTEG reports on the state of the database as of when it starts, not as of when it completes. Note: a command such as ls -l command shows sparse files at their full size, but does not show actual disk usage. Use a command such as du -sh to see actual disk usage.
- The environment variable GTM\_BAKTMPDIR (unlike most GT.M environment variables, this is upper case; it is the same directory used for MUPIP BACKUP -ONLINE) can be used to indicate a directory where MUPIP should place the snapshot files (used by MUPIP INTEG -ONLINE). If GTM\_BAKTMPDIR does not exist, MUPIP places the snapshot files in the current directory at the time you issue the INTEG command. MUPIP and GT.M processes automatically cleans up these temporary snapshot files under a wider variety of conditions.
- Temporary directory security settings must allow write access by the MUPIP process and read access by all processes updating the database. MUPIP creates the temporary file with the same access as the database file so processes updating the database can write to the temporary file. If the database is encrypted, the updating processes write encrypted blocks to the snapshot file and the MUPIP INTEG process must start with access to appropriate key information as it does even -NOONLINE.

## General Database Management

- MUPIP INTEG -NOONLINE [-FAST] {-REGION|-FILE} clears the KILLs in progress and Abandoned Kills flags if the run includes the entire database and there are no incorrectly marked busy blocks.
- Only one online integ can be active per database region. If an online integ is already active, a subsequent one issues an error and immediately terminates. If an online integ does not successfully complete, GT.M cleans it up in one of the following ways:
  1. A subsequent online integ detects that an earlier one did not successfully complete and releases the resources held by the prior online integ before proceeding.
  2. If a MUPIP STOP was issued to the online integ process, the process cleans up any resources it held. Note: since the process was stopped the results of the integ may not be valid.
  3. subsequent MUPIP RUNDOWN ensures the release of resources held by prior unsuccessful online integs for the specified regions.
  4. For every 64K transactions after the online integ initiation, online integ checks GT.M health for improperly abandoned online integs and releases resources held by any it finds.
- Incompatible with: -FILE, -TN\_RESET (there should be no need to use -TN\_RESET on a GT.M V5 database).

### -Region

Specifies that the INTEG parameter identifies one or more regions rather than a database file. The format of the REGION qualifier is:

```
-R[EGION]
```

- MUPIP INTEG -REGION does not require stand alone access to databases. Instead, it freezes updates (but not reads) to the database during the check. The region-list argument may specify more than one region of the current Global Directory in a list separated with commas. INTEG -REGION requires the environment variable gtmgbldir to specify a valid Global Directory. For more information on defining gtmgbldir, refer to the "Global Directory Editor" chapter.
- Because a KILL may briefly defer marking the blocks it releases "free" in the bit maps, INTEG -REGION may report spurious block incorrectly marked busy errors. These errors are benign. If these errors occur in conjunction with a "Kill in progress" error, resolve the errors after the "Kill in progress" error is no longer present.
- By default, INTEG operates -FILE.
- Incompatible with: -FILE, -TN\_RESET

### -Subscript

Specifies a global or a range of keys to INTEG. The global key may be enclosed in quotation marks (" "). Identify a range by separating two subscripts with a colon (:). -SUBSCRIPT cannot detect incorrectly marked busy errors. The format of the SUBSCRIPT qualifier is:

```
-S[UBSCRIPT]=subscript
```

Specify SUBSCRIPT only if the path to the keys in the subscript is not damaged. If the path is questionable or known to be damaged, use DSE to find the block(s) and INTEG -BLOCK.

Incompatible with: -BLOCK, -TN\_RESET

## -TN\_reset

Resets block transaction numbers and backup event recorded transaction numbers to (one) 1, and the current transaction number to two (2) which makes the backup event recorded transaction numbers more meaningful and useful. It also issues an advisory message to perform a backup.

The format of the TN\_RESET qualifier is:

```
-TN[_RESET]
```

- Transaction numbers can go up to 18,446,744,073,709,551,615. This means that a transaction processing application that runs flat out at a non-stop rate of 1,000,000 updates per second would need a TN reset approximately every 584,554 years.
- The -TN\_RESET qualifier rewrites all blocks holding data. If the transaction overflow resets to zero (0) database operation is disrupted.
- The -TN\_RESET qualifier is a protective mechanism that prevents the transaction overflow from resetting to 0.
- By default, INTEG does not modify the block transaction numbers.



### Important

There should never be a need for a -TN\_RESET on a database with only V5 blocks, even when cleaning up after a runaway process.

- The -TN\_RESET qualifier is incompatible with the -FAST, -BLOCK, -REGION, and -SUBSCRIPT qualifiers.



### Note

Any time a GT.M update opens a database file that was not properly closed, GT.M increments the transaction number by 1000. This automatic increment prevents problems induced by abnormal database closes, but users must always consider this factor in their operational procedures. The rate at which GT.M "uses up" transaction numbers is a function of operational procedures and real database updates.

## -TRANSACTION

Specifies the maximum number of block transaction- number-too-large errors that MUPIP INTEG reports. The format of the TRANSACTION qualifier is:

```
-[NO]TR[ANSACTION][=integer]
```

- -NOTRANSACTION removes limits on transaction reporting so MUPIP INTEG reports all transaction number errors.
- -NOTRANSACTION does not accept assignment of an argument.
- A system crash may generate many "block transaction number too large" errors. These errors can cause problems for BACKUP -INCREMENTAL and for transaction processing. Normally, the automatic increment of 1000 blocks that GT.M adds when a database is reopened averts these errors. If a problem still exists after the database is reopened, users can use a value in the DSE CHANGE -FILEHEADER -CURRENT\_TN= command to quickly fix "block transaction number too large number" errors.

- By default, INTEG reports a maximum of 10 block transaction errors (-TRANSACTION=10).

## Examples for MUPIP INTEG

Example:

```
$ mupip integ -block=4 mumps.dat
```

This command performs a MUPIP INTEG operation on the BLOCK 4 of mumps.dat.

Example:

```
$ mupip integ -adjacency=20
```

A sample output from the above command follows:

Type	Blocks	Records	% Used	Adjacent
Directory	2	5	4.150	NA
Index	18	1151	77.018	1
Data	1137	94189	97.894	1030
Free	43	NA	NA	NA
Total	1200	95345	NA	1031

This example performs a MUPIP INTEG operation assuming that logically related data occupies 20 data blocks in the current database. The sample output shows that out of 1137 data blocks, 1030 data blocks are adjacent to each other. One can improve the performance of a database if the all blocks are as adjacent as possible.

Example:

```
$ mupip integ -brief mumps.dat
```

This command performs a MUPIP INTEG operation on the database mumps.dat. A sample output from the above command follows:

No errors detected by integ.

Type	Blocks	Records	% Used	Adjacent
Directory	2	2	2.490	NA
Index	1	1	2.343	1
Data	1	3	6.738	1
Free	96	NA	NA	NA
Total	100	6	NA	2

Example:

```
$ mupip integ -fast mumps.dat
```

This command performs a MUPIP INTEG operation only on the index block of the database file mumps.dat. A sample output from the above command follows:

No errors detected by fast integ.

Type	Blocks	Records	% Used	Adjacent
------	--------	---------	--------	----------

## General Database Management

Directory	2	2	2.490	NA
Index	1	1	2.343	1
Data	1	NA	NA	NA
Free	96	NA	NA	NA
Total	100	NA	NA	1

Note the NA entries (highlighted in bold) for Data type. It means that the MUPIP INTEG -FAST operation checked only index blocks.

```
$ mupip integ -full mumps.dat
```

The sample output from the above command follows:

Directory tree

Level	Blocks	Records	% Used	Adjacent
1	1	1	2.343	NA
0	1	1	2.636	NA

Global variable ^Dinosaur

Level	Blocks	Records	% Used	Adjacent
1	1	6	8.398	1
0	6	500	83.902	6

No errors detected by integ.

Type	Blocks	Records	% Used	Adjacent
Directory	2	2	2.490	NA
Index	1	6	8.398	1
Data	6	500	83.902	6
Free	91	NA	NA	NA
Total	100	508	NA	7

Example:

```
$ mupip integ -map=20 -maxkeysize=20 -transaction=2 mumps.dat
```

This command performs a MUPIP INTEG operation and restricts the maximum number of "key size too large" errors to 20.

Example:

```
$ mupip integ -map=20 -transaction=2 mumps.dat
```

This command performs a MUPIP INTEG operation and restricts the maximum number of "block transaction- number-too-large errors" to 2.

```
$ mupip integ -file mumps.dat -tn_reset
```

This command resets the transaction number to one in every database block.

Example:

```
$ mupip integ -subscript="^Parrots" mumps.dat
```

This example performs a MUPIP INTEG operation on the global variable ^Parrots in the database file mumps.dat.

Example:

```
$ mupip integ -subscript="^Amsterdam(100)": "^Bolivia("Chimes")" -region DEFAULT
```

This example performs a MUPIP INTEG operation all global variables greater than or equal to ^Amsterdam (100) and less than or equal to ^Bolivia("Chimes") in the default region(s).



### Note

To specify a literal in the command string, use two double quotation marks for example, ^b("c").

## INTRPT

Sends an interrupt signal to the specified process. The signal used is [POSIX] SIGUSR1. The format of the MUPIP INTRPT command is:

```
INTRPT process-id
```



### Important

Ensure that signal SIGUSR1 is not be used by any C external function calls or any (initially non-GT.M) processes that use call-in support, as it is interpreted by GT.M as a signal to trigger the \$ZINTERRUPT mechanism.

- To INTRPT a process belonging to its own account, a process requires no UNIX privilege.
- To INTRPT a process belonging to its own GROUP, a process requires UNIX membership in the user group of the target process privilege. To INTRPT a process belonging to an account outside its own GROUP, a process requires UNIX superuser privilege.

## JOURNAL

Analyzes, extracts, reports, and recovers data using journal files. For a description of the JOURNAL command, refer to Chapter 6: “*GT.M Journaling*” (page 130).

## LOAD

Puts the global variable names and their corresponding data values into a GT.M database from a sequential file.

The format of the LOAD command is:

```
L[OAD]
[
  -BE[GIN]=integer
  -E[ND]=integer
  -FI[LLFACTOR]=integer
  -FO[RMAT]={GO|B[INARY]|Z[WR]}
  -S[TDIN]
]
```

file-name



## Caution

From an application perspective, performing a MUPIP LOAD operation while an application is running may result in an inconsistent application state for the database.

- MUPIP LOAD uses the Global Directory to determine which database files to use.
- LOAD supports user collation routines.
- LOAD takes its input from the file defined by the file-name
- LOAD may take its input from a UNIX file on any device that supports such files.
- MUPIP LOAD command considers a sequential file as encoded in UTF-8 if the environment variable gtm\_chset is set to UTF-8. Ensure that MUPIP EXTRACT commands and corresponding MUPIP LOAD commands execute with the same setting for the environment variable gtm\_chset.
- For information on loading with an M "percent utility," refer to the %GI section of the "M Utility Routines" chapter in *GT.M Programmer's Guide*. LOAD is typically faster, but the %GI utility can be customized.
- Press <CTRL-C> to produce a status message from LOAD. Entering <CTRL-C> twice in quick succession stops LOAD. A LOAD that is manually stopped or stops because of an internal error is incomplete and may lack application level integrity, but will not adversely affect the database structure unless terminated with a kill -9.



## Note

The MUPIP EXTRACT or MUPIP LOAD procedure for large databases are time consuming due to the volume of data that has to be converted from binary to ZWR format (on source) and vice versa (on target). One must also consider the fact that the extract file can be very large for a large database. Users must ensure adequate storage support the size of the extract file and the space occupied by the source and target databases. In order to reduce the total time and space it takes to transfer database content from one endian platform to another, it is efficient to convert the endian format in-place for a database and transfer the converted database. See MUPIP ENDIANCVT for more information on converting the endian format of a database file.

The following sections describe the optional qualifiers of the MUPIP LOAD command.

## -FOrmat

Specifies the format of the input file. The format must match the actual format of the input file for LOAD to operate.

The format codes are:

GO - Global Output format  
 B[INARY] - Binary format  
 Z[WR] - ZWRITE format

- By default, LOAD uses FORMAT=ZWR.
- -FORMAT=GO expects the data in record pairs. Each global node requires one record for the key and one for the data.

- -FORMAT=ZWR expects the data for each global node in a single record.
- -FORMAT=BINARY only applies to Greystone Database Structure (GDS) files. A BINARY format file loads significantly faster than a GO or ZWR format file. -FORMAT=BINARY works with data in a proprietary format. -FORMAT=BINARY has one header record, therefore LOAD -FORMAT=BINARY starts active work with record number two (2).

## -Bgin

Specifies the record number of the input file with which LOAD should begin. Directing LOAD to begin at a point other than the beginning of a valid key causes an error. The format of the BEGIN qualifier is:

```
-BE[GIN]=integer
```



### Important

Always consider the number of header records for choosing a -BEGIN point. See FORMAT qualifier for more information.

- For -FORMAT=GO input, the value is usually an odd number. As -FORMAT=BINARY requires important information from the header, this type of load requires an intact file header regardless of the -BEGIN value.
- For -FORMAT = ZWR input, each record contains a complete set of reference and data information. The beginning values are not restricted, except to allow two records for the header.
- By default, LOAD starts at the beginning of the input file.

## -End

Specifies the record number of the input file at which LOAD should stop. -END=integer must be greater than the -BEGIN=integer for LOAD to operate. LOAD terminates after processing the record of the number specified by -END or reaching the end of the input file. The format of the END qualifier is:

```
-E[ND]=integer
```

The value of -FORMAT=GO input should normally be an even number. By default, LOAD continues to the end of the input file.

## -Fill\_factor

Specifies the quantity of data stored in a database block. Subsequent run-time updates to the block fill the remaining available space reserved by the FILL\_FACTOR. Blocks that avoid block splits operate more efficiently. The format of the FILL\_FACTOR qualifier is:

```
-FI[LL_FACTOR]=integer
```

- Reserves room and avoid unnecessary block split to accommodate the forecasted growth in a global variable that may experience significant rate of additions over a period.
- Users having database performance issues or a high rate of database updates must examine the defined FILL\_FACTORS. Unless the application only uses uniform records, which is not typical for most applications, FILL\_FACTORS do not work precisely.

- By default, LOAD uses `-FILL_FACTOR=100` for maximum data density.



## Note

`FILL_FACTOR` is useful when updates add or grow records reasonably uniformly across a broad key range. If updates are at ever ascending or descending keys, or if the record set and record sizes are relatively static in the face of updates, `FILL_FACTOR` won't provide much benefit.

## -Stdin

Specifies that MUPIP LOAD takes input from standard input (stdin). The format of the STDIN qualifier is:

```
-S[TDIN]
```

## Examples for MUPIP LOAD

Example:

```
$ mupip load ex_file.go
```

This command loads the content of the extract file `ex_file.go` to the current database.

Example:

```
$ mupip load -format=go big.glo
```

This command loads an extract file `big.glo` in the current database.

Example:

```
$ mupip load -begin=5 -end=10 rs.glo
```

This command begins MUPIP LOAD operation from record number 5 and ends at record number 10. Note that the value for `BEGIN` is an odd number. A sample output from the above command follows:

```
GT.M MUPIP EXTRACT
02-MAR-2011 18:25:47 ZWR
Beginning LOAD at record number: 5

LOAD TOTAL  Key Cnt: 6  Max Subsc Len: 7  Max Data Len: 1
Last LOAD record number: 10
```

Example:

```
$ mupip load -fill_factor=5 reobs.glo
```

This command set the `FILL_FACTOR` to 5 for loading an extract file in the current database.

Example:

```
$cat big.glo | mupip load -stdin
$mupip load -stdin < big.glo
```

These commands loads the extract file big.glo using -stdin.

## REORG

Improves database performance by defragmenting and reorganizing database files and attempts to reduce the size of the database file. MUPIP REORG runs concurrently with other database activity, including updates. Competing activity generally increases the time required to perform a REORG, as well as that of the competing operations.

The format of the REORG command is:

```
REO[RG]
[
  -D[OWNGRADE]
  -E[XCLUDE]=global-name-list
  -FI[LL_FACTOR]=integer
  -I[NDEX_FILL_FACTOR]=integer
  -R[ESUME]
  -S[ELECT]=global-name-list
  -T[RUNCATE][=percentage]
  -UP[GRADE]
  -REG[ION]=region-list
]
```



### Note

While REORG optimizes the GDS structure of database files, it does not handle native file system file fragmentation. In most cases, fragmentation at the native file system level is more likely than fragmentation at the GDS structure level. Therefore, FIS recommends users create files with appropriate allocations and extensions, on disks with large amounts of contiguous free space. Use native utilities and MUPIP utilities (depending on operational procedures) to eliminate file fragmentation when database files have been extended more than a dozen times.

- Using REORG concurrently with normal database access affects the performance of normal operation. To reduce this impact, lower the priority of the process performing the REORG.
- MUPIP REORG does not change the logical contents of the database, and can run on either the originating instance or replicating instance of an LMS application. In such cases, resuming REORGs in process should be part of the batch restart. See "GT.M Database Replication" chapter for more information about running REORG on a dual site application.
- If you run MUPIP STOP for an ongoing MUPIP REORG process, it may leave the database blocks in an inconsistent state. In this situation, GT.M converts the ongoing KILLs flag to abandoned KILLs flag. If a subsequent MUPIP REORG encounters these abandoned KILLs flags, it gives a message and then starts its REORG actions.
- <CTRL-C> terminates REORG. A REORG terminated abnormally by operator action or error is incomplete but does not adversely affect the database structure, unless terminated with a kill -9.



### Caution

REORG focuses on optimum adjacency and a change to even a single block can cause it to perform a large number of updates with only marginal benefit. Therefore, FIS recommends not running successive REORGs close together in time as that can provide minimal benefit for a significant increase in database and journal

activity. For the same reason, FIS recommends careful research and planning before using the -RESUME qualifier or complex uses of -EXCLUDE and -SELECT.

Assume two scenarios of putting values of ^x(1) to ^x(10000). In the first scenarios, fill values in a sequential manner. In the second scenario, enter values for odd subscripts and then enter values for the even subscripts.

Scenario 1:

At the GT.M prompt, execute the following command sequence:

```
GTM>for i=1:1:10000 set ^x(i)=$justify(i,200)
```

Then, execute the following MUPIP INTEG command.

```
$ mupip integ -region "*"
```

This command produces an output like the following:

```
Integ of region DEFAULT
No errors detected by integ.
```

Type	Blocks	Records	% Used	Adjacent
Directory	2	2	2.490	NA
Index	29	2528	95.999	1
Data	2500	10000	82.811	2499
Free	69	NA	NA	NA
Total	2600	12530	NA	2500

Note the high density (percent used) for index and data blocks from the report.

Scenario 2:

At the GT.M prompt, execute the following command sequence:

```
GTM>for i=1:2:10000 s ^x(i)=$justify(i,200)
```

```
GTM>for i=2:2:10000 set ^x(i)=$justify(i,200)
```

Then, execute the following command:

```
$ mupip integ -region "*"
```

This command produces an output like the following:

```
Integ of region DEFAULT
No errors detected by integ.
```

Type	Blocks	Records	% Used	Adjacent
Directory	2	2	2.490	NA
Index	153	3902	29.211	57
Data	3750	10000	55.856	1250
Free	95	NA	NA	NA

Total	4000	13904	NA	1307
-------	------	-------	----	------

Note that there are more and less dense index and data blocks used than in scenario 1. MUPIP REORG addresses such issues and makes the database (depending on the FILL\_FACTOR) more compact.

The optional qualifiers for MUPIP REORG are:

**-Exclude**

Specifies that REORG not handle blocks that contain information about the globals in the associated list–this means they are neither reorganized nor swapped in the course of reorganizing other globals; -EXCLUDE can reduce the efficiency of REORG because it complicates and interferes with the block swapping actions that try to improve adjacency.

The format of the EXCLUDE qualifier is:

```
-E[XCLUDE]=global-name-list
```

- Assume that a single MUPIP command organizes a subset of the globals in a database or region. If a second MUPIP REORG command selects the remaining globals, it may tend to disrupt the results of the first REORG by de-optimizing the previously organized blocks. This is because there is no information passed from the previous MUPIP REORG command to the next command. The EXCLUDE qualifier allows users to list the name of the previously REORGed globals, so that the MUPIP REORG bypasses the GDS blocks containing these globals.
- If global-name-list contains globals that do not exist, REORG issues a message to the terminal and continues to process any specified globals that exist. If REORG is unable to process any globals, it terminates with an error.
- Global-name-list can be an individual global name, a range of global names, or a list of names and prefixes followed by the wildcard symbol. For example:
  1. A global name, such as ACN.
  2. A range of global names, such as A7:B7.
  3. A list, such as A,B,C.
  4. Global names with the same prefix such as TMP\*.

In the first case, REORG only excludes global ^ACN. In the second case, REORG excludes all global names in the collating sequence A7 to B7. For the third case, REORG excludes A, B, and C. In the last case, REORG excludes all globals prefixed with TMP.

- Enclose wildcards in double-quotes ("") to prevent inappropriate expansion by the shell. The caret symbol (^) in the specification of the global is optional.
- By default, REORG does not EXCLUDE any globals.
- In case any global appears in the argument lists of both -SELECT and -EXCLUDE, REORG terminates with an error.

**-Fill\_factor**

Specifies how full you want each database block to be. This is a target number. Individual blocks may be more or less full than the fill factor. The format of the FILL\_FACTOR qualifier is:

```
F[ILL_FACTOR]=integer
```

## General Database Management

- The arguments for the FILL\_FACTOR qualifier must be integers from 30 to 100. These integers represent the percentage of the data block that REORG can fill. By default, the FILL\_FACTOR value is 100 for maximum data density.
- Users who come upon database performance issues or a high rate of database updates must examine the defined FILL\_FACTORS. Unless the application uses entirely uniform records, which is not typical for most applications, FILL\_FACTORS do not work precisely.
- The FILL\_FACTOR for data that is relatively static, or grows by the addition of new nodes that collate before or after pre-existing nodes, should be 100 percent. The FILL\_FACTOR for data that is growing by additions to existing nodes may be chosen to leave room in the typical node for the forecast growth for some period. Generally, this is the time between the LOAD and first REORG, or between two REORGs. This is also true for additions of nodes that are internal to the existing collating sequence.

### -Index\_fill\_factor

Directs REORG to leave free space within index blocks for future updates. Arguments to this qualifier must be integers from 30 to 100 that represent the percentage of the index block that REORG can fill. REORG uses this number to decide whether to place more information in an index block, or create space by moving data to another block. The format of the INDEX\_FILL\_FACTOR qualifier is:

```
-I[INDEX_FILL_FACTOR]=integer
```

Under certain conditions, especially with large database block sizes, it may be possible to achieve faster throughput by using a smaller fill factor for index blocks than for data blocks. By default, the INDEX\_FILL\_FACTOR is the value of FILL\_FACTOR regardless of whether that value is explicitly specified or implicitly obtained by default.

### -Resume

For an interrupted REORG operation, -RESUME allows the user to resume the REORG operation from the point where the operation stopped. REORG stores the last key value in the database file header. The format of the RESUME qualifier is:

```
-R[RESUME]
```

- With RESUME specified, the program retrieves the last key value, from the database file header, and restarts operations from that key.

### -Region

Specifies that REORG operate in the regions in the associated list and restricts REORG to the globals in those regions that are mapped by the current global directory; it does not have the same interactions as -EXCLUDE and -SELECT, but it does not mitigate those interactions when combined with them.

The format of the REGION qualifier is:

```
-R[EGION]=region-list
```

### -Select

Specifies that REORG reorganizes only the globals in the associated list; globals not on the list may be modified by block swaps with selected globals unless they are named with -EXCLUDE; -SELECT can be difficult to use efficiently because it tends to deoptimize unselected globals unless they are name in an -EXCLUDE list (which introduces inefficiency).

The format of the SELECT qualifier is:

```
-S[ELECT]=global-name-list
```

- By default, REORG operates on all globals in all database files identified by the current Global Directory for the process executing the MUPIP command.
- One of the functions performed by REORG is to logically order the globals on which it operates within the file. Unless the EXCLUDE and SELECT qualifiers are properly used in tandem, repeating the command with different selections in the same file wastes work and leaves only the last selection well organized.
- If you enter the REORG -SELECT=global-name-list command and the specified globals do not exist, REORG issues a message to the screen and continues to process any specified globals that exist. If REORG is unable to process any globals, it terminates with an error.
- Arguments for this qualifier may be an individual global name, a range of global names, or a list of names and prefixes followed by the wildcard symbol. The caret symbol (^) in the specification of the global is optional.
- The global name can be:
  1. A global name, such as ACN
  2. A range of global names, such as A7:B7
  3. A list, such as A,B,C.
  4. Global names with the same prefix such as TMP\*.
- In the first case, REORG only includes global ^ACN. In the second case, REORG includes all global names in the collating sequence A7 to B7. For the third case, REORG includes A, B, and C. In the last case, REORG includes all globals prefixed with TMP.
- By default, REORG selects all globals.

## -Truncate

Specifies that REORG, after it has rearranged some or all of a region's contents, should attempt to reduce the size of the database file and return free space to the file system. The format of the TRUNCATE qualifier is:

```
-T[RUNCATE][=percentage]
```

The optional percentage (0-99) provides a minimum amount for the reclamation; in other words, REORG won't bother performing a file truncate unless it can give back at least this percentage of the file; the default (0) has it give back anything it can. TRUNCATE always returns space aligned with bit map boundaries, which fall at 512 database block intervals. TRUNCATE analyses the bit maps, and if appropriate, produces before image journal records as needed for recycled (formerly used) blocks; The journal extract of a truncated database file may contain INCTN records having the inctn opcode value 9 indicating that the specific block was marked from recycled to free by truncate.



### Note

TRUNCATE does not complete if there is a concurrent online BACKUP or use of the snapshot mechanism, for example by INTEG.

## Examples for MUPIP REORG

Example:

```
$ mupip reorg -exclude="^b2a,^A4gsEQ2e:^A93"
```

This example performs a MUPIP REORG operation on all globals excluding ^b2a and all globals ranging from ^A4gsEQ2e to ^A93.

Example:

If the forecasted growth of a global is 5% per month from relatively uniformly distributed updates, and REORGs are scheduled every quarter, the `FILL_FACTOR` for both the original LOAD and subsequent REORGs might be 80 percent  $100 - ((3 \text{ months} + 1 \text{ month "safety" margin}) * \text{five percent per month})$ . The REORG command based on the above assumptions is as follows:

```
$ mupip reorg -fill_factor=80
```

## REPLICATE

Control the logical multi-site operation of GT.M. For more information on the qualifiers of the MUPIP REPLICATE command, refer to Chapter 7: “*Database Replication*” (page 173) .

## RESTORE

Integrates one or more BACKUP -INCREMENTAL files into a corresponding database. The transaction number in the first incremental backup must be one more than the current transaction number of the database. Otherwise, MUPIP RESTORE terminates with an error.

The format of the RESTORE command is:

```
RE[STORE] [-[NO]E[XTEND]] file-name file-list
```

- **file-name** identifies the name of the database file that RESTORE uses as a starting point.
- **file-list** specifies one or more files produced by BACKUP -INCREMENTAL to RESTORE into the database. The file-name are separated by commas (,) and must be in sequential order, from the oldest transaction number to the most recent transaction number. RESTORE may take its input from a UNIX file on any device that supports such files.
- The current transaction number in the database must match the starting transaction number of each successive input to the RESTORE.
- If the BACKUP -INCREMENTAL was created using -TRANSACTION=1, create a new database with MUPIP CREATE and do not access it, except the standalone MUPIP commands INTEG -FILE, EXTEND, and SET before initiating the RESTORE.

### -Extend

Specifies whether a MUPIP RESTORE operation should extend the database file automatically if it is smaller than the size required to load the data.

The format of the EXTEND qualifier is:

```
-[NO]E[XTEND]
```

M activity between backups may automatically extend a database file. Therefore, the database file specified as the starting point for a RESTORE may require an extension before the RESTORE. If the database needs an extension and the command specifies -NOEXTEND, MUPIP displays a message and terminates. The message provides the sizes of the input and output database files and the number of blocks by which to extend the database. If the RESTORE specifies more than one incremental backup with a file list, the database file may require more than one extension.

By default, RESTORE automatically extends the database file.

### Examples for MUPIP RESTORE

```
$ mupip restore backup.dat $backup_dir/backup.bk1, $backup_dir/backup.bk2, $backup_dir/backup.bk3
```

This command restores backup.dat from incremental backups stored in directory specified by the environment variable backup\_dir.

```
$ mupip restore gtm.dat ""gzip -d -c online5pipe.inc.gz |""
```

This command uses a pipe to restore gtm.dat since its last DATABASE backup from the bytestream backup stored in online5pipe.inc.gz.

### RUNDOWN

When database access has not been properly terminated, RUNDOWN properly closes currently inactive databases, removes abandoned GT.M database semaphores, and releases any IPC resources used. Under normal operations, the last process to close a database file performs the RUNDOWN actions, and a MUPIP RUNDOWN is not required. If a database file is already properly rundown, a MUPIP RUNDOWN has no effect. If in doubt, it is always safe to perform a rundown. FIS recommends the following method to shutdown a GT.M application or the system:

- Terminate all GT.M processes, and
- Rundown any and all database files that may be active.

MUPIP RUNDOWN checks for version mismatch. If there is a mismatch, it skips the region and continues with the next region. This makes it easier for multiple (non-interacting) GT.M versions to co-exist on the same machine. Note that GT.M does not support concurrent access to the same database file by multiple versions of the software.

The format of the RUNDOWN command is:

```
RU[NDOWN] [-FILE file-name|-REGION region-list]}
```

MUPIP RUNDOWN clears certain fields in a file that is already closed. This facilitates recovery from a system crash or other operational anomaly.

Use RUNDOWN after a system crash or after the last process accessing a database terminates abnormally. RUNDOWN ensures that open databases are properly closed and ready for subsequent use. RUNDOWN has no effect on any database that is actively being accessed at the time the RUNDOWN is issued.

To ensure database integrity, all system shutdown algorithms should include scripts that stop at GT.M processes and perform RUNDOWN on all database files.

The RUNDOWN command may include one of the following qualifiers:

-F[file]  
-R[region]

If the RUNDOWN command does not specify either -File or -Region, it checks all the IPC resources (shared memory) on the system and if they are associated with a GT.M database, attempts to rundown that file.

## -File

Specifies that the argument is a file-name for a single database file. The -FILE qualifier is incompatible with the REGION qualifier. If the rundown parameter consists of a list of files, the command only operates on the first item in the list.

Incompatible with: -REGION

## -Region

Specifies that the argument contains one or more region-names that identify database files mapped by the current Global Directory. Use the wild card "\*" to rundown all inactive regions in a global directory.

Incompatible with: -FILE

When MUIP RUNDOWN has no qualifier, it performs rundown on all inactive database memory sections on the node. Because this form has no explicit list of databases, it does not perform any clean up on regions that have no abandoned memory segments but may not have been shutdown in a crash.

## -Override

Overrides the protection that prevents MUIP RUNDOWN from performing a rundown of a replication-enabled (with BEFORE\_IMAGE) database or a non-replicated NOBEFORE-journaled database that was abnormally shutdown. The protection involves issuing the MUUSERLBK error for a previously crashed replication-enabled (with BEFORE IMAGE journaling) database and the MUUSERECOV error for a non-replicated or NOBEFORE-journaled database. Both these errors prevent complications related to data recovery from a journal file or a replication-enabled database.

## SET

Modifies certain database characteristics. MUIP SET operates on either regions or files.



### Note

In regions that have journaling enabled and on, users can switch journal files without either requiring standalone access or freezing updates.

The format of the SET command is:

```
SE[T] {-FI[LE] file-name|-REG[ION] region-list}
-A[CCCESS_METHOD]={BG|MM}
-B[YPASS]
-DE[FER_TIME]=seconds
-E[XTENSION_COUNT]=integer(no of blocks)
-F[LUSH_TIME]=integer
-G[LOBAL_BUFFERS]=<integer>
```

```
-JN[LFILE]
-JO[URNAL]=journal-option-list
-L[OCK_SPACE]=integer
-M[UTEX_SLOTS]=integer
-[NO]INST[_FREEZE_ON_ERROR]
-PA[RTIAL_RECOV_BYPASS]
-REP[LICATION]={ON|OFF}
-RES[ERVED_BYTES]=integer
-S[TANDALONENOT]
-V[ERSION]={V4|V6}
-W[AIT_DISK]=integer
```

- The file-name (or region-list) identifies the target of the SET.
- The SET command must include one of the following qualifiers which determine whether the argument to the SET is a file-name or a region-list.
- Exclusive access to the database is required if the MUPIP SET command specifies -ACCESS\_METHOD, -GLOBAL\_BUFFERS, -LOCK\_SPACE or -NOJOURNAL, or if any of the -JOURNAL options ENABLE, DISABLE, or BUFFER\_SIZE are specified.

The following section describe the qualifiers of the MUPIP SET command.

### -Access\_method

Specifies the access method (GT.M buffering strategy) for storing and retrieving data from the global database file. The format of the ACCESS\_METHOD qualifier is:

```
-A[CCCESS_METHOD]=code
```

### -Partial\_recov\_bypass

Sets the CORRUPT\_FILE flag in the database fileheader to FALSE. The CORRUPT\_FILE flag indicates whether a region completed a successful recovery. For more information, refer to the CORRUPT\_FILE qualifier in Chapter 10, *Database Structure Editor* [279].

### -File

Specifies that the argument is a file-name for a single database file. The format of the FILE qualifier is:

```
-F[ILE]
```

Incompatible with: -REGION

### -Region

Specifies that the argument is a region-list which identifies database file(s) mapped by the current Global Directory. The format of the REGION qualifier is:

```
-R[EGION]
```

SET -REGION changes multiple files when the parameter contains a list and/or wildcards.

Incompatible with: -FILE

### **-Extension\_count**

Specifies the number of GDS blocks by which an existing database file extends. A file or region name is required. This qualifier requires standalone access.

The format of the EXTENSION\_COUNT qualifier is:

```
-E[XTENSION_COUNT]=integer
```

### **-Flush\_time**

Specifies the amount of time between deferred writes of stale cache buffers. The default value is 1 second and the maximum value is 1 hour. -FLUSH\_TIME requires standalone access. The format of the FLUSH\_TIME qualifier is:

```
-F[LUSH_TIME]=[[[HOURS:]MINUTES:]SECONDS:]CENTISECONDS
```

### **-Global\_buffers**

Specifies the number of cache buffers for a BG database. This qualifier requires standalone access. The format of the GLOBAL\_BUFFERS qualifier is:

```
-G[LOBAL_BUFFERS]=integer
```

For more information on ways to determine good working sizes for GLOBAL\_BUFFERS, refer to Chapter 4: “*Global Directory Editor*” (page 32).

In general, increasing the number of global buffers improves performance by smoothing the peaks of I/O load on the system. However, increasing the number of global buffers also increases the memory requirements of the system, and a larger number of global buffers can increase the probability of the buffers getting swapped out. If global buffers are swapped out, any performance gain from increasing the number of global buffers will be more than offset by the performance impact of swapping global buffers. Most applications use from 1,000 to 4,000 global buffers for database regions that are heavily used. FIS does not recommend using fewer than 256 buffers except under special circumstances.

The minimum is 64 buffers and the maximum is 65536 buffers. By default, MUPIP CREATE establishes GLOBAL\_BUFFERS using information entered in the Global Directory.

On many UNIX systems, default kernel parameters may be inadequate for GT.M global buffers, and may need to be adjusted by a system administrator.

### **-Lock\_space**

Specifies the number of pages allocated to the management of M locks associated with the database. The size of a page is always 512 bytes.

The format of the LOCK\_SPACE qualifier is:

```
-L[OCK]_SPACE=integer
```

- The maximum LOCK\_SPACE is 65,536 pages.

- The minimum LOCK\_SPACE is 10 pages.
- The default LOCK\_SPACE is 40 pages.
- A file or region name is required to assign lock space.
- For more information on LOCK\_SPACE, refer to Chapter 4: “*Global Directory Editor*” (page 32).
- This qualifier requires standalone access.

### **-INST\_freeze\_on\_error**

Enables or disables custom errors in a region to automatically cause an Instance Freeze. This flag modifies the "Inst Freeze on Error" file header flag.

For more information on creating a list of custom errors that automatically cause an Instance Freeze, refer to “Instance Freeze” (page 193).

For more information on promptly setting or clearing an Instance Freeze on an instance irrespective of whether any region is enabled for Instance, refer to the “Starting the Source Server” (page 230) section of the Database Replication chapter.

### **-Mutex\_slots**

Sets the size of a structure that GT.M uses to manage contention for the principal critical section for a database. Performance issues may occur when there are many processes contending for database access and if this structure cannot accommodate all waiting processes. Therefore, FIS recommends setting this value to a minimum of slightly more than the maximum number of concurrent processes you expect to access the database.

The minimum value is 64 and the maximum value is 32768. The default value is 1024.

### **-Journal**

Specifies whether the database allows journaling and, if it does, characteristics for the journal file. The format of the JOURNAL qualifier is:

```
-[NO]J[OURLIST][=journal-option-list]
```

- -NOJOURNAL specifies that the database does not allow journaling. And also it does not accept an argument assignment.
- -JOURNAL specifies journaling is allowed. It takes one or more arguments in a journal-option-list.
- For detailed description of the all JOURNAL qualifiers and its keywords, refer to Chapter 6: “*GT.M Journaling*” (page 130).

### **-Qdbrundown**

Quickens normal process shutdown where a large number of processes accessing a database file are required to shutdown almost simultaneously, for example, in benchmarking scenarios. When a terminating GT.M process observes that a large number of processes are attached to a database file and QDBRUNDOWN is enabled, it bypasses checking whether it is the last process accessing the database. Such a check occurs in a critical section and bypassing it also bypasses the usual RUNDOWN actions which accelerates process shutdown removing a possible impediment to process startup. By default, QDBRUNDOWN is disabled.

Note that with QDBRUNDOWN there is a possibility of race condition that might leave the database fileheader and IPC resources in need of cleanup. Although QDBRUNDOWN minimizes the probability of such a race condition, it cannot eliminate it. When using QDBRUNDOWN, FIS recommends an explicit MUPIP RUNDOWN of the database file after the last process exits, to ensure the cleanup of database fileheader and IPC resources.

### -REServed\_bytes

Specifies the size to be reserved in each database block. RESERVED\_BYTES is generally used to reserve room for compatibility with other implementations of M or to observe communications protocol restrictions. The format of the RESERVED\_BYTES qualifier is:

```
-RES[ERVED_BYTES]=size
```

- RESERVED\_BYTES may also be used as a user-managed fill factor.
- The minimum RESERVED\_BYTES is 0 bytes. The maximum RESERVED\_BYTES is the block size minus the size of the block header which is 7 or 8 depending on your platform. Realistic determinations of this amount should leave room for at least one record of maximum size.

### -Version

Sets the block format version (Desired DB Format field in the file header) for all subsequent new blocks. The format of the VERSION qualifier is:

```
-V[ERSION]={V4|V6}
```

- MUPIP UPGRADE and MUPIP REORG -UPGRADE set the Desired DB Format field in the database file header to V6 while MUPIP REORG -DOWNGRADE sets it to V4.
- To set the version to V4, the current transaction number (CTN) of the database must be within the range of a 32-bit maximum.
- V6 block format is compatible with the V5 block format. The longer key and longer records (spanning nodes) features of V6 format are automatically disabled when used with GT.M V5.\* versions.
- For more information on the upgrading or downgrading your database, refer to the release notes document of your current GT.M version.

## Examples for MUPIP SET

Example:

```
$ mupip set -journal=on,nobefore -region "*"
```

This example enables NOBEFORE image journaling and turns on journaling for all regions.

```
$ mupip set -version=V4 -file mumps.dat
```

```
Database file mumps.dat now has desired DB format V4
```

This example sets the block format to V4 for all subsequent new blocks in V6 database file mumps.dat.

Example:

```
$ mupip set -version=v6 -file mumps.dat
```

Database file mumps.dat now has desired DB format V5

This example sets the block format to V6 for all subsequent new blocks in V4 database file mumps.dat.

Example:

```
mupip set -flush_time=01:00:00:00 -region DEFAULT
```

This example sets flush time to 1 hour. You can also specify flush time in any combination of [[HOURS:]MINUTES:]SECONDS:]CENTISECONDS. MUPIP interprets -FLUSH\_TIME=360000 or -FLUSH\_TIME=00:60:00:00 as -FLUSH\_TIME=01:00:00:00.

Example:

```
$ mupip set -region REPTILES -inst_freeze_on_error
```

This example enables custom errors in region REPTILES to cause an Instance Freeze.

## SIZE

Estimates and reports the size of global variables using a format that is similar to the one that appears at the end of the MUPIP INTEG -FULL report. In comparison with MUPIP INTEG -FAST -FULL, MUPIP SIZE provides the option of choosing any one of the three estimation techniques to estimate the size of global variables in a database file. These techniques vary in measurement speed and estimate accuracy. The format of the MUPIP SIZE command is:

```
MUPIP SI[ZE] [-h[uristic]=estimation_technique] [-s[elect]=global-name-list]
▶ [-r[egion]=region-list]
```



The optional qualifiers of MUPIP SIZE are:

*-Heuristic=estimation\_technique*

Specifies the estimation technique that MUPIP SIZE should use to estimate the size of global variables. The format of the -HEURISTIC qualifier is:

```
-h[uristic]={sc[an][,level=<lvl>] | a[rsample][,samples=<smp[ls]>] |
▶ i[mps[ample][,samples=<smp[ls]>]}
```



- *smp[ls]* is the number of samples and must be greater than zero (0)
- *lvl* is a positive or negative tree level designation and  $-(\text{level of the root block}) \leq \text{lvl} \leq (\text{level of the root block})$

*estimation-technique* is one of the following:

- *scan,level=<lvl>*

## General Database Management

Traverses the global variable tree and counts the actual number of records and blocks at levels from the root down to the level specified by `lvl` (default is 0, the data blocks). If the given level is non-negative, it is the lowest block level of the global for which the count is requested. So, 0 means all blocks, 1 means all index blocks, 2 means all index blocks of level 2 and above, and so on. SCAN counts a negative level from the root of the global tree where -1 means children of the root.

- `arsample,samples=<smpls>`

Uses acceptance/rejection sampling of random tree traversals to estimate the number of blocks at each level. It continues until the specified number of samples (default is 1,000) is accepted.

- `impsample,samples=<smpls>`

Uses importance sampling of random tree traversals to weight each sample of the specified number of samples (default is 1,000) in order to estimate size of the tree at each level.

- If `-HEURISTIC` is not specified, MUPIP SIZE uses the `ARSAMPLE,SAMPLE=1000` estimation technique.



### Important

For large databases, MUPIP SIZE is faster than MUPIP INTEG `-FAST -FULL`. `IMPSAMPLE` is expected to be the fastest estimation technique, followed by `ARSAMPLE` and then `SCAN`.

In terms of accuracy, MUPIP INTEG `-FAST -FULL` is the most accurate.

### *-Select*

Specifies the global variables on which MUPIP SIZE runs. If `-SELECT` is not specified, MUPIP SIZE selects all global variables.

The format of the `SELECT` qualifier is:

```
-s[elect]=global-name-list
```

*global-name-list* can be:

- A comma separated list of global variables.
- A range of global variables denoted by `start:end` syntax. For example, `-select="g1:g4"`.
- A global variable with wildcards, for example, `"g*"` (the name must be escaped to avoid shell filename expansion)
- `"**"` to select all global variables.

### *-Region*

Specifies the region on which MUPIP SIZE runs. If `REGION` is not specified, MUPIP SIZE selects all regions. The format of the `REGION` qualifier is:

```
-R[EGION]=region-list
```

Examples:

```
$ mupip size -heuristic="impsample,samples=2000" -select="y*" -region="AREG"
```

This example estimates the size of all global variable starting with "y". It uses importance sampling with 2000 samples on the region AREG.

```
$ mupip size -heuristic="scan,level=-1"
```

This example counts the number of blocks and records at 1 level below the root of the database tree.

```
$ mupip size -heuristic="arsample" -select="g1:g3"
```

This example estimates the size of global variables g1, g2 and g3 using accept/reject sampling with the default number of samples regardless of the region in which they reside.



### Note

Apart from randomness caused by sampling heuristics, MUPIP SIZE also has randomness from concurrent updates because it does not use the snapshot technique that MUPIP INTEG uses.

## STOP

Terminates a GT.M image. The image executes an orderly disengagement from all databases that are currently open by the process, and then exits. A MUPIP STOP performs a kill -15 and therefore may also be used to stop non-GT.M images.

The format of the STOP command is:

```
MUPIP ST[OP] process-id
```

- Use the shell command `ps` to display a list of active process names and process identifiers (PIDs).
- To STOP a process belonging to its own account, a process requires no privileges. To STOP a process belonging to another account, MUPIP STOP must execute as root.

## TRIGGER

Examines or loads trigger definitions. The format of the MUPIP TRIGGER command is:

```
TRIGGER {-TRIG[GERFILE]=<trigger_definitions_file> [-NOPR[OMPT]]|  
[-SELE[CT]][=name-list|*][<select-output-file>]}
```

Before you run the MUPIP TRIGGER command:

1. Set the value of the environment variable **gtmgblidir**: to specify the value of a current global directory.
2. Ensure that the key size, record size, block size of your database is sufficient for storing trigger definition. You may have to set the key and record sizes larger than the database content would otherwise require.

The qualifiers of the MUPIP TRIGGER command are as follows:

*TRIGgerfile*=<trigger\_definitions\_file>

Loads a trigger definition file to the database. The format of the TRIGGERFILE qualifier is:

```
-TRIG[GERFILE]=<trigger_definitions_file> [-NOPR[OMPT]]
```

## General Database Management

- For information on the syntax and usage of a trigger definition file, refer to *GT.M Programmer's Guide*.
- A MUPIP TRIGGER -TRIGGERFILE operation occurs within a transaction boundary, therefore, if even one trigger from the trigger definition file fails to parse correctly, MUPIP TRIGGER rolls back the entire trigger definition file load. MUPIP TRIGGER operations have an implicit timeout of zero (0), meaning the read must succeed on the first try or the command will act as if it received no input.
- MUPIP TRIGGER -TRIGGERFILE ignores blank lines and extra whitespace within lines. It treats lines with a semi-colon in the first position as comments and ignores their content.
- MUPIP TRIGGER compiles the XECUTE action string and rejects the load if the compilation has errors.
- Always specify the same value for the environment variable **gtm\_chset** during loading and executing triggers. If you specify different values of **gtm\_chset** during loading and executing triggers, MUPIP TRIGGER generates a run-time error (TRIGINVCHSET). GT.M does not prevent a process from updating different nodes with triggers using a different character set, however, GT.M prevents a process from updating the same triggering node with different character sets. Your coding practice, for all database updates, should be to ensure that you provide the same value for **gtm\_chset** during load compilation and run-time compilation.
- Incompatible with: **-SELECT**



### Note

The trigger update summary reports count not only names and option changes as "modified" but also cases where a -COMMANDS list changed, even though those are functionally additions or deletions of separate trigger definitions.

*SELECT=name-list*

Provides a facility to examine the current trigger definition. SELECT produces a list of the current triggers for a comma-separated list of global variables or trigger names. The format of the SELECT qualifier is:

**-SELE[CT][=name-list[\*]][\*] [ <select-output-file> ]**

1. Name-list can include global names, delimited with a leading caret (^), and/or trigger names (user-defined or auto-generated) with no leading caret. You can specify a trailing asterisk(\*) with either.
2. With no arguments specified, GT.M treats -SELECT as -SELECT="" and extracts a list of all current triggers.
3. Optionally, you can specify a file name to redirect the output of the command. If you do not specify a file name, MUPIP TRIGGER prompts for a file name. If you respond with an empty string (RETURN), MUPIP TRIGGER directs the output to STDOUT.



### Note

The output from the MUPIP TRIGGER -SELECT command may not be identical to your trigger definition file. This is because GT.M converts some semantically identical syntax into a single internal representation; while -SELECT output may not be identical to the -TRIGGERFILE input, it has the same meaning. Additionally, MUPIP TRIGGER -SELECT displays a field called "Cycle" as part of a comment. Cycle is the number of trigger definition updates (addition, modification, or deletion) performed on a global.

## Examples for MUPIP TRIGGER

This section provides step-by-step instructions for creating, modifying, and deleting triggers. Triggers affect all processes updating a database unlike, for example, environment variables such as \$gtmroutines which work on a per process basis. Therefore, FIS recommends that you should always have carefully planned procedures for changing triggers in your production environment.

### To create a new trigger for global node ^Acct("ID"):

- Using your editor, create a trigger definition file called triggers.trg with the following entry:

```
+^Acct("ID") -name=ValidateAccount -commands=S -xecute="Write ""Hello Earth!"""
```

- Execute a command like the following:

```
$ mupip trigger -triggerfile=triggers.trg
```

This command adds a trigger for ^Acct("ID"). On successful trigger load, this command displays an output like the following:

```
File triggers.trg, Line 1: ^Acct trigger added with index 1
=====
1 triggers added
0 triggers deleted
0 trigger file entries not changed
0 triggers modified
=====
```

Now, every S[et] operation on the global node ^Acct("ID") executes the trigger.

- Execute a command like the following:

```
$ mupip trigger -select="^Acct*"
```

This command displays the triggers. A sample output looks like the following:

```
;trigger name: ValidateAccount# cycle: 1
+^Acct("ID") -name=ValidateAccount -commands=S -xecute="Write ""Hello Earth!"""
```

### To modify an existing trigger for global node ^Acct("ID"):

You cannot directly replace an existing trigger definition with a new one. With the exception of -NAME and -OPTIONS, to change an existing trigger, you have to delete the existing trigger definition and then add the modified trigger definition as a new trigger. Note that GT.M performs two different trigger comparisons to match trigger definitions depending on whether or not S[ET] is the trigger invocation command. If there is a S[ET], then the comparison is based on the global name and subscripts, PIECES, [Z]DELIM, and XECUTE. If there is no SET, GT.M compares only the global node with subscripts and the -XECUTE code value.

- Begin by executing the following command:

```
$ mupip trigger -select="^Acct*"
Output file:
```

- Specify *trigger\_mod.trg* as the output file. This file contains entries like the following:

```
;trigger name: ValidateAccount# cycle: 1
+^Acct("ID") -name=ValidateAccount -commands=S -xecute="Write ""Hello Earth!"""
```

## General Database Management

- Using your editor, open `trigger_mod.trg` and change + (plus) to - (minus) for the trigger definition entry for `ValidateAccount` and add a new trigger definition for `^Acct("ID")`. To avoid inconsistent application behavior, it is important to replace an old trigger with a new one in the same transaction (Atomic). The `trigger_mod.trg` file should have entries like:

```
;trigger name: ValidateAccount# cycle: 1-^Acct("ID") -name=ValidateAccount -commands=Set -xecute="Write
""Hello
Earth!""
;trigger name: ValidateAccount#+^Acct("ID") -name=ValidateAccount -commands=Set -xecute="Write ""Hello Mars!""
```



- Execute a command like the following:

```
$ mupip trigger -triggerfile=trigger_mod.trg
```

- This command displays an output like the following:

```
File trigger_mod.trg, Line 1: ^Acct trigger deleted
File trigger_mod.trg, Line 3: ^Acct trigger added with index 1
=====
1 triggers added
1 triggers deleted
0 trigger file entries not changed
0 triggers modified
=====
```

Congratulations! You have successfully modified the `xecute` string of `ValidateAccount` with the new one.

### To delete an existing trigger for global node `^Acct("ID")`:

- Begin by executing the following command:

```
$ mupip trigger -select="^Acct*"
Output file:
```

- Specify `trigger_delete.trg` as the output file. This file contains entries like the following:

```
;trigger name: ValidateAccount# cycle: 3
+^Acct("ID") -name=ValidateAccount -commands=S -xecute="Write ""Hello Mars!""
```

- Using your editor, change + (plus) to - (minus) for the trigger definition entry for **ValidateAccount**. Alternatively, you can create a file with an entry like **-ValidateAccount**.
- Now, execute a command like the following:

```
$ mupip trigger -triggerfile=trigger_delete.trg
```

This command displays an output like the following:

```
File trigger_delete.trg, Line 2: ^Acct trigger deleted
=====
0 triggers added
1 triggers deleted
0 trigger file entries not changed
0 triggers modified
=====
```

You have successfully deleted trigger "ValidateAccount".

**To change a trigger name for global node ^Acct("ID"):**

1. Using your editor, create a new file called *trigger\_rename.trg* and add a trigger definition entry for **ValidateAcct** with the same trigger signature as **ValidateAccount**. Your trigger definition would look something like:

```
+^Acct("ID") -name=ValidateAcct -commands=S -execute="Write ""Hello Mars!"""
```

2. Verify that the **ValidateAccount** trigger exists by executing the following command:

```
$ mupip trigger -select="^Acct*"
Output file:
```

3. Respond with an empty string (Press Enter). Confirm that the trigger summary report contains an entry like the following:

```
;trigger name: ValidateAccount# cycle: 3
+^Acct("ID") -name=ValidateAccount -commands=S -execute="Write ""Hello Mars!"""
```

4. Now, execute a command like the following:

```
$ mupip trigger -triggerfile=trigger_rename.trg
```

This command displays an output like the following:

```
=====
0 triggers added
0 triggers deleted
0 trigger file entries not changed
1 triggers modified
=====
```

You have successfully changed the trigger name **ValidateAccount** to **ValidateAcct**.

## UPGRADE

Upgrades the file-header of a database. The format of the MUPIP UPGRADE command is:

```
UP[GRADE]
```

- It increases the size from 4 bytes to 8 bytes of file-header fields such as current transaction number (CTN), maximum TN and others that contain transaction numbers.
- It resets the various trace counters and changes the database format to V5. This change does not upgrade the individual database blocks but sets the database format flag to V5.
- It also initializes a counter of the current blocks that are still in V4 format. It decrements this counter each time an existing V4 format block is converted to V5 format. When the counter is 0, the entire database gets converted.

### Example for MUPIP UPGRADE

Example:

```
$ mupip upgrade mumps.dat
```

This example upgrades the file-header of mumps.dat to V5 format.

## MUPIP CommandSummary

COMMAND	OBJECTS	MAIN QUALIFIER
B[ACKUP]	region-name file-name	-BK[UPDBJNL]=DISABLE   OFF -B[YTESTREAM] -NET[TIMEOUT]=seconds -C[OMPREHENSIVE] -DA[TABASE] -REPLA[CE] -DBG -I[NCREMENTAL] -[NO]J[OURNAL][=journal-options-list] -NETTIMEOUT -[NO]NEWJNLFILES[=[NO]PREVLINK], [NO]S[YNC_IO] -O[NLINE] -RECORD -REPLI[NSTANCE]=OFF   ON -S[INCE]={DATABASE BYTESTREAM RECORD} -T[RANSACTION=hexa;transaction_number]
CR[EATE]	-	-R[EGION]=region-name
EN[DIANCVT]	file-name	-OUTDB=<outdb-file> -OV[ERRIDE]
EXI[T]	-	-
EXTE[ND]	region-name	-B[LOCKS]=blocks
EXTR[ACT]	-	-FO[RMAT]=GO B[INARY] Z[WR] -FR[EEZE] -LA[BEL]=text -[NO]L[OG] -S[ELECT]=global-name-list -O[CHSET]=character-set
F[REEZE]	region-list	-DBG -OF[F] -OV[ERRIDE]

General Database Management

COMMAND	OBJECTS	MAIN QUALIFIER
		-ON[-R[ECORD]]
FT[OK]	File-name	-D[B] -J[NLPOOL] -R[ECVPOOL]
H[ELP]	command-option	-
I[NTEG]	File-name or region-list	-A[DJACENCY]=integer -BL[OCK]=hexa;block-number -BR[IEF] -FA[ST] -FI[LE] -FU[LL] -NO]K[EYRANGES -[NO][MAP]=integer -[NO]MAXK[EYSIZE]=integer -R[EGION] -S[UBSCRIPT]=subscript -TN[_RESET] -[NO]TR[ANSACTION][=integer]
J[OURNALS]	file-name	-EX[TRACT][=file-specification -stdout] -REC[OVER]   -RO[LLBACK] -SH[OW][=show-option-list] -[NO]V[ERIFY] -BA[CKWARD]   -FO[RWARD]
L[OAD]	file-name	-BE[GIN]=integer -BLOCK_DENSITY -E[ND]=integer -FI[LLFACTOR]=integer -FO[RMAT]=GO B[INARY] Z[WR] -S[TDIN]
REO[RG]		-DOWNGRADE

General Database Management

COMMAND	OBJECTS	MAIN QUALIFIER
		-E[XCLUDE]=global-name-list -FI[LL_FACTOR]=integer -I[NDEX_FILL_FACTOR]=integer -REG[ION] -RES[UME] -SA[FEJNL] -S[ELECT]=global-name-list -STA[RTBLK]=hexa -STO[PBLK]=hexa -T[RUNCATE][=percentage] -UP[GRADE] -USER_DEFINED_REORG=reorg_list
REP[LICATE]	file-name	-E[DITINSTANCE] -I[NSTANCE_CREATE] -R[ECEIVER] -S[OURCE] -UPDA[TEPROC] -UPDH[ELPER]
RE[STORE]	file-name or file-list	-[NO]E[XTEND]
RU[NDOWN]	file-name or region-name	-F[ILE] -R[EGION]
SE[T]	file-name or region-name	-A[CCCESS_METHOD=BG MM] -B[YPASS] -DB[FILENAME]=database_file -DE[FER_TIME]=seconds -E[XTENSION_COUNT]=integer(no of blocks) -FILE -F[LUSH_TIME]= integer -G[LOBAL_BUFFERS]=integer -JN[LFILE]

### General Database Management

COMMAND	OBJECTS	MAIN QUALIFIER
		-JO[URNAL]=journal-option-list -L[OCK_SPACE]=integer -PA[RTIAL_RECOV_BYPASS] -PR[EVJNLFILE]=jnl_file_name -RE[GION] -REPLI[CATION]=ON OFF -REPL_[STATE]=ON OFF -RES[ERVED_BYTES]=integer -S[TANDALONENOT] -V[ERSION]=V4 V5 -W[AIT_DISK]=integer
ST[OP]	process-id	process-id
UP[GRADE]	-	-

Main Qualifier	MUPIP Command	Options/Qualifiers
-EDITINSTANCE	REPLICATE	-CHANGE -DETAIL -OFFSET=hexa -VALUE=hexa -SIZE=hexa -VALUE=hexa
-FENCES=<fence-options-list>	JOURNAL-RECOVER-ROLLBACK	ALWAYS NONE PROCESS
-JOURNAL=<journal-options-list>	BACKUP SET	ALIGNSIZE=integer ALLOCATION=integer AUTOSWITCHLIMIT=integer BEFORE_IMAGES BUFFER_SIZE=integer DISABLE

General Database Management

Main Qualifier	MUPIP Command	Options/Qualifiers
		ENABLE EPOCH_INTERVAL=integer EXTENSION=integer FILENAME=file_name OFF ON SYNC_IO YIELD_LIMIT=integer
-LOOKBACK_LIMIT=lookback-option-list	-RECOVER -ROLLBACK	TIME="time" OPERATIONS=integer
-RECEIVER	REPLICATE	-BUFFSIZE=integer -CHANGELOG -CHECKHEALTH -CMLVL=integer -FILTER=filter_name -he[lpers]=[m[,n]] -LISTENPORT=integer -LOG=logfile -LOG_INTERVAL=integer -SHOWBACKLOG -SHUTDOWN -START -STATSLOG=[ON OFF] -STOPSOURCEFILTER TIMEOUT=seconds -UPDATEONLY -UPDATERESYNC
-RECOVER	JOURNAL	-AFTER=time -APPLY_AFTER_IMAGE -BACKWARD

General Database Management

Main Qualifier	MUPIP Command	Options/Qualifiers
		<ul style="list-style-type: none"> <li>-BEFORE=time</li> <li>-BROKENTRANS=file</li> <li>-CHAIN</li> <li>-CHECKTN</li> <li>-[NO]ER[ROR_LIMIT][=integer]</li> <li>-FENCES=fence-option-list</li> <li>-FORWARD</li> <li>-FULL</li> <li>-GLOBAL=&lt;global_list&gt;</li> <li>-ID=&lt;pid_list&gt;</li> <li>-INTERACTIVE</li> <li>-LOOKBACK_LIMIT=&lt;lookback_limit_options&gt;</li> <li>-LOSTTRANS[=file]</li> <li>-RED[IRECT]=file-pair-list</li> <li>-SINCE=time</li> <li>-VERBOSE</li> <li>-VERIFY</li> </ul>
-EXTRACT	JOURNAL	<ul style="list-style-type: none"> <li>-AFTER=time</li> <li>-BEFORE=time</li> <li>-BROKENTRANS=file</li> <li>-CHAIN</li> <li>-CHECKTN</li> <li>-[NO]ER[ROR_LIMIT]=integer]</li> <li>-FENCES=fence-option-list</li> <li>-FULL</li> <li>-GLOBAL=&lt;global_list&gt;</li> <li>-ID=&lt;pid_list&gt;</li> <li>-INTERACTIVE</li> <li>-LOOKBACK_LIMIT=&lt;lookback_limit_options&gt;</li> </ul>

General Database Management

Main Qualifier	MUPIP Command	Options/Qualifiers
		-LOSTTRANS[=file] -SINCE=time -VERBOSE -VERIFY
-ROLLBACK	JOURNAL	-APPLY_AFTER_IMAGE -BACKWARD -BEFORE=time -BROKENTRANS=file -[NO]ER[ROR_LIMIT][=integer] -FENCES=fence-option-list -FETCHRESYNC -LOOKBACK_LIMIT=<lookback_limit_options> -LOSTTRANS[=file] -RES[YNC]=hexa;journal_sequence_number -VERBOSE -VERIFY
-SHOW=<show-option-list>	JOURNAL	-ACTIVE_PROCESSES -ALL -BROKEN_TRANSACTIONS -HEADER -PROCESSES -STATISTICS -AFTER=time -USER=user-list -TRANSACTION=[KILL SET] -INTERACTIVE -GLOBAL=<global_list> -ID=<pid_list> -INTERACTIVE

General Database Management

Main Qualifier	MUPIP Command	Options/Qualifiers
-SINCE	BACKUP	-BYTESTREAM -COMPREHENSIVE -DATABASE -INCREMENTAL -RECORD
-SO[URCE]	REPLICATE	-ACTIVATE -BUFFSIZE=Buffer_size -CHANGELOG -CHECKHEALTH -CMPLVL=integer -CONNECTPARAMS=connection_options -DEACTIVATE -DETAIL -FILTER=filter_name -INSTSECONDARY=secondary_instance name -JNLPOOL-LOG=log_file -LOG_INTERVAL=integer -LOSTTNCOMPLETE -NEEDRESTART -PASSIVE -PROPAGATEPRIMARY -ROOTPRIMARY -SECONDARY=secondary_instance_name -SHOWBACKLOG -SHUTDOWN -START -STATSLOG -STOPSOURCEFILTER

### General Database Management

Main Qualifier	MUPIP Command	Options/Qualifiers
		-TIMEOUT=seconds

---

## Chapter 6. GT.M Journaling

Revision History		
Revision V6.0-001/1	22 March 1013	<ul style="list-style-type: none"><li>Improved the formatting of all command syntaxes.</li><li>In “SET Object Identifying Qualifiers” (page 141), added information about the -repl_state and -dbfilename qualifiers.</li></ul>
Revision V6.0-001	27 February 2013	In “-EXtract[=<file-name> -stdout]” (page 153), added information about the gtm_extract_nocol environment variable.
Revision V6.0-000/1	21 November 2012	Updated “SET -JOURNAL Options” (page 143) and the journaling limits for V6.0-000.
Revision V5.5-000/4	6 June 2012	<ul style="list-style-type: none"><li>In “-ROLLBACK [{-ON[LINE]} -NOO[NLINE]]” [155], added the description of the new -ONLINE qualifier for -ROLLBACK.</li><li>In “Journal Extract Formats” [168], added the definitions of unam, clntnam, and ALIGN records.</li></ul>
Revision V5.5-000/3	2 May 2012	Added the definition of tid.
Revision V5.5-000/2	19 March 2012	Updated the Journal Extract Formats section for V5.5-000.
Revision 3	26 December 2011	Added the Journal Extract format for ZTRIG.
Revision 2	2 December 2011	Improved the description of -EXTRACT and corrected the heading levels of some sections under MUPIP JOURNAL.
Revision V5.4-002B	24 October 2011	Conversion to documentation revision history reflecting GT.M releases with revision history for each chapter.

---

## Introduction

The four key properties of transaction processing systems, the so-called "ACID" properties are: Atomicity, Consistency, Isolation, and Durability. GT.M transaction processing provides the first three by means of the TStart and TCommit commands and Durability through journaling.

GT.M, like virtually all high performance databases, uses journaling (called “logging” by some databases) to restore data integrity and provide continuity of business after an unplanned event such as a system crash.

Note that, journaling is not a substitute for good system configuration and design. For example, if a database and its journal files are on the same disk controller, a hardware failure on that controller can damage both files, and prevent recoverability. Journaling complements other techniques to build a robust system.

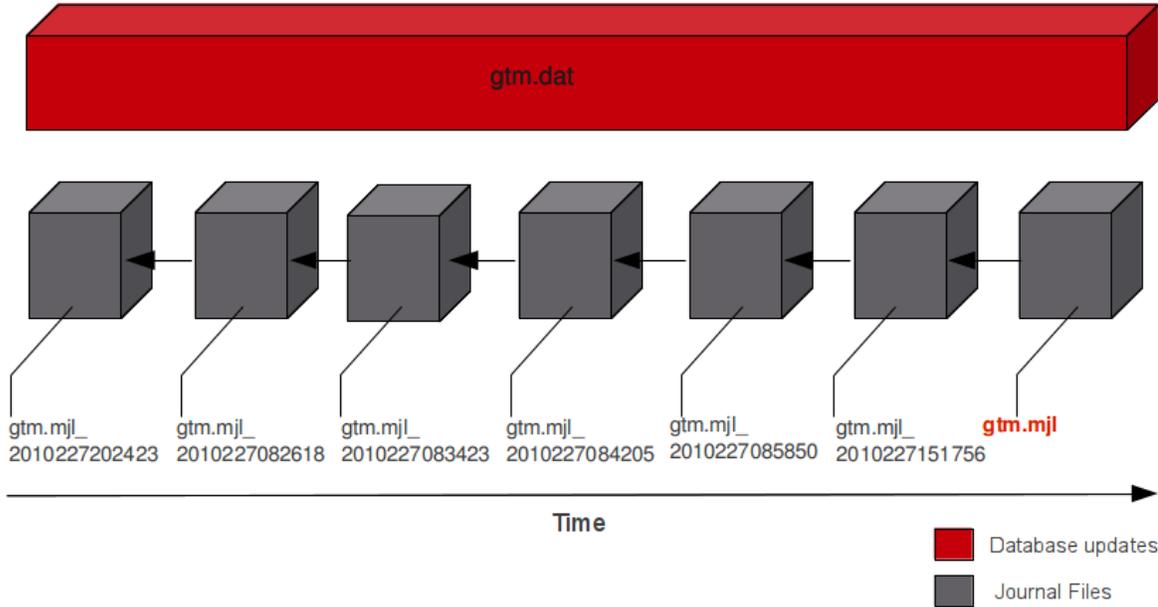
Journaling requires no M programming. However, the GT.M commands described later in this chapter may enhance the value of journaling.

## Journal Files

GT.M journaling uses journal files to record information pertaining to database updates. A journal file has a default extension of mjl. If the new journal filename (the one specified in the FILENAME option or the default) already exists, GT.M renames the existing journal file by appending a string that denotes the time of creation of the journal file in the form of "\_YYYYJJJHHMMSS" where:

YYYY	4-digit-year	such as 2010
JJJ	3-digit-Julian-day (between 1 and 366)	such as 199
HH	2-digit-hour in 24 hr format	such as 14
MM	2-digit minute	such as 40
SS	2-digit seconds	such as 30

The following animation describes how GT.M uses journal files to record information pertaining to database updates on gtm.dat (the default database file created by gtmprofile).



At any given time the database file (gtm.dat) has a single active journal file (gtm.mjl) with links to predecessor ("previous generation") journal files. The black arrow between the journal files demonstrate how a journal file is back-linked to its predecessor with file name in the form of gtm.mjl\_YYYYJJJHHMMSS to form a chain of journal files. When a switch of journal files occurs, either implicitly (for example, when AUTOSWITCHLIMIT is reached) or explicitly (for example, on a backup event or MUPIP SET -JOURNAL=ON), GT.M renames the existing journal file with the timestamp of its last modification. GT.M creates a new journal file with the name of the journal file for that database, and specifies the previous generation journal file name (after the rename), in the newly created journal file's header. GT.M journaling provides mechanisms for durable recovery/extract from the journal files, replaying database updates to an active database, reverting the database state to a previous consistent state for when replication is in use, and so on. GT.M automatically turns off journaling on encountering run-time conditions such as no available disk space or no authorization for a process attempting to auto-switch a journal file. In such a case, GT.M also logs an appropriate message to the operator log to alert the operational staff. If GT.M detects that the rename-logic yields a filename that already exists (a condition when journal files are switched in the same second), the string "\_N[N[N[N...]]]" is appended to the renamed filename where "N[N[N[N...]]]" denotes a sequence of numbers as follows:

**0,1,2,3,4,5,6,7,8,9,90,91,92,93,94,95,96,97,98,99,990,991,...**

GT.M tries all numbers from the order in the above sequence until it finds a non-existing rename-filename. In the above illustration, if `gtm.mjl_2010227_082618` is switched in the same second and `gtm.mjl_2010227_082618_0` already exists, the renamed journal file would be `gtm.mjl_2010227_082618_1`. If the existing file renaming scheme or the default journal file naming scheme discussed above results in a filename longer than 255 characters (due to the suffix creation rules), GT.M produces an error and turns off journaling.



### Note

In a very short time window just before switching a journal file, GT.M create a temporary file with an `.mjl_new` extension and attempts to write a few initialization journal records. After performing an initial verification, GT.M renames the `.mjl_new` file to the current `.mjl` file. In rare cases, you might see an `.mjl_new` file if the journal file creation process was interrupted midway (possibly due to permission or disk space issues). If a subsequent MUIP process detects an `.mjl_new` file and no `.mjl` file, it automatically deleted it and creates a new `.mjl` file.

There are two switches to turn on journaling - `ENable / DISable` and `ON/OFF`. Enabling or disabling journaling requires stand alone access to the database. Turning journaling on and off can be done when the database is in use. Note: Whenever GT.M implicitly turns off journaling due to run-time conditions such as no available disk space or no authorization for a process attempting to auto-switch a journal file (and so on) , it produces an error with accompanying messages to alert operation staff. GT.M on selected platforms can encrypt data in database and journal files. Encryption protects against unauthorized access to data by an unauthorized process which is able to access disk files, that is, encryption protects data at rest (DAR). Rather than build encryption into GT.M, a plug-in architecture facilitates use of your preferred encryption software. For more information, refer to Chapter 12: “*Database Encryption*” (page 363).

## Recovery from a Journal File

The following two procedures enable recovery of a database from a journal file:

1. Forward Recovery (roll forward by applying )
2. Backward Recovery (roll back to a checkpoint, optionally followed by a subsequent roll forward)



### Note

In a multi-site database replication configuration, you might use these recovery procedures to refresh a replicating instance from the backup of an originating instance. However, the steps for both these recovery procedures are different.

## Forward Recovery

Forward recovery "replays" all database updates in forward direction till the specified point in the journal file. Forward recovery on a backup database starts from when the backup was taken and continues till the specified point in the journal files. Forward recovery on an empty database starts from the beginning of the journal files.

Suppose a system crash occurred at 08:50 hrs and a backup of the database was taken at 08:26 hrs. Using forward recovery, you can replay the database updates between 08:26 hrs to 8:50 hrs (in blue) on the backup copy of the database and restore the database to a state prior to the crash. In the process you can also identify unfinished or broken transactions that might have occurred at the time of the crash. In the following illustration, X denotes the crash time and the blue updates denote forward processing.

## GT.M Journaling



A command like **mupip journal -recover -forward -before="--8:50" gtm.mjl** performs this operation. From the current journal file, forward recovery moves back to the point where the begin transaction number of a journal file matches the current transaction number of the active database (the point when the backup was taken) and begins forward processing. Since a journal file is back-linked to its predecessor, GT.M facilitates forward processing by activating temporary forward links between journal files that appear only during recovery. These forward links are temporary because they are expensive to maintain as new journal files are created. Note: Forward recovery, by design, begins from a journal file whose "Begin Transaction" matches the "Current Transaction" of the active database. This condition occurs only when a new journal file is created (switched) immediately after a backup. If a database is backed up with MUPIP BACKUP -NONEWJNLFILES (a backup option where journal files are not switched), forward recovery cannot find a journal file whose Begin Transaction matches the Current Transaction and therefore cannot proceed with forward recovery. Always use a backup option that switches a journal file or switch journal files explicitly after a backup. Also, once a database has been recovered using forward recovery, you can no longer use it for a future recovery unless you restore the database again from the backup.

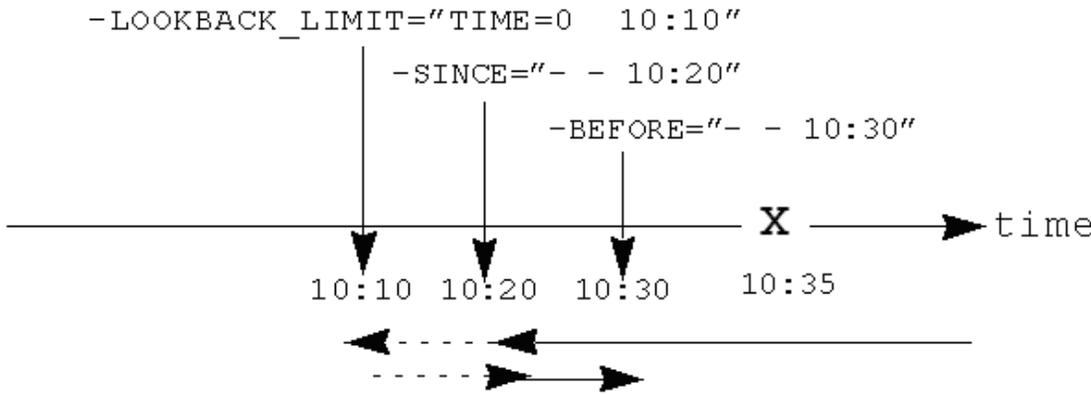
## Backward Recovery

Backward recovery restores a journaled database to a prior state. Backward processing starts by rolling back updates to a checkpoint (specified by **-SINCE** or **-AFTER**) prior to the desired state and replaying database updates forward till the desired state.

Backward Recovery uses "BEFORE\_IMAGE" journaling. With BEFORE\_IMAGE journaling, GT.M captures the database updates, as well as "snapshots" of portions of the database immediately prior to the change caused by the update. Unlike forward recovery which works on a backup database, backward recovery works only on production (current) database provided it is usable and BEFORE\_IMAGE journaling is enabled.

Suppose a system crash occurred at 10:35 hrs, a command like **mupip journal recover backward -lookback\_limit="TIME=0 10:10" -since="-- 10:20" -before="-- 10:30"** performs backward recovery. The following illustration demonstrates how GT.M performs a recovery after a system crash at 10:35. Backward recovery "un-does" the database updates backward to 10:20, then applies updates forward until the crash. By adding **-BEFORE="-- - 10:30"** to the command, the recovery stops when forward processing encounters updates that originally occurred after 10:30. If the application includes ZTSTART and ZTCOMMIT commands to fence a group of transactions, backward processing may continue back prior to 10:10 searching to resolve fenced transactions that were incomplete at 10:20.

## GT.M Journaling



-LOOKBACK\_LIMIT controls the maximum amount of additional backward processing, in this case, 10 minutes. Note that the -SINCE time in this example is slightly exaggerated for the sake of the graphical representation. If the application includes TSTART and TCOMMIT commands to fence transactions, backward processing does not require LOOKBACK\_LIMIT because TSTART/TCOMMIT transactions automatically resolve open transaction fences. So, in the above example if the transactions are fenced with TSTART/TCOMMIT, backward recovery automatically increases the backward processing by 10 minutes.



### Important

ZTSTART and ZTCOMMIT are deprecated in favor of TSTART and COMMIT. FIS no longer validates ZTSTART/ZTCOMMIT and -LOOPBACK\_LIMIT (since it applies to ZTSTART/ZTCOMMIT).

## rolled\_bak\* files

GT.M adds a prefix **rolled\_bak\_** to the journal file whose entire contents are eliminated (rolled back) by a backward recovery. GT.M does not use these files after a successful recovery therefore you might want to consider moving or deleting them. You should never use rolled\_bak\* files for any future database recovery. If there is a need to process rolled\_bak\* files, you should extract the journal records and process them using an M program. FIS recommends that you rename the roll back journal file immediately after a rollback if you want to save it, to prevent a subsequent rollback from overwriting it.

## Journal Files Access Authorization

GT.M propagates access restrictions to the journal files, backup, and snapshot temporary files. Therefore, generally journal files should have the same access authorization characteristics as their corresponding database files. In the rare case where database access is restricted but the owner is not a member of either the database group nor the group associated with the \$gtm\_dist directory, you should provide world read-write access to the journal files. As long as the operating system permits the access, GT.M allows access to database files and journals in cases where the system has no user or group information available for the file. Such an unusual situation can arise, for example, when the user and group are provided via NIS, but if NIS is not currently operational the owner and group cannot be determined; or perhaps a user id is deleted while the GT.M process is active.

## Triggers in Journal Files

GT.M manages "trigger definitions" and "triggered updates" differently during journaling and replication. Trigger definitions appear in both journal files and replication streams so the definitions propagate to recovered and replicated databases. Triggered updates appear in the journal file, since MUPIP JOURNAL -RECOVER/-ROLLBACK does not invoke triggers.

However, they do not appear in the replication stream since the Update Process on a replicating instance apply triggers and process their logic.

GT.M implicitly wraps a trigger as an M transaction. Therefore, a journal extract file for a database that uses triggers always has Type 8 and 9 (TSTART/TCOMMIT) records even if the triggers perform no updates (that is, are effectively No-ops).

When journaling is ON, GT.M generates journal records for database updates performed by trigger logic. For an explicit database update, a journal record specifies whether any triggers were invoked as part of that update. GT.M triggers have no effect on the generation and use of before image journal records, and the backward phase of rollback / recovery. A trigger associated with a global in a region that is journaled can perform updates in a region that is not journaled. However, if triggers in multiple regions update the same node in an unjournaled region concurrently, the replay order for recovery or rollback might differ from that of the original update and therefore produce a different result; therefore this practice requires careful analysis and implementation. Except when using triggers for debugging, FIS recommends journaling any region that uses triggers. If your database uses triggers, always ensure that unjournaled globals do not perform triggered updates in journaled globals and create procedures to handle trigger updates in the broken/lost transaction files. In broken/lost transaction files, you can identify these entries as + or - and appropriately deal with them using MUPIP TRIGGER and \$ZTRIGGER().

## BEFORE\_IMAGE Journaling

BEFORE\_IMAGE is a form of Journaling that creates "mini-backups" preceding each database update. Backward Recovery uses these mini-backups to restore the database as far back in time then it replays the database updates. "BEFORE\_IMAGE" journaling requires more disk I/O and storage space than M-level (or NOBEFORE) journaling but delivers faster recovery times from system failures .



### Note

As stated in the GDE chapter, the MM database access method bypasses the BG buffer pool and relies entirely on the operating/file system to manage traffic between memory and disk. Because with MM, GT.M has no control over the timing of disk updates, BEFORE\_IMAGE journaling is not an option with MM; attempts to use these two facilities together produce an error.

## NOBEFORE\_IMAGE Journaling

"NOBEFORE\_IMAGE" is a form of M-level Journaling that sequentially stores each database update in a journal file. A forward recovery operation restore the database by replaying these database updates. "NOBEFORE\_IMAGE" consumes less I/O bandwidth in normal use and helps obtain more throughput from the available servers.

## Choosing between BEFORE\_IMAGE and NOBEFORE\_IMAGE

The choice between BEFORE\_IMAGE journaling and NOBEFORE\_IMAGE journaling is important especially in a logical multi-site database replication deployment. If an application pushes the I/O bandwidth of the servers on which it runs, NOBEFORE\_IMAGE journaling may help obtain more throughput from available servers. BEFORE\_IMAGE journaling could be the likely choice if an application requires quicker recovery in the unlikely event of a crash. For a comprehensive discussion on the choosing the type of Journaling, refer to "Choosing between BEFORE\_IMAGE and NOBEFORE\_IMAGE journaling" (page 200).

## Broken Transaction File

In the case of a catastrophic event, it is unlikely that GT.M can properly complete writing all journal records to the file. GT.M reports the unfinished records or incomplete fenced transactions as "broken transactions". GT.M extracts broken transactions into a file called the broken transaction file.

## Lost Transaction File

Any complete transaction that occurs after a broken transaction is a lost transaction. GT.M does not play it into the database but extracts it as a lost transaction into a file called the lost transaction file.

All broken and lost transactions are made available as the result of a database recovery.

The operational procedures and the application tools should provide a means of recovering and reprocessing the information in the broken and lost transaction files after a recovery or rollback that places content in these files.

If there are no fences, repair any application-level integrity problems. In either case, MUPIP INTEG -FAST provides an excellent quick test of whether the database can support new updates with relative safety.

## Journaling Benefits

It is important to understand the benefits of Journaling before you enable Journaling on your database. M database management ensures that multiple concurrent updates and retrievals of the same information (or information "close together" in ordered sequence) occur in a predictable and logical fashion. Sometimes a database manager may have to change multiple records, usually indices, as a result of a single update. Interrupting a process that is performing such a "multi-point" update violates a design assumption of the M implementation and also results in a malformed database. Under normal operation, the database logic handles interruptions by deferring their recognition until the update is complete. However, occurrences such as power failures or a KILL-9 can cause such interruptions. GT.M Journaling helps maintain data integrity and continuity of business in the event of such interruptions.

Other benefits include (but not limited to):

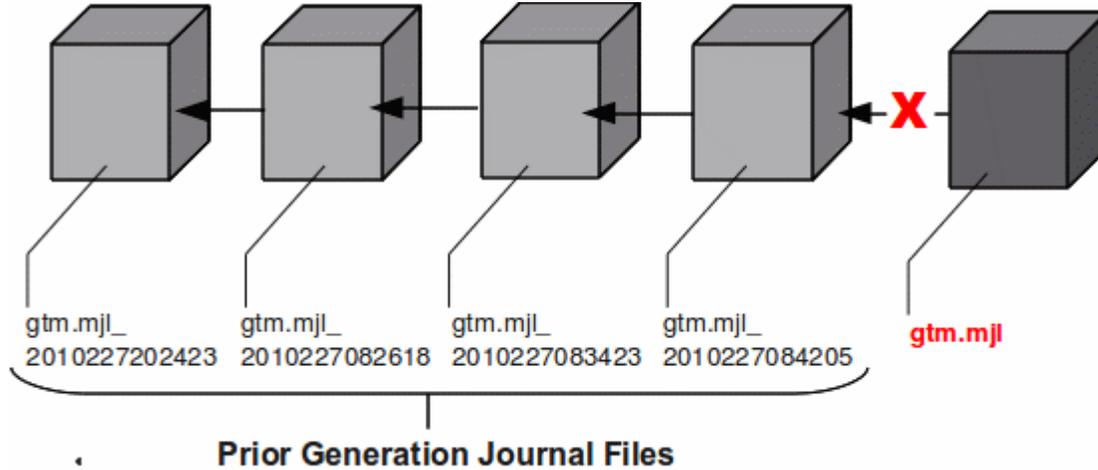
- Automatic replay of work to the last committed update recorded in a journal file. Note that with the use of transaction processing and journaling, GT.M provides full ACID properties.
- Quick recovery options, such as processing only the information recorded immediately prior to failure. For example, you can recover just the last minute of work instead of replaying the entire journal file.
- Recorded database updates formatted appropriately for processing by an M program. For example, MUPIP JOURNAL -EXTRACT produces records specified by time, user, the process identification number, global variable, process name, and transaction type.
- Identification of processes active when the system failed. -SHOW identifies these processes, as well as what transactions were not completed, and other information about the database updates and processes contained in the journal file.

## Backup Journal Files

FIS recommends separate backup schemes for database files and journal files. MUPIP BACKUP creates a backup copy of the database. You should backup journal files separately.

## GT.M Journaling

MUPIP BACKUP uses the `-BKUPDBJNL` and `-NEWJNLFILES` to interact with journal files. As stated in the General Database Management chapter, `BKUPDBJNL` enables or turns off the journaling characteristics of the backup database and `NEWJNLFILES` sets the journaling characteristics of the database being backed up. The following illustration describes how `MUPIP BACKUP -NEWJNLFILES=NOPREVLINK` cuts the back link between the newly created journal file and the prior generation journal files.



Since `-NEWJNLFILES=NOPREVLINK` cuts back link of the newly created journal file, any subsequent recovery or rollback will not be able to go back past this discontinuity.



### Note

When `MUPIP SET` changes the journal state from `DISABLED` or `OFF` to `ON`, `GT.M` creates new journal files with no back-links which, like the above example, indicates a fresh start of journaling for the database.

## Select database files for Journaling

You should journal any databases whose integrity you care about. Conversely, you need not journal any database that you are prepared to delete in the event of an untoward event like a system crash.

FIS recommends considering the following aspects before you select database files for Journaling.

- **Always journal data that is worth preserving:** You can journal some or all database files. A quickly understood method of selecting database files for Journaling is as follows:
  - Do not journal any database that you are prepared to delete in the event of an untoward event like a system crash. Never journal temporary data.
  - Truly static does not require journaling but produces no journal impact when held in journaled regions.
  - Move temporary information to separate database files that do not require journaling. If the globals contains process-local(temporary) information or possibly static information, move them to one or more separate database files and use other means (for example, `MUPIP CREATE` or `MUPIP BACKUP`) to manage the information in their region(s).
- **Weigh the deltas associated with manual re-entry and automatic re-play of transactions:** Most of the overhead costs associated with recovering from a failure usually derive from maintaining a state of preparedness for the manual recovery and the potential risk to the organization from damage to the information during the relatively infrequent and "abnormal"

handling of a recovery. Therefore, always weigh the cost of reduced computer throughput or alternatively the additional hardware to support journaling with the same level of performance, against the reduced likelihood of a prolonged manual re-entry with its associated drawbacks.

- **Journal both frequently updated globals and infrequently updated globals:** You might journal only heavily updated globals. However, infrequently changed globals generate little additional load and may present significant control problems if not journaled, you might decide that these globals should also be journaled to maintain application integrity.
- **Separate the point of failure:** Always use different disks and different disk controllers (where possible) for the journal and the associated database files.

## Fencing Transactions

The programming practice of fencing logical transactions protects database integrity during a system interruption. A logical transaction is a logical unit that is not complete unless all parts of the transaction are captured. For instance, the logical transaction "transfer funds between accounts" consists of a debit update to one account and a credit update to another account.

Establishing fences around a logical transaction assures that the transaction is committed as a unit, thereby avoiding logical inconsistencies. These logical inconsistencies, sometimes referred to as application-level database integrity problems, manifest themselves as run-time errors, inappropriate branching, and incorrect reports.

The four ACID properties are Atomicity, Consistency, Isolation and Durability. GT.M provides Durability with Journaling and Atomicity, Consistency, and Isolation with TSTART and TCOMMIT commands. The TSTART and TCOMMIT commands are replacements for the ZTSTART and ZTCOMMIT commands. The following table shows the benefits and drawbacks of each set of TSTART/TCOMMIT versus ZTSTART/ZTCOMMIT commands with their application transaction-fencing requirement.

TSTART/TCOMMIT	ZTSTART/ZTCOMMIT
Provide a transaction management facility that is fully ACID-compliant.	Provide journal enhancement to improve the quality of recoveries. With ZTSTART/ZTCOMMIT, programming logic, usually LOCK protocols, must ensure Consistency and Isolation.
All updates stay private until the time of TCOMMIT. This ensures Atomicity.	Atomicity is only ensured (within operationally set parameters) during journal recovery
No cascading rollbacks	A long-running transaction can trigger cascading rollbacks.
TS[TART][:tvexpr] [( [lvn...]   lvn   * ] [ :keyword (keyword...) ]	ZTS[TART][:tvexpr]
TSTART can manage local variable state on restarts.	
Depth of "nested" transactions for TSTART and TCOMMIT is 127.	Depth of "nested" transactions for ZTSTART and ZTCOMMIT is 25.



### Important

The term cascading roll-back describes the situation that occurs when dropping one transaction causes previous transactions to be sequentially dropped, until potentially all transactions are dropped. If an application violates this assumption, a JOURNAL -RECOVER may create a database with application-level integrity problems. M LOCKs ensure the isolation of a sequence of updates from interaction with any other updates. TSTART and TCOMMIT transaction fences implicitly exhibit the required isolation whether fences are used with or without associated LOCKs.

For more information on TSTART/TCOMMIT, refer to the "Commands" chapter of the *GT.M Programmer's Guide* for more information.



## Note

As stated in the beginning of this chapter, ZTSTART and TZTCOMMIT are deprecated in favor of TSTART and TCOMMIT. FIS no longer validate the ZTSTART and ZTCOMMIT functionality so you should always use TSTART and TCOMMIT to fence your transactions.

## Deciding Whether to Use Fencing

You might fence some, all, or no application programs. When you program with fences, it is possible to force a recovery to ignore the fences by using additional qualifiers to MUPIP JOURNAL -RECOVER. The following lists advantages and disadvantages for fencing transactions.

### Fencing Advantages

- Faster recovery
- Minimum risk recovery
- Databases recovered from journals that include fences do not require post-recovery checks and repairs for logical consistency

Note that TSTART/TCOMMIT pairs are the preferred method of fencing; see the sections on Transaction Processing in the *GT.M Programmer's Guide* for addition benefits of this approach.

### Fencing Disadvantages

- Must be programmed into the M code
- If the application is already structured to minimize logical transaction breakage problems, inserting the fencing commands may be a largely mechanical task. In less structured applications, inserting fences immediately "inside" the M LOCKs associated with transactions may provide an excellent first approximation of proper fencing.
- Fencing adds some entries to the journal file(s)
- Fencing may duplicate methods of recovery already established to address these issues
- An application structured so that all information for each logical transaction is stored in a single global node (while other nodes hold only redundant information), permits rebuild programs to completely correct logical inconsistencies. With less restrictive designs, logical inconsistencies may be corrected manually or by using semi-automated techniques.

## VIEW Keywords

GT.M provides the JNLFLUSH and JNLWAIT keywords as arguments to the VIEW command. Normal operation does not require VIEW commands to control journaling. However, under special circumstances, such as debugging, VIEW commands with journal keywords allow an M program to ensure that GT.M has transferred all its updates to the journal file(s).

VIEW "JNLFLUSH":region initiates a complete transfer of all buffered journal records for a given region from memory to the disk. Normally, the transfer of journal buffers to disk happens automatically. The transfer is triggered by room requirements

to hold new journal records and/or the passage of time since the last update. VIEW "JNLFLUSH" (without a specified region) flushes all regions in the current Global Directory.

VIEW "JNLWAIT" causes GT.M to suspend process execution until all updates initiated by the process in all regions have been transferred to the journal file (on disk). Updates within M TRANSACTIONS typically behave as if they included an implicit VIEW "JNLWAIT" with their final TCOMMIT. TRANSACTIONS with a TRANSACTION ID="BATCH" or "BA" are exempted from the implicit "JNLWAIT". Normally, process execution for updates outside of M transactions continues asynchronously with the transfer of journal records to disk.

For more information on the VIEW command, refer to the "Commands" chapter in the *GT.M Programmer's Guide*.

## \$VIEW() Keywords

GT.M provides the JNLACTIVE, JNLFILE, REGION and JNLTRANSACTION keywords as arguments to the \$VIEW function. Normal operation does not require \$VIEW() to examine journaling status. However, under certain circumstances, such as during debugging of logical transaction design and implementation, \$VIEW() may provide a useful tool.

\$VIEW("JNLACTIVE", region) returns a zero (0) indicating journaling is disabled for the region, one (1) indicating journaling is enabled but OFF, or two (2) indicating journaling is enabled and ON for the named region.

\$VIEW("JNLFILE", region) returns the journal file name. If no journal filename has been established it returns a null string. Otherwise it is a fully translated filename.

\$VIEW("REGION", expr) where expr evaluates to a gvn, returns the name of the region associated with the named gvn. This parameter may be used in conjunction with the above two parameters (JNLACTIVE & JNLFILE), to get journaling status in a configuration-independent manner.

\$VIEW("JNLTRANSACTION") returns the difference between the number of ZTSTARTs that have been issued and the number of ZTCOMMITs. If no fenced transaction is in progress, then a zero (0) is returned. This serves an analogous function to \$TLEVEL for transactions that use TSTART and TCOMMIT.

For more information on \$VIEW(), refer to the "Functions" chapter in the *GT.M Programmer's Guide*.

---

## SET

MUPIP SET is the primary utility used to establish and activate journaling (using the -JOURNAL) and replication (using the -REPLICATION).

When GDE creates a Global Directory, it stores either the explicitly specified journaling information, or the GDE default value (refer to "SET -JOURNAL Options" (page 143)) for any unspecified characteristics.

MUPIP CREATE copies existing journaling information from the Global Directory to the database file, establishing journaling characteristics for all GDE supported journal-options.



### Important

GT.M applies journaling information in the Global Directory to a database file only when it is created. Thereafter use MUPIP, or possibly DSE, to change journaling characteristics in database files. Be sure to use

GDE to reflect current journaling needs so that the next time you use MUPIP CREATE you get the desired journaling characteristics.

DSE DUMP -FILEHEADER displays the current values for all established journaling characteristics.

This section provides a description of the MUPIP SET command with specific reference to the journaling related qualifiers. For information on the other MUPIP SET qualifiers, refer to Chapter 5: “*General Database Management*” (page 67).

MUPIP SET -JOURNAL can change some database characteristics when journaling is active for a specific file or region(s). The first run of MUPIP SET -JOURNAL on an older database automatically changes the maximum/minimum journal settings to match those required by the current GT.M version. MUPIP SET operates on either regions or files.

The format for the MUPIP SET command is:

```
MUPIP SE[T] -qualifier... {-F[ILE] file-name|-REG[ION] region-list}
```

The file-specification or region-list identifies the target of the SET. Region-names separated by commas (,) make up a region-list.

To establish journaling characteristics, use the MUPIP SET command with the -[NO]JOURNAL[=journal-option-list] qualifier and one of the following SET object identifying qualifiers:

```
-F[ILE]
-R[EGION]
```

Together with one or more of the SET action qualifiers:

```
-[NO]JOURNAL[=journal-option-list] -REPLICATION=<replication-option>'
```

## SET Object Identifying Qualifiers

The following qualifiers identify the journaling targets:

```
-F[ILE]
```

Specify that the argument to the SET is a file-specification for a single database file. A Journal file's name can now include characters in Unicode.

Old journal files stay open for about 10 seconds after a switch to a new journal file.

```
-R[EGION]
```

Specify that the argument to the SET is a list of one or more region-names, possibly including wildcards, which, through the mapping of the current Global Directory, identifies a set of database files. SET -REGION modifies multiple files when the parameter contains more than one name.

The -REGION qualifier is incompatible with the -FILE qualifier.

```
-J[NLFILE]
```

Specifies that the target for SET is a journal file. The format of the JNLFILE qualifier is:

```
-jnlfile jnl_file [-[no]prevjnlfile=jnlfilename] [-bypass] [-repl_state={on|off}] [-dbfilename=file_name]
```

*jnl\_file* specifies the name of the target journal file.

**-prevjnlfile=*jnlfilename***

Changes the name of the previous generation of the journal file in the header of *jnl\_file* to *jnlfilename* (for example, when moving the previous generation journal file to a different location). The file name can be a full path-name or a relative path name; however, before the file-name is stored in the header, it is expanded to its full path-name.

**-noprevjnlfile**

Cuts the generation link of the journal file *jnl\_file*. The name of the previous generation journal file is nullified in the header of *jnl\_file*. Such an operation is appropriate when it is assured that there will never be a reason for a rollback to the previous generation journal file.

**-bypass**

Override the requirement that database files (or their corresponding journal files) affected by the set command be available standalone.



**Caution**

Changing the previous generation file link when a rollback operation is in progress or when the Source Server is actively replicating, can damage the journal file and hamper recoverability.

**-repl\_state={on|off}**

Change the replication state of a journal file; this command is intended for use only under instructions from your GT.M support provider.

**-dbfilename=*file\_name***

Associates a journal file with a different database file; this command may be useful in arranging unusual RECOVER or ROLLBACK scenarios.

## SETAction Qualifiers

The -JOURNAL and -REPLICATION qualifiers are the only SET qualifiers relevant for journaling. For information on the other MUPIP SET qualifiers, refer to Chapter 5: “General Database Management” (page 67).

**-[NO]J[OURNALL][=*journal-option-list*]**

Enables or disables journaling for the specified database file or region(s). MUPIP SET commands with this qualifier also establish the characteristics for journal files. FIS believes the defaults and minimum for journal file characteristics are in line with current hardware capabilities and suitable for a production environment.

The *journal-option-list* contains keywords separated with commas (,) enclosed in double quotes ". These double quotes are optional when the list contains only one keyword. This option list is a super set of the journal-option-list available through GDE.

- -NOJOURNAL specifies that the database does not allow journaling, or disables journaling for a database that currently has it enabled. It is equivalent to -JOURNAL=DISABLE.

## GT.M Journaling

- -NOJOURNAL does not accept an argument assignment. It does not create new journal files. When a database has been SET -NOJOURNAL, it appears to have no journaling file name or other characteristics.
- -JOURNAL= enables journaling for a database file. -JOURNAL= takes one or more arguments in a journal-option-list. As long as journaling is ENABLED and turned ON at the end of the command, SET -JOURNAL= always creates a new version of the specified journal file(s).
- -NOJOURNAL specifies that the database does not allow journaling, or disable journaling for a database where journaling is active.
- Enable BEFORE\_IMAGE or NOBEFORE\_IMAGE journaling for a database file.
- As long as journaling is ENABLED and turned ON at the end of the command, SET -JOURNAL= always creates a new version of the specified journal file(s).
- Every MUPIP SET -JOURNAL command on a database file that specifies an ON or OFF journal-activation option causes the values of all explicitly specified journal-file-options to be stored in the database overriding any previously established characteristics for those options.
- If you specify both -JOURNAL and -NOJOURNAL in the same command line, the latter takes effect.
- Whenever MUPIP SET creates a new journal file, it uses all values for journal-file-options that the user explicitly specifies in the command line for the new journal file. If you do not specify a journal-file-option, MUPIP SET takes the characteristics of the existing journal file.
- MUPIP SET supports qualifiers (like -ACCESS\_METHOD, and so on) to change non-journaling characteristics of database file(s). If you specify these qualifiers -JOURNAL, MUPIP SET modifies the non-journaling characteristics first and then moves on to modify the journaling characteristics. Command execution stops when it encounters an error. If MUPIP SET encounters an error in processing the command line or the non-journaling characteristics, it makes no changes to any characteristics. However, if MUPIP SET encounters an error in processing the journaling characteristics, the non-journaling characteristics have already been successfully changed.
- -NOJOURNAL is equivalent to -JOURNAL=DISABLE.
- -NOJOURNAL does not accept an argument assignment. It does not create new journal files. When a database has been SET -NOJOURNAL, it appears to have no journaling file name or other characteristics.

For details on the *journal-option-list* refer to “SET -JOURNAL Options ” (page 143).

### **-REPLI[CATION]=*replication-option***

-REPLICATION sets journal characteristics and changes the replication state simultaneously. It can also be used with the -JOURNAL qualifier. If journaling is ENABLED and turned ON, SET -REPLICATION=ON creates new set of journal files, cuts the back-link to the prior generation journal files, and turns replication ON.

## SET -JOURNAL Options

### **ALI[GNSIZE]=*blocks***

- Specifies the number of 512-byte-blocks in the ALIGNSIZE of the journal file.
- If the ALIGNSIZE is not a perfect power of 2, GT.M rounds it up to the nearest power of 2.

## GT.M Journaling

- The default and minimum ALIGNSIZE value is 4096 blocks. The maximum value is 4194304 (=2 GigaBytes).
- A journal file consists of a sequential stream of journal records each of varying size. It is typically not easy to detect the beginning of the last valid journal record in an abnormally terminated journal file (for example, system crash). To facilitate journal recovery in the event of a system crash, the GT.M run-time system ensures that offsets in the journal file which are multiple of ALIGNSIZE (excepting offset 0 which houses the journal file header) is always the beginning of a valid journal record. In order to ensure this, GT.M run-time system writes padding data (if necessary) in the form of ALIGN journal records just before the ALIGNSIZE boundary. These ALIGN records also help in skipping past invalid records in the middle of a journal file allowing MUPIP JOURNAL -EXTRACT -FORWARD -FULL to extract as much data of a corrupt journal file as possible.
- While a larger align size trade off crash recovery time in favor of increased journaling throughput, especially when before image journaling is in use, there is marginal value in using an align size larger than a few MB.
- The minimum ALIGNSIZE supported is always be greater than or equal to the maximum journal record size, which in turn depends on the maximum database block size.
- Note that a large value of ALIGNSIZE implies less aligned boundaries for recovery to use and hence slows backward recovery down so drastically that, for example, the maximum value of 4194304 causes backward recovery (in case of a crash) to take as much time as forward recovery on that journal file.

### ALL[OCATION]=*blocks*

Sets the allocation size of the journal file. GT.M uses this information to determine when it should first review the disk space available for the journal file. The size of the journal file at creation time is a constant (depending on the GT.M version) but once the journal file reaches the size specified by ALLOCATION, every extension produces a check of free space available on the device used for the journal file.

GT.M issues informational messages to the system log whenever the free space available is not much more than the extension size. GT.M provides these extension checks as an operational aid for identifying, before space runs out, that a file system holding the journal file is low on space. When there is no more free space available on the file system holding a journal file, GT.M shuts off journaling for the corresponding database file.

The default ALLOCATION value is 2048 blocks. The minimum value allowed is 2048. The maximum value is 8,388,607 (4GB-512 bytes, the maximum journal file size).

### AU[TOSWITCHLIMIT]=*blocks*

Specifies the limit on the size of a journal file. When the journal file size reaches the limit, GT.M automatically performs an implicit online switch to a new journal file.



#### Note

It is possible to set the AUTOSWITCHLIMIT to a value higher than the maximum file size (in blocks) for the file system. Currently GT.M does not attempt to check for this condition at specification time. GT.M produces a run-time error when a journal file reaches the maximum size for the file system. Therefore, ensure that the AUTOSWITCHLIMIT never exceeds the file-system limit.

The default value for AUTOSWITCHLIMIT is 8386560 & the maximum value is 8388607 blocks (4GB-512 bytes). The minimum value for AUTOSWITCHLIMIT is 16384. If the difference between the AUTOSWITCHLIMIT and the allocation value is not a multiple of the extension value, GT.M rounds-down the value to make it a multiple of the extension value and displays an

## GT.M Journaling

informational message. GT.M produces an error when the rounded value of AUTOSWITCHLIMIT is less than the minimum value.

If you specify values for ALLOCATION, EXTENSION, and AUTOSWITCHLIMIT for a region such that (ALLOCATION +EXTENSION>AUTOSWITCHLIMIT), either using GDE or MUPIP SET -JOURNAL, GT.M sets ALLOCATION to match the AUTOSWITCHLIMIT, and produces a JNLALLOGGROW message.

At journal extension time, including journal autoswitch time, if (ALLOCATION+EXTENSION>AUTOSWITCHLIMIT) for a region, GT.M uses the larger of EXTENSION and AUTOSWITCHLIMIT as the increment to warn of low available journal disk space. Otherwise, it uses EXTENSION.

### [NO]BEFORE\_IMAGES

Controls whether the journal should capture BEFORE\_IMAGES of GDS blocks that an update is about to modify. A SET -JOURNAL=ON must include either BEFORE\_IMAGES or NOBEFORE\_IMAGES in the accompanying *journal-option-list*.

If you specify both NOBEFORE\_IMAGES and BEFORE\_IMAGES in the same journal-option-list, the last specification overrides any previous one(s).

As GT.M creates new journal files only with the ON option and every ON specification must include either BEFORE\_IMAGES or NOBEFORE\_IMAGES. If the user specifies [NO]BEFORE\_IMAGES along with the OFF option serve no purpose.

Although it is possible to perform an online switch of a database from (or to) NOBEFORE-IMAGE journaling to (or from) BEFORE-IMAGE journaling, it is important to understand that backward recovery can never succeed if it encounters even one in a set of journal files for a database without BEFORE-IMAGES.

### BU[FFER\_SIZE]=*blocks*

Specifies the amount of memory used to buffer journal file output.

MUPIP requires standalone access to the database to modify BUFFER\_SIZE. Therefore, GT.M restricts the use of the BUFFER\_SIZE option to change the current journal-buffer-size as part of an online switch of the journal files.

The default value is 2308 blocks. The minimum BUFFER\_SIZE is 2307 blocks. The maximum BUFFER\_SIZE is 32K blocks which means that the maximum buffer you can set for your journal file output is 16MB.

### DISABLE

Equivalent to the -NOJOURNAL qualifier of MUPIP SET. It specifies that journaling is not an option for the region or file named. If the user specifies DISABLE, then MUPIP SET ignores all other options in the *journal-option-list*.

### ENABLE

Makes the database file or region available for journaling. By default, ENABLE turns journaling ON unless OFF is specified in the same option list. A command that includes ENABLE must also specify BEFORE\_IMAGES or NOBEFORE\_IMAGES.

### EP[OCH\_INTERVAL]=*seconds*

*seconds* specifies the elapsed time interval between two successive EPOCHs. An EPOCH is a checkpoint, at which all updates to a database file are committed to disk. All journal files contain epoch records.

A smaller EPOCH\_INTERVAL reduces the time to recover after a crash at the cost of increased I/O load on the run-time system (due to more frequent checkpoints). A larger EPOCH\_INTERVAL has the opposite effect. Therefore, set EPOCH=interval for a more efficient run-time with larger values of interval and more efficient ROLLBACK processing with smaller values of interval.

## GT.M Journaling

The default EPOCH\_INTERVAL value is 300 seconds (5 minutes). The minimum value is 1 second. The maximum value is 32,767 (one less than 32K) seconds, or approximately 9.1 hours. If you enable journaling and do not specify a value for EPOCH\_INTERVAL, GT.M inherits the value of EPOCH\_INTERVAL of the last journal file in that region. EPOCH\_INTERVAL only makes takes effect when the user turns journaling ON and there is no earlier journal file.

**EX[TENSION]=*blocks***

*blocks* specifies the size of the journal file extension by which file expands and becomes full.

**EXTENSION=*blocks*** specifies when GT.M should review disk space available for the journal file after the ALLOCATION has been used up. It also specifies how much space should be available at each review.

As UNIX file systems use lazy allocation schemes, allocation and extension values do not result in physical disk block allocation for the journal file.

The values determine when GT.M checks the file systems to see if it has enough space to hold an extension worth of journal data. When a journal file reaches the size of ALLOCATION and any multiple of EXTENSION, GT.M checks the file system for room, and if the available space is less than three times the EXTENSION, it writes warnings to the operator log. GT.M provides these extension checks as an operational aid for identifying, before space runs out, that a file system holding the journal file is low on space. When there is no more free space available ) on the file system holding a journal file or there is no authorization of a process attempting to autoswitch a journal file, GT.M shuts off journaling for the corresponding database file.

The default EXTENSION value is 2048 blocks. The minimum EXTENSION is zero (0) blocks and the maximum is 1073741823 (one less than 1 giga) blocks.

**F[ILENAME]=*journal\_filename***

*journal\_filename* specifies the name of the journal file. FILENAME is incompatible with SET -REGION, if you specify more than one region.

GT.M treats the filename as having two components - basename and extension. The format of the journal filename is *basename.extension*, where extension does not contain any periods (.), but if the filename contains more than one period (.), *basename* contains all but the last period (.). Also note that "extension" is the empty string ("") if the filename does not contain any periods (.).

The convention of the default value for the FILENAME is as follows:

- GT.M takes the basename of the database filename as the basename for the journal file with an extension of mjl if the database has a dat extension. For example, database name *mumps.dat* results in a default name *mumps.mjl*. If the database filename does not have a dat extension, GT.M replaces all occurrences of periods (.) with underscores (\_) with an extension of mjl and takes the full database filename. For example, database name *mumps.acn* results in a default name *mumps\_acn.mjl*. Therefore, by default, a journal file has an extension of mjl unless you explicitly specify a different extension with the FILENAME journal option. If the new journal filename (the one specified in the FILENAME option or the default) already exists, GT.M renames the existing file with the string "\_YYYYJJJHHMMSS" appended to the existing file extension where the string denotes the time of creation of the existing journal file in the following format:

YYYY	4-digit-year	such as 2011
JJ	3-digit-Julian-day (between 1 and 366)	such as 199
HH	2-digit-hour in 24 hr format	such as 14
MM	2-digit minute	such as 40
SS	2-digit seconds	such as 30

Assuming the above example for the string value, GT.M renames a journal file *mumps.mjl* to *mumps.mjl\_2010199144030* when it switches to a new journal file.

## GT.M Journaling

- If GT.M detects that the rename-logic yields a filename that already exists, the string "\_N[N[N[N...]]]" is appended to the renamed filename where "N[N[N...]]" denotes the sequence of numbers 0,1,2,3,4,5,6,7,8,9,90,91,92,93,94,95,96,97,98,99,990,991,...

GT.M tries all numbers from the order in the above sequence until it finds a non-existing rename-filename.

Taking the same example as above, in case mumps.mjl\_2010199144030 and mumps.mjl\_2010119144030\_0 already exists, the rename string would be mumps.mjl\_2010199144030\_1.

- If the existing file renaming scheme or the default journal file naming scheme discussed above results in a filename longer than 255 characters (due to the suffix creation rules), GT.M produces an error and turns off journaling.

A journal file name can include characters in Unicode.



### Note

Whenever GT.M implicitly turns off journaling due to run-time conditions such as no available disk space or no authorization for a process attempting to auto-switch a journal file (and so on), it produces an error and accompanying messages identify the reason for that condition.

For journal recovery, GT.M maintains a field in every journal file's header that stores the name of the previous generation journal file for the same database file. When a MUPIP SET changes the journal state from DISABLED or OFF to ON, GT.M creates new journal files with no previous generation journal file name. This indicates that this is a fresh start of journaling for the particular database. When journaling is already ON, and GT.M is implicitly (due to AUTOSWITCHLIMIT being reached) or explicitly (due to MUPIP SET JOURNAL) required to create new journal files, GT.M maintains the previous generation journal filename (after any appropriate rename), in the new journal file's header.

In all cases where journaling is ON both before and after a journal file switch, GT.M maintains the previous generation journal file name in the new journal file's header except when GT.M creates a new journal file due to an implicit switch because it detects an abnormal termination of the current journal file or if the current journal file was not properly closed due to a system crash and the database was the subject of a MUPIP RUNDOWN afterwards.



### Note

In the event of a crash, FIS strongly recommends performing a MUPIP JOURNAL ROLLBACK on a database with replication, MUPIP JOURNAL RECOVER on a journaled database, and MUPIP RUNDOWN only if using neither journaling nor replication. GT.M error messages provide context-specific instructions to promote this decision-making model which helps protect and recover data after a crash.

The previous generation journal filename is a back link from the current generation journal.

GT.M produces an error and makes no change to the journaling state of the database when the FILENAME is an existing file and is not the active journal file for that database. In this way, GT.M prevents possible cycles in the back-links (such as, a3.mjl has a back-link to a2.mjl which in turn has a back-link to a1.mjl which in turn has a back-link to a3.mjl thereby creating a cycle). Cycles could prevent journal recovery. Also, note that cycles in back-links are possible only due to explicit FILENAME specifications and never due to an existing FILENAME characteristics from the database or by using the default FILENAME.

## NOPREVJNLFILE

Eliminates the back link of a journal file.

[NO]S[YNC\_IO]

## GT.M Journaling

Directs GT.M to open the journal file with certain additional IO flags (the exact set of flags varies by the platform where SYNC\_IO is supported, for example on Linux you might utilize the O\_DIRECT flag). Under normal operation, data is written to but not read from the journal files. Therefore, depending on your actual workload and your computer system, you may see better throughput by using the SYNC\_IO journal option.

You should empirically determine the effect of this option, because there is no way to predict the performance gain or impact in advance. There is no functional difference in GT.M behavior with the use of SYNC\_IO. If you determine that different workloads perform best with a different setting of SYNC\_IO, you can change it with MUPIP SET at any time.

The default is NOSYNC\_IO. If you specify both NOSYNC\_IO and SYNC\_IO in the same journal-option-list, GT.M uses the last occurrence.

### OFF

Stops recording subsequent database updates in the journal file. Specify OFF to establish journaling characteristics without creating a journal file or starting journaling.

The default for SET -JOURNAL= is ON.

### ON

Records subsequent database updates in that journal file. MUPIP SET -JOURNAL=ON must include either BEFORE\_IMAGES or NOBEFORE\_IMAGES in the accompanying journal-option-list. By default GT.M sets journal operation to BEFORE\_IMAGE if this command changes the database replication state (refer to Chapter 7: “Database Replication” [173] for more information) from OFF to ON and JOURNAL=NOBEFORE\_IMAGE is not specified.



### Important

ON keyword works only on previously ENABLED regions. GT.M ignores ON if Journaling is DISABLED. In other words, an ENable / DISable is like the power switch on the back of many television sets and ON/OFF is like the ON/OFF on the remote control. The ON/OFF on the remote control works only when the power switch on the back of the television set is enabled.

If the current generation journal file is damaged/missing, MUPIP SET -JOURNAL=ON implicitly turns off journaling for the specified region, creates a new journal file with no back pointers to the prior generation journal file, and turns journaling back on. Further, if replication is enabled, MUPIP SET -JOURNAL=ON temporarily switches the replication WAS\_ON state in the time window when MUPIP SET command turns off journaling and returns normal as long as it operates out of the journal pool buffer and doesn't need to reference the damaged journal file(s). During this operation, MUPIP SET -JOURNAL=ON also sends the PREJNLLINKCUT message for the region to the application and the operator log. While this operation ensures that journaling continues even if the current generation journal file is damaged/missing, creating a new journal file with no back pointers creates a discontinuity with the previous journal files. Therefore, FIS recommends taking a database backup at the earliest convenience because a MUPIP RECOVER/ROLLBACK will not be able to go back past this discontinuity. Also, consider switching the journal files on all regions in the instance (with REGION "") to ensure the RECOVER/ROLLBACK for other regions remains unaffected.

The default for SET -JOURNAL= is ON.

### Y[IELD\_LIMIT]=*yieldcount*

*yieldcount* specifies the number of times a process that tries to flush journal buffer contents to disk yields its timeslice and waits for additional journal buffer content to be filled-in by concurrently active processes, before initiating a less than optimal I/O operation.

A smaller YIELD\_LIMIT is appropriate for light load conditions while larger values are appropriate as the load increases.



## Note

A small YIELD\_LIMIT may cause performance loss due to partial page writes while a large YIELD\_LIMIT may cause performance loss due to significant idle times (due to a lot of yields).

The minimum YIELD\_LIMIT is zero (0), the maximum YIELD\_LIMIT is 2048 and the default YIELD\_LIMIT is 8.

As the disk can only write entire blocks of data, many I/O subsystems perform a READ-MODIFY-WRITE operation when data to be written is a partial block as opposed to simple writes for an entire block. The YIELD\_LIMIT qualifier tries to reduce the frequency of sub-optimal partial block writes by deferring such writes as much as possible in the hope that in the meantime the journal buffer accumulates more content and qualifies for an optimal entire block write.

## Examples for MUPIP SET

```
$ mupip set -journal="enable,nobefore" -file mumps.dat
```

This example enables NOBEFORE\_IMAGE journaling on mumps.dat. If journaling is already enabled, this command switches the current journal file.

Example:

```
$ mupip set -journal=on,enable,before -region "*"
```

This example turn on journaling with BEFORE\_IMAGE journaling. If journaling is already enabled, this command switches the current journal file for all regions.

```
$ mupip set -file -journal="nobefore,buff=2307" gtm.dat
```

This example initiates NOBEFORE\_IMAGE journaling for the database file gtm.dat with a journal buffer size of 2307 blocks. It also switches to new journal file. This command assumes that some prior MUPIP SET -JOURNAL specified ENABLE for gtm.dat.

Example:

```
$ mupip set -region -journal=enable,before_images,allocation=50000,ext=5000 "*"
```

This example enables journaling with BEFORE\_IMAGES on all regions of the current Global Directory and gives each journal file an ALLOCATION of 50000 blocks and an EXTENSION of 5000 blocks. If the regions have significantly different levels of update, use several MUPIP SET -FILE or -REGION commands.

Example:

```
$ mupip set -region -journal="enable,before" areg,breg
```

This example declares journaling active with BEFORE\_IMAGES for the regions areg and breg of the current Global Directory.

Example:

```
$ mupip set -file -nojournal mumps.dat
```

## GT.M Journaling

This example disables journaling on the database file mumps.dat.

Example:

```
$ mupip set -journal="ENABLE,BEFORE_IMAGES" -region "AREG"  
$ mupip set -journal="ON,BEFORE_IMAGES" -region "*"
```

This example turns on journaling only for the region AREG. Note that AREG is the only region that is "available" for journaling.

Example:

```
$ mupip set -access_method=MM -file gtm.dat
```

This example sets MM (Memory Mapped) as the access method or GT.M buffering strategy for storing and retrieving data from the database file gtm.dat. Since MM is not supported with BEFORE\_IMAGE journaling, this example produces an error on a database with BEFORE\_IMAGE journaling enabled. You can also use -access\_method=BG to set BG (Buffered Global) as your buffering strategy. For more information on the implications of these access methods, refer to "Segment Qualifiers" (page 58).

Example:

```
$ mupip set -journal=before,noprevjnlfile,file=newmumps.mjl -file mumps.dat
```

The above command cuts the back link of the newly created journal file newmumps.mjl.

---

## JOURNAL

MUPIP JOURNAL command analyzes, extracts from, reports on, and recovers journal files. The format for the MUPIP JOURNAL command is:

```
MUPIP J[OURNAL] -qualifier[...] file-selection-argument
```

*file-selection-argument* is a comma-separated list of journal files.

**-qualifier [...]** is a combination of Action, Direction, Time, Sequence Number, Control, and Selection qualifiers that perform various MUPIP JOURNAL operations. To create any MUPIP JOURNAL command, select an appropriate combination of qualifiers by moving horizontally from the Action column extending to the Selection column:

Action	Direction	Time (optional)	Sequence Number (optional)	Control (optional)	Selection (optional)
One or more -EX[TRACT] [=file specification] -REC[OVER] -RO[LLBACK] -SH[OW] [=show option list] -[NO]V[ERIFY]	Only one -BA[CKWARD] -FO[RWARD]	One or more -A[FTER]=time -BE[FORE]=time -[NO] LOO[KBACK_LIMIT] [=lookback option list] -SI[NCE]=time	Only one -FE[T]CHRESYNC] = port number -RES[YNC] = jnl sequence number	One or more -[NO]API[PLY_AFTER_IMAGES] -BR[OKENTRANS] = extract file name -[NO] CHA[IN] -[NO] CHE[CKTN] -[NO] ER[ROR_LIMIT] [=integer] -FE[NCES] = fence option -FU[LL] -[NO] IN[TERACTIVE] -LOST[TRANS] = extract file name -RED[IRECT] = file pair list -VERB[OSE] -DE[TAIL]	One or more -G[LOBAL]= global list -ID= pid list -T[RANSACTION] = transaction type -U[SER]= user list

## GT.M Journaling

Also ensure that you adhere to the following rules:

1. -BEFORE is compatible with all other JOURNAL qualifiers except -ROLLBACK.
2. -AFTER is incompatible with -BACKWARD and all action qualifiers, except -EXTRACT, -SHOW, and -VERIFY.
3. -APPLY\_AFTER\_IMAGE is compatible only with -RECOVER, or -ROLLBACK.
4. -BACKWARD is incompatible with -FORWARD, -AFTER, -CHECKTN, -NOCHAIN, and -REDIRECT.
5. -BROKENTRANS is compatible only with -RECOVER, -ROLLBACK, or -EXTRACT.
6. -CHAIN is only compatible with -FORWARD.
7. -DETAIL is compatible only with -EXTRACT.
8. -FETCHRESYNC or -RESYNC are compatible only with -ROLLBACK.
9. -FORWARD is incompatible with -BACKWARD, -ROLLBACK, -SINCE, and -LOOKBACK\_LIMIT.
10. -FULL is compatible only with -EXTRACT, -SHOW, or -VERIFY.
11. -LOSTTRANS is compatible only with -RECOVER, -ROLLBACK, or -EXTRACT.
12. -REDIRECT is compatible only with -RECOVER.
13. -ROLLBACK is incompatible with -RECOVER, FORWARD, -CHAIN, -CHECKTN, -REDIRECT, time qualifiers of -SHOW.
14. -SINCE is incompatible with -FORWARD.
15. -TRANSACTION is compatible only with -EXTRACT and -SHOW.
16. -USER is compatible only with -EXTRACT and -SHOW.
17. file list must not be asterisk (\*) for -REDIRECT.
18. file list must be asterisk (\*) for -ROLLBACK.
19. Journal selection qualifiers are incompatible with -RECOVER, -ROLLBACK, and -VERIFY.
20. Journal time qualifiers are incompatible with -ROLLBACK.

For example, `MUPIP JOURNAL -EXTRACT=gtm.mjf -FORWARD -DETAIL` is a valid command which performs forward processing to extract detailed the journal records to gtm.mjf. However, `MUPIP JOURNAL -EXTRACT -REDIRECT=gtm.dat=test/gtm.dat -FORWARD` is an invalid command because -REDIRECT is not compatible with -EXTRACT.

MUPIP JOURNAL manipulates an inactive journal file that is available for exclusive (standalone) use. You can transcribe Journal files to tape. However, you must always restore them to disk for processing by MUPIP JOURNAL.

Press CTRL+C to stop JOURNAL processing. A JOURNAL command that terminates abnormally by operator action or error produces an incomplete result. In this case, the resulting database may be corrupt. If you stop a JOURNAL operation by mistake, reissue the command to produce the proper result for -RECOVER (or -ROLLBACK) -BACKWARD. For -RECOVER -FORWARD, restore the database from backup and reissue the command.

## Journal Action Qualifiers

This section describes the journaling action qualifiers.

### **-EXtract[=<file-name>|-stdout]**

Transfers information from journal files into files formatted for processing by M routines. It reports the journal time stamps using the \$H format, as controlled by the time zone setting from the OS and the process environment for the process running the EXTRACT.

-EXTRACT takes <file-name> or -stdout as an optional argument.

<file-name> specifies the name of the output file. -stdout specifies that -EXTRACT write to standard output (stdout) instead of writing to a file.

With no arguments, MUPIP JOURNAL derives the output file specification of the extract file using the name of the first journal file that is processed in the forward processing phase and a file type of .mjf. Note that, if multiple journal names are specified in the command line the first journal specified might be different from the first journal processed in the forward phase. When -EXTRACT is specified with -RECOVER (or -ROLLBACK), the -JOURNAL command extracts all the journal records processed during a -RECOVER -FORWARD command or the forward phase of (-RECOVER or -ROLLBACK) -BACKWARD command.

-EXTRACT applies to forward processing of the journal file; if the combined state of the journal file and the Journal Time qualifiers does not cause forward processing, -EXTRACT does not create an output file.

When used independent of -RECOVER (or -ROLLBACK), -EXTRACT option can produce a result even though the database file does not exist, although it does try to access the database if it is available.

If a database having custom collation is inaccessible or the replication instance is frozen with a critical section required for the access held by another process and the environment variable gtm\_extract\_nocol is defined and evaluates to a non-zero integer or any case-independent string or leading substrings of "TRUE" or "YES", MUPIP JOURNAL -EXTRACT issues the DBCOLLREQ warning and proceeds with the extract using the default collation. If gtm\_extract\_nocol is not set or evaluates to a value other than a positive integer or any case-independent string or leading substrings of "FALSE" or "NO", MUPIP JOURNAL -EXTRACT exits with the SETEXTRENV error if it encounters such a situation. Note that if default collation is used for a database with custom collation, the subscripts reported by MUPIP JOURNAL -EXTRACT are those stored in the database, which may differ from those read and written by application programs.

Note that, a broken transaction, if found, is extracted to a broken transaction file (refer to “Journal Control Qualifiers” (page 163) for details), and all future complete transactions are considered as lost transactions, and are extracted to a lost transaction file (refer to “Journal Control Qualifiers” (page 163) for details).

To avoid broken transaction or lost transaction processing and instead extract all journal records into one file, use the control qualifier -FENCES=NONE. FIS strongly recommended against using -FENCES=NONE if -RECOVER/-ROLLBACK is also specified.

### **-RECover**

Instructs MUPIP JOURNAL to initiate database recovery. -RECOVER initiates the central JOURNAL operation for non-replicated database. From the list of JOURNAL action qualifiers, select RECOVER alone or with any other action qualifiers except -ROLLBACK.

-RECOVER -FORWARD with time qualifiers initiates forward recovery. Forward recovery ignores the current journaling state of the target database file. It disables journaling of the target database file, (if currently ENABLE and ON), while playing

## GT.M Journaling

forward the database updates. However, it restores the journaling state of the database at the end of a successful recovery (if necessary), except when journaling is ENABLE'd and ON before the recovery. In the latter case, the journaling state at the end of a successful recovery, is switched to ENABLE and OFF. No journaling is performed for the logical updates to the database for JOURNAL -RECOVER -FORWARD. If the target database's current transaction number is less than first transaction number to be processed in the specified journal file for that region, -RECOVER attempts to include previous generation journal file(s) in its processing, unless the -NOCHAIN qualifier is specified. Following the successive previous links of journal files -RECOVER tries to include previous generations of journal files until the transaction number when the journal file was created is less than, or equal to that of the target database. -RECOVER issues one or more informational messages when it includes previous generation journal files. If target database's current transaction number is not equal to the first transaction number of the earliest journal file to be processed for a region, -RECOVER exits with an error. If multiple journal files for a single region are specified with -RECOVER -FORWARD, it behaves as if -NOCHAIN was specified. If the journal files are not a complete set (for example mumps1.mjl and mumps3.mjl were specified, with mumps2.mjl missing from the command line), MUPIP JOURNAL produces an error because the journal files specified are discontinuous in terms of database transaction numbers. On the other hand, specifying just mumps3.mjl automatically includes mumps2.mjl and mumps1.mjl in the recovery.

-RECOVER -BACKWARD with time qualifiers initiates backward recovery. For backward recovery, the target database file should be the same as when GT.M wrote the last complete transaction to the journal. Because the database may be in an indeterminate state due to a failure, exact checks for this match are not possible. If the target database has journaling DISABLE'd (or ENABLE, OFF), -RECOVER -BACKWARD exits with an error message.

If the target database has journaling ENABLE, ON, but the journal file name in database file header does not match the latest generation journal file name specified for that region, -RECOVER exits with an error.

During forward processing phase of JOURNAL -RECOVER -BACKWARD, MUPIP journals the logical updates to the database. It also creates before images. It is always required to have journaling ENABLE'd and ON for -RECOVER -BACKWARD or -ROLLBACK.

If a transaction is found with incomplete fence, it is considered broken. During forward phase of recovery, if a complete transaction (fenced or unfenced) is found after a broken transaction. -RECOVER increments the error count. If -ERRORLIMIT is reached, the complete transaction goes to lost transaction file, otherwise, it is applied to the database.

All broken and lost transactions are made available as the result of the -RECOVERY. They are written as journal extract format in two different text files. They are the broken transaction file and the lost transaction file. Refer to the sections on BROKENTRANS and LOSTTRANS in "Journal Control Qualifiers" (page 163).

When performing JOURNAL -RECOVER with fences (FENCES="PROCESS" or FENCES="ALWAYS"), it is essential for the command to include all the journal files corresponding to the complete set of database files that make up the logical database. If the specified set of journals is incomplete, the recovery reports all transactions that included any missing region as broken. Typically, this means that the results of the recovery are unsatisfactory or even unusable.

MUPIP JOURNAL -RECOVER requires exclusive access to database files before recovery can occur. It keeps the exclusive access to the database files, which means that the database files become inaccessible during the time of recovery.

If time qualifiers are not specified, -BACKWARD -RECOVER/-ROLLBACK performs optimal recovery. An optimal recovery checks whether the database is in a wholesome state and attempts to perform an automatic recovery if there is a crash. If needed, optimal recovery goes back to include some previous generation files in order to get a consistent starting point and then comes forward as far as the available journal record allow it to while preserving consistent application state. At the end, the journaling state of the database stays ENABLE, ON. Note that the gtm script performs an optimal recovery on every run.

When a database file is rolled back by -RECOVER -BACKWARD, the corresponding journal file is also rolled back so that the two are synchronized. -RECOVER -BACKWARD then creates a new journal file. If no forward play of journal records is necessary, the newly created journal file stays empty and the database points to the new journal file. The values for journal

allocation and extension in the new journal file, are copied over from the database. The autoswitchlimit value in the new journal file is the maximum of the autoswitchlimit values of all journal files from the latest generation journal file until the turnaround point journal file generation (turnaround point is the point in the journal file where backward processing stops and forward processing begins). The journal allocation/extension values in the new journal file are picked up from the earliest generation of the set of those journal files sharing the maximum autoswitchlimit value.

GT.M adds a prefix `rolled_bak_` to the journal file whose entire contents are eliminated (rolled back) by `-RECOVER -BACKWARD`. GT.M does not use these files after a successful recovery therefore you might want to consider moving or deleting them. You should never use `rolled_bak*` files for any future database recovery. If there is a need to process `rolled_bak*` files, you should extract the journal records from `rolled_bak*` files and process them using a M program.

### **-ROLLBACK [{-ON[LINE]}|-NOO[NLINE]]**

`-ROLLBACK` initiates the central JOURNAL operation for a replicated database. MUPIP JOURNAL commands may specify `-ROLLBACK` with other action qualifiers but not with `-RECOVER`. If you do not use `-FETCHRESYNC`, the database rolls back to the last consistent state. Only asterisk (\*) qualifier is allowed for the journal file selection, that is, `-ROLLBACK` selects journal files by itself.

### **-NOO[NLINE]**

Specifies that ROLLBACK requires exclusive access to the database and the replication instance file. This means that the database and the replication instance files are inaccessible during a `-ROLLBACK -NOONLINE`.

By default, MUPIP JOURNAL `-ROLLBACK` is `-NOONLINE`.

### **-ON[LINE]**

Specifies that ROLLBACK should run without requiring exclusive access to the database and the replication instance file.

GT.M increments ISV `$ZONLNRLBK` every time a process detects a concurrent MUPIP JOURNAL `-ONLINE -ROLLBACK`.

If the logical state of the database after the completion of MUPIP JOURNAL `-ONLINE -ROLLBACK` matches the logical state of the database at the start of MUPIP JOURNAL `-ONLINE -ROLLBACK`, that is, only removes or commits an uncommitted TP transaction or non-TP mini-transaction, any transaction (TP or Non-TP) incurs a restart.

If MUPIP JOURNAL `-ONLINE -ROLLBACK` changes the logical state of the database, the behavior is as follows:

- `-ONLINE -ROLLBACK` increments ISV `$ZONLNRLBK`
- In a TP transaction including trigger code within a transaction, `-ONLINE -ROLLBACK` restarts the transaction.
- In a non-TP mini-transaction, including within an implicit transaction caused by a trigger, `-ONLINE -ROLLBACK` produces a `DBROLLEDBACK` error, which, in turn, invokes the error trap if `$ETRAP` or `$ZTRAP` are in effect.

Any utility/command attempted while MUPIP JOURNAL `-ONLINE -ROLLBACK` operates waits for ROLLBACK to complete; the `$gtm_db_startup_max_wait` environment variable configures the wait period. For more information on `$gtm_db_startup_max_wait`, refer to “Environment Variables” (page 17).



### **Note**

Because MUPIP ROLLBACK `-ONLINE` can take a database backwards in state space, please make sure that you understand what it is that you intend it to do when you invoke it. FIS developed it as a step towards a much larger project and anticipates that it will not be broadly useful as is.

## GT.M Journaling

-ROLLBACK -BACKWARD exits with an error message for following conditions:

1. Any database region corresponding to a journal file processed has replication state turned OFF. Note that, a configuration where there are replicated regions and at least one non-replicated-but-journaled region is not permitted by the replication source server. The source server errors out at startup on such a configuration without having set up the journal pool. Since all GT.M updates to replicated regions need the source server to have set up a journal pool, no updates are possible until the configuration is changed to have only replicated regions or non-replicated-and-non-journaled regions.
2. Any database region corresponding to a journal file identified by the command argument has journaling state DISABLE'd or ENABLE'd and OFF.
3. Any database region corresponding to a journal file has journal state ENABLE'd and ON, but the journal file name specified in the database file header is different than one identified by the command argument.

If a transaction is found with incomplete fence, it is considered broken. For the duration of the rollback, replication is turned OFF on all regions and turned back ON at the end of the rollback.

During the forward phase of rollback, if a complete transaction (fenced or unfenced) is found after a broken transaction, it is considered as a lost transaction. During forward phase of rollback, MUPIP journals the logical updates to the database. All broken and lost transactions are made available as a result of the rollback. These are written as journal extract format in two different text files.

When a database file is rolled back by -ROLLBACK, the corresponding journal file is also rolled back so that the two are synchronized. -ROLLBACK then creates a new journal file. If no forward play of journal records is necessary, the newly created journal file is empty and the database points to the new journal file. The journal allocation/extension/autoswitchlimit values in the new journal file is set in the way as described for -RECOVER -BACKWARD in the previous section under -RECOVER.

A prefix rolled\_bak\_ is added to the journal file, whose entire contents are eliminated by a -ROLLBACK. These files are not used by GT.M after the MUPIP JOURNAL -RECOVER, and can be moved/deleted as needed.

For -ROLLBACK the target database file should be the same as when GT.M wrote the last complete transaction to the journal.

If the -FETCHRESYNC or -RESYNC qualifiers are not specified, MUPIP does an optimal rollback (that is, check whether the database is in a wholesome state and attempt to automatically recover a database if there is a crash).



### Note

If ROLLBACK (either -NOONLINE or -ONLINE) terminates abnormally (say because of a kill -9), it leaves the database in a potentially inconsistent state indicated by the FILE corrupt field in the database file header. When a ROLLBACK terminates leaving this field set, all other processes receive DBFLCORRP errors any time they attempt to interact with the database. You can clear this condition as following in descending order of risk:

- Rerun ROLLBACK to completion
- MUPIP SET -FILE -PARTIAL\_RECOV\_BYPASS
- DSE CHANGE -FILEHEADER -CORRUPT=FALSE -NOCRIT

However, the MUPIP and DSE actions do not ensure that the database has consistent state; check for database integrity with MUPIP INTEG.

**-SHOW=show-option-list**

Specifies which information for the JOURNAL command to display about a journal file.

Use -FORWARD with -SHOW together (but without -RECOVER ) to process the entire journal file. Specify -SHOW with -RECOVER (or -ROLLBACK) to consider all the journal files/records processed during a -RECOVER -FORWARD command or forward phase of a -RECOVER (or -ROLLBACK) -BACKWARD command. Without -RECOVER (or -ROLLBACK), -SHOW does not require database access.

The show-option-list includes (these are not case-sensitive):

1. **AL[L]**

Displays all the available type of information about the journal file. ALL is the default if you omits the show-option-list.

2. **AC[TIVE\_PROCESSES]**

Displays all processes active at the end of the period specified implicitly or explicitly by the JOURNAL command time qualifiers.

3. **B[ROKEN\_TRANSACTIONS]**

Display all processes that had incomplete fenced transactions at the end of the period covered by the JOURNAL command.

4. **H[EADER]**

Displays the journal file header information. If the MUIPIP JOURNAL command includes only the -SHOW=HEADER action qualifier, GT.M processes only the journal file header (not the contents) even if you specify -BACKWARD or -FORWARD with it. The size of a journal file header is 64K.

HEADER displays almost all the fields in the journal file header. The NODE field is printed up to a maximum of the first 12 characters. The following is an example of SHOW=HEADER output:

```
-----
SHOW output for journal file /home/jdoe/.fis-gtm/V6.0-000_x86/g/gtm.mjl
-----

Journal file name      /home/jdoe/.fis-gtm/V6.0-000_x86/g/gtm.mjl
Journal file label     GDSJNL23
Database file name     /home/jdoe/.fis-gtm/V6.0-000_x86/g/gtm.dat
Prev journal file name /home/jdoe/.fis-gtm/V6.0-000_x86/g/gtm.mjl_2012310190106
Next journal file name

Before-image journal   ENABLED
Journal file header size      65536 [0x00010000]
Virtual file size         2048 [0x00000800] blocks
Journal file checksum seed  2272485152 [0x87735F20]
Crash                    FALSE
Recover interrupted      FALSE
Journal file encrypted    FALSE
Journal file hash         00000000000000000000000000000000
Blocks to Upgrade Adjustment      0 [0x00000000]
End of Data               65960 [0x000101A8]
Prev Recovery End of Data      0 [0x00000000]
Endian Format              LITTLE
```

## GT.M Journaling

```
Journal Creation Time      2012/11/06 17:30:33
Time of last update       2012/11/06 17:30:33
Begin Transaction          1 [0x0000000000000001]
End Transaction            1 [0x0000000000000001]
Align size                 2097152 [0x00200000] bytes
Epoch Interval            300
Replication State          CLOSED
Jnlfile SwitchLimit        8386560 [0x007FF800] blocks
Jnlfile Allocation         2048 [0x00000800] blocks
Jnlfile Extension          2048 [0x00000800] blocks
Maximum Journal Record Length 1048672 [0x00100060]
Turn Around Point Offset   0 [0x00000000]
Turn Around Point Time     0
Start Region Sequence Number 1 [0x0000000000000001]
End Region Sequence Number 1 [0x0000000000000001]
```

Process That Created the Journal File:

PID	NODE	USER	TERM	JPV_TIME
000006706	jdoe-laptop	jdoe	0	2012/11/06 17:30:33

Process That First Opened the Journal File:

PID	NODE	USER	TERM	JPV_TIME
000006706	jdoe-laptop	jdoe	0	2012/11/06 17:30:33

### 5. P[ROCESSES]

Displays all processes active during the period specified implicitly or explicitly by the JOURNAL command time qualifiers.

### 6. S[TATISTICS]

Displays a count of all journal record types processed during the period specified implicitly or explicitly by the JOURNAL command time qualifiers. The following is an example of SHOW=STATISTICS output:

```
-----
SHOW output for journal file /home/jdoe/.fis-gtm/V6.0-000_x86/g/gtm.mjl
-----
```

Record type	Count
*BAD*	0
PINI	2
PFIN	2
ZTCOM	0
KILL	1333533
FKILL	0
GKILL	0
SET	0
FSET	0
GSET	0
PBLK	4339
EPOCH	2
EOF	1

```

TKILL      0
UKILL      0
TSET       0
USET       0
TCOM       0
ALIGN      49
NULL       0
ZKILL      0
FZKIL      0
GZKIL      0
TZKIL      0
UZKIL      0
INCTN      4314
AIMG       0
TZTWO      0
UZTWO      0
TZTRI      0
UZTRI      0
TRUNC      0

```

```
%GTM-S-JNLSUCCESS, Show successful
```

```
%GTM-S-JNLSUCCESS, Verify successful
```

```
%GTM-I-MUJNLSTAT, End processing at Tue Nov 6 17:42:21 2012
```

The following example displays the cryptographic hash of the symmetric key stored in the journal file header (the output is one long line).

```
$ mupip journal -show -backward mumps.mjl 2>&1 | grep hash
```

```

Journal file hash F226703EC502E975784
8EEC733E1C3CABE5AC146C60F922D0E7D7CB5E
2A37ABA005CE98D908B219249A0464F5BB622B72F5FDA
0FDF04C8ECE52A4261975B89A2

```

## -[NO]Verify

Verifies a journal file for integrity. This qualifier cannot have a value. -VERIFY scans the files and checks if it is in legal form, if not, it terminates without affecting the database files.

-VERIFY when specified along with -FORWARD verifies the entire journal file. For -NOVERIFY -FORWARD, only the tail of a journal file is verified for cross region integrity. In both cases, if -RECOVER is also specified, the forward play of journal records is done in a separate pass only after the verification pass is complete and error-free.

-VERIFY along with -BACKWARD verifies all journal records from the end of the journal file till the turn around point. When -VERIFY -BACKWARD is specified along with -RECOVER or -ROLLBACK, backward processing involves two passes, the first pass to do the verification until the turn around point, and the second pass to apply before image (PBLK) records.

When -NOVERIFY -BACKWARD is specified along with -RECOVER or -ROLLBACK, PBLKs are applied to the database in the same pass as the verification. This speeds up processing. But the disadvantage of this approach is that in the event of verification terminating in the middle of backward processing, there is no protection of cross-region integrity. FIS recommends the use of -VERIFY with -RECOVER or -ROLLBACK.

When used independent of -RECOVER (or -ROLLBACK), -[NO]VERIFY option does not need database access. The default is -VERIFY.

## Journal Direction Qualifiers

The following two qualifiers control the journal processing direction:

### -BACKWARD

Specifies that MUPIP JOURNAL processing should proceed from the end of the journal file. If the actions include -RECOVER, JOURNAL -BACKWARD restores before-images from the end-of the file back to an explicitly or implicitly specified point (the turn around point), before it reverses and processes database updates in the forward direction (the forward phase).



#### Note

-BACKWARD is incompatible with -FORWARD.

### -FORWARD

Specifies that MUPIP JOURNAL processing for the specified action qualifier should proceed from the beginning of the given journal file. When processing a -RECOVER action qualifier, in certain cases, MUPIP JOURNAL may need to go before the first record of the specified journal file, that is, it can start from a previous generation journal file (refer to “-RECOVER” (page 153) for details).

If multiple journal files are specified in the command line, -FORWARD sorts the journal files within each region based on creation time and processes them starting from the earliest journal file. Unless the -NOCHECKTN qualifier is specified, -FORWARD performs checks on journal files corresponding to each region to ensure they are contiguous, both in terms of time span, as well as, transaction number span. -FORWARD errors out if it detects a discontinuity.



#### Note

-FORWARD is incompatible with -BACKWARD and -ROLLBACK.

## Journal Time Qualifiers

Journal qualifiers specifying time accept arguments in absolute or delta time format. Enclose time arguments in quotation marks (" "). Include a back-slash (\) delimiter before both, the beginning and ending quotation marks to escape it from being processed by the UNIX shell.

Absolute format is *day-mon-yyyy hh:mm:ss*, where *day* denotes the date of the month, *mon* indicates the abbreviated 3-letter month name (for example, Jan, Feb,..) and the year *yyyy* and hour *hh* are separated by a space. Absolute time may indicate today's date with "--" before the hours.

Delta format is *day hh:mm:ss*, indicating the number of days, hours, minutes, and seconds; where the day and the hours (hh) are separated by a space. If delta time is less than a day, it must start with zero (0) followed by a space.

Delta time is always relative to the maximum time of the last record in all journal files specified by arguments to the MUPIP JOURNAL command.



#### Note

All time qualifiers are incompatible with -ROLLBACK.

The following section describes the time qualifiers in more detail:

**-A[FTER]=time**

Specifies reference time stamps in the journal and identifies the point after which JOURNAL starts processing in the journal file(s). This time qualifier applies to -FORWARD only.

If -AFTER= provides a time following the last time recorded in the journal file or following any -BEFORE= time, JOURNAL processing produces no result and a warning message is displayed. If -AFTER provides a time preceding the first time recorded in the journal file specified in the command line, and, previous generation journal file(s) exists for that journal file, then previous generation journal file(s) are not included for the processing. You must specify previous generation journal files explicitly in the command line in order for them to be considered.

Using -BEFORE with -AFTER restricts processing to a particular period of time in the journal file.

**-BE[FORE]=time**

Specifies an ending time for any action -FORWARD or -BACKWARD. The time specified references time stamps in the journal files. If -BEFORE= specifies a time preceding the first time recorded in the journal file, or preceding any -AFTER= or -SINCE= time, JOURNAL processing produces no result, and a warning message is displayed.

If -BEFORE= time exceeds the last time recorded in journal files, JOURNAL processing effectively ignores the qualifier and terminates at the end of the journal file. By default, JOURNAL processing terminates at the end of the journal file.

**-[NO]LOO[KBACK\_LIMIT][=lookback-option-list]**

Specifies how far JOURNAL -BACKWARD processes past the turnaround point (the explicit or implicit point in journal file up to which -RECOVER proceeds backward before it reverses and processes database in forward direction), while attempting to resolve open transaction fences. This option is applicable only for transactions fenced with ZTSTART and ZTCOMMIT. For transaction fenced with TSTART and TCOMMIT, -RECOVER always resolves open transaction fences.

-LOOKBACK\_LIMIT=options, include time and transaction counts. -NOLOOKBACK\_LIMIT specifies that JOURNAL -BACKWARD can process all the way to the beginning of the journal file, if necessary, to resolve open transaction fences. -LOOKBACK\_LIMIT= is incompatible with -FORWARD.

When -FENCES=NONE, JOURNAL processing ignores -LOOKBACK\_LIMIT.

The -LOOKBACK\_LIMIT options are:

- **TIME=***time*

This limits LOOKBACK by a specified amount of delta or absolute journal time.

- **OPERATIONS=***integer*

This limits LOOKBACK to the specified number of database transactions.

The TIME LOOKBACK option name and its value must be enclosed in quotes ("").

For example:

**-lookback="time=0 00:00:30"**

When `-LOOKBACK_LIMIT=` specifies both options, they must be separated by a comma (,), for example:

```
-lookback="\time=0 00:00:30,operations=35"
```

When `-LOOKBACK_LIMIT=` specifies both options, the first limit reached terminates the LOOKBACK.

By default, MUPIP JOURNAL uses `-LOOKBACK_LIMIT="\TIME=0 00:05\"` providing five minutes of journal time prior to `-SINCE=` to resolve open fences. A `-LOOKBACK_LIMIT` that specifies a limit much before the beginning of the earliest journal file acts as if `-NOLOOKBACK_LIMIT` was specified.

```
-SI[NCE]=time
```

Specifies a starting (or checkpoint) time for an action qualifier with `-BACKWARD`, that is, `-SINCE` specifies how far back in time JOURNAL `-BACKWARD` should process (from the end of the journal file), before starting the forward processing.

The time specified references time stamps in the journal files. If there are open fenced transactions when JOURNAL `-BACKWARD` locates the `-SINCE=` time, it continues processing backward to resolve them, unless the command also specifies `-FENCES=NONE`. If `-SINCE=` time exceeds the last time recorded in the journal files or, follows any `-BEFORE=time`, JOURNAL processing effectively ignores the qualifier, and displays a warning message.

By default, `-SINCE=` time is `0 00:00:00` which denotes the time at the end of the journal file (the time when the last journal record was updated).

## Journal Sequence Number Qualifiers

These qualifiers are compatible only with `-ROLLBACK`.

```
-FET[CHRESYNC]=<port number>
```

In an LMS configuration, rollbacks the replicating instance to a common synchronization point from which the originating instance can transmit updates to allow it to catch up. This command rolls back a former originating instance to the journal sequence number at which the current originating instance took over. The format of the fetchresync qualifier is:

```
-fetchresync=<port number> -losttrans=<extract file> file-list
```

The `<port number>` is the communication port number that the rollback command uses when fetching the reference point. Always use the same `<port number>` on the originating instance for rollback as the one used by the Receiver Server.



### Important

FIS recommends you to unconditionally script the mupip journal `-rollback -fetchresync` command prior to starting any Source Server on the replicating instance to avoid a possible out-of-sync situation.

The reference point sent by the originating instance is the `RESYNC_SEQNO` (explained later) that the originating instance once maintained. The database/journal files are rolled back to the earlier `RESYNC_SEQNO` (that is, the one received from originating instance or the one maintained locally). If you do not use `-fetchresync`, the database rolls back to the last consistent replicating instance state.

The system stores extracted lost transactions in the file `<extract file>` specified by this mandatory qualifier. The starting point for the search for lost transactions is the `JNL_SEQNO` obtained from the originating instance in the `-fetchresync` operation. If `-fetchresync` is not specified, `<extract file>` lists the post-consistent-state transactions that were undone by the rollback procedure to reach a consistent state.



## Note

The extracted lost transactions list may contain broken transactions due to system failures that occurred during processing. Do not resolve these transactions --they are not considered to be committed.



## Caution

The database header may get corrupted if you suspend an ongoing ROLLBACK -FETECHRESYNC operation or if the TCP connection between the two instances gets broken. The workaround is to restart the ROLLBACK -FETCHRESYNC operation or wait 60 seconds for the FETCHRESYNC operation to timeout.

Example:

```
$ mupip journal -rollback -fetchresync=2299 -losttrans="glo.lost" -backward
```

This command performs a ROLLBACK -FETCHRESYNC operation on a replicating instance to bring it to a common synchronization point from where the originating instance can begin to transmit updates to allow it to catch up. It also generates a lost transaction file glo.lost of all those transactions that are present on the replicating instance but not on the originating instance at port 2299.

```
-RES[YNC]=<journal sequence number>
```

Specifies the journal sequence number to which GT.M must rollback the database/journal files need to be rolled back to a specific point. If you specify a journal sequence number that is greater than the last consistent state, GT.M rolls back the database/journal files to the last consistent state. Under normal operating conditions, this qualifier is not needed.

## Journal Control Qualifiers

The following qualifiers control journal processing:

```
-[NO]AP[PLY_AFTER_IMAGE]
```

Specifies that after image records (AIMG) be applied to the database as part of forward processing of backward recovery or rollback. AIMG are "snapshots" of the database updates captured by GTM immediately after the change caused by a DSE update. By default, during forward phase of backward recovery or rollback, AIMG records are applied to the database.

By default, -RECOVER -FORWARD does not apply AIMG record into the database. -APPLY\_AFTER\_IMAGE is compatible with -RECOVER, or -ROLLBACK action qualifiers only.

```
-BR[OKENTRANS]=<extract file>
```

-BROKENTRANS is an optional qualifier for -ROLLBACK, -RECOVER and -EXTRACT. If this is not specified and a broken transaction file creation is necessary, MUPIP JOURNAL creates one using the name of the current journal file being processed with a .broken extension.

Note that, if selection qualifiers are specified, the broken transaction determination (and therefore lost transaction determination as well) is done based on the journal file that is filtered by the selection qualifiers. This means that a transaction's journal records may be considered complete or broken or lost, depending on the nature of the selection qualifiers. Using -FENCES=NONE along with the selection qualifiers will result in every journal record to be considered complete and hence prevent broken or lost transaction processing.

**-[NO]CHA[IN]**

-CHAIN allows JOURNAL processing to include previous generations of journal files with -FORWARD. If JOURNAL -RECOVER needs to process previous generation journal file(s) and -NOCHAIN is specified, MUPIP JOURNAL exits with an error.

-CHAIN is the default.

**-[NO]CHE[CKTN]**

-CHECKTN specifies that JOURNAL -FORWARD must verify for each region that the begin transaction number of the earliest journal file to be processed for that region is same as the current transaction in the database file and that the end transaction number of every journal file is equal to the begin transaction number of the next generation journal file for a given region. By default, -FORWARD uses -CHECKTN.

-NOCHECKTN forces forward recovery by overriding inbuilt mechanisms for checking transaction integrity. Use -NOCHECKTN with caution because it may lead to integrity issues in the recovered database and journal files.

-CHECKTN is incompatible with -BACKWARD and -ROLLBACK.

**-[NO]ER[ROR\_LIMIT][=integer]**

Specifies the number of errors that MUPIP JOURNAL processing accepts. When the number of errors exceeds the -ERROR\_LIMIT, the -INTERACTIVE qualifier determines whether JOURNAL processing halts or defers to the operator. -NOERROR\_LIMIT prevents MUPIP JOURNAL from stopping because of errors. Journal processing continues until it reaches the end of the journal file, regardless of the number of errors.

Note that, -NOERROR\_LIMIT is not the same as -ERROR\_LIMIT=0.

By default, MUPIP JOURNAL uses -ERROR\_LIMIT=0, causing the first error to initiate the appropriate error action. In case of a crash there could be some incomplete journal records at the end of a journal file. MUPIP JOURNAL does not consider these as errors. In addition, fenced transactions that are broken are not considered as errors.

During the forward phase of recovery, if a broken transaction is found, all the logical records processed afterwards are considered suspect. If a complete transaction is found after any broken transactions, MUPIP JOURNAL -RECOVER increments the error count and, if it is less than the error limit, it is applied to the database. Otherwise, it is treated as a lost transaction and extracted. If a complete transaction is found after any broken transactions, MUPIP JOURNAL -ROLLBACK treats it as a lost transaction and extracts it irrespective of the error limit.

If MUPIP JOURNAL needs to increment error count during its processing, a warning message is issued for every error encountered except in the following cases when the error count is incremented but no warning message is displayed:

- When a complete transaction is found after a broken transaction
- When -EXTRACT -FULL encounters errors

If MUPIP JOURNAL completes successfully with a non-zero value of error count, the return status is not a success, but a warning.

**-FE[NCES][=fence-option]**

Specifies how JOURNAL processes fenced transactions. Fenced transactions are logical transactions made up of database updates preceded by a TSTART or ZTSTART command and followed, respectively, by a TCOMMIT or ZTCOMMIT command.

## GT.M Journaling

All updates between a TSTART or ZTSTART and a TCOMMIT or ZTCOMMIT are designed to occur together so that after journal recovery the database contains either all the updates corresponding to a fenced transaction, or none of them.

The argument values for -FENCES option for MUPIP -RECOVER are not case-sensitive.

The fence options are:

- **NONE**

This causes MUPIP JOURNAL to apply all individual updates as if transaction fences did not exist. Note that, this means a SET/KILL within a TP or ZTP transaction would be played as if it was an unfenced SET/KILL. This may cause the database and new journals created (if -BACKWARD is specified), to be different from the state before the recovery took place.

- **ALWAYS**

This causes MUPIP JOURNAL to treat any unfenced or improperly fenced updates as broken transactions.

- **PROCESS**

This causes MUPIP JOURNAL to accept unfenced database updates, and also to observe fences when they appear, generating broken transaction files in the case of a TSTART with no corresponding TCOMMIT, or a ZTSTART with no corresponding ZTCOMMIT. It also generates broken transactions if a multi-region transaction with TSTART and TCOMMIT expects N regions to participate, but the number of TSTART/TCOMMIT pairs found is less than N. The same happens for multi-region ZTSTART and ZTCOMMIT.

By default, MUPIP JOURNAL uses -FENCES=PROCESS.

### -FU[LL]

-FULL when used with -EXTRACT, specifies that all journal records be extracted. A journal file's contents can be rolled back in case of backward recovery or rollback (refer to “-RECOVER ” (page 153) or “-ROLLBACK [{-ON[LINE]}|-NOO[NLINE]] ” (page 155) for more details) in order to keep the database and journal in sync. This is achieved not by truncating the contents of the journal file but instead setting a field in the journal file header, which shows up as "Prev Recovery End of Data" in a MUPIP JOURNAL -SHOW=HEADER output, to indicate the end of the journal file before rolling back and setting another field in the file header to indicate the new end of the journal file (this field shows up as "End of Data" in a MUPIP JOURNAL -SHOW=HEADER output). Once a journal file's contents are rolled back, all future MUPIP JOURNAL commands (including -EXTRACT) operate on the rolled back journal file only. But if -FULL is specified along with -EXTRACT, the entire journal file contents (including those records that were rolled back) are extracted. This qualifier is to be used only as a diagnostic tool and not in normal operation.

-FULL qualifier is compatible with -EXTRACT only.

### -[NO]IN[TERACTIVE]

Specifies whether, for each error over the -ERROR\_LIMIT, JOURNAL processing prompts the invoking operator for a response to control continuation of processing. If the operator responds that processing should not continue, the MUPIP JOURNAL command terminates.

-NOINTERACTIVE terminates the journal processing as soon as the MUPIP JOURNAL command generates the number of errors specified in -ERROR\_LIMIT.

This qualifier is applicable only to interactive mode or terminal mode. The default is -INTERACTIVE.

`-LOST[TRANS]=<extract file>`

`-LOSTTRANS` is an optional qualifier for `-RECOVER`, `-ROLLBACK` and `-EXTRACT`. If this is not specified and a lost transaction file creation is necessary, MUPIP JOURNAL creates one using the name of the current journal file being processed with a `.lost` extension.

Any complete transactions after a broken transaction is considered a lost transaction. They are written into the lost transaction file. For `-RECOVER` it might be considered as good transaction and applied to the database, if `-ERROR_LIMIT` qualifier allows it to do so.

Note that, if selection qualifiers are specified, the broken transaction determination (and therefore lost transaction determination as well) is done based on the journal file that is filtered by the selection qualifiers. This means that a transaction's journal records may be considered complete or broken or lost, depending on the nature of the selection qualifiers. Using `-FENCES=NONE` along with the selection qualifiers will result in every journal record to be considered complete and hence prevent broken or lost transaction processing.

In the case of a replicated database, lost transaction can have an additional cause. If failover occurs (that is, the originating Source Server, A, fails and the replicating Source Server, B, assumes the originating instance's role), some transactions committed to A's database may not be reflected in B's database. Before the former originating instance becomes the new replicating instance, these transactions must be rolled back. These transactions are known as "lost transactions". Note that these are complete transactions and different from a broken transaction. MUPIP JOURNAL `-ROLLBACK` stores extracted lost transactions in the extract-file specified by this qualifier. The starting point for the search for lost transactions is the journal sequence number obtained from the originating Source Server in the `-FETCHRESYNC` operation.

`-RED[IRECT]=file-pair-list`

Replays the journal file to a database different than the one for which it was created. Use `-REDIRECT` to create or maintain databases for training or testing.

This qualifier applies to `-RECOVER` action and `-FORWARD` direction qualifier only. JOURNAL rejects `-REDIRECT` unless it appears with `-RECOVER`.

The file-pair-list consists of one or more pairs of file-names enclosed in parentheses () and separated by commas (.). The pairs are separated by an equal sign in the form:

**old-file-name=new-file-name**

where the old file-name identifies the original database file and the new file-specification file-name identifies the target of the `-RECOVER`. The old-file-specification can always be determined using `-SHOW`.

By default, JOURNAL directs `-RECOVER` to the database file from which the journal was made. `-REDIRECT` is not compatible with `-ROLLBACK`.

Example:

```
$ mupip journal -recover -forward -redirect="bgddb.dat=test.dat" bgddb.mjl
```

This JOURNAL command does a forward recovery that `-REDIRECTs` the updates in `bgddb.mjl` from `bgddb.dat` to `test.dat`.

`-VERB[OSE]`

Prints verbose output in the course of processing. It is not negatable and it is set to OFF by default.

## Journal Selection Qualifiers

Journal Selection Qualifiers are compatible with `-EXTRACT` and `-SHOW` operations only. This is because most applications are not constructed to safely remove a subset of transactions based on criteria that is exterior to the application design. To exclude transactions from a recovery based on some selection criteria, the methodology is to `-EXTRACT` the records, and then reapply them through application logic rather than by journal recovery. This approach permits the application logic to appropriately handle any interactions between the removed and the retained transactions. Note that, selection qualifiers might result in only a subset of a fenced transaction's journal records to be extracted (for example, a `TSTART` record may not be extracted because the first update in that transaction was filtered out by a selection qualifier, while the corresponding `TCOMMIT` record may get extracted). This can cause a fenced transaction to seem broken when actually it is not.

The following qualifiers control the selection criteria for journal processing.

Except for `-TRANSACTION`, all qualifiers allow for specifying a comma (,) separated list of values.

```
-G[GLOBAL]=global-list
```

Specifies globals for MUPIP JOURNAL to include or exclude from processing. You might find this qualifier useful for extracting and analyzing specific data.

The `global-list` contains one or more `global-names` (without subscripts) preceded by a caret symbol (^). To include more than one global use one of the following syntaxes.

```
$ mupip journal -forward -extract -global="^A*,^C" mumps.mjl
```

or

```
$ mupip journal -forward -extract -global="(^A*,^C)" mumps.mjl
```

The names may include the asterisk (\*) wildcard. That is, `-GLOBAL="^A*"` selects all global variables with names starting with A. The entire list or each name may optionally be preceded by a tilde sign (~), requiring JOURNAL to exclude database updates to the specified global(s). When the `global-list` with a MUPIP JOURNAL `-GLOBAL` does not start with a tilde sign (~), JOURNAL processes only the explicitly named globals. By default, JOURNAL processes all globals.

To specify subscripts, using `-GLOBAL="^A(1)"` results in all keys under the `^A(1)` tree to be included, that is, it is equivalent to using `-GLOBAL="^A(1,*)"`. An asterisk (\*) or a percent (%) anywhere in the global specification is permitted. Percent (%) matches any character, and asterisk (\*) matches any string (possibly zero length too). The asterisk (\*) or percent (%) specification can be used for `-USER` qualifier too.

Example:

To extract all `^GBL*` except for `^GBLTMP`:

```
$ mupip journal -extract -global="^GBL*,~^GBLTMP" -forward mumps.mjl
```

To extract all `^GBL` except for `^GBL(1,"TMP")`:

```
$ mupip journal -extract -global="\^GBL,~^GBL\{1,\"\"TMP\"\"\}\" -forward mumps.mjl
```

The backslash (\) delimiter characters are required in UNIX to pass MUPIP the double quotes (") of the string subscript.

An INVGLOBALQUAL error is issued along with the error offset in the command line, whenever a parse error of the global qualifier string is encountered.

`-ID=pid-list`

Specifies that JOURNAL processing include or exclude database updates generated by one or more processes, identified by process identification numbers (PIDs). The entire list or each PID may optionally be preceded by a tilda sign (~), requiring JOURNAL to exclude database updates initiated by the specified PID. You may use this qualifier for trouble shooting or analyzing data.

By default, JOURNAL processes database updates regardless of the PID that initiated it.

`-T[ANSACTION]=transaction-type`

Specifies transaction-types for JOURNAL to include or exclude from processing. For example, you may use this qualifier to report only on KILL operations to locate possible causes for missing data.

The transaction-types are SET and KILL and can be negated. These types correspond to the M commands of the same names. When the transaction-type with a JOURNAL -TRANSACTION is not negated, JOURNAL processes only transactions of the type named (for example, -TRANSACTION=KILL), whereas if it is negated, JOURNAL does not process transactions of the type named (for example, -TRANSACTION=NOKILL).

By default, JOURNAL processes transactions, regardless of its type.

`-U[SER]=user-list`

Specifies that MUPIP JOURNAL processing include or exclude database updates generated by one or more users. You can use this qualifier to audit the actions of a particular user. The user-list contains names of one or more users. Indicate multiple users by separating the names with commas (.). The names may include the wildcard asterisk (\*). The entire list or each name may optionally be preceded by a minus sign (-) tilda sign (~), requiring JOURNAL to exclude database updates initiated by the specified user(s). When the user-list with a JOURNAL -USER does not start with a tilda sign (~), JOURNAL processes only those database updates, which are generated by explicitly named users. The asterisk (\*) or percent (%) specification can be used for -USER qualifier. Percent (%) matches any character, and asterisk (\*) matches any string (possibly zero length too).

By default, JOURNAL processes database updates regardless of the user who initiated them.

---

## Journal Extract Formats

Journal EXTRACT files always start with a label. For the current release of GT.M, the label is GDSJEX06 for a simple journal extract file. This label is necessary to identify the format of the file.

If the environment variable gtm\_chset is set of UTF-8, then file format label is followed by another label called "UTF-8" to indicate UTF-8 mode.

After this label, the journal record extracts follow. These journal record extracts include fields or pieces delimited by a back slash (\).

The first piece of an -EXTRACT output record contains a two-digit decimal transaction record type (for example, 01 for a process initialization record). The second piece contains the full date and time of the operation, represented in the \$HOROLOG format. The third piece contains a GT.M assigned number (database transaction number) which uniquely identifies the transaction within the time covered by the journal file. The fourth piece contains the process ID (PID) of the process that performed the operation, represented as a decimal number. The remainder of the record depends on the record type.

## GT.M Journaling

Records of type SET, KILL, ZKILL, TSTART, and TCOMMIT include the token\_seq as part of the output. It is the sixth field in the output of the journal record extract. When replication is in use, token\_seq is a journal sequence number (jsnum) that uniquely identifies each transaction (for more information on journal sequence number refer to Chapter 7: “Database Replication” (page 173)). When replication is not in use and the transaction is a TP or ZTP transaction, token\_seq is an 8-byte token that uniquely identifies the entire TP or ZTP transaction. For non-replicated, non-TP, and non-ZTP journal records, token\_seq has a zero (0) value.

The format of the plain journal extract is as follows:

```
NULL      00\time\tnum\pid\clntpid\jsnum\strm_num\strm_seq
PINI(U)   01\time\tnum\pid\nnam\unam\term\clntpid\clntnnam\clntunam\clntterm
PINI(V)
▶ 01\time\tnum\pid\nnam\unam\term\mode\logintime\image_count\pname\clntpid\clntnnam\clntunam\clntterm\clntmode
\clntlogintime\clntimage_count\clntpname
PFIN      02\time\tnum\pid\clntpid
EOF       03\time\tnum\pid\clntpid\jsnum
KILL      04\time\tnum\pid\clntpid\token_seq\strm_num\strm_seq\updnum\nodeflags\node
SET       05\time\tnum\pid\clntpid\token_seq\strm_num\strm_seq\updnum\nodeflags\node=sarg
ZTSTART   06\time\tnum\pid\clntpid\token
ZTCOM     07\time\tnum\pid\clntpid\token\partners
TSTART    08\time\tnum\pid\clntpid\token_seq\strm_num\strm_seq
TCOM      09\time\tnum\pid\clntpid\token_seq\strm_num\strm_seq\partners\tid
ZKILL     10\time\tnum\pid\clntpid\token_seq\strm_num\strm_seq\updnum\nodeflags\node
ZTWORM    11\time\tnum\pid\clntpid\token_seq\strm_num\strm_seq\updnum\ztwormhole
ZTRIG     12\time\tnum\pid\clntpid\token_seq\strm_num\strm_seq\updnum\nodeflags\node
```



where:

- 01 record indicates a process/image-initiated update (PINI) into the current journal file for the first time.
- 02 record indicates a process/image dropped interest (PFIN) in the current journal file.
- 03 record indicates all GT.M images dropped interest in this journal file and the journal file was closed normally.
- 04 record indicates a database update caused by a KILL command.
- 05 record indicates a database update caused by a SET command.
- 06 record indicates a ZTSTART command.
- 07 record indicates a ZTCOMMIT command.
- 08 record indicates a TSTART command.
- 09 record indicates a TCOMMIT command.
- 10 record indicates a database update caused by a ZKILL command.
- 11 records indicates a value for/from \$ZTWORMHOLE (when replication is turned on).
- 12 record indicates a ZTRIGGER command.

Journal extracts contain NULL records only in a multisite replication configuration where triggers or external M-filters are active. Here are two examples when NULL records are sent to the journal files:

- An external filter on an instance transforms a SET record to a NULL record that has a different schema.
- If the source side has triggers enabled and its receiver side either runs a pre-trigger version of GT.M or runs on a platform where triggers are not supported (for example, AXP VMS or HPUX HPPA), trigger definition journal records from the source side are transformed to NULL records on the receiver side.



### Important

A NULL record does not have global information. Therefore, it resides in the alphabetically last replicated region of the global directory.

The format of the detail journal extract is as follows:

```
PINI(U) time\tnum\chksum\pid\nam\unam\term\clntpid\clntnam\clntunam\clntterm
PINI(V)
▶ time\tnum\chksum\pid\nam\unam\term\mode\logintime\image_count\pname\clntpid\clntnam\clntunam\clntterm\clntmode
\clntlogintime\clntimage_count\clntpname
PFIN time\tnum\chksum\pid\clntpid
EOF time\tnum\chksum\pid\clntpid\jsnum
SET time\tnum\chksum\pid\clntpid\token_seq\strm_num\strm_seq\updnum\nodeflags\node=sarg
KILL time\tnum\chksum\pid\clntpid\token_seq\strm_num\strm_seq\updnum\nodeflags\node
ZKILL time\tnum\chksum\pid\clntpid\token_seq\strm_num\strm_seq\updnum\nodeflags\node
ZTWORM time\tnum\chksum\pid\clntpid\token_seq\strm_num\strm_seq\updnum\ztwormhole
ZTRIG time\tnum\chksum\pid\clntpid\token_seq\strm_num\strm_seq\updnum\nodeflags\node
TSTART time\tnum\chksum\pid\clntpid\token_seq\strm_num\strm_seq
TSET time\tnum\chksum\pid\clntpid\token_seq\strm_num\strm_seq\updnum\nodeflags\node=sarg
TKILL time\tnum\chksum\pid\clntpid\token_seq\strm_num\strm_seq\updnum\nodeflags\node
TZKILL time\tnum\chksum\pid\clntpid\token_seq\strm_num\strm_seq\updnum\nodeflags\node
TZTWORM time\tnum\chksum\pid\clntpid\token_seq\strm_num\strm_seq\updnum\ztwormhole
TZTRIG time\tnum\chksum\pid\clntpid\token_seq\strm_num\strm_seq\updnum\nodeflags\node
USET time\tnum\chksum\pid\clntpid\token_seq\strm_num\strm_seq\updnum\nodeflags\node=sarg
UKILL time\tnum\chksum\pid\clntpid\token_seq\strm_num\strm_seq\updnum\nodeflags\node
UZKILL time\tnum\chksum\pid\clntpid\token_seq\strm_num\strm_seq\updnum\nodeflags\node
UZTWORM time\tnum\chksum\pid\clntpid\token_seq\strm_num\strm_seq\updnum\ztwormhole
UZTRIG time\tnum\chksum\pid\clntpid\token_seq\strm_num\strm_seq\updnum\nodeflags\node
TCOM time\tnum\chksum\pid\clntpid\token_seq\strm_num\strm_seq\partners\tid
INCTN time\tnum\chksum\pid\clntpid\opcode\incdetail
EPOCH time\tnum\chksum\pid\clntpid\jsnum\blks_to_upgrd\free_blocks\total_blks\fully_upgraded[\strm_num
\strm_seq]...
PBLK time\tnum\chksum\pid\clntpid\blknum\bsiz\blkhdrtn\ondskbver
AIMG time\tnum\chksum\pid\clntpid\blknum\bsiz\blkhdrtn\ondskbver
NULL time\tnum\pid\clntpid\jsnum\strm_num\strm_seq
ZTSTART time\tnum\chksum\pid\clntpid\token
FSET time\tnum\chksum\pid\clntpid\token_seq\strm_num\strm_seq\updnum\nodeflags\node=sarg
FKILL time\tnum\chksum\pid\clntpid\token_seq\strm_num\strm_seq\updnum\nodeflags\node
FZKILL time\tnum\chksum\pid\clntpid\token_seq\strm_num\strm_seq\updnum\nodeflags\node
GSET time\tnum\chksum\pid\clntpid\token_seq\strm_num\strm_seq\updnum\nodeflags\node=sarg
GKILL time\tnum\chksum\pid\clntpid\token_seq\strm_num\strm_seq\updnum\nodeflags\node
GZKILL time\tnum\chksum\pid\clntpid\token_seq\strm_num\strm_seq\updnum\nodeflags\node
ZTCOM time\tnum\chksum\pid\clntpid\token\partners
ALIGN time\tnum\chksum\pid\clntpid
```



where:

AIMG records are unique to DSE action and exist because those actions do not have a "logical" representation.

EPOCH records are status records that record information related to check pointing of the journal.

NCTN records are the transaction numbers of the sequence of critical sections in which the process and marked the database blocks of the globals as previously used but no longer in use in the bit maps.

PBLK records are the before image records of the bit maps.

ALIGN records pad journal records so every alignsize boundary (set with MUPIP SET -JOURNAL and is visible in DSE DUMP - FILEHEADER output) in the journal file starts with a fresh journal record. The sole purpose of these records is to help speed up journal recovery.

Legend (All hexadecimal fields have a 0x prefix. All numeric fields otherwise are decimal):

<b>tnum</b>	<b>Transaction number</b>
<b>chksum</b>	Checksum for the record.
<b>fully_upgraded</b>	1 if the db was fully upgraded (indicated by a dse dump -file -all) at the time of writing the EPOCH
<b>pid</b>	Process id that wrote the jnl record.
<b>clntpid</b>	If non-zero, clntpid is the process id of the GT.CM client that initiated this update on the server side.
<b>jsnum</b>	Journal sequence number.
<b>token</b>	Unique 8-byte token.
<b>strm_num</b>	If replication is true and this update originated in a non-supplementary instance but was replicated to and updated a supplementary instance, this number is a non-zero value anywhere from 1 to 15 (both inclusive) indicating the non-supplementary stream number. In all other cases, this stream # value is 0. In case of an EPOCH record, anywhere from 0 to 16 such "strm_num" numbers might be displayed depending on how many sources of supplementary instance replication have replicated to the instance in its lifetime.
<b>strm_seq</b>	If replication is true and this update originated in a non-supplementary instance but was replicated to and updated a supplementary instance, this is the journal sequence number of the update on the originating non-supplementary instance. If replication is true and this update originated in a supplementary instance, this is the journal sequence number of the update on the originating supplementary instance. In all other cases, this stream sequence number is 0. Note that the journal seqno is actually 1 more than the most recent update originating on that stream number. In case of an EPOCH record, anywhere from 0 to 16 such "strm_seq" numbers might be displayed depending on how many sources of supplementary instance replication have replicated to the instance in its lifetime.
<b>tid</b>	TRANSACTIONID string (BATCH or any string of descriptive text chosen by the application) specified as an argument of the corresponding TSTART command. If TRANSACTIONID is not specified with TSTART, GT.M sets <b>tid</b> to null. TRANSACTIONID can specify any value for <b>tid</b> but affects GT.M behavior only when TRANSACTIONID specifies BATCH or BA.
<b>token_seq</b>	If replication is turned on, it is the journal sequence number. If not, it is a unique 8-byte token.
<b>updnum</b>	=n where this is the nth update in the TP or ZTP transaction. n=1 for the 1st update etc. 0 for non-TP.
<b>nodeflags</b>	Decimal number interpreted as a binary mask.. Currently only 5 bits are used.

### GT.M Journaling

	<ul style="list-style-type: none"> <li>• 00001 (1) =&gt; update journaled but NOT replicated (For example, update inside a trigger)</li> <li>• 00010 (2) =&gt; update to a global that had at least one trigger defined, even if no trigger matched this update</li> <li>• 00100 (4) =&gt; \$ZTWORMHOLE holds the empty string ("") at the time of this update or was not referenced during this update</li> <li>• 01000 (8) =&gt; update did not invoke any triggers even if they existed (For example, MUPIP LOAD)</li> <li>• 10000 (16) =&gt; whether the update (set or kill) is a duplicate. In case of a KILL, it is a kill of some non-existing node aka duplicate kill. Note that the dupkill occurs only in case of the Update Process. In case of GT.M, the KILL is entirely skipped. In both cases (duplicate set or kill), only a jnl record is written, the db is untouched.</li> </ul> <p>Combinations of the above bits would mean each of the individual bit characteristics. For example, 00011 =&gt; update within a trigger context, and to a global with at least one trigger defined. For example, 00011 =&gt; update inside a trigger and to a global with at least one trigger defined. Certain bit combinations are impossible. For example, 01001 since GT.M replicates any update that does not invoke triggers.</p>
<b>node</b>	Key that is being updated in a SET or KILL.
<b>sarg</b>	Right-hand side argument to the SET (that is, the value that the key is being SET to).
<b>partners</b>	Number of journaled regions participating in this TP or ZTP transaction (TCOM/ZTCOM record written in this TP or ZTP) .
<b>opcode</b>	Inctn opcode. See gdsfhead.h inctn_opcode_t for all possible values.
<b>blknum</b>	Block number corresponding to a PBLK or AIMG or INCTN record.
<b>bsiz</b>	Block size from the header field of a PBLK or AIMG record.
<b>blkhdrtn</b>	Transaction number from the block header of a PBLK or AIMG record.
<b>ondskbver</b>	On disk block version of this block at the time of writing the PBLK or AIMG record. 0 => V4, 1 => V5.
<b>incdetail</b>	0 if opcode=1,2,3; blks2upgrd if opcode=4,5,6; blknum if opcode=7,8,9,10,11,12,13
<b>ztwormhole</b>	string corresponding to \$ZTWORMHOLE
<b>blks2upgrd</b>	# of new V4 format bitmap blocks created if opcode=4,5; csd->blks_to_upgrd if opcode=6
<b>uname</b>	Name of the user that wrote this PINI record.
<b>clntunam</b>	If non-empty, clntunam is the name of the GT.CM client that initiated this update on the server side.

## Chapter 7. Database Replication

Revision History		
Revision V6.1-000/1	04 September 2014	In “Procedures ” (page 203), corrected the downloadable scripts example (msr_proc2.tar.gz) for setting up an A->P replication configuration.
Revision V6.1-000	01 August 2014	<ul style="list-style-type: none"> <li>• In “Starting the Source Server” (page 230), added the description of -TLSID, -[NO]PLAINTEXTFALLBACK, and -RENEGOTIATE_INTERVAL qualifiers.</li> <li>• In “Starting the Receiver Server ” (page 243), added the descriptions of -TLSID, -START, and -LOG qualifiers..</li> <li>• In “Procedures ” (page 203), added a new topic called “Setting up a secured TLS replication connection” (page 223). This topic includes a downloadable example that creates two databases and sets up TLS replication between them.</li> <li>• Added a new section called “TLS/SSL Replication” (page 194).</li> </ul>
Revision V6.0-003/1	19 February 2014	<ul style="list-style-type: none"> <li>• In “Starting the Source Server” (page 230), added information about the use of the gtm_ipv4_only environment variable.</li> <li>• In “REORG ” (page 102), added a caution note about running successive reorgs.</li> </ul>
Revision V6.0-003	27 January 2014	<ul style="list-style-type: none"> <li>• In “Procedures ” (page 203), added downloadable script examples for A-&gt;B and A-&gt;P replication scenarios.</li> <li>• In “Activating a Passive Source Server” (page 237), added information about the ACTIVE_REQUESTED mode.</li> <li>• In “Deactivating an Active Source Server” (page 238), added information about the PASSIVE_REQUESTED mode.</li> <li>• In “Starting the Source Server” (page 230), added information about the IPv6 support.</li> </ul>
Revision V6.0-001/1	22 March 2013	Improved the formatting of all command syntaxes and corrected the description of the -helper qualifier.
Revision V6.0-001	27 February 2013	<ul style="list-style-type: none"> <li>• In “Instance Freeze” [193], added a note about the behavior when a process configured for instance freeze encounters an error with journaling.</li> <li>• Improved the description of -UPDATERESYNC.</li> </ul>

## Database Replication

		<ul style="list-style-type: none"> <li>• In “Procedures ” [203], added the following topics: <ul style="list-style-type: none"> <li>• “Setting up a new replicating instance of an originating instance (A-&gt;B, P-&gt;Q, or A-&gt;P)” [221]</li> <li>• “Replacing the replication instance file of a replicating instance (A-&gt;B and P-&gt;Q)” [221]</li> <li>• “Replacing the replication instance file of a replicating instance (A-&gt;P)” [221]</li> <li>• “Setting up a new replicating instance from a backup of the originating instance (A-&gt;P)” [222]</li> <li>• “Changing the database schema of a replicating instance (A-&gt;B and A-&gt;P)” [223]</li> </ul> </li> <li>• In “Displaying/Changing the attributes of Replication Instance File and Journal Pool” [228], corrected the description of the -offset and -size qualifiers of -EDITINSTANCE.</li> </ul>
Revision V6.0-000/1	21 November 2012	<ul style="list-style-type: none"> <li>• In “Commands and Qualifiers” [226], added the description of the -log_interval qualifier.</li> <li>• In “Starting the Receiver Server ” [243], added the description of the -initialize qualifier and updated the description of -updateresync for V6.0-000.</li> </ul>
Revision V6.0-000	19 October 2012	<ul style="list-style-type: none"> <li>• In “Starting the Source Server” [230], added the description of the -freeze qualifier.</li> <li>• Added a new section called “Instance Freeze” (page 193).</li> </ul>
Revision V5.5-000/10	28 September 2012	In “Starting the Receiver Server ” (page 243), improved the description of -UPDATERESYNC.
Revision V5.5-000/9	14 August 2012	Improved the description of the replication WAS_ON state and added the “Recovering from the replication WAS_ON state ” (page 219) section.
Revision V5.5-000/6	19 July 2012	Removed excessive spacing around SVG diagrams in screen and print pdfs.
Revision V5.5-000/5	17 July 2012	Updated for V5.5-000.
Revision V5.5-000/4	6 June 2012	Removed -LOG as an option for -STATSLOG.
Revision V5.5-000/3	2 May 2012	<ul style="list-style-type: none"> <li>• In “Rolling Back the Database After System Failures” (page 249), corrected the command syntax and the example.</li> <li>• In “Starting the Source Server” (page 230), corrected the quoting of -filter examples.</li> </ul>

## Database Replication

Revision V5.5-000/1	5 March 2012	Improved the description of the -instsecondary qualifier. Added the “Stopping a Source Server” [241] section and the “Shutting Down an instance” [209] procedure.
Revision V5.5-000	27 February 2012	In “Starting the Source Server” [230], added information about using external filters for pre-V5.5-000 versions.
Revision 4	13 January 2012	In “Starting the Source Server” [230], added an example of a replication filter.
Revision 2	2 December 2011	Corrected the usage of the term propagating instance, improved the description of -stopsourcefilter, and changed -nopropagatingprimary to -propagatingprimary.
Revision 1	10 November 2011	In the “Startup” [204] section, changed “Restore the replication instance file” to “Recreate the replication instance file”.
Revision V5.4-002B	24 October 2011	Conversion to documentation revision history reflecting GT.M releases with revision history for each chapter.

---

## Introduction

GT.M replication provides logical equivalence between multiple databases. It facilitates continuous application availability, real-time decision support, warehousing, analytics, and auditing. There are two types of replication:

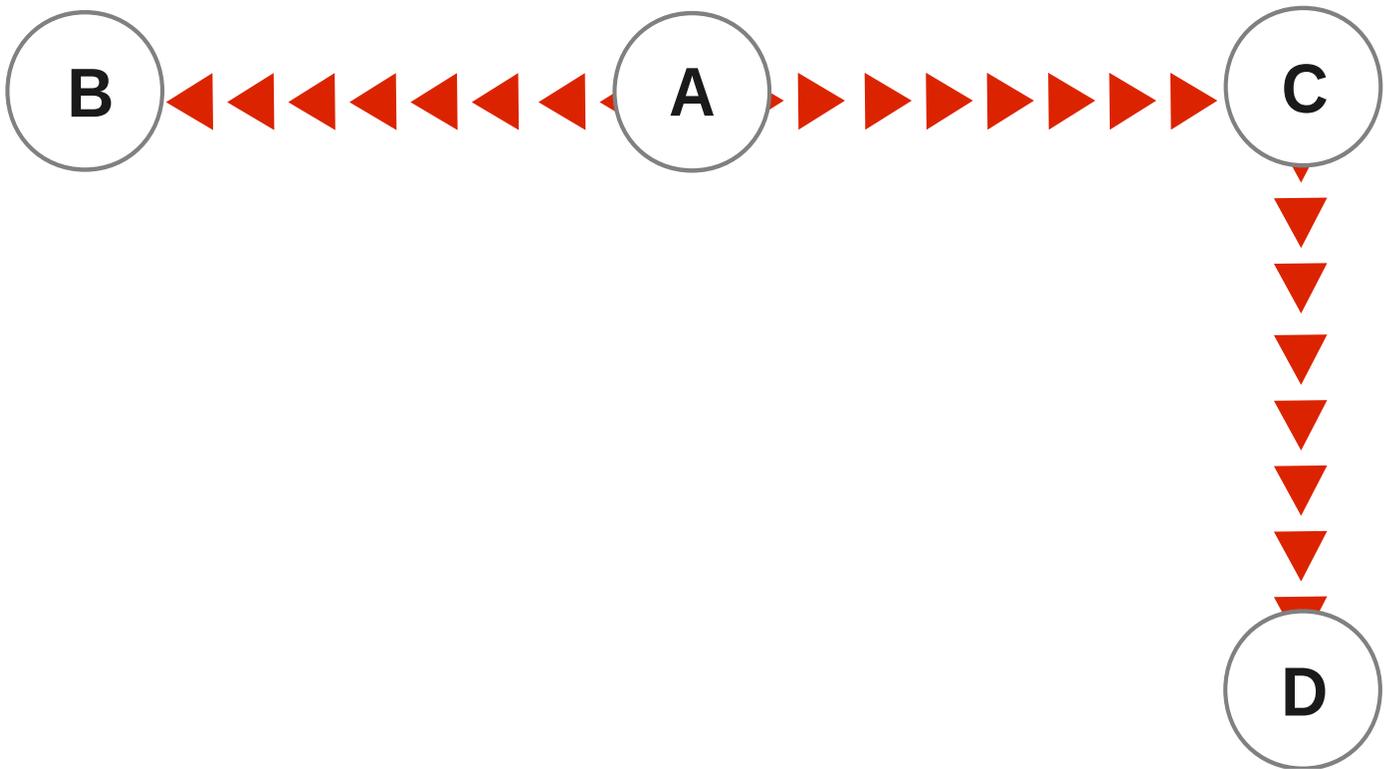
1. Business Continuity (BC) replication
2. Supplementary Instance (SI) replication

BC replication provides business continuity for systems of record. Updates applied at an originating instance replicate in near real-time to a replicating instance. To help ensure this consistency, BC replication prohibits locally generated database updates on a replicating secondary instance. For example with instances named A and B, business logic processed on instance A can be streamed to instance B so that should A ever go down, B can immediately take over and provide continuity. In order to ensure that B produces results consistent with A, B can contain only material state information that is also in A.

Updates applied at an originating instance replicate in near real-time to as many as sixteen replicating instances each of which can propagate further down to as many as sixteen replicating instances. Each replicating instance can further replicate to as many as sixteen replicating instances and so on. When an originating instance becomes unavailable, any downstream instance can become the replacement originating instance to keep the application available.

In the following illustration, A is the originating primary instance and B and C are its replicating instances. C is also a propagating primary instance because it further replicates to D. This BC replication configuration can also be described as a B<-A->C->D configuration.

## Database Replication

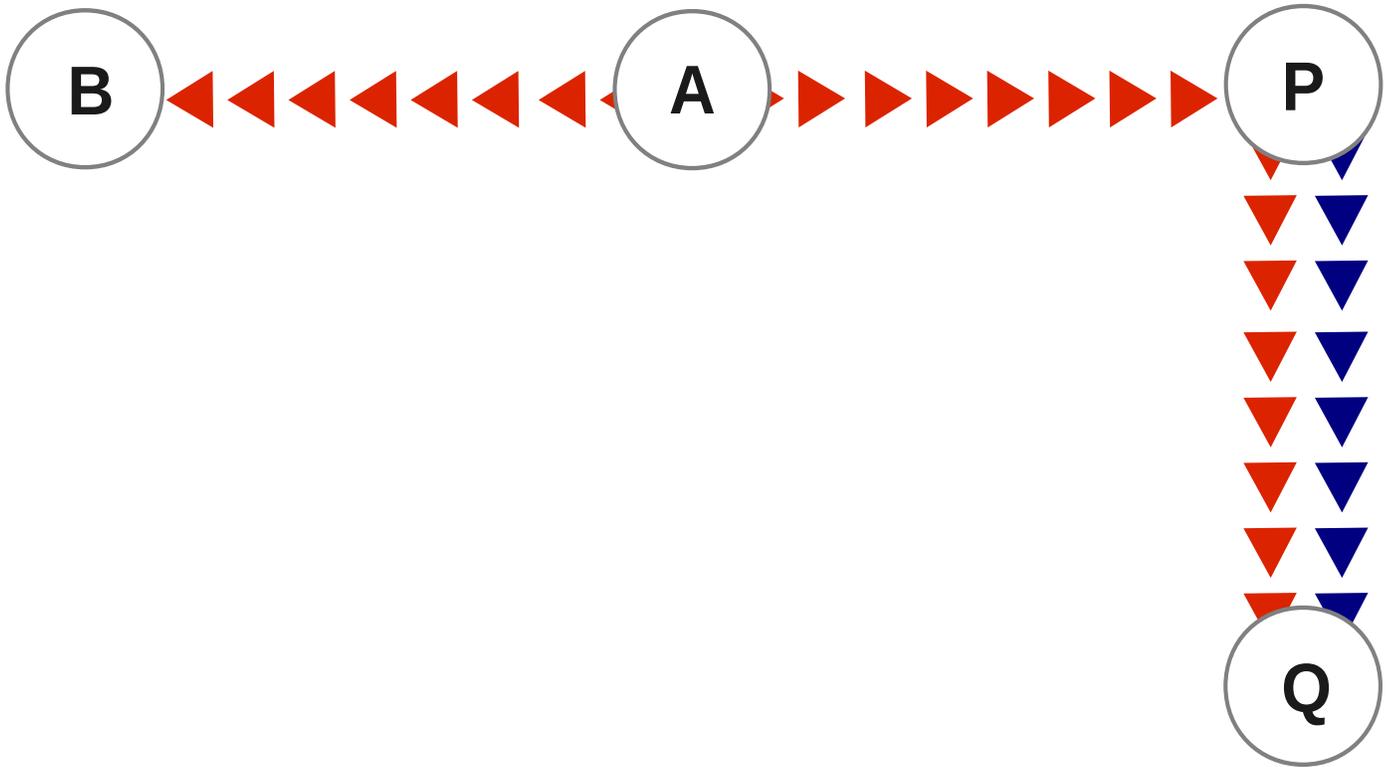


BC replication is intended for mission-critical applications that must be available 24 hours a day, 365 days a year, in the face of both unplanned events (such as system crashes) as well planned events (such as system and software upgrades).

With BC replication, you can create a logical multi-site (LMS) replication configuration for mission critical applications that must always be available not only in face of unplanned events (such as system or data center failures) but also in the face of planned events such as computing platform upgrades, OS upgrades, GT.M upgrades and even application upgrades. Deploying a BC replication configuration should take into account available network capacities, operational preferences, risk assessments, and business continuity requirements.

SI replication allows replication from an instance A to another originating primary instance P. P can execute its own business logic, compute, and commit its own updates to its database, while receiving a replication stream. In turn, P can have its own replicating secondary instance Q and A can have its own replicating instance B. In such an SI replication configuration, only originating primary instances A and P can execute business logic and compute database updates. Replicating secondary instances B and Q are only permitted to receive and apply replication streams from their originating primary instances. The following diagram illustrates this configuration.

## Database Replication



In this diagram, A is an originating primary instance having B and P as its replicating instance. P is another originating primary instance (supplementary instance) having Q as its replicating instance. This SI replication can also be described as a B<-A->P->Q configuration.

SI replication is intended to be a general purpose mechanism whose utility includes applications such as real-time decision support, warehousing, analytics, and auditing.



### Note

In this book, instances {A, B, C...} denote system of records (BC replication) and instances {P, Q, R...} denote instances that receive SI replication.

GT.M replication is asynchronous, which in turn means that the source and receiver ends of a replication connection are at an identical state only when there is no activity underway. To maintain consistency, and to restore it when restarting a replication connection, instances maintain a common, mutually coherent, instance-wide serial number called a journal sequence number. Each journal sequence number is tagged with two fields that identify the source of the update- a stream # that can take on values 0 through 15 and a stream sequence number (the journal sequence number of the update on the originating instance). Because replication deals with an entire global variable namespace, regardless of the mapping of global variables to database files, all updates participate in this numbering, even when modifying different database files.

On an originating primary instance that is not the recipient of an SI replication stream, the journal sequence number and the stream sequence number are the same.

Suppose sequence numbers in P are 100, 101, and 102. If the first and third transactions are locally generated and the second is replicated, the tagged journal sequence numbers might be something like {100,0,10}, {101,1,18}, {102,0,11}. The 0 stream # for 100 and 102 indicates those transactions are generated locally on P whereas stream # 1 indicates those transactions were generated

## Database Replication

in A. It is important to note that if, as part of restarting replication with A, P needs to roll {101,1,18} off its database, database update serialization also requires it to roll {102,0,11} off as well, and both will appear in the Unreplicated Transaction Log (also known as Lost Transaction File).

The same transaction that has a P sequence number of 101 will have a different sequence number of 18 on A and B. That is, the journal sequence number on A becomes the stream sequence number on P. The replication instance file in P contains information that allows GT.M to determine this mapping, so that when P rolls {101,1,18} off its database, A knows that Malvern has rolled off A.s transaction 18.

If P in turn implements BC replication to another instance Q, the tagging is propagated to Q, such that if A and P both go down (e.g., if they are co-located in a data center that loses electricity), B and C can take over the functions of A and P respectively, and Q can perform any synchronization needed in order to start accepting a replication stream from B as being a continuation of the updates generated by A, and B in turn accepts Q as the successor to P.

An LMS Group is a set of one or more instances that receive updates from the same originating primary instance and represent the same logical state. GT.M implicitly forms an LMS Group by storing the identification of the originating primary instance in the replication instance file of each replicating instance. There are two types of LMS Groups:

1. BC Group: An LMS Group whose originating primary instance is a BC instance. A BC Group can have BC and SI instances as members.
2. SI Group: An LMS Group whose originating primary instance is an SI instance. An SI Group can have only SI instances as members and can receive replication only from a BC member of a BC Group.

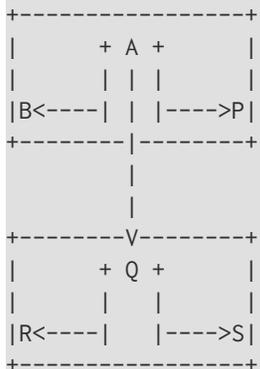
BC members of a BC Group can replicate downstream to other BC and SI groups whereas an SI Group cannot replicate downstream to other groups.



### Important

Instances can change their roles within an LMS group but they cannot move between groups. However data from one instance / group can be loaded into another group.

The following example illustrates a replication configuration where instance A from A's BC Group replicates instance Q in Q's SI Group.



### Note

In this replication configuration, instance B can also replicate to instance Q. However, instance P cannot replicate to an instance in Q's group because it is an SI member of A's BC group.

## Database Replication

GT.M imposes no distance limits between instances. You can place instances 20,000 kilometers apart (the circumference of Planet Earth is 40,000 kilometers) or locally within the same data center.

Using TCP connections, GT.M replicates between instances with heterogeneous stacks of hardware, operating system, endian architecture<sup>1</sup> and even GT.M releases. A GT.M instance can source a BC replication stream to or receive a BC replication stream from GT.M V5.1-000 or later. However, SI replication requires both source and receive side must be GT.M V5.5-000. SI replication is only supported on POSIX editions of GT.M - in other words, it is not supported by GT.M on OpenVMS. GT.M replication can even be used in configurations with different application software versions, including many cases where the application software versions have different database schema. This also means that a number of inexpensive system - such as GNU/Linux commodity servers - can be placed in locations throughout an organization. Replicating instances can be used for decision support, reporting, and analytics. Because GT.M databases can be encrypted, these commodity servers are potentially usable in environments outside secure data centers as long as their operating systems and software are secured.

GT.M replication requires journaling to be enabled and turned on for replicated regions. Unreplicated regions (for example global variables contain information that is meaningful only on one instance and only as long as the instance is operating - such as process ids, temporary working files, and so on) need not be replicated or journaled.

GT.M replication mechanism is designed in such a way that a network failure between instances will not stop an application from being available-which is a limitation of techniques such as high availability clustering<sup>2</sup>. There are mechanisms in place for edge cases like processing "in flight" transactions and common cases like handling backlog of updates after recovery from a network failure. While it is not possible to provide absolute continuity of business in our imperfect universe, an LMS configuration gives you the flexibility to choose application configurations that match your investment to a risk level that best meets the business needs of your organization.

## Database Transaction Number

Every transaction applied to a database file increments the database transaction number for that file. Each block records the database transaction number at which it was updated, and the Current transaction field in the file header shows the value for the next transaction or mini-transaction to use. The following database file header fields all show database transaction numbers: Last Record Backup, Last Database Backup, Last Bytestream Backup, Maximum TN, and Maximum TN Warn.

Database transaction numbers are currently unsigned 64-bit integers.

While database activity uses database transaction numbers sequentially, not every transaction number can be found in a database block. For a Kill increments the database transaction number, but can remove blocks with earlier database transaction numbers from the database.

Note that database transaction numbers are updated in memory and only periodically flushed to secondary storage, so in cases of abnormal shutdown, the on-disk copies in the file header might be somewhat out-of-date.

## Journal Sequence Number

While the database transaction number is specific to a database file, replication imposes a serialization of transactions across all replicated regions. As each transaction is placed in the Journal Pool it is assigned the next journal sequence number. When a database file in a replicated instance is updated, the Region Seqno field in the file header records the journal sequence number for that transaction. The journal sequence number for an instance is the maximum Region Seqno of any database file in that

---

<sup>1</sup>BC replication from/to GT.M on OpenVMS is supported only to/from instances on OpenVMS and little endian UNIX/Linux platforms.

<sup>2</sup>GT.M database replication is compatible with clustering - each instance can be a "hot-warm" cluster where if one node fails, another node can recover the database and continue operation. Since GT.M LMS application configurations provides better business continuity in the face of a greater range of eventualities than clusters, if you wish to use clusters, consider their use in conjunction with, rather than instead of, GT.M LMS configurations.

## Database Replication

instance. While it uses them in processing, GT.M stores journal sequence numbers only in journal files. In database file headers, Zqgblmod Seqno and Zqgblmod Trans are journal sequence numbers.

Except for transactions in Unreplicated Transaction Logs, the journal sequence number of a transaction uniquely identifies that transaction on the originating primary instance and on all replicating secondary instances. When replicated via SI replication, the journal sequence number becomes a stream sequence number (see below) and propagated downstream, thus maintaining the unique identity of each transaction.

Journal sequence numbers cannot have holes - missing journal sequence numbers are evidence of abnormal database activity, including possible manual manipulation of the transaction history or database state.

Journal sequence numbers are 60-bit unsigned integers.

## Stream Sequence Number

The receiver of a SI replication stream has both transactions that it receives via replication as well as transactions that it computes locally from business logic. As discussed earlier, while journal sequence numbers can uniquely identify a series of database updates, they cannot identify the source of those updates. Therefore, we have the concept of a stream sequence number.

On an originating primary instance that is not the recipient of an SI replication stream, the journal sequence number and the stream sequence number are the same.

On a primary instance that is the recipient of an SI replication stream, the journal sequence number uniquely identify and serialize all updates, whether received from replication or locally generated. However, there is also a stream sequence number, which is the journal sequence number for locally generated transactions, and for replicated updates, the combination of a non-zero 4 bit tag (that is, with values 1 through 15) and the journal sequence number for the transaction on the system from which it was replicated. These stream sequence numbers are propagated to downstream replicating secondary instances.

Stream sequence numbers are 64-bit unsigned integers.

## Examples

To make the following scenarios easier to understand, each update is prefixed with the system where it was originally generated and the sequence number on that system and any BC replicating secondary instances.

### Simple Example

The three systems initially operate in roles O (Originating primary instance), R (BC Replicating secondary instance) and S (recipient of an SI replication stream).

<b>Ardmore</b>	<b>BrynMawr</b>	<b>Malvern</b>	<b>Comments</b>
O: ... <b>A95, A96, A97, A98, A99</b>	R: ... <b>A95, A96, A97, A98</b>	S: ... <u>M34, A95, M35, M36, A96, A97, M37, M38</u>	<b>Ardmore</b> as an originating primary instance at transaction number <b>A99</b> , replicates to <i>BrynMawr</i> as a BC replicating secondary instance at transaction number <b>A98</b> and <u>Malvern</u> as a SI that includes transaction number <b>A97</b> , interspersed with locally generated updates. Updates are recorded in each instance's journal files using before-image journaling.
Crashes	O: ... <b>A95, A96, A97, A98, B61</b>	... <u>M34, A95, M35, M36, A96, A97, M37, M38</u>	When an event disables <b>Ardmore</b> , <i>BrynMawr</i> becomes the new originating primary, with <b>A98</b> as the latest transaction in its database, and starts processing application logic to maintain

## Database Replication

Ardmore	BrynMawr	Malvern	Comments
			business continuity. In this case where <u>Malvern</u> is not ahead of <u>BrynMawr</u> , the Receiver Server at <u>Malvern</u> can remain up after <b>Ardmore</b> crashes. When <u>BrynMawr</u> connects, its Source Server and <u>Malvern</u> 's Receiver Server confirms that <u>BrynMawr</u> is not behind <u>Malvern</u> with respect to updates received from <b>Ardmore</b> , and SI replication from <u>BrynMawr</u> picks up where replication from <b>Ardmore</b> left off.
-	O: ... <b>A95, A96, A97, A98, B61, B62</b>	S: ... <u>M34, A95, M35, M36, A96, A97, M37, M38, A98, M39, B61, M40</u>	<u>Malvern</u> operating as a supplementary instance to <u>BrynMawr</u> replicates transactions processed on <u>BrynMawr</u> , and also applies its own locally generated updates. Although <b>A98</b> was originally generated on <b>Ardmore</b> , <u>Malvern</u> received it from <u>BrynMawr</u> because <b>A97</b> was the common point between <u>BrynMawr</u> and <u>Malvern</u> .
... <b>A95, A96, A97, A98, A99</b>	O: ... <b>A95, A96, A97, A98, B61, B62, B63, B64</b>	S: ... <u>M34, A95, M35, M36, A96, A97, M37, M38, A98, M39, B61, M40, B62, B63</u>	<u>Malvern</u> , continuing as a supplementary instance to <u>BrynMawr</u> , replicates transactions processed on <u>BrynMawr</u> , and also applies its own locally generated updates. <b>Ardmore</b> meanwhile has been repaired and brought online. It has to roll transaction <b>A99</b> off its database into an Unreplicated Transaction Log before it can start operating as a replicating secondary instance to <u>BrynMawr</u> .
R: ... <b>A95, A96, A97, A98, B61, B62, B63, B64</b>	O: ... <b>A95, A96, A97, A98, B61, B62, B63, B64, B65</b>	S: ... <u>M34, A95, M35, M36, A96, A97, M37, M38, A98, M39, B61, M40, B62, B63, M41, B64</u>	Having rolled off transactions into an Unreplicated Transaction Log, <b>Ardmore</b> can now operate as a replicating secondary instance to <u>BrynMawr</u> . This is normal BC Logical Multi-Site operation. <u>BrynMawr</u> and <u>Malvern</u> continue operating as originating primary instance and supplementary instance.

## Ensuring Consistency with Rollback

Whereas in the last example Malvern was not ahead when starting SI replication from BrynMawr, in this example, asynchronous processing has left it ahead and must rollback its database state before it can receive the replication stream.

Ardmore	BrynMawr	Malvern	Comments
O: ... <b>A95, A96, A97, A98, A99</b>	R: ... <b>A95, A96, A97</b>	S: ... <u>M34, A95, M35, M36, A96, A97, M37, M38, A98, M39, M40</u>	<b>Ardmore</b> as an originating primary instance at transaction number <b>A99</b> , replicates to <u>BrynMawr</u> as a BC replicating secondary instance at transaction number <b>A97</b> and <u>Malvern</u> as a SI that includes transaction number <b>A98</b> , interspersed with locally generated updates. Updates are recorded in each instance's journal files using before-image journaling.
Crashes	O: ... <b>A95, A96, A97</b>	... <u>M34, A95, M35, M36, A96, A97, M37, M38, A98, M39, M40</u>	When an event disables <b>Ardmore</b> , <u>BrynMawr</u> becomes the new originating primary, with <b>A97</b> the latest transaction in its database. <u>Malvern</u> cannot immediately start replicating from <u>BrynMawr</u> because the database states would not be consistent - while <u>BrynMawr</u> does not have <b>A98</b> in its database and its next update may implicitly or explicitly depend on that absence, <u>Malvern</u> does, and may have relied on <b>A98</b> to compute <u>M39</u> and <u>M40</u> .
-	O: ... <b>A95, A96, A97, B61, B62</b>	S: ... <u>M34, A95, M35, M36, A96, A97, M37, M38, B61</u>	For <u>Malvern</u> to accept replication from <u>BrynMawr</u> , it must roll off transactions generated by <b>Ardmore</b> , (in this case <b>A98</b> ) that <u>BrynMawr</u> does not have in its database, as well as any additional

## Database Replication

Ardmore	<i>BrynMawr</i>	<u>Malvern</u>	Comments
			transactions generated and applied locally since transaction number <b>A98</b> from <b>Ardmore</b> . <sup>a</sup> This rollback is accomplished with a mupip journal -rollback -fetchresync operation on <u>Malvern</u> . <sup>b</sup> These rolled off transactions ( <b>A98</b> , <u>M39</u> , <u>M40</u> ) go into the Unreplicated Transaction Log and can be subsequently reprocessed by application code. <sup>c</sup> Once the rollback is completed, <u>Malvern</u> can start accepting replication from <i>BrynMawr</i> . <sup>d</sup> <i>BrynMawr</i> in its Originating Primary role processes transactions and provides business continuity, resulting in transactions <i>B61</i> and <i>B62</i> .
-	O: ... <b>A95</b> , <b>A96</b> , <b>A97</b> , <i>B61</i> , <i>B62</i> , <i>B63</i> , <i>B64</i>	S: ... <u>M34</u> , <b>A95</b> , <u>M35</u> , <u>M36</u> , <b>A96</b> , <b>A97</b> , <u>M37</u> , <u>M38</u> , <i>B61</i> , <i>B62</i> , <u>M39a</u> , <u>M40a</u> , <i>B63</i>	<u>Malvern</u> operating as a supplementary instance to <i>BrynMawr</i> replicates transactions processed on <i>BrynMawr</i> , and also applies its own locally generated updates. Note that <u>M39a</u> & <u>M40a</u> may or may not be the same updates as the <u>M39</u> & <u>M40</u> previously rolled off the database.

<sup>a</sup>As this rollback is more complex, may involve more data than the regular LMS rollback, and may involve reading journal records sequentially; it may take longer.

<sup>b</sup>In scripting for automating operations, there is no need to explicitly test whether *BrynMawr* is behind Malvern - if it is behind, the Source Server will fail to connect and report an error, which automated shell scripting can detect and effect a rollback on Malvern followed by a reconnection attempt by *BrynMawr*. On the other hand, there is no harm in Malvern routinely performing a rollback before having *BrynMawr* connect - if it is not ahead, the rollback will be a no-op. This characteristic of replication is unchanged from releases prior to V5.5-000.

<sup>c</sup>GT.M's responsibility for them ends once it places them in the Unreplicated Transaction Log.

<sup>d</sup>Ultimately, business logic must determine whether the rolled off transactions can simply be reapplied or whether other reprocessing is required. GT.M's \$ZQGBLMOD() function can assist application code in determining whether conflicting updates may have occurred.

## Rollback Not Desired or Required by Application Design

In the example above, for Malvern to start accepting SI replication from *BrynMawr* with consistency requires it to rollback its database because it is ahead of *BrynMawr*. There may be applications where the design of the application is such that this rollback neither required nor desired. GT.M provides a way for SI replication to start in this situation without rolling transactions off into an Unreplicated Transaction File.

Ardmore	<i>BrynMawr</i>	<u>Malvern</u>	Comments
O: ... <b>A95</b> , <b>A96</b> , <b>A97</b> , <b>A98</b> , <b>A99</b>	R: ... <b>A95</b> , <b>A96</b> , <b>A97</b>	S: ... <u>M34</u> , <b>A95</b> , <u>M35</u> , <u>M36</u> , <b>A96</b> , <b>A97</b> , <u>M37</u> , <u>M38</u> , <b>A98</b> , <u>M39</u> , <u>M40</u>	<b>Ardmore</b> as an originating primary instance at transaction number <b>A99</b> , replicates to <i>BrynMawr</i> as a BC replicating secondary instance at transaction number <b>A97</b> and <u>Malvern</u> as a SI that includes transaction number <b>A98</b> , interspersed with locally generated updates. Updates are recorded in each instance's journal files using before-image journaling.
Crashes	O: ... <b>A95</b> , <b>A96</b> , <b>A97</b> , <i>B61</i> , <i>B62</i>	... <u>M34</u> , <b>A95</b> , <u>M35</u> , <u>M36</u> , <b>A96</b> , <b>A97</b> , <u>M37</u> , <u>M38</u> , <b>A98</b> , <u>M39</u> , <u>M40</u>	When an event disables <b>Ardmore</b> , <i>BrynMawr</i> becomes the new originating primary, with <b>A97</b> the latest transaction in its database and starts processing application logic. Unlike the previous example, in this case, application design permits (or requires) <u>Malvern</u> to start replicating from <i>BrynMawr</i> even though <i>BrynMawr</i> does not have <b>A98</b> in its database and <u>Malvern</u> may have relied on <b>A98</b> to compute <u>M39</u> and <u>M40</u> .
-	O: ... <b>A95</b> , <b>A96</b> , <b>A97</b> , <i>B61</i> , <i>B62</i>	S: ... <u>M34</u> , <b>A95</b> , <u>M35</u> , <u>M36</u> , <b>A96</b> , <b>A97</b> , <u>M37</u> , <u>M38</u> , <b>A98</b> , <u>M39</u> , <u>M40</u> , <i>B61</i> , <i>B62</i>	With its Receiver Server started with the -noresync option, <u>Malvern</u> can receive a SI replication stream from <i>BrynMawr</i> , and replication starts from the last common transaction shared by <i>BrynMawr</i> and

## Database Replication

Ardmore	<i>BrynMawr</i>	<u>Malvern</u>	Comments
			<u>Malvern</u> . Notice that on <i>BrynMawr</i> no <b>A98</b> precedes <i>B61</i> , whereas it does on <u>Malvern</u> , i.e., <u>Malvern</u> was ahead of <i>BrynMawr</i> with respect to the updates generated by <b>Ardmore</b> .

## Two Originating Primary Failures

Now consider a situation where **Ardmore** and Malvern are located in one data center, with BC replication to *BrynMawr* and *Newtown* respectively, located in another data center. When the first data center fails, the SI replication from **Ardmore** to Malvern is replaced by SI replication from *BrynMawr* to *Newtown*.

Ardmore	<i>BrynMawr</i>	<u>Malvern</u>	<i>Newtown</i>	Comments
O: ... <b>A95, A96, A97, A98, A99</b>	R: ... <b>A95, A96, A97, A98</b>	S: ... <u>M34, A95, M35, M36, A96, M37, A97, M38</u>	R: ... <u>M34, A95, M35, M36, A96, M37</u>	<b>Ardmore</b> as an originating primary instance at transaction number <b>A99</b> , replicates to <i>BrynMawr</i> as a BC replicating secondary instance at transaction number <b>A98</b> and <u>Malvern</u> as a SI that includes transaction number <b>A97</b> , interspersed with locally generated updates. <u>Malvern</u> in turn replicates to <i>Newtown</i> .
Goes down with the data center	O: ... <b>A95, A96, A97, A98, B61, B62</b>	Goes down with the data center	... <u>M34, A95, M35, M36, A96, M37</u>	When a data center outage disables <b>Ardmore</b> , and <u>Malvern</u> , <i>BrynMawr</i> becomes the new originating primary, with <b>A98</b> as the latest transaction in its database and starts processing application logic to maintain business continuity. <i>Newtown</i> can receive the SI replication stream from <i>BrynMawr</i> , without requiring a rollback since the receiver is not ahead of the source.
-	O: ... <b>A95, A96, A97, A98, B61, B62</b>	-	S: ... <u>M34, A95, M35, M36, A96, M37, A97, A98, N73, B61, N74, B62</u>	<i>Newtown</i> receives SI replication from <i>BrynMawr</i> and also applies its own locally generated updates. Although <b>A97</b> and <b>A98</b> were originally generated on <b>Ardmore</b> , <i>Newtown</i> receives them from <i>BrynMawr</i> . <i>Newtown</i> also computes and applies locally generated updates
... <b>A95, A96, A97, A98, A99</b>	O: ... <b>A95, A96, A97, B61, B62, B63, B64</b>	... <u>M34, A95, M35, M36, A96, M37, A97, M38</u>	S: ... <u>M34, A95, M35, M36, A96, M37, A97, A98, N73, B61, N74, B62, N75, B63, N76, B64</u>	While <i>BrynMawr</i> and <i>Newtown</i> , keep the enterprise in operation, the first data center is recovered. Since <b>Ardmore</b> has transactions in its database that were not replicated to <i>BrynMawr</i> when the latter started operating as the originating primary instance, and since <u>Malvern</u> had transactions that were not replicated to <i>Newtown</i> when the latter took over, <b>Ardmore</b> and <u>Malvern</u> must now rollback their databases and create Unreplicated Transaction Files before receiving BC replication streams from <i>BrynMawr</i> and <i>Newtown</i> respectively.

## Database Replication

<b>Ardmore</b>	<b>BrynMawr</b>	<b>Malvern</b>	<b>Newtown</b>	<b>Comments</b>
				<b>Ardmore</b> rolls off <b>A98</b> and <b>A99</b> , <u>Malvern</u> rolls off <b>A97</b> and <u>M38</u> .
R: ... <b>A95, A96, A97</b> , <i>B61, B62, B63, B64</i>	O: ... <b>A95, A96, A97</b> , <i>B61, B62, B63, B64</i> , B65	R: ... <u>M34</u> , <b>A95</b> , <u>M35</u> , <u>M36</u> , <b>A96</b> , <u>M37</u> , <b>A97</b> , <b>A98</b> , <i>N73, B61, N74</i> , <i>B62, N75, B63, N76</i> , <i>B64</i>	S: ... <u>M34</u> , <b>A95</b> , <u>M35</u> , <u>M36</u> , <b>A96</b> , <u>M37</u> , <b>A97</b> , <b>A98</b> , <i>N73, B61, N74</i> , <i>B62, N75, B63, N76</i> , <i>B64, N77</i>	Having rolled off transactions into an Unreplicated Transaction Log, <b>Ardmore</b> can now operate as a replicating secondary instance to <i>BrynMawr</i> . This is normal BC Logical Multi-Site operation. <i>BrynMawr</i> and <u>Malvern</u> continue operating as originating primary instance and supplementary instance. Note that having rolled <b>A97</b> off its database, <u>Malvern</u> receives that transaction from <i>Newtown</i> as it catches up.

## Replication and Online Rollback

Consider the following example where **Ardmore** rolls back its database in state space while an application is in operation. using the mupip journal -rollback -backward -online feature.

<b>Ardmore</b>	<b>BrynMawr</b>	<b>Malvern</b>	<b>Comments</b>
O: ... <b>A95, A96, A97</b> , <b>A98, A99</b>	R: ... <b>A95, A96, A97</b>	S: ... <u>M34</u> , <b>A95</b> , <u>M35</u> , <u>M36</u> , <b>A96</b> , <b>A97</b> , <u>M37</u> , <u>M38</u> , <b>A98</b> , <u>M39</u> , <u>M40</u>	<b>Ardmore</b> as an originating primary instance at transaction number <b>A99</b> , replicates to <i>BrynMawr</i> as a BC replicating secondary instance at transaction number <b>A97</b> and <u>Malvern</u> as a SI that includes transaction number <b>A98</b> , interspersed with locally generated updates. Updates are recorded in each instance's journal files using before-image journaling.
Rolls back to <b>A96</b> with <b>A97</b> through <b>A99</b> in the Unreplicated Transaction Log	Rolls back automatically to <b>A96</b> (assume Receiver Server started with -autorollback - refer to the V5.5-000 Release Notes for details.	-	Instances receiving a replication stream from <b>Ardmore</b> can be configured to rollback automatically when <b>Ardmore</b> performs an online rollback by starting the Receiver Server with -autorollback. If <u>Malvern</u> 's Receiver Server is so configured, it will roll <b>A97</b> through <u>M40</u> into an Unreplicated Transaction Log. This scenario is straightforward. But with the -noresync qualifier, the Receiver Server can be started configured to simply resume replication without rolling back, and that scenario is developed here.
O: ... <b>A95, A96</b> , <b>A97a, A98a, A99a</b>	R: ... <b>A95, A96</b> , <b>A97a, A98a</b>	S: ... <u>M34</u> , <b>A95</b> , <u>M35</u> , <u>M36</u> , <b>A96</b> , <b>A97</b> , <u>M37</u> , <u>M38</u> , <b>A98</b> , <u>M39</u> , <u>M40</u> , <b>A97a</b> , <u>M41</u> , <b>A98a</b> , <u>M42</u>	Transactions <b>A97a</b> through <b>A99a</b> are different transactions from <b>A97</b> through <b>A99</b> (which are in an Unreplicated Transaction File on <b>Ardmore</b> and must be reprocessed). Note that <u>Malvern</u> has both the original <b>A97</b> and <b>A98</b> as well as <b>A97a</b> and <b>A98a</b> . <b>A99</b> was never replicated to <u>Malvern</u> - <b>Ardmore</b> rolled back before it was replicated, and <b>A99a</b> has not yet made it to <u>Malvern</u> (it will soon, unless <b>Ardmore</b> rolls back again).

## Limitations - SI Replication

SI replication is only supported on POSIX editions of GT.M - in other words, it is not supported by GT.M on OpenVMS. Furthermore, starting V5.5-000, GT.M does not support replication between OpenVMS and POSIX platforms, or on POSIX platforms with GT.M releases prior to V5.1-000. To upgrade to GT.M V5.5-000, or to upgrade from GT.M on OpenVMS to GT.M V5.5-000 on a POSIX platform, first upgrade to GT.M V5.1-000 or later on a POSIX platform as an intermediate step.

### Database Replication

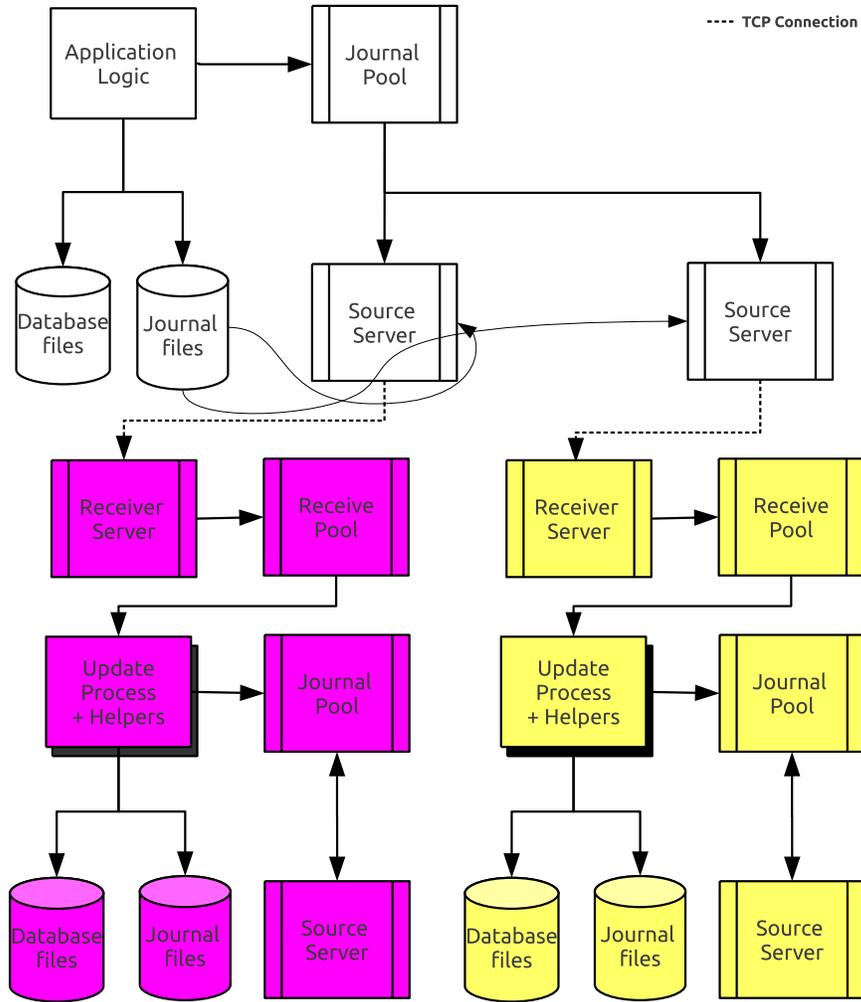
Although a receiver of SI replication can source a BC replication stream for downstream propagation, it cannot source an SI replication stream. So, in the example above, while Malvern can receive SI replication from **Ardmore** or *BrynMawr*, and it can source a BC replication stream to *Newtown*, which can in turn source a BC replication stream to Oxford. Thus, none of Malvern, *Newtown* or Oxford can source an SI replication stream.

Also an instance can only receive a single SI replication stream. Malvern cannot receive SI replication from an instance other than **Ardmore** (or an instance receiving BC replication from **Ardmore**, such as *BrynMawr*). *Newtown* or Oxford are replicating secondary instances and can receive no updates other than from Malvern.

The total number of replication streams that an instance can source is sixteen, with any combination of BC and SI replication.

## Replication Architecture

The following diagram illustrates a BC replication configuration deployed as B<-A->C. White (top) is the originating instance processing business logic, while Rose (left) and Yellow (right) are replicating instances. The dotted line represents a TCP connection and the red dots show the movement of transactions. If White goes down in an unplanned or planned event, either Rose or Yellow can become the originating instance within seconds to tens of seconds, and the other instance can become a replicating instance to the new originating instance. When White recovers, it rejoins as a replicating instance to the new originating instance. At some suitable future time, when so desired, White can again be made the originating instance.



When a process commits a transaction on White, GT.M provides Durability by writing and "hardening" an update record to the journal file and then the database file. The same process also writes the update records to an area of shared memory called a Journal Pool as part of committing the transaction, but does not wait for Rose and Yellow to commit the transaction (this means

## Database Replication

that a failure of the network between instances does not stop application operation on White). Two Source Server processes, one for Rose and one for Yellow, read journal update records from the Journal Pool and stream updates over TCP connections to Receiver Server processes on the replicating instances they serve.

Under normal conditions, White Source Servers stream update records from the Journal Pool to the Rose and Yellow Receiver Servers. The Journal Pool is a shared memory segment that does not expand after its initial creation. If updates for the database state to which the replicating instance needs to catch up are no longer in the Journal Pool, the Source Server finds the updates in journal files, until the replicating instance catches up to the point where the remaining required updates can again come from the Journal Pool. The diagram represents this with the curved lines from the journal file to the Source Server processes.

A Source Server<sup>3</sup> can be in either of two modes--active mode or passive mode.

In active mode, a Source Server connects to the Receiver Server on its replicating instance and transfers update records from the Journal Pool via the communication channel. If an active Source Server is not connected to its Receiver Server, it makes repeated attempts to connect until it succeeds. When an active Source Server connects with its Receiver Server, they ensure their two instances are in sync before proceeding with replication. In the diagram, the White Source Servers are in active mode. When an active Source Server receives a command to switch to passive mode, it closes the connection with its Receiver Server and "goes to sleep" until it receives a command to become active.

In passive mode, a Source Server is in a stand-by state. In the diagram, the Rose and Yellow Source Servers are in passive mode. When a passive Source Server receives a command to switch to active mode, it attempts to establish a connection with the specified Receiver Server on its replicating instance.

Under typical operating conditions, with no system or network bottlenecks, GT.M moves a transaction off the originating instance and into the care of the network moving towards its replicating instance in sub-millisecond time frames. Network transit times then determine how long the transaction message takes to reach the replicating instance. Because it uses a change- or delta-based protocol, GT.M Replication uses network bandwidth efficiently. Furthermore, the Source Server can compress the byte stream which the Receiver Server then decompresses; alternatively network routers can perform the compression and decompression. You can use standard techniques at the stack or router for encrypting TCP connections to secure replication.

On Rose and Yellow instances, a Receiver Server receives update records sent by the White Source Server and puts them in the Receive Pool, which is in a shared memory segment. Source and Receiver Server processes implement flow control to ensure that the Receive Pool does not overflow. The Update Process picks these update records and writes them to the journal file, the database file, and the Journal Pool. The Update Process on a replicating instance performs operations analogous to "Application Logic" on the originating instance.

## Helper Processes

Helper processes accelerate the rate at which an Update Process can apply an incoming replication stream to the database on a replicating instance.

The GT.M database engine performs best when multiple processes concurrently access the database, cooperating with one another to manage it. Therefore, it is possible for the tens, hundreds or thousands of application processes executing on an originating instance to outperform a replicating instance with only a single Update Process. Helper processes enable the update process to apply database updates faster and thereby keep up.

There are two types of helper processes:

1. Reader: Reader processes read the update records in the Receive Pool and attempt to pre-fetch database blocks into the global buffer pools, so they are more quickly available for the Update Process.

---

<sup>3</sup>The first Source Server process started on an instance creates the Journal Pool.

2. **Writer:** Writer processes help the Update Process by flushing database and journal records from shared memory (global and journal buffers) to the file system.

A certain number of each type of helper process maximizes throughput. As a practical matter, as long as the file system bandwidth on a replicating instance is equal to or greater than that of the originating instance providing its replication stream, there need be little concern about having too many helper processes.



### Note

There may be other reasons for a replicating instance to lag behind its originating instance during replication. Helper processes cannot improve situations such as the following:

- There is a bottleneck in the network between the originating and replicating instances--increase the network bandwidth or use compression.
- The hardware of the replicating instance is not as capable as that of the hardware on the originating instance--upgrade the hardware of the replicating instance.

## Filters

A Filter is a conversion program that transforms a replication stream to a desired schema. It operates as a traditional UNIX filter, reading from STDIN and writing to STDOUT. Both input and output use the GT.M journal extract format. A filter can operate on an originating instance or a replicating instance. When the originating instance is an older application version, a filter can change the update records from the old schema to the new schema. When the originating instance is the newer application version, a filter can change the update records from the new schema to the old schema. Once you have logic for converting records in the old schema to the new schema, the per record code serves as the basis for the filter by replacing the scanning logic with logic to read the extract format and extract the update and completing the filter by reassembling the revised record(s) into the GT.M extract format.

For complete redundancy during rolling upgrades, you must also have a filter that changes transactions from the new schema to the old schema. The principal restriction in creating schema change filters is that the filter must not change the number of transactions in the replication stream, since GT.M uses the journal sequence numbers for checking and restoring the logical equivalence of instances.

This means:

- If a replication stream contains transactions, for each input transaction, the filter must produce one and exactly one output transaction. It's acceptable for a transaction to be empty, that is, to make no database updates.
- If an update in a replication stream is outside a transaction, it is considered a transaction in that the journal sequence number is to be incremented by one.
- If the schema change requires a single database update, simply emit the new update in the output stream.
- If the schema change requires no database updates in the new schema, emit a single empty transaction.
- If the schema change requires multiple database updates in the new schema, create a transaction, and package all the updates inside that transaction.

## Replication Instance File

A Replication Instance file maintains the current state of an instance. It also serves as a repository of the history of the journal sequence numbers that are generated locally or received from other instances.

It includes 3 sections -- File Header, Source Server slots, and History Records.

The File Header section records information about the current instance, such as semaphore and shared memory ids of the Journal and Receive Pool, journal sequence number of the current instance.

The Source Server slots store information for each replicating instance for which a Source Server is started. A slot stores the name of the replicating instance, the last transmitted sequence number, and the sequence number when the Source Server was last connected to the originating instance (Connect Sequence Number).

A Replication Instance file has 16 slots. Initially, all are unused. A Source Server replicating to a replicating instance for the first time utilizes an unused slot to store the information and any future Source Server process replicating to the same replicating instance updates this information.

If an unused slot is not available, the first time a Source Server is started to replicate to an instance, the slot for the least recently started replicating instance is reused, and the information that is previously stored in that slot is overwritten. Any subsequent mupip replic -source on the preempted replicating instance generates a REPLINSTSECNONE message.

Preemption of slots does not occur if an instance connects to no more than 16 different replicating instances throughout its lifetime.

In the History Records section, the history of an instance is maintained as a set of records. A new history record is added to the tail of the instance file whenever an instance changes from being an originating instance to replicating instance or vice versa --the only exception being when a history record is removed from the tail of the instance file when updates are rolled back from the database as part of a mupip journal -rollback. Every record identifies a range of journal sequence numbers and the name of the originating instance that generated those journal records. The first history record starts with the current journal sequence number of the instance.

When an originating instance transmits a sequence number to a replicating instance, the originating instance name is recorded as "Root Primary Instance Name" in the replication instance file history of both the instances. The same rule applies when a replicating instance is acting as an originating instance for another replicating instance downstream.

This history is crucial for determining the journal sequence numbers through which both instances are synchronized when two instances attempt to connect. This journal sequence number is determined by going back in the history of both the instance files and finding the earliest shared journal sequence number that was generated by the same originating instance. If the shared journal sequence number matches the current journal sequence number of the replicating instance, the Receiver Server on the replicating instance continues with normal replication. Otherwise, a mupip journal -rollback -fetchresync must be performed on the replicating instance to rollback to a common synchronization point from which the originating instance can transmit updates to allow it to catch up.



### Note

It is necessary to take a backup of the Replication Instance file that is consistent with the snapshot of the database files in the backup. mupip backup -replinstance creates a backup of the Replication Instance file. If the replication instance file is damaged or deleted, a new instance file must be created, and all replicating instance downstream must be recreated from backups.

---

## Implementing Replication and Recovery

A transaction processing application makes a series of database updates. GT.M executes these updates online or from data-driven logic, commonly called "batch."

1. Online Update: An online update arrives at GT.M as a message from a client.
2. Driven by internal information, such as balances in end-of-day, or external information, such as a list of checks from a clearinghouse.

The processing model in each case is a transaction or a unit of work initiated by client input such as a request to transfer funds from one account to another, or as the next logical unit of work such as posting interest on the next account. This general model holds both for applications where users login directly to a host (perhaps using terminal emulation from a workstation) and those where a client communicates with a host server process. This section lists key considerations for a transaction processing application to:

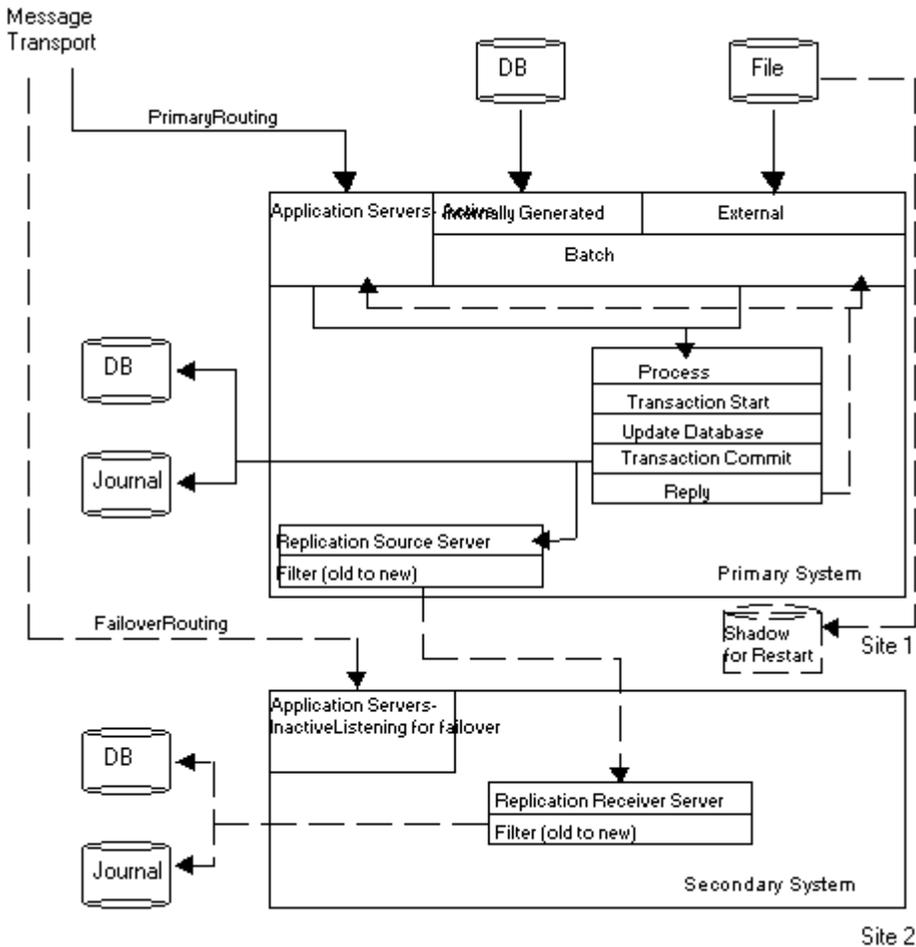
- reliably perform online and batch updates on GT.M
- implement an LMS configuration in a tiered environment, and
- facilitate recovery in a cutover event.

## Application Architecture

FIS recommends you to plan upfront for database consistency while designing the architecture of an LMS application. Some of the planning parameters for application's architecture may include:

- Always package all database updates into transactions that are consistent at the level of the application logic using the TSTART and TCOMMIT commands. For information on commands, refer to the "Commands" chapter in the GT.M Programmer's Guide. For any updates not so packaged, ensure that the database is logically consistent at the end of every M statement that updates the database; or that there is application logic to check, and restore application-level consistency when the database recovers after a crash.
- Ensure that internally driven batch operations store enough information in the database to enable an interrupted batch operation to resume from the last committed transaction. In case an originating instance fails in the middle of a batch process, a new originating instance (previously a replicating instance) typically must resume and complete the batch process.
- If the application cannot or does not have the ability to restart batch processes from information in the database, copy a snapshot of the database to a replicating instance just before the batch starts. In case an originating instance fails, restore the new originating instance to the beginning of the batch operations, and restart the batch operation from the beginning on the new originating instance.
- Ensure that externally driven batch processing also has the ability to resume. The external file driving the batch operation must be available on the replicating instance before starting the batch operation on the originating instance. This is required to handle originating instance failure during the batch process.
- GT.M produces an error for updates outside the set of database files defined by the instance file. External references are not prohibited as such. In other words, there can be global directory and instance configurations where an external reference update falls within the instance and works correctly. Read references outside an instance are permitted because they currently do not engage replication.

## Database Replication



This diagram illustrates an application architecture that can reliably perform batch and online updates in a tiered environment. It addresses the online updates via the Message Transport (which has to be able to reroute communications to the current originating instance after a cutover) and batch updates via an external file (which has to be made available to the current originating instance after a cutover).

An application server is a GT.M process that accepts, processes, and responds to messages provided through the Message Transport. They may exist as a bunch of application servers in a “cloud” of size determined by the size of the node and the needs of the application. On the originating instance, an application server process receives messages and processes application transactions. The application logic issues the TSTART command and a series of SET (also KILL and MERGE) commands that [potentially/provisionally] update the database, then a TCOMMIT command to finalize the transaction. The process may directly WRITE a reply, but another process may act as an agent that takes that reply from a database record and sends it to the originator.

## Implement a Message Delivery System

This section describes how a well-designed messaging system makes an application's architecture more cutover-ready by using an example in which the originating instance fails after the TCOMMIT, but before the system generates a reply and transmits it to the client.

As noted in the previous section, application servers on the originating instance respond to messages from clients delivered over a network for online operations in a tiered environment. Each client message results in zero (inquiry) or one update transaction on the server. The network delivering messages must be robust. This means each message must either be delivered

## Database Replication

exactly once to an application server on the originating instance, or result in client notification of the delivery failure. The messaging system must handle situations such as failure on the originating instance after the client transmits the message but before the originating instance receives it. Integration of the message delivery system with the logic determining whether an instance is an originating instance or replicating instance at any time reduces risk and switch over time.

Application logic typically responds to client messages with a reply generated immediately after the TCOMMIT for a transaction. The application and the message architecture must handle the scenario in which the originating system fails after the TCOMMIT, but before the system generates a reply and transmits it to the client. In such a scenario, the client waits for a response and eventually timesout and retries the message.

An LMS application can handle this situation by designing the message structure to have a unique message identifier (MSGID), and the application to include the MSGID in the database as part of the TCOMMIT.

If the originating instance crashes after committing the transaction and the cutover logic makes the former replicating instance the new originating instance--This new originating instance, then, receives the retried message that has the same MSGID from the client. In this case, one of the following can occur:

- The database shows that the transaction corresponding to the MSGID in the message was processed. The server could then reply that this transaction was processed. A more sophisticated approach computes the response to the client within the transaction, and to stores it in the database as part of the transaction commit. Upon receipt of a message identified as a retry of a previously processed message, the server returns the stored response from the database to the client.
- The database shows the transaction as unprocessed. In this case, the new originating instance processes the transaction. At this time, it is unknown whether the former originating instance processed the transaction before going down. If it was not processed, there is no issue. If it was processed but not replicated, GT.M rollback logic rolls it back when the former originating instance comes up as a replicating instance, and it must be reconciled either manually or automatically, from the rollback report (since the result of processing the first time may be different from the result of processing the second time).

## System Requirements

This section describes the system requirements that are necessary to implement an application with an LMS configuration.

### Root Primary Status Identification

GT.M does not make any decisions regarding originating or replicating operations of an instance. You must explicitly specify `-ROOTPRIMARY` to identify an instance as current originating instance during application startup.

To implement a robust, continuously available application, each application instance must come up in the correct state. In particular, there must be exactly one originating instance (`-ROOTPRIMARY`) at any given time. All database update operations on replicated databases must take place on the originating instance. LMS prohibits independent logical database updates on instances other than the originating instance.



#### Note

MUPIP BACKUP -ONLINE and MUPIP REORG -ONLINE update control information or physical representations, not the logical database contents, and can operate freely on a replicating instance.

## Cutover

Cutover is the process of reconfiguring an LMS application so that a replicating instance takes over as the current originating instance. This might be a planned activity, such as bringing down the originating instance for hardware maintenance, or it

may be unplanned such as maintaining application availability when the originating instance or the network to the originating instance goes down.

Implementing and managing cutover is outside the scope of GT.M. FIS recommends you to adhere to the following rules while designing cutover:

1. Always ensure that there is only one originating instance at any given time where all database updates occur. If there is no originating instance, the LMS application is also not available.
2. Ensure that messages received from clients during a cutover are either rejected, so the clients timeout and retry, or are buffered and sent to the new originating instance.
3. Always configure a former originating instance to operate as a replicating instance whenever it resumes operations or comes back online after a crash.
4. Failing to follow these rules may result in the loss of database consistency between an originating instance and its replicating instances.



### Important

A cutover is a wholesome practice for maximizing business continuity. FIS strongly recommends setting up a cutover mechanism to keep a GT.M application up in the face of disruptions that arise due to errors in the underlying platform. In environments where a cutover is not a feasible due to operational constraints, consider setting up an Instance Freeze mechanism for your application. For more information, refer to “Instance Freeze” (page 193).

## Instance Freeze

In the event of run-time conditions such as no disk space, I/O problems, or disk structure damage, some operational policies favor deferring maintenance to a convenient time as long as it does not jeopardize the functioning of the GT.M application. For example, if the journal file system runs out of disk space, GT.M continues operations with journaling turned off and moves to the replication WAS\_ON state until journaling is restored. If there is a problem with one database file or journal file, processes that update other database regions continue normal operation.

Some operational policies prefer stopping the GT.M application in such events to promptly perform maintenance. For such environments, GT.M has a mechanism called "Instance Freeze".

The Instance Freeze mechanism provides an option to stop all updates on the region(s) of an instance as soon as a process encounters an error while writing to a journal or database file. This mechanism safeguards application data from a possible system crash after such an error.

The environment variable `gtm_custom_errors` specifies the complete path to the file that contains a list of errors that should automatically stop all updates on the region(s) of an instance. The error list comprises of error mnemonics (one per line and in capital letters) from the GT.M Message and Recovery Guide. The GT.M distribution kits include a `custom_errors_sample.txt` file which can be used as a target for the `gtm_custom_errors` environment variable.



### Note

When a processes that is part of an instance configured for instance freeze behavior encounters an error with journaling, it freezes the instance and invokes its own error trap even if it does not have the `gtm_custom_errors` environment variable set.

## Database Replication

You can enable the Instance Freeze mechanism selectively on any region(s) of an instance. For example, a region that represents a patient or financial record may qualify for an Instance Freeze whereas a region with an easily rebuilt cross reference index may not. You can also promptly freeze an instance irrespective of whether any region is enabled for Instance Freeze.

`MUPIP SET -[NO]INST[_FREEZE_ON_ERROR] [-REGION|-FILE]` enables custom errors in region to automatically cause an Instance Freeze. `MUPIP REPLICATE -SOURCE -FREEZE={ON|OFF} -[NO]COMMENT[="string"]` promptly sets or clears an Instance Freeze on an instance irrespective of whether any region is enabled for Instance Freeze (with `MUPIP SET -INST_FREEZE_ON_ERROR`).

A process that is not in a replicated environment ignores `$gtm_custom_errors`. The errors in the custom errors file must have a context in one of the replicated regions and the process recognizing the error must have the replication Journal Pool open. For example, an error like UNDEF cannot cause an Instance Freeze because it is not related to the instance. It also means that, for example, standalone MUPIP operations can neither cause nor honor an Instance Freeze because they do not have access to the replication Journal Pool. A process with access to the replication Journal Pool must honor an Instance Freeze even if does not have a custom error file and therefore cannot initiate an Instance Freeze.

Depending on the error, removing an Instance Freeze is operator driven or automatic. GT.M automatically removes Instance Freezes that are placed because of no disk space; for all other errors, Instance Freeze must be cleared manually by operator intervention. For example, GT.M automatically places an Instance Freeze when it detects a DSKNOSPCAVAIL message in the operator log. It automatically clears the Instance Freeze when an operator intervention clears the no disk space condition. During an Instance Freeze, GT.M modifies the NOSPACEEXT message from error (-E-) to warning (-W-) to indicate it is performing the extension even though the available space is less than the specified extension amount. The following errors are listed in the `custom_errors_sample.txt` file. Note that GT.M automatically clears the Instance Freeze set with DSKNOSPCAVAIL when disk space becomes available. All other errors require operator intervention.

- Errors associated with database files caused by either I/O problems or suspected structural damage: DBBMLCORRUPT, DBDANGER, DBFSYNCERR, DSKNOSPCAVAIL, GBLOFLOW, GVDATAFAIL, GVDATAGETFAIL, GVGETFAIL, GVINCRFAIL, GVKILLFAIL, GVORDERFAIL, GVPUTFAIL, GVQUERYFAIL, GVQUERYGETFAIL, GVZTRIGFAIL, OUTOFSPACE, TRIGDEFBAD.
- Errors associated with journal files caused by either I/O problems or suspected structural damage: JNLACCESS, JNLCLOSE, JNLCLOSED, JNLEXTEND, JNLFILECLOSERR, JNLFILEXTERR, JNLFILOPN, JNLFLUSH, JNLFSYNCERR, JRTNULLFAIL, JNLRDERR, JNLREAD, JNLVSIZE, JNLWRERR.

During an Instance Freeze, attempts to update the database and journal files hang but operations like journal file extract which do not require updating the database file(s) continue normally. When an Instance Freeze is cleared, processes automatically continue with no auxiliary operational or programmatic intervention. The Instance Freeze mechanism records both the freeze and the unfreeze in the operator log.



### Note

As of V6.0-000, the Instance Freeze facility is a field test grade implementation. Because there are a large number of errors that GT.M can recognize and because GT.M has several operational states, the GT.M team has tested the errors in the `custom_errors_sample.txt` which are consistent with what we expect to be common usage. If you experience problems trying to add other errors or have concerns about plans to add other errors, please consult your GT.M support channel.

## TLS/SSL Replication

GT.M includes a plugin reference implementation that provides the functionality to secure the replication connection between instances using Transport Layer Security (TLS; previously known as SSL). Just as database encryption helps protect against unauthorized access to a database by an unauthorized process that is able to access disk files (data at rest), the plugin reference

implementation secures the replication connection between instances and helps prevent unauthorized access to data in transit. FIS has tested GT.M's replication operations of the TLS plugin reference implementation using OpenSSL (<http://www.openssl.org>). A future GT.M release may include support for popular and widely available TLS implementations / cryptography packages other than OpenSSL. Note that a plug-in architecture allows you to choose a TLS implementation and a cryptography package. FIS neither recommends nor supports any specific TLS implementation or cryptography package and you should ensure that you have confidence in and support for whichever package that you intend to use in production.



### Note

Database encryption and TLS/SSL replication are just two of many components of a comprehensive security plan. The use of database encryption and TLS replication should follow from a good security plan. This section discusses encrypted GT.M replicating instances and securing the replication connection between them using TLS/SSL; it does not discuss security plans.



### Important

You can setup TLS replication between instances without using GT.M Database Encryption. GT.M Database Encryption is not a prerequisite to using TLS replication.

The general procedure of creating a TLS replication setup includes the following tasks:

1. Create a new database or use an existing one.
2. “Creating a root-level certification authority” (page 195)
3. “Creating leaf-level certificates” (page 196)
4. “Creating a configuration file” (page 198)
5. Enabling replication and starting the Source and Receiver Servers with the TLSID qualifier.

## Creating a root-level certification authority

To use TLS, the communicating parties need to authenticate each other. If the authentication succeeds, the parties encrypt the subsequent communication. TLS authentication uses certificates signed by Certificate Authorities (CAs). Certificate Authorities' certificates themselves are signed (and trusted) by other CAs eventually leading to a Root CA, which self-signs its own certificate. Although the topic of certificates, and the use of software such as OpenSSL is well beyond the scope of GT.M documentation, the steps below illustrate the quick-start creation of a test environment using Source and Receiver certifications with a self-signed Root CA certificate.

Creating a root certificate authority involves three steps.

1. Generate a private key with the OpenSSL command: **openssl genrsa -des3 -out ca.key 4096**. The command prompts for a password with which to protect the private key.
2. Generate a self-signed certificate with the OpenSSL command: **openssl req -new -x509 -days 365 -key ca.key -out ca.crt**. The command first prompts for the password of the private key followed by a series of interactive queries regarding the attributes of the certificate. Below is sample output:

```
Enter pass phrase for ca.key:
```

```
You are about to be asked to enter information that will be incorporated into your certificate request.
```

```
What you are about to enter is what is called a Distinguished Name or a DN.
```

## Database Replication

There are quite a few fields but you can leave some blank  
For some fields there will be a default value,  
If you enter '.', the field will be left blank.

-----

```
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:PA
Locality Name (eg, city) []:Malvern
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Example Pvt. Ltd
Organizational Unit Name (eg, section) []:Certificate Authority
Common Name (e.g. server FQDN or YOUR name) []:www.example.com
Email Address []:example@example.com
```

At this point, ca.crt is a root certificate that can be used to sign other certificates (including intermediate certificate authorities). The private key of the root certificate must be protected from unauthorized access.

## Creating leaf-level certificates

The root certificate is used to sign regular, leaf-level certificates. Below are steps showing the creation of a certificate to be used to authenticate a GT.M Source Server with a GT.M Receiver Server (and vice-versa).

1. Generate a private key. This is identical to step (a) of root certificate generation.
2. Generate a certificate sign request with the OpenSSL command **openssl req -new -key client.key -out client.csr**. The command first prompts for the password of the private key followed by a series of interactive queries regarding the attributes of the certificate. Below is sample output:

```
Enter pass phrase for client.key:
You are about to be asked to enter information that will be incorporated into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
```

-----

```
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:PA
Locality Name (eg, city) []:Malvern
Organization Name (eg, company) [Internet Widgits Pty Ltd]:XYZQ International
Organizational Unit Name (eg, section) []: OurSourceServer
Common Name (e.g. server FQDN or YOUR name) []:www.xyzq.com
Email Address []:xyzq@xyzq.com
Please enter the following 'extra' attributes to be sent with your certificate request
A challenge password []:challenge
An optional company name []:XYZQ Pvt. Ltd
```

Typically, organization that generates the certificate sign then sends it to a certificate authority (or a root certificate authority), which audits the request and signs the certificate with its private key, thereby establishing that the certificate authority trusts the company/organization that generated the certificate and requested its signing. In this example, we sign the certificate sign request with the root certificate generated above.

3. Sign the certificate sign request with an OpenSSL command like:

## Database Replication

```
openssl ca -config $PWD/openssl.cnf -in client.ccr -out client.crt
```

The output of this command looks like the following:

```
>You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
Country Name (2 letter code) [US]: US
State or Province Name (full name) [Philadelphia]:Illinois
City (e.g., Malvern) [Malvern]:Chicago"
Organization Name (eg, company) [FIS]:FIS
Organizational Unit Name (eg, section) [GT.M]:GT.M
Common Name (e.g. server FQDN or YOUR name) [localhost]:fisglobal.com
Ename Address (e.g. helen@gt.m) []:root@gt.m
```

```
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
Using configuration from /usr/lib/ssl/openssl.cnf
Enter pass phrase for ./certs/ca.key:
Check that the request matches the signature
Signature ok
```

```
Certificate Details:
Serial Number: 14 (0xe)
Validity
Not Before: Jun 11 14:06:53 2014 GMT
Not After : Jun 12 14:06:53 2014 GMT
Subject:
countryName = US
stateOrProvinceName = Illinois
organizationName = FIS
organizationalUnitName = GT.M
commonName = fisglobal.com
emailAddress = helen@gt.m
X509v3 extensions:
X509v3 Basic Constraints:
CA:FALSE
Netscape Comment:
OpenSSL Generated Certificate
X509v3 Subject Key Identifier:
96:FD:43:0D:0A:C1:AA:6A:BB:F3:F4:02:D6:1F:0A:49:48:F4:68:52
X509v3 Authority Key Identifier:
keyid:DA:78:3F:28:8F:BC:51:78:0C:5F:27:30:6C:C5:FE:B3:65:65:85:C9
```

```
Certificate is to be certified until Jun 12 14:06:53 2014 GMT (1 days)
Sign the certificate? [y/n]:y
```

```
1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```



## Important

Keep the self-signed root certificate authority and leaf-level certificates in a secure location. Protect their directories with 0500 permissions and the individual files with 0400 permissions so that unauthorized users cannot access them.

Please refer to OpenSSL documentation <http://www.openssl.org/docs/> for information on how to create intermediate CAs, Diffie-Hellman parameters, Certificate Revocation Lists, and so on.

## Creating a configuration file

The configuration file is divided into two sections—Database encryption section and the TLS section. The database encryption section contains the list of database files and their corresponding key files and the TLS section contains a TLSID label that identifies the location of root certification authority certificate in PEM format and leaf-level certificate with their corresponding private key files. Note that the use of the `gtmconfig` environment variable require the `libconfig` library to be installed.

After creating a leaf-level certificate that is signed by a self-signed root certificate, create a configuration file (one for Source and the other for Receiver Server) with the following format:

```
tls: {
  verify-depth: 7;
  CAfile: "/path/to/ca.crt";
  tls : {
    format: "PEM";
    cert: "/path/to/client.crt";
    key: "/path/to/client.key";
  };
};
```

where **tls** specifies the TLSID that is used to start the Source/Receiver Server, **CAfile** specifies the path to the root certification authority, **cert** specifies the the path to leaf-level certificate and **key** specifies the path to the private key file.

Set the `gtmconfig` environment variable to point to the path to the configuration file. The environment variable `gtmtls_passwd_<tlsid>` must specify an obfuscated version of the password for the client's private key. Use the `maskpass` utility provided with your GT.M distribution to create an obfuscated password.

Here is a sample configuration file:

```
/* Database encryption section */
database: {
  keys: (
    {
      dat: "/tmp/mumps.dat"; /* Encrypted database file. */
      key: "/tmp/mumps.key"; /* Encrypted symmetric key. */
    },
    {
      dat: "/tmp/a.dat";
      key: "/tmp/a.key";
    },
    ...
  );
}
```

## Database Replication

```
/* TLS section */

tls: {
    /* Certificate Authority (CA) verify depth provides an upper limit on the number of CAs to look up for
    verifying
    ▶ a given
        * certificate. The depth count is described as ''level 0:peer certificate'', ''level 1: CA certificate'',
        * ''level 2: higher level CA certificate'', and so on. The default verification depth is 9.
        */
    verify-depth: 7;

    /* CAfile: points to a file, in PEM format, describing the trusted CAs. The file can contain several CA
    ▶ certificates identified by:
        * -----BEGIN CERTIFICATE-----
        * ... (CA certificate in base64 encoding) ...
        * -----END CERTIFICATE-----
        * sequences.
        */
    CAfile: "/home/jdoe/current/tls/certs/CA/gtmCA.crt";

    /* CApath: points to a directory containing CA certificates in PEM format. The files each contain one CA
    ▶ certificate. The files are
        * looked up by the CA subject name hash value, which must hence be available. If more than once certificate
        with
    ▶ the same
        * name hash value exists, the extension must be different (e.g. 9d66eef0.0, 9d66eef0.1 etc). The directory
        is
    ▶ typically
        * created by the OpenSSL tool 'c_rehash'.
        */
    CApath: "/home/jdoe/current/tls/certs/CA/";

    /* Diffie-Hellman parameters used for key-exchange. Either none or both have to be specified. If neither is
    ▶ specified, then
        * then the data is encrypted with the same keys that are used for authentication.
        */
    dh512: "/home/jdoe/current/tls/dh512.pem";
    dh1024: "/home/jdoe/current/tls/dh1024.pem";

    /* crl: points to a file containing list of revoked certificates. This file is created by the openssl
    utility. */
    crl: "/home/jdoe/current/tls/revocation.crl";

    /* Timeout (in seconds) for a given session. If a connection disconnects and resumes within this time
    interval,
    ▶ the session
        * is reused to speed up the TLS handshake. A value of 0 forces sessions to not be reused. The default value
        is 1
    ▶ hour.
        */
    session-timeout: 600;

    /* List of certificate/key pairs specified by identifiers. */
    PRODUCTION: {
        /* Format of the certificate and private key pair. Currently, the GT.M TLS plug-in only supports PEM
```

## Database Replication

```
format. */
    format: "PEM";
    /* Path to the certificate. */
    cert: "/home/jdoe/current/tls/certs/Malvern.crt";
    /* Path to the private key. If the private key is protected by a passphrase, an obfuscated version of
the
password
    * should be specified in the environment variable which takes the form gmtls_passwd_<identifier>.
For instance,
    * for the below key, the environment variable should be 'gmtls_passwd_PRODUCTION'.
    * Currently, the GT.M TLS plug-in only supports RSA private keys.
    */
    key: "/home/jdoe/current/tls/certs/Malvern.key";
};

DEVELOPMENT: {
    format: "PEM";
    cert: "/home/jdoe/current/tls/certs/BrynMawr.crt";
    key: "/home/jdoe/current/tls/certs/BrynMawr.key";
};
};
```



If you are using the environment variable `gtm_dbkeys` to point to the master key file for database encryption, please convert that file to the libconfig configuration file format as pointed to by the `$gtmencrypt_config` environment variable at your earliest convenience. Effective V6.1-000, the `gtm_dbkeys` environment variable and the master key file it points to are deprecated in favor of the `gtmencrypt_config` environment variable. Although V6.1-000 supports the use of `$gtm_dbkeys` for database encryption, FIS plans to discontinue support for it in the very near future. To convert master key files to libconfig format configuration files, please click on  to download the `CONVDBKEYS.m` program and follow instructions in the comments near the top of the program file. You can also download `CONVDBKEYS.m` from <http://tinco.pair.com/bhaskar/gtm/doc/articles/downloadables/CONVDBKEYS.m>.

## Network Link between Systems

GT.M replication requires a durable network link between all instances. The database replication servers must be able to use the network link via simple TCP/IP connections. The underlying transport may enhance message delivery, (for example, provide guaranteed delivery, automatic cutover and recovery, and message splitting and re-assembly capabilities); however, these features are transparent to the replication servers, which simply depend on message delivery and message receipt.

## Choosing between `BEFORE_IMAGE` and `NOBEFORE_IMAGE` journaling

Between `BEFORE_IMAGE` journaling and `NOBEFORE_IMAGE` journaling, there is no difference in the final state of a database / instance recovered after a crash. The difference between before image and nobefore journaling is in:

- the sequence of steps to recover an instance and the time required to perform them.
- the associated storage costs and IO bandwidth requirements.

## Recovery

When an instance goes down, its recovery consists of (at least) two steps: recovery of the instance itself: hardware, OS, file systems, and so on - say  $t_{sys}$ ;  $t_{sys}$  is almost completely<sup>4</sup> independent of the type of GT.M journaling.

For database recovery:

- With BEFORE\_IMAGE journaling, the time is simply that is needed to execute a mupip journal recover backward "\*" command or, when using replication, mupip journal recover -rollback. This uses before image records in the journal files to roll the database files back to their last epochs, and then forward to the most current updates. If this takes  $t_{bck}$ , the total recovery time is  $t_{sys}+t_{bck}$ .
- With NOBEFORE\_IMAGE journaling, the time is that required to restore the last backup, say,  $t_{rest}$  plus the time to perform a mupip journal -recover -forward "\*" command, say  $t_{fwd}$ , for a total recovery time of  $t_{sys}+t_{rest}+t_{fwd}$ . If the last backup is available online, so that "restoring the backup" is nothing more than setting the value of an environment variable,  $t_{rest}=0$  and the recovery time is  $t_{sys}+t_{fwd}$ .

Because  $t_{bck}$  is less than  $t_{fwd}$ ,  $t_{sys}+t_{bck}$  is less than  $t_{sys}+t_{fwd}$ . In very round numbers,  $t_{sys}$  may be minutes to tens of minutes,  $t_{fwd}$  may be tens of minutes and  $t_{bck}$  may be in tens of seconds to minutes. So, recovering the instance A might (to a crude first approximation) be a half order of magnitude faster with BEFORE\_IMAGE journaling than with NOBEFORE\_IMAGE journaling. Consider two deployment configurations.

1. Where A is the sole production instance of an application, halving or quartering the recovery time of the instance is significant, because when the instance is down, the enterprise is not in business. The difference between a ten minute recovery time and a thirty minute recovery time is important. Thus, when running a sole production instance or a sole production instance backed up by an underpowered or not easily accessed, "disaster recovery site," before image journaling with backward recovery is the preferred configuration ofis the preferred configuration ofbetter suits a production deployment. Furthermore, in this situation, there is pressure to bring A back up soon, because the enterprise is not in business - pressure that increases the probability of human error.
2. With two equally functional and accessible instances, A and B, deployed in an LMS configuration at a point in time when A, running as the originating instance replicating to B, crashes, B can be switched from a replicating instance to an originating instance within seconds. An appropriately configured network can change the routing of incoming accesses from one instance to the other in seconds to tens of seconds. The enterprise is down only for the time required to ascertain that A is in fact down, and to make the decision to switch to B— perhaps a minute or two. Furthermore, B is in a "known good" state, therefore, a strategy of "if in doubt, switchover" is entirely appropriate. This time,  $t_{swch}$ , is independent of whether A and B are running BEFORE\_IMAGE journaling or NOBEFORE\_IMAGE journaling. The difference between BEFORE IMAGE journaling and NOBEFORE\_IMAGE journaling is the difference in time taken subsequently to recover A, so that it can be brought up as a replicating instance to B. If NOBEFORE\_IMAGE journaling is used and the last backup is online, there is no need to first perform a forward recovery on A using its journal files. Once A has rebooted:
  - Extract the unreplicated transactions from the crashed environment
  - Connect the backup as a replicating instance to B and allow it to catch up.



### Note

Applications that can take advantage the forthcoming LMX capability will essentially make  $t_{swch}$  zero when used with a suitable front-end network.

<sup>4</sup>The reason for the "almost completely" qualification is that the time to recover some older file systems can depend on the amount of space used.

## Comparison other than Recovery

Cost	The cost of using an LMS configuration is at least one extra instance plus network bandwidth for replication. There are trade-offs: with two instances, it may be appropriate to use less expensive servers and storage without materially compromising enterprise application availability. In fact, since GT.M allows replication to as many as sixteen instances, it is not unreasonable to use commodity hardware <sup>a</sup> and still save total cost.
Storage	Each extra instance of course requires its own storage for databases and journal files. Nobefore journal files are smaller than the journal files produced by before image journaling, with the savings potentially offset if a decision is made to retain an online copy of the last backup (whether this nets out to a saving or a cost depends on the behavior of the application and on operational requirements for journal file retention).
Performance	<p>IO bandwidth requirements of nobefore journaling are less than those of before image journaling, because GT.M does not write before image journal records or flush the database.</p> <ul style="list-style-type: none"> <li>• With before image journaling, the first time a database block is updated after an epoch, GT.M writes a before image journal record. This means that immediately after an epoch, given a steady rate of updates, there is an increase in before image records (because every update changes at least one database block and generates at least one before image journal record). As the epoch proceeds, the frequency of writing before image records falls back to the steady level<sup>b</sup> – until the next epoch. Before image journal records are larger than journal records that describe updates.</li> <li>• At epochs, both before image journaling and nobefore journaling flush journal blocks and perform an fsync() on journal files<sup>c</sup>. When using before image journaling, GT.M ensures all dirty database blocks have been written and does an fsync()<sup>d</sup>, but does not take these steps.</li> </ul> <p>Because IO subsystems are often sized to accommodate peak IO rates, choosing NOBEFORE_IMAGE journaling may allow more economical hardware without compromising application throughput or responsiveness.</p>

<sup>a</sup>GT.M absolutely requires the underlying computer system to perform correctly at all times. So, the use of error correcting RAM, and mirrored disks is advised for production instances. But, it may well be cost effective to use servers without redundant power supplies or hot-swappable components, to use RAID rather than SAN for storage, and so on.

<sup>b</sup>How much the steady level is lower depends on the application and workload.

<sup>c</sup>Even flushing as many as 20,000 journal buffers, which is more than most applications use, is only 10MB of data. Furthermore, when GT.M's SYNC\_IO journal flag is specified, the fsync() operation requires no physical IO.

<sup>d</sup>The volume of dirty database blocks to be flushed can be large. For example, 80% of 40,000 4KB database blocks being dirty would require 128MB of data to be written and flushed.

## Database Repair

A system crash can, and often will, damage a database file, leaving it structurally inconsistent. With before image journaling, normal MUPIP recovery/rollback repairs such damage automatically and restores the database to the logically consistent state as of the end of the last transaction committed to the database by the application. Certain benign errors may also occur (refer to the "Maintaining Database Integrity" chapter). These must be repaired on the (now) replicating instance at an appropriate time, and are not considered "damage" for the purpose of this discussion. Even without before image journaling, a replicating instance (particularly one that is multi-site) may have sufficient durability in the aggregate of its instances so that backups (or copies) from an undamaged instance can always repair a damaged instance.



### Note

If the magnetic media of the database and/or the journal file is damaged (e.g., a head crash on a disk that is not mirrored), automatic repair is problematic. For this reason, it is strongly recommended that organizations use hardware mirroring for magnetic media.



### Caution

Misuse of UNIX commands, such as kill-9 and ipcrm, by processes running as root can cause database damage.

Considering the high level at which replication operates, the logical dual-site nature of GT.M database replication makes it virtually impossible for related database damage to occur on both originating and replicating instances.

To maintain application consistency, do not use DSE to repair or change the logical content of a replicated region on an originating instance.



### Note

Before attempting manual database repair, FIS strongly recommends backing up the entire database (all regions).

After repairing the database, bring up the replicating instance and backup the database with new journal files. MUPIP backup online allows replicating to continue during the backup. As stated in the Journaling chapter, the journal files prior to the backup are not useful for normal recovery.

## Procedures

GT.M database replication ensures that the originating instance and its replicating instances are logically identical, excluding latency in replication. During rolling upgrades, the originating instance and its replicating instances are logically equivalent but the exact storage of M global variables may differ. FIS recommends you to ensure that all database updates to replicated regions must occur only on the originating instance. GT.M replicates all changes on the originating instance to its replicating instances.

## Normal Operation

LMS applications require procedures for a number of situations, including both normal operation and failure scenarios. Although LMS applications can operate manually, the requirements are complex enough to warrant automation via scripting. It is essential to carefully design, implement, and test the procedures to provide an automatic cutover to a replicating instance within seconds. GT.M provides the required functionality to implement continuous availability via LMS; however, smooth operation requires significant system engineering to implement.

- Determine current and prior instance status (originating/replicating).
- Perform necessary database recovery/rollback. Different failure modes have different recovery scenarios, which may include rollback of previously committed transactions for subsequent processing. Reconcile rolled back transactions automatically or manually.
- Create new journal files. Although not required, this simplifies system administration.

## Database Replication

- Bring up the Source Server on each replicating instance to establish the Journal Pool. In case of a cutover, the new originating instance needs the Journal Pool established to replicate data. On the originating instance, bring up the Source Server in active mode.
- On a replicating instance, bring up the Source Server in passive mode. Start the GT.M Receiver Server on the replicating instance.
- When you start an originating instance, start any application servers. Optionally start application servers on a replicating instance to facilitate a faster cutover; however, they must not perform updates to any replicating instance (Always remember the golden rule: all database updates should be performed only on the originating instance).
- If you are starting a new originating instance and batch operations were in process when the former originating instance went down, restart those batch operations on the new originating instance.

## Startup

An initial startup of an application instance has no prior status. There are two ways to bring up an LMS application: bring up the originating instance followed by the replicating instance, or bring both up at the same time.

## Single-Site

- When launching a multi-site application as a conventional single-site, versus launching it as a single-site for which a replicating instance will be launched later, always establish the Journal Pool before any M processes access the database. Bring up the Source Server in passive mode to create the Journal Pool. Then bring up the application. Later, switch the Source Server to active mode when the replicating instance comes up.
- Perform database recovery/rollback to the last consistent state, since the system may previously have crashed. If the database was shut down cleanly, the recovery/rollback to the last consistent state is essentially a "no-op." This operation restores the database to the last committed transaction. (This assumes there is no media damage.)
- Create new journal files.
- Start the Source Server in passive mode.
- Start the application servers.
- If the state of the database indicates that batch operations were in process, restart batch operations.

## Dual-site - Concurrent Startup

FIS does not recommend concurrent startup of instances in an LMS configuration because of the possibility of a network or timing problem resulting in multiple originating or replicating instances. However, in a dual-site configuration it is possible to concurrently startup the originating and replicating instance. The steps are as follows:

- Use an external agent to identify the originating and replicating instances and then bring up both sites simultaneously.
- Ensure that both databases are logically identical and have the same journal sequence number. (When starting up, an instance considers its journal sequence number to be the maximum `reg_seqno` of any replicated region. Thus, if the originating instance and its replicating instance do not have identical files, that is if they are logically identical but configured differently, ensure that at least one region on the replicating instance has `reg_seqno` and `resync_seqno` the same as the largest `reg_seqno` on the originating instance.) No rollback/recovery should be required on either site.
- If there is any possibility that the two are not identical, do not bring both up concurrently.

## Single-Site to Multi-Site

FIS recommends you to bring up the application as a single-site first and then bring up any replicating instances.

### *Setting up an A->B replication configuration for the first time*

Perform the following steps to setup an originating instance Philadelphia and its replicating instance Shanghai.

- On **Philadelphia**:
  - Turn on replication.
  - Create the replication instance file.
  - Start the Source Server.
- On **Shanghai**:
  - Turn on replication.
  - Create the replication instance file.
  - Start the passive Source Server.
  - Start the Receiver Server.



## Download Example

**msr\_proc1.tar.gz** contains a ready-to-run example of setting up an A->B replication configuration. Click  to download **msr\_proc1.tar.gz** or open directly from [http://tinco.pair.com/bhaskar/gtm/doc/books/ao/UNIX\\_manual/msr\\_proc1.tar.gz](http://tinco.pair.com/bhaskar/gtm/doc/books/ao/UNIX_manual/msr_proc1.tar.gz). Use the following command sequence to run this example:

#	Philadelphia - originating instance	Shanghai - replicating instance
1	<code>\$ source ./env Philadelphia</code>	<code>\$ source ./env Shanghai</code>
2	<code>\$ ./repl_setup</code>	<code>\$ ./repl_setup</code>
3	<code>\$ ./originating_start Philadelphia Shanghai</code>	<code>\$/replicating_start Shanghai</code>
4	<code>\$ mumps -r %XCMD 'set ^A="Philadelphia"</code>	<code>\$ mumps -r %XCMD 'write ^A' Philadelphia</code>
5	<code>\$/originating_stop</code>	<code>\$/replicating_stop</code>

This example demonstrates a command sequence that can be used to set up an A->B replication configuration on a local test system. No claim of copyright is made with respect to the scripts used in this example. YOU MUST UNDERSTAND, AND APPROPRIATELY ADJUST, THIS MODULE BEFORE USING IN A PRODUCTION ENVIRONMENT.

### *Setting up an A->P replication configuration for the first time*

## Database Replication

Perform the following steps to setup an originating instance Philadelphia and its supplementary instance Perth.

- On **Philadelphia**:
  - Turn on replication.
  - Create the replication instance file.
  - Start the Source Server.
- On **Perth**:
  - Turn on replication.
  - Create the replication instance file with the -supplementary qualifier.
  - Start the passive Source Server.
  - Start the Receiver Server and the Update Process with -updateresync="/path/to/bkup\_orig\_repl\_inst\_file" -initialize. Use the -updateresync -initialize qualifiers only once.
  - For subsequent Receiver Server and Update Process startups, perform -rollback -fetchresync start the Receiver Server and Update Process without the -updateresync -initialize qualifiers.



### Download Example

**msr\_proc2.tar.gz** contains a ready-to-run example of setting up an A->P replication configuration. Click  to download **msr\_proc2.tar.gz** or open directly from [http://tinco.pair.com/bhaskar/gtm/doc/books/ao/UNIX\\_manual/msr\\_proc2.tar.gz](http://tinco.pair.com/bhaskar/gtm/doc/books/ao/UNIX_manual/msr_proc2.tar.gz). Use the following command sequence to run this example:

#	Philadelphia - originating instance	Perth - supplementary instance
1	<code>\$ source ./env Philadelphia</code>	<code>\$ source ./env Perth</code>
2	<code>./db_create</code> <code>./repl_setup</code>	-
3	<code>./originating_start_suppl Philadelphia Perth</code>	-
4	<code>./backup_repl</code> #Note down the target location of orig_ri_backup	<code>./db_create</code> <code>./suppl_setup /path/to/orig_ri_backup</code>
5	<code>\$ mumps -r %XCMD 'set ^A=1'</code>	<code>\$ mumps -r %XCMD 'write ^A set ^B=2 write ^B'</code>
6	-	<code>./replicating_stop</code>
7	-	<code>./replicating_start_suppl_n Perth</code> #all subsequent startups of the Receiver Server.

## Database Replication

#	Philadelphia - originating instance	Perth - supplementary instance
8	<code>\$/originating_stop</code>	<code>\$/replicating_stop</code>

This example demonstrates a command sequence that can be used to set up an A->P replication configuration on a local test system. No claim of copyright is made with respect to the scripts used in this example. YOU MUST UNDERSTAND, AND APPROPRIATELY ADJUST, THIS MODULE BEFORE USING IN A PRODUCTION ENVIRONMENT.

### *Replicating Instance Starts after a Shut Down or Crash*

If the replicated regions are untouched after a shutdown or crash, simply restart the replicating instance. It automatically catches up with the originating instance. Perform the following steps to start a replicating instance after a shutdown operation or a crash:

- Recover/rollback database to last consistent state.
- Create new journal files.
- First start the passive Source Server and then the Receiver Server.
- Start the passive application servers, if appropriate.

### *Replicating Instance Starts from a copy of Originating Instance*

In this scenario, the database used to start up a replicating instance is logically identical to the database with which the originating instance started up, and has the same journal sequence number. When the replicating instance starts, all transactions applied to the originating instance replicate to the replicating instance as it catches up with the originating instance.

Perform the following steps to start a replicating instance from a copy of the originating instance:

- Load/restore the database files.
- Recreate the replication instance file.
- Turn replication on.
- Start passive Source Server, and then the Receiver Server.
- Start the passive application servers, if appropriate.
- Preserve all journal files on the originating instance along with back pointers. This is because the Source Server goes to the journal file to read transactions that are no longer in the journal pool.



### **Note**

Cease all GT.M activity in the replicating instance and RUNDOWN any open database files before copying database files onto a replicating instance, or creating new database files and loading them with extracts from the originating instance.

## Database Replication

### *Replicating Instance Starts from Backup of Originating Instance*

The more common scenario for bringing up a replicating instance is to take a backup of the originating instance and bring it up as a replicating instance. If the backup is a comprehensive backup, the file headers store the journal sequence numbers.

The backup should use the `-newjnlfiles` switch of MUPIP backup to create new journal files. Once the replicating instance becomes operational, the Source Server does not need to go back prior to the backup to find transactions to send to the replicating instance.

Perform the following steps to start a replicating instance from the backup of an originating instance:

- Load/restore the database. If the replicating database is not from a comprehensive or database backup from the originating instance, set the journal sequence number from the originating at the instant of the backup for at least one replicated region on the replicating instance.
- Create new journal files without back pointers to previous generations of journal files with the `-noprevjnlfile` flag. Since this database represents the beginning of this instance, it is not meaningful to have a previous generation journal file.
- Start the passive Source Server and then the Receiver Server with the `-updateresync` qualifier, along with the other startup qualifiers for the Receiver Server. As the originating instance stores the last journal sequence number transmitted to a replicating instance, this qualifier is required to force replication to start from the actual journal sequence number in the replicating instance. Without `-updateresync`, the Receiver Server may refuse to start replication because the journal sequence number in the replicating instance may be higher than what the originating instance expects.
- Start the passive application servers, if appropriate.
- Since the originating instance should not need to be rolled back to a state prior to the start of the backup, the generation link on the originating instance can be cut in the journal files created by the online backup command on the originating instance. Use `MUPIP SET -jnlfile <jnl_file> -noprevjnlfile` to cut the previous generation link for the journal file. Use the `-bypass` qualifier to override the standalone requirements of the SET command.

### **Multi-Site to Single-Site**

Under normal operation, when going from multi-site to single-site, shut down the Receiver Servers and Update Process as well as any passive Source Servers and any inactive application servers. Then perform the MUPIP RUNDOWN operation of the database.

For an extended shut down, switch the active Source Server on the originating instance to passive to prevent it from trying to connect with a replicating instance. Under normal scenarios, the originating instance operates alone, so that when a replicating instance comes back, replication resumes where it left off.

### **Normal Cutover**

Follow these steps to perform a controlled cutover in which the originating instance and a replicating instance switch roles. Such a step is necessary to test cutover or to bring the originating instance down for maintenance. In the following steps, A is the former originating instance and new replicating instance, while B is the former replicating instance and new originating instance.

1. Choose a time when database update rates are low to minimize the chances clients may time out and retry their messages, and when no batch processes are running.
2. The external system responsible for originating/replicating status identification should be made aware that Site B should now be the originating instance and Site A should now be the replicating instance. If the messaging layer between clients

## Database Replication

and servers differs from the external control mechanism, command it to route messages to the replicating instance. There may be a need to hold messages briefly during the cutover.

On A:

- Stop the application servers.
- Shut down the Source Server with an appropriate timeout. The timeout should be long enough to replicate pending transactions to the replicating instance, but not too long to cause clients to conclude that the application is not available. The GT.M default Source Server wait period is up to 120 seconds.
- Wait for B to become functional as the originating instance, and then query B (using `-EDITINSTANCE -SHOW`) for the journal sequence number when it became the originating instance, and roll back to this number. Transactions that are rolled off the originating instance become "non-replicated transactions" that must subsequently be reconciled on B, either automatically or manually.
- Create new journal files.
- Start the passive Source Server, and then the Receiver Server.
- Start the passive application servers, if appropriate.

On B:

- Shut down the Receiver Server with an appropriate timeout.
- Create new journal files.
- Make the passive Source Server active.
- Start the application servers (if they were previously passive, make them active).
- If the state of the database indicates that batch operations were in process, restart batch operations.
- Begin accepting online transactions.

### Shutting Down an instance

To shutdown an originating instance:

- Shut down all GT.M and mupip processes that might be attached to the Journal Pool.
- In case the originating instance is also a supplementary instance, shutdown the Receiver Server(s) (there might be more than one Receiver Server in future GT.M versions).
- Shut down all active and/or passive Source Servers.
- Execute `mupip rundown -region` to ensure that the database, Journal Pool, and Receiver Pool shared memory is rundown properly.

To shutdown a propagating instance:

- Shut down all replicating instance servers (Receiver Server, Update Process and its Helper processes).

## Database Replication

- Shutdown the originating instance servers (all active and/or passive Source Servers).
- On its replicating instances, ensure that there are no GT.M or MUIPIP processes attached to the Journal Pool as updates are disabled (they are enabled only on the originating instance).
- Execute `mupip rundown -region` to ensure that the database, Journal Pool, and Receiver Pool shared memory is rundown properly.

## Failures

### Network Failures

Perform a cutover from the originating instance to a replicating instance if the network from clients to the originating instance fails, and the network from the clients to the replicating instance is still functioning.

If the network from clients to both the originating and all available replicating instances fails, the application is no longer available.

If the network between the originating instance and replicating instance fails, no action is required to manage GT.M replication. The originating instance continues to make the application available. Replicating instances will catch up with the originating instance when the network is restored.

If the network from the clients to the replicating instance fails, no action is required to manage GT.M replication, although it would be prudent to make the network operational as soon as possible.

### Single-Site Failure

#### *Replicating Instance Fails*

When the replicating instance comes back up after failure, it will still be the replicating instance. Refer to the preceding "Secondary Starts After a Shut Down or Crash" description for further details.

#### *Originating instance Fails*

If the originating instance fails, the replicating instance should take over; when the originating instance comes back up, it should come up as the new replicating instance.

- The external control mechanism should detect that the originating instance has failed, and take action to switch the replicating instance to originating mode, and either route transactions to the new originating instance (former replicating instance) or notify clients to route transactions to the new originating instance.
- If the former originating instance did not respond to certain transactions, one cannot be certain whether they were processed and whether or not the database updates were committed to the former replicating instance. These transactions must now be processed on the new replicating instance. When the former originating instance comes up as the new replicating instance, GT.M rolls off the processed transactions and database updates for reconciliation on the new originating instance.

On new originating instance (former replicating instance)

- Stop the Replication Server.
- Create new journal files.
- Switch the Source Server from passive to active mode to start replicating to the new replicating instance (former originating instance) when it comes back up.

## Database Replication

- Start the application servers, or if they were passive, they should be activated. The new originating instance is now ready to receive transactions from clients.
- If the state of the database indicates that batch operations were in process, restart batch operations.

When the new replicating instance (the former originating instance) comes back up, query the originating instance for the journal sequence number at which it became the originating instance (refer to “Displaying/Changing the attributes of Replication Instance File and Journal Pool” (page 228) for more information), and roll back the replicating instance to this point (refer to Section : “Rolling Back the Database After System Failures” (page 249) for more information). Reconcile the transaction from the lost transaction file or possibly the broken transaction file resulting from the rollback operation.

- Create new journal files.
- Start the Source Server in passive mode.
- Start the Receiver Server to resume replication as the new replicating instance. Dual-site operation is now restored.
- As appropriate, start the passive application servers.

## Dual-Site Failures

Various dual-site failure scenarios are possible. Each case is a full system outage - that is, the application is no longer available (which is why multi-site may be attractive). This section identifies the recovery mechanism for each scenario. In each scenario, Site A is initially the originating instance and Site B is initially the replicating instance, before any failure.

### *Replicating Instance (Site B) Fails First*

In the following scenarios, the replicating instance fails first. Once the replicating instance fails, the originating instance continues to operate. The originating instance’s replication Source Server is unable to send updates to the replicating instance. Thus, there is a queue of non-replicated transactions at the failure point on the originating instance. Then, the originating instance fails before the replicating instance recovers, which leads to a dual-site outage. Operating procedures differ according to which site recovers first.

### *Site A recovers first*

On Site A:

- Rollback the database to the last committed transaction (last application-consistent state).
- Create new journal files.
- Start the Source Server.
- Start the application servers. Application availability is now restored.
- If the state of the instance indicates that batch operations were in process, restart batch operations.

### *On Site B, when it recovers:*

- Rollback the database to the last committed transaction.
- Create new journal files.

## Database Replication

- Start the Source Server in passive mode.
- Start the Receiver Server. Dual-site operation is now restored.
- Start the passive application servers, as appropriate.

### *Site B recovers first*

#### On Site B:

- Rollback the database to the last committed transaction (last application-consistent state).
- Create new journal files.
- Start the Source Server.
- Start the application servers. Application availability is now restored.
- If the state of the database indicates that batch operations were in process, restart batch operations.

#### On Site A, when it recovers:

- Query the originating instance (refer to “Displaying/Changing the attributes of Replication Instance File and Journal Pool” (page 228)) for the journal sequence number at which it became the originating instance, and roll back the replicating instance to this point. Then apply the lost transaction files to the originating instance for reconciliation/reapplication.
- Create new journal files.
- Start the Source Server in passive mode.
- Start the Receiver Server to resume replication as the new secondary. Dual-site operation is now restored.
- Start the passive application servers, as appropriate.

### **Originating Instance (Site A) Fails First**

In the following scenarios, the originating instance fails first, causing a cutover to Site B. Site B operates as the originating instance and then fails.

### *Site B Recovers First*

#### On Site B:

- Roll back the database to the last committed transaction (last application-consistent state).
- Create new journal files.
- Start the Source Server.
- Start the application servers. Application availability is now restored. If the state of the database indicates that batch operations were in process, restart batch operations.

## Database Replication

On Site A, when it recovers:

- Query the originating instance (refer to Section : “Displaying/Changing the attributes of Replication Instance File and Journal Pool” (page 228) for more information) for the journal sequence number at which it became primary, and the rollback the replicating instance to this point. Then apply the lost transaction files to the originating instance for reconciliation/reapplication. The broken transaction file contains information about transactions that never completed and may be useful in researching where work should be restarted. However, broken transactions, unlike the lost transactions, are inherently incomplete.
- Create new journal files.
- Start the Source Server in passive mode.
- Start the Receiver Server to resume replication as the new secondary. Dual-site operation is now restored.
- Start the passive application servers, as appropriate.

### *Site A Recovers First*

On Site A:

- Roll back the database to the last committed transaction (last application-consistent state).
- Create new journal files.
- Start the Source Server.
- Start the application servers. Application availability is now restored.
- If the state of the database indicates that batch operations were in process, restart batch operations.

On Site B, when it recovers:

- Roll back all transactions that were processed when it was the primary. Transmit the transactions backed out of the database by the rollback to the originating instance for reconciliation/reapplication.
- Create new journal files.
- Start the Source Server in passive mode.
- Start the Receiver Server to resume replication as the new secondary. Dual-site operation is now restored.
- Start the passive application servers, as appropriate.

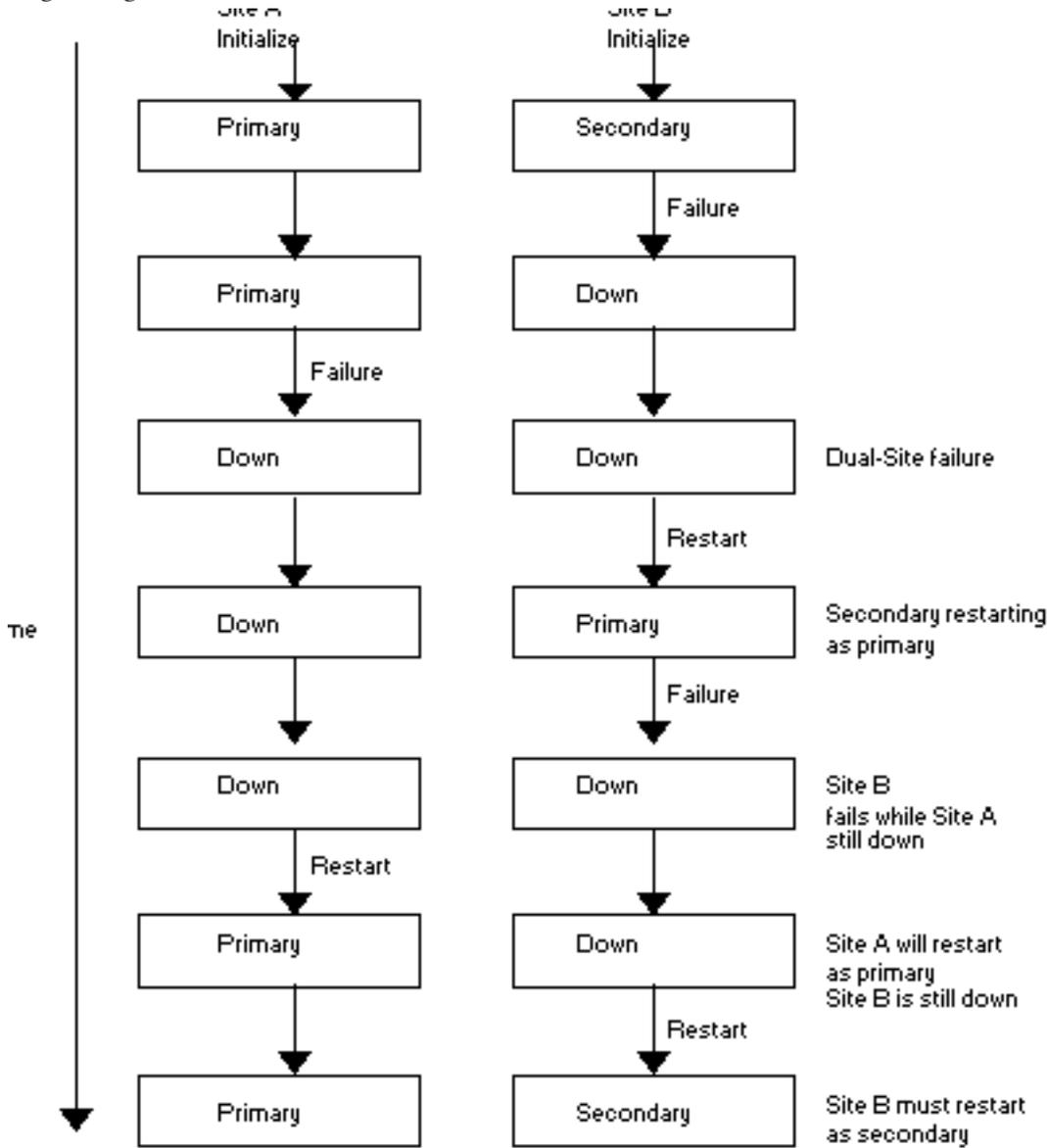
## Complex Dual-Site Failure

A complex dual-site failure may result in a large number of non-replicated transactions, depending on how the instances restart. This situation occurs if both instances are down, the replicating instance comes up for a period, assumes originating instance status, and then fails again while the first originating instance remains down.

If the first originating instance comes up first after the second failure, represented as Site A in the following figure, it will be the originating instance. When Site B comes up, it will act as the replicating instance and rollback all transactions performed while

## Database Replication

it was the originating instance and Site A was down. These transactions become non-replicated transactions. If Site B came up first, then the non-replicated transactions would occur when Site A restarted. These would be the transactions while A was the originating instance after B failed.



When recovering from these situations, the originating instance is always the current system of record when the replicating instance comes up. The replicating instance must roll back to the transaction with the highest journal sequence number common in both the instance and "catch up" from there. On the originating instance, reconcile all the lost and possibly broken transactions files that resulted from the rollback operation.

## Rolling Software Upgrades

A rolling software upgrade provides continuous service while applying a new software release to the system. In other words, a rolling software upgrade removes the need for application downtime as while each system is updated with the new software release independently, the other system acts as the originating instance during that period.

## Dual-Site Rolling Upgrade

This rolling upgrade sequence assumes that there is only one replicating instance, which is down during the upgrade. Thus, if the originating instance goes down during the upgrade, the application becomes unavailable.



### Note

A properly set LMS configuration avoids this exposure.

Assuming that Site A is the originating instance before the upgrade, here are the steps to perform a rolling upgrade:

- Site A continues to operate normally, (that is, to the originating instance, a replicating instance upgrade looks like a replicating instance failure).

On Site B:

- Shut down the Source and Receiver Servers and the application. Perform a MUPIP RUNDOWN operation and make a backup copy.
- Upgrade the software.
- If there is no change to the database layout or schema, bring the replicating instance up and proceed with the planned cutover (as follows).
- Note the largest journal sequence number in any replicated database region.
- Upgrade the database.
- Set the journal sequence number of any replicated region to the largest journal sequence number noted above.
- If there was no change to the database schema, bring the replicating instance back up and proceed with the planned cutover (as follows).
- If there was a change to the database schema, bring up the replicating instance with the new-to-old filter on the Source Server, and the old-to-new filter on the Receiver Server.
- At this point, dual-site operation is restored with Site B running the new software and Site A running the old software. It may be appropriate to operate in this mode for some time to verify correct operation.
- After which you can execute the following to complete the upgrade:
  - Perform a controlled cutover between systems.
  - Make Site A the replicating instance and Site B the originating instance for the remainder of the upgrade.

On Site A:

- Once you are satisfied with the operation of the new software on Site B, shut down the Source and Receiver Servers and the application. Run down the database and take a backup copy.
- Upgrade the software.

## Database Replication

- Note the largest journal sequence number in any replicated database region.
- Upgrade the database.
- Set the journal sequence number of any replicated region to the largest journal sequence number noted above.
- If there was no change to the database schema, bring the replicating instance back up. Normal operation is now restored, and the upgrade is complete.
- If there are filters, Bring up Site A as the secondary. Use the `-stopsourcefilter` qualifier on the Receiver Server on Site A to turn off the filter on Site B. This restores normal operation.

## Rolling SI Replication Upgrade

Although V5.5-000 supports only one source stream for SI replication, the architecture allows for fifteen externally sourced streams (numbers 1 through 15), with stream 0 being locally generated updates. Under normal conditions, on a supplementary instance, one will see updates from streams 0 and 1. If the recipient of an SI stream has been moved from receiving replication from one source to another, you may see other stream numbers (corresponding to updates from other streams) as well.

When adding SI replication, the rules to remember are that (a) both source and receiver sides of SI replication must be V5.5-000, (b) upgrading an instance to V5.5-000 requires a new replication instance file because the replication instance file format for SI is not compatible with those of prior releases and (c) the `-updateresync` qualifier requires the name of a prior replication instance file when both source and receiver are V5.5-000.

Remember that except where an instance is an unreplicated sole instance, you should upgrade replicating secondary instances rather than originating primary instances. Starting with BC replication (e.g., **Ardmore** as originating primary and *BrynMawr* as replicating secondary), the simplest steps to start SI replication to Malvern are:

- Bring *BrynMawr* down and upgrade it to V5.5-000. *BrynMawr* requires a new replication instance file. Please refer to the relevant release notes for details of upgrading database files and global directories; unless otherwise instructed by FIS, always assume that object and journal files are specific to each GT.M release.
- Resume BC replication from **Ardmore** to *BrynMawr*. Since **Ardmore** is at an older GT.M release than V5.5-000, when starting the Receiver Server for the first time at *BrynMawr*, the `-updateresync` qualifier does not require the name of a prior replication instance file.
- Create supplementary instance Malvern from a backup of *BrynMawr* or **Ardmore**, if that is more convenient. Malvern will require a new replication instance file, created with the `-supplementary` qualifier.
- Start SI replication from *BrynMawr* to Malvern. Since Malvern and *BrynMawr* are both V5.5-000, the `-updateresync` qualifier used when the Malvern Receiver Server starts for the first time requires the old replication instance file copied, perhaps as part of a BACKUP, up from *BrynMawr* as its value.

At your convenience, once *BrynMawr* is upgraded you can:

- Switchover so that *BrynMawr* is the originating primary instance with BC replication to **Ardmore** and SI replication to Malvern. This is unchanged from current LMS procedures. SI replication from *BrynMawr* to Malvern can operate through the switchover.
- Bring **Ardmore** down and upgrade it to V5.5-000. It requires a new replication instance file.
- Start BC replication from *BrynMawr* to **Ardmore**. Since **Ardmore** and *BrynMawr* are both V5.5-000, the `-updateresync` qualifier for **Ardmore**'s first Receiver Server start requires the name of a prior replication instance file. As it cannot use

**Ardmore's** pre-V5.5-000 format replication instance file, in this special case, use a backup copy from *BrynMawr* as that prior file.

### Creating a new instance file without an `-updateresync` qualifier

On the source side:

- Use the MUPIP BACKUP command with the `-REPLINSTANCE` qualifier to backup the instance to be copied.
- Ship the backed up databases and instance file to the receiving side.

On the receiving side:

- Run the MUPIP REPLIC `-EDITINST` command on the backed up instance file to change the instance name to reflect the target instance. This makes the source replication instance file usable on the target instance while preserving the history records in the instance file.
- Create new journal files, start a passive Source Server and a Receiver Server (without an `-updateresync` qualifier).
- Allow a Source Server to connect.

### Upgrade Replication Instance File

To upgrade the replication instance file, perform the following steps:

- Shut down all mumps, MUPIP and DSE processes except Source and Receiver Server processes; then shut down the Receiver Server (and with it, the Update Process) and all Source Server processes. Use MUPIP RUNDOWN to confirm that all database files of the instance are closed and there are no processes accessing them.
- Rename the existing replication instance file after making a backup copy.
- Create a new replication instance file (you need to provide the instance name and instance file name, either with command line options or in environment variables, as documented in the Administration and Operations Guide):
  - If this instance is to receive SI replication (Malvern in the examples above) or to receive BC replication from an instance that receives SI replication (Newtown in the examples above), use the command:

```
mupip replicate -instance_create -supplementary
```

- Otherwise use the command:

```
mupip replicate -instance_create
```

- Prepare it to accept a replication stream:
  - Start a passive Source Server using the `-updok` flag.
  - Start the Receiver Server using the `updateresync` flag, e.g.: `mupip replicate -receiver -start -updateresync=filename flag` where `filename` is the prior replication file if the source is V5.5-000 and no filename if it is an older GT.M release (with other required command line flags, as documented in the Administration and Operations Guide).
- Start a Source Server on a root or propagating primary instance to replicate to this instance. Verify that updates on the source instance are successfully replicated to the receiver instance.

The `-updateresync` qualifier indicates that instead of negotiating a mutually agreed common starting point for synchronization the operator is guaranteeing the receiving instance has a valid state that matches the source instance currently or as some point in the past. Generally this means the receiving instance has just been updated with a backup copy from the source instance.



### Note

A GT.M V5.5-000 instance can source a BC replication stream to or receive a BC replication stream from older GT.M releases, subject to limitations as discussed in the Limitations section of this document. It is only for SI replication that both source and recipient must both be V5.5-000.

## Creating a Supplementary Instance

A supplementary instance cannot be the first or sole instance that you upgrade to V5.5-000 - you must already have created an instance running V5.5-000 to provide a replication stream to the supplementary instance.

You can create a supplementary instance from (a) a backup copy of another instance, a supplementary instance, an originating primary or replicating secondary by giving it a new identity, or (b) a freshly created, new instance. An instance used to create a supplementary instance must already be upgraded to V5.5-000.

Starting with a backup of another instance, follow the procedures above under Upgrade Replication Instance File using the `-supplementary` flag to the `mupip replicate -instance_create` command.

Creating a supplementary instance from a backup of an existing instance creates an SI replication instance with all the database state in the existing instance and is perhaps the most common application need. But there may be situations when a supplementary instance only needs new data, for example if it is to provide a reporting service only for new customers, and historical data for old customers is just excess baggage. In this case, you can also create a supplementary instance by creating a new instance, pre-loading it with any data required, enabling replication for the database files (since an Update Process will only apply updates to a replicated - and journaled - database region), and following the procedures above under above under Upgrade Replication Instance File using the `-supplementary=on` flag to the `mupip replicate -instance_create` command. Ship a replication instance file from the source to provide the `-updateresync=filename` qualifier required when starting the Receiver Server for the first time.

## Starting / Resuming SI Replication

For starting SI replication on an originating primary supplementary instance P in a B<-A->P-<Q configuration, the procedure is similar to a non-supplementary instance except that one also needs to start a receiver server (after having started a source server to set up the journal pool) to receive updates from the corresponding non-supplementary instance (A or B in this case). Similarly, as part of shutting down the originating primary supplementary instance, an added step is to shut down the Receiver Server (if it is up and running) before shutting down the Source Server.

Remember that for GT.M replication, the Receiver Server listens at a TCP port and the Source Server connects to it. If the Receiver Server is not ahead of the Source Server, replication simply starts and streams updates from the source to the receiver. When the Receiver Server is ahead of the Source Server, the cases are different for BC and SI replication.

For either BC or SI replication, if the Receiver Server is started with the `-autorollback` qualifier, it performs an online rollback of the receiving instance, so that it is not ahead of the originating instance, creating an Unreplicated Transaction Log of any transactions that are rolled off the database. When started without the `-autorollback` qualifier, a Receiver Server notified by its Source Server of a rollback, logs the condition and exits so an operator can initiate appropriate steps.

For SI replication the `-noresync` qualifier tells the Receiver Server not to rollback the database even if the receiver is ahead of the source. In this case, the Source Server starts replication from the last journal sequence number common to the two instances.

## Changing the Replication Source

When changing the source of a supplementary replication stream to another in the same family of instances (for example in the example where **Ardmore** and Malvern crash and *Newtown* is to receive a replication stream from *BrynMawr*, start the Receiver Server normally (with either `-autorollback` or `-noresync`, as appropriate) and allow *BrynMawr* to connect to it. Instead of using `-autorollback`, you can also perform a `mupip journal -rollback -backward -fetchresync` before starting the Receiver Server.

To migrate a supplementary instance from receiving SI replication from one set of BC instances to a completely different set - for example, if Malvern is to switch from receiving SI replication from the set of {**Ardmore**, *BrynMawr*} to a completely unrelated set of instances {Pottstown, Sanatoga}

## Switchover between instances having triggers

GT.M stores trigger definitions in the regions to which they map and the maintenance-related information in the default region. To invoke a trigger on a global variable, GT.M uses the in-region information but for `$ZTRIGGER()` and `MUPIP TRIGGER` it searches the lookup information in the default region to find the region that holds its trigger definition.

Take this aspect into account while designing trigger maintenance actions and switchover processes for an LMS configuration having a mix of replicated and unreplicated regions.

You should exercise caution when you set up an LMS configuration where the default region is replicated but some regions that store trigger definitions are unreplicated. If the triggers in the unreplicated regions are needed for proper operation in the event of a switchover, you must define and maintain them separately on the originating instance as well as on each replicating instance that may become an originating instance. To define or maintain triggers on a replicating instance, follow these steps:

1. Shutdown the Receiver Server of the replicating instance.
2. Apply the trigger definitions using `MUPIP TRIGGER` or `$ZTRIGGER()`.
3. Set the sequence number of the default region to match the sequence number of the default region on the originating instance.
4. Restart the Receiver Server of the replicating instance.

## Schema Change Filters

Filters between the originating and replicating systems perform rolling upgrades that involve database schema changes. The filters manipulate the data under the different schemas when the software revision levels on the systems differ.

GT.M provides the ability to invoke a filter; however, an application developer must write the filters specifically as part of each application release upgrade when schema changes are involved.

Filters should reside on the upgraded system and use logical database updates to update the schema before applying those updates to the database. The filters must invoke the replication Source Server (new schema to old) or the database replication Receiver Server (old schema to new), depending on the system's status as either originatig or replicating. For more information on Filters, refer to "Filters" (page 188).

## Recovering from the replication WAS\_ON state

If you notice the replication `WAS_ON` state, correct the cause that made GT.M turn journaling off and then execute `MUPIP SET -REPLICATION=ON`.

## Database Replication

To make storage space available, first consider moving unwanted non-jounaled and temporary data. Then consider moving the journal files that predate the last backup. Moving the currently linked journal files is a very last resort because it disrupts the back links and a rollback or recover will not be able to get back past this discontinuity unless you are able to return them to their original location.

If the replication WAS\_ON state occurs on the originating side:

If the Source Server does not reference any missing journal files, -REPLICATION=ON resumes replication with no downtime.

If the Source Server requires any missing journal file, it produces a REPLBRKNTRANS or NOPREVLINK error and shuts down. Note that you cannot rollback after journaling turned off because there is insufficient information to do such a rollback.

In this case, proceed as follows:

1. Take a backup (with MUPIP BACKUP -BKUPDBJNL=OFF -REPLINST=<bckup\_inst>) of the originating instance.
2. Because journaling was turned off in the replication WAS\_ON state, the originating instance cannot be rolled back to a state prior to the start of the backup. Therefore, cut the previous generation link of the journal files on the originating instance and turn replication back on. Both these operations can be accomplished with MUPIP SET -REPLICATION="ON".
3. Restore the replicating instance from the backup of the originating instance. Change the instance name of <bckup\_inst> to the name of the replicating instance (with MUPIP REPLIC -EDITINST -NAME).
4. Turn on replication and journaling in the restored replicating instance. Specify the journal file pathname explicitly with MUPIP SET -JOURNAL=filename=<repinst\_jnl\_location> (as the backup database has the originating instance's journal file pathname).
5. Restart the Source Server process on the originating instance.
6. Start the Receiver Server (with no -UPDATERESYNC) the replicating instance.

If the replication WAS\_ON state occurs on the receiving side:

Execute MUPIP SET -REPLICATION=ON to return to the replication ON state. This resumes normal replication on the receiver side. As an additional safety check, extract the journal records of updates that occurred during the replication WAS\_ON state on the originating instance and randomly check whether those updates are present in the receiving instance.

If replication does not resume properly (due to errors in the Receiver Server or Update Process), proceed as follows:

1. Take a backup (with MUPIP BACKUP -BKUPDBJNL=OFF -REPLINST=<bckup\_inst>) of the originating instance.
2. Restore the replicating instance from the backup of the originating instance. Change the instance name of <bckup\_inst> to the name of the replicating instance (with MUPIP REPLIC -EDITINST -NAME).
3. Turn on replication and journaling in the restored replicating instance. Specify the journal file pathname explicitly with MUPIP SET -JOURNAL=filename=<repinst\_jnl\_location> (as the backup database has the originating instance's journal file pathname).
4. Restart the Source Server process on the originating instance. Normally, the Source Server might still be running on the originating side.
5. Start the Receiver Server (with no -UPDATERESYNC) the replicating instance.

## Setting up a new replicating instance of an originating instance (A->B, P->Q, or A->P)

To set up a new replicating instance of an originating instance for the first time or to replace a replicating instance if database and instance file get deleted, you need to create the replicating instance from a backup of the originating instance, or one of its replicating instances.

If you are running GT.M V5.5-000 or higher:

- Take a backup of the database and the replication instance file of the originating instance together at the same time with `BACKUP -REPLINSTANCE` and transfer them to the location of the replicating instance. If the originator's replicating instance file was newly created, take its backup while the Source Server is running to ensure that the backup contains at least one history record.
- Use `MUPIP REPLICATE -EDITINST -NAME=<secondary-instname>` to change the replicating instance's name.
- Start the replicating instance without `-udpateresync`.

If you are running GT.M pre-V5.5-000:

- Create a new replication instance file on the replicating instance.
- Start the replicating instance with this new replication instance file with the `-updateresync` qualifier (no value specified).

## Replacing the replication instance file of a replicating instance (A->B and P->Q)

In this case, it is possible that the replicating instance's database files are older than the originating instance. Note that to resume replication there is no need to transfer the backup of the originating instance's database and replication instance files.

To replace the existing replicating instance file with a new replication instance file, follow these steps:

If you are running GT.M V5.5-000 or higher:

- Take a backup of just the replication instance file (no database files with `BACKUP -REPLINST=</path/to/bkup-orig-repl-inst-file>`) and transfer it to the site of the replicating instance.
- Start the replicating instance with `-updateresync=</path/to/bkup-orig-repl-inst-file>`.

In this case, the Receiver Server determines the current instance's journal sequence number by taking a maximum of the Region Sequence Numbers in the database file headers on the replicating instance and uses the input instance file to locate the history record corresponding to this journal sequence number, and exchanges this history information with the Source Server.

If you are running GT.M pre-V5.5-000:

- Create a new replication instance file on the replicating instance.
- Start the replicating instance with this new instance file with the `-updateresync` qualifier (no value specified).

## Replacing the replication instance file of a replicating instance (A->P)

On P:

- Use the `-SUPPLEMENTARY` qualifier with the `MUPIP REPLICATE -INSTANCE_CREATE` command to indicate this is a supplementary instance file.

## Database Replication

- Start a Source Server on P with -UPDOK to indicate local updates are enabled on P.
- Start the Receiver Server on P with the -UPDATERESYNC=</path/to/bkup-orig-repl-inst-file> qualifier and -RESUME. -RESUME indicates that A and P had been replicating before. The Receiver Server looks at the local (stream #0) sequence numbers in the database file headers on P and takes the maximum value to determine the journal sequence number of the new stream on P. It then uses this as the instance journal sequence number on A to resume replication.

### Setting up a new replicating instance from a backup of the originating instance (A->P)

On A:

- Take a backup of the replication instance file and the database together at the same time with BACKUP -REPLINSTANCE and transfer it to P. If the A's replication instance file was also freshly created, then take the backup while the Source Server is running on the originating instance. This ensures that the backed up replication instance file contains at least one history record.

On P:

- Create a new replication instance file. Use the -SUPPLEMENTARY qualifier with the MUPIP REPLICATE -INSTANCE\_CREATE command to indicate this is a supplementary instance file.
- Restore the database backups from A to P or use MUPIP LOAD to load the data.
- Start a Source Server on P with -UPDOK to indicate local updates are enabled on P.
- Start the Receiver Server on P with the -UPDATERESYNC=</path/to/bkup-orig-repl-inst-file> qualifier and -INITIALIZE. The -INITIALIZE indicates this is the first time A and P are replicating.

In this case, the Receiver Server uses the current journal sequence number in the </path/to/bkup-orig-repl-inst-file> as the point where A starts sending journal records. GT.M updates the stream sequence number of Stream # 1 in the instance file on P to reflect this value. From this point, GT.M maps the journal sequence number on A to a stream journal sequence number (for example, stream # 1) on P.

### Setting up an A->P configuration for the first time if P is an existing instance (having its own set of updates)

On P:

- Turn off passive Source Server and Receiver Server (if they are active).
- Turn off replication and run down the database.

On A:

- Take a backup of the replication instance file and the database together with BACKUP -REPLINSTANCE and transfer it to P. If A's instance file was also freshly created, take the backup while the Source Server is running on the originating instance. This ensures that the backed up replication instance file contains at least one history record.

On P:

- Do not create a new instance file. Continue using the existing instance file to preserve updates that have already occurred on P.
- Start the Source Server with -UPDOK to indicate that local updates are enabled on P.

## Database Replication

- Start the Receiver Server with the `-UPDATERESYNC=</path/to/bkup-orig-repl-inst-file>` qualifier and `-INITIALIZE`. The `-INITIALIZE` indicates this is the first time A and P are replicating.

The Receiver Server uses the current journal sequence number in the `</path/to/bkup-orig-repl-inst-file>` as the point where A starts sending journal records. GT.M updates the stream sequence number (for example, of Stream # 1) in the instance file on P to reflect this value. Going forward, the journal sequence number on A will always map to a stream journal sequence number (for example, of stream # 1) on P.

### Changing the database schema of a replicating instance (A->B and A->P)

1. Shut down the Receiver Server, passive Source Server, turn off replication, and rundown the database.
2. Open DSE and execute `DUMP -FILEHEADER -ALL` and note down the Region Seqno field of each replicated region(s). `FIND -REGION=""` displays all regions and `FIND -REGION=<region-name>` switches between regions. Identify the highest *Region Seqno* of all the Region Seqnos. If you are running an A->P configuration, note down the Stream and Reg Seqno of all the streams.
3. Change the schema by modifying the global directory and relocating any data that needs to be moved.
4. Open DSE and run `CHANGE -FILEHEADER -REG_SEQNO=<the-highest-region-seqno>` to set the Region Seqno field in the database file header to the highest *Region Seqno* identified in step 2. If you are running an A->P configuration, run the `CHANGE -FILEHEADER -STRM_NUM=1 -STRM_REG_SEQNO=<hexa>` command to set the Stream and Reg Seqno of the streams noted in step 2.
5. Turn replication ON, start the passive Source Server, and start the Receiver Server without `-UPDATERESYNC`.
6. GT.M resumes receiving replication from the point of interruption.

When replication is ON, GT.M increments the Region Seqno field in the database file header for every update that occurs on a region. When replication is OFF, database updates do not affect the Region Seqno field. Except step 4, no other step affects the Region Seqno field. When you set the Region Seqno in step 4, you instruct GT.M to resume receiving replication from point replication turned off in step 1.

### Setting up a secured TLS replication connection

**tlsrepl.tar.gz** contains scripts that create two databases and sets up a secure (TLS/SSL) replication connection between them.

Please click  to download **tlsrepl.tar.gz** and follow the below instructions. You can also download **tlsrepl.tar.gz** from [http://tinco.pair.com/bhaskar/gtm/doc/books/ao/UNIX\\_manual/tlsrepl.tar.gz](http://tinco.pair.com/bhaskar/gtm/doc/books/ao/UNIX_manual/tlsrepl.tar.gz). The scripts in **tlsrepl.tar.gz** are solely for the purpose of explaining the general steps required to encrypt replication data in motion. You must understand, and appropriately adjust, the scripts before using them in a production environment. The following steps create two instances and a basic framework required for setting up a TLS replication connection between them. Note that all certificates created in this example are for the sake of explaining their roles in a TLS replication environment. For practical applications, use certificates signed by a CA whose authority matches your use of TLS.

On instance Helen:

1. Specify the value of `$gtm_dist` and `$gtmroutines` in the `env` script and execute the following command:

```
$ source ./env Helen
```

This creates a GT.M environment for replication instance name Helen. When prompted, enter a password for `gtmtls_passwd_Helen`.

## Database Replication

2. Create a certificate directory setup in the current directory and a self-signed root certification authority.

```
$ ./cert_setup
```

This creates a self-signed root certification authority (ca.crt) in \$PWD/certs/. ca.crt needs to be available on both Source and Receiver Servers. In a production environment, ensure that you keep root-level certificates in directories with 0500 permissions and the individual files with 0400 permissions so that unauthorized users cannot access them.

The output of this script is like the following"

```
Generating RSA private key, 512 bit long modulus
.....+++++++
.....+++++++
e is 65537 (0x10001)
Enter pass phrase for /home/jdoe/certs/ca.key:
Verifying - Enter pass phrase for /home/jdoe/certs/ca.key:
Enter pass phrase for /home/jdoe/certs/ca.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
Country Name (2 letter code) [US]:US
State or Province Name (full name) [Philadelphia]:Philadelphia
City (e.g., Malvern) [Malvern]:Malvern
Organization Name (eg, company) [FIS]:FIS
Organizational Unit Name (eg, section) [GT.M]:GT.M
Common Name (e.g. server FQDN or YOUR name) [localhost]:fis-gtm.com
Email Address (e.g. helen@gt.m) []:root@gt.m
```

3. Create the global directory and the database for instance Phil.

```
$ ./db_setup
```

This also creates a leaf-level certificate for Helen. The openssl.conf used to create this example is available in the download for your reference. Ensure that your openssl.cnf file is configured according to your environment and security requirements.

The output of this script is like the following:

```
Generating RSA private key, 512 bit long modulus
.+++++++
.....+++++++
e is 65537 (0x10001)
Enter pass phrase for /home/jdoe/certs/Helen.key: [Specify the password that you entered for gmtls_passwd_Helen]
Verifying - Enter pass phrase for /home/jdoe/certs/Helen.key:
Enter pass phrase for /home/jdoe/certs/helen.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
Country Name (2 letter code) [US]:
State or Province Name (full name) [Philadelphia]:Illinois
City (e.g., Malvern) [Malvern]:Chicago
Organization Name (eg, company) [FIS]:FIS
Organizational Unit Name (eg, section) [GT.M]:GT.M
Common Name (e.g. server FQDN or YOUR name) [localhost]:fisglobal.com
Email Address (e.g. helen@gt.m) []:helen@gt.m
```

## Database Replication

```
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
Using configuration from /usr/lib/ssl/openssl.cnf
Enter pass phrase for ./certs/ca.key:
Check that the request matches the signature
Signature ok
Certificate Details:
  Serial Number: 14 (0xe)
  Validity
    Not Before: Jun 11 14:06:53 2014 GMT
    Not After : Jun 12 14:06:53 2014 GMT
  Subject:
    countryName = US
    stateOrProvinceName = Illinois
    organizationName = FIS
    organizationalUnitName = GT.M
    commonName = fisglobal.com
    emailAddress = helen@gt.m
  X509v3 extensions:
  X509v3 Basic Constraints:
    CA:FALSE
  Netscape Comment:
    OpenSSL Generated Certificate
  X509v3 Subject Key Identifier:
    96:FD:43:0D:0A:C1:AA:6A:BB:F3:F4:02:D6:1F:0A:49:48:F4:68:52
  X509v3 Authority Key Identifier:
    keyid:DA:78:3F:28:8F:BC:51:78:0C:5F:27:30:6C:C5:FE:B3:65:65:85:C9

Certificate is to be certified until Jun 12 14:06:53 2014 GMT (1 days)
Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

4. Create the `$gtmconfig` file. Specify the names of all participating instances.

```
./gen_gc Helen Phil
```

5. Turn replication on and create the replication instance file:

```
$ ./repl_setup
```

6. Start the originating instance Helen:

```
$ ./originating_start Helen Phil
```

On instance Phil:

1. Specify the value of `$gtm_dist` and `$gtmroutines` in the `env` script and execute the following command:

```
$ source ./env Phil
```

This creates a GT.M environment for replication instance name Phil. When prompted, enter a password for `gtmtls_passwd_Phil`.

2. Create the global directory and the database for instance Phil.

## Database Replication

```
$ ./db_setup
```

This also creates a leaf-level certificate for Phil. The openssl.conf used to create this example is available in the download for your reference. Ensure that your openssl.cnf file is configured according to your environment and security requirements.

3. Create the \$gtmconfig file. Specify the names of all participating instances.

```
./gen_gc Helen Phil
```

Ensure that ca.key generated on Instance Helen is available on instance Phil.

4. Create the \$gtmconfig file. Specify the names of all participating instances.

```
$ ./gen_gc Helen Phil
```

5. Turn replication on and create the replication instance file:

```
$ ./repl_setup
```

6. Start the replicating instance Phil.

```
$ ./replicating_start
```

For subsequent environment setup, use the following commands:

```
source ./env Phil or source ./env Helen  
./replicating_start or ./originating_start Helen Phil
```

---

## Commands and Qualifiers

The following MUPIP commands and qualifiers control database replication in a GT.M environment.

### Turning Replication On/Off

Command Syntax:

```
mupip set {-file db-file|-region reg-list} -replication={ON|OFF}
```

Qualifiers:

*-file and -region*

Use these qualifiers in the same manner that you would use them for a MUPIP SET. For more information refer to Chapter 5: “General Database Management” [67].

*-replication=replication-state*

Switches the GT.M replication subsystem ON/OFF and possibly modify the current journaling [no-]before image field (which is stored in the database file header).

*replication-state* is either of the following keywords:

*OFF*

Disable replication of the database file(s) or region(s). Even if you turn off replication, journaling continues to operate as before.



## Important

GT.M creates a new set of journal files and cuts the back link to the previous journal files if the replication-state is OFF and then turned ON again. The database cannot rollback to a state prior to ON. Therefore, ensure that replication-state remains ON throughout the span of database replication. Turn replication-state OFF only if database replication is no longer needed or the instance is about to be refreshed from the backup of the originating instance.

*ON*

Enables replication for the selected database file(s) or region(s). When the JOURNAL qualifier is not specified, this action turns BEFORE\_IMAGE journaling on. Specify -JOURNAL=NOBEFORE\_IMAGE to enable replication with no-before-image journaling. In both cases, GT.M creates a new journal file for each database file or region, and switches the current journal file. FIS recommends you to specify the desired journaling characteristics (MUPIP SET -JOURNAL=BEFORE\_IMAGE or MUPIP SET -JOURNAL=NOBEFORE\_IMAGE).

When replication is ON, a MUPIP SET REPLICATION=ON command with no JOURNAL qualifier assumes the current journaling characteristics (which are stored in the database file header). By default GT.M sets journal operation to BEFORE\_IMAGE if this command changes the replication state from OFF to ON and JOURNAL=NOBEFORE\_IMAGE is not specified. Therefore, conservative scripting should always specify the desired journaling characteristics using the JOURNAL qualifier of the MUPIP SET command.

The replication state ON in the file header denotes normal replication operation.

*[WAS\_ON] OFF*

Denotes an implicit replication state when GT.M attempts to keep replication working even if run-time conditions such as no available disk space or no authorization for a process attempting to auto-switch a journal file cause GT.M to turn journaling off. Even with journaling turned off, the Source Server attempts to continue replication using the records available in the replication journal pool. In this state, replication can only continue as long as all the information it needs is in the replication journal pool. Events such as an operationally significant change on the replicating instance(s) or communication problems are likely to cause the Source Server to need information older than that in the replication journal pool and because it cannot look for that information in journal files, at that point the Source Server shuts down.



## Note

If the replication ON state is like a bicycle running smoothly on the road, replication WAS\_ON is like a bicycle with a flat front tire being ridden like a unicycle - the system is operating outside its intended mode of use and is more subject to misfortune.

WAS\_ON is an implicit replication state. At all times during the WAS\_ON state, you can see the current backlog of transactions and the content of the Journal Pool (MUPIP REPLICATE -SOURCE -SHOWBACKLOG and MUPIP REPLICATE -SOURCE -JNLPOOL -SHOW). This information is not available in the replication OFF state.

For more information on recovering originating and replicating instances from WAS\_ON, refer to Recovering from the replication WAS\_ON state (page 219).

Example:

```
$ mupip set -replication=on -file mumps.dat
```

This example enables database replication and turns before-image journaling on for mumps.dat.

## Database Replication

```
$ mupip set -replication=on -journal=nobefore_image -file mumps.dat
```

This example enables database replication and turns no-before-image journaling on for mumps.dat.

```
$ mupip set -replication=off -file mumps.dat
```

This example turns off database replication for mumps.dat.

## Creating the Replication Instance File

Command Syntax:

```
mupip replicate -instance_create -name=<instance name> [-noreplace] [-supplementary]
```

Qualifiers:

*-instance\_create*

Creates a replication instance file. `mupip replicate -instance_create` takes the file name of the replication instance file from the environment variable `gtm_repl_instance`.

If an instance file already exists, GT.M renames it with a timestamp suffix, and creates a new replication instance file. This behavior is similar to the manner in which GT.M renames existing journal files while creating new journal files. Creating an instance file requires standalone access.

*-name*

Specifies the instance name that uniquely identifies the instance and is immutable. The instance name can be from 1 to 16 characters. GT.M takes the instance name (not the same as instance file name) from the environment variable `gtm_repl_instname`. If `gtm_repl_instname` is not set and `-name` is not specified, GT.M produces an error.

*-noreplace*

Prevents the renaming of an existing replication instance file.

*-supplementary*

Specifies that the replication instance file is suitable for use in a supplementary instance.

Example:

```
$ export gtm_repl_instance=mutisite.repl
$ export gtm_repl_instname=America
$ mupip replicate -instance_create
```

This example creates a replication instance file called `mutisite.repl` specified by `gtm_repl_instance` with an instance name `America` specified by environment variable `gtm_repl_instname`.

## Displaying/Changing the attributes of Replication Instance File and Journal Pool

Command Syntax:

```
mupip replicate
```

```
-edit[instance] {<instance-file>|-source -jnlpool}
{-show [-detail]|-change [-offset=] [-size=] [-value=]}
[-name=<new-name>]
```

*-editinstance*

Displays or changes the attributes of the specified *instance-file*. Use *-editinstance* in combination with SHOW or CHANGE qualifiers.

*-jnlpool*

Displays or changes the attributes of Journal Pool. Always specify *-source* with *-jnlpool*. Use *-jnlpool* in combination with SHOW or CHANGE qualifiers.

*-change*

The CHANGE qualifier is intended only for use under the guidance of FIS and serves two purposes. When used with *-editinstance -offset -size*, it changes the contents of the replication instance file. When used with *-jnlpool*, it changes the contents of journal pool header. Although MUPIP does not enforce standalone access when using this feature on the instance file or the journal pool, doing so when replication is actively occurring can lead to catastrophic failures.

*-name=<new-name>*

Changes the instance name in the replication instance file header to the new-name. Note that changing an instance name preserves the instance history.

*-show*

Displays File Header, Source Server slots, and History Records from the Replication Instance file.

*-detail*

When specified, all fields within each section are displayed along with their offset from the beginning of the file and the size of each field. Use this qualifier to find the *-offset* and *-size* of the displayed field. To edit any displayed field, use the *-change* qualifier.

*-size*

Indicates the new size of the new value in bytes. The value of size can be either 1, 2, 4, or 8.

*-offset*

Takes a hexadecimal value that is a multiple of *-size*. With no *-offset* specified, GT.M produces an error. GT.M also produces an error if the offset is greater than the size of the instance file or the journal pool header.

*-value*

Specifies the new hexadecimal value of the field having the specified *-offset* and *-size*. With no value specified, GT.M displays the current value at the specified offset and does not perform any change. Specifying *-value=<new\_value>* makes the change and displays both the old and new values.

**Caution**

Change the instance file or the journal pool only on explicit instructions from FIS.

*-show*

The SHOW qualifier serves two purposes. When used with `-editinstance`, it displays the content of the replication instance file. When used with `-jnlpool`, it displays the content of the journal pool.

Example:

```
$ mupip replicate -editinstance -show -detail multisite.repl
```

This example displays the content of the replication instance file `multisite.repl`. The optional detail qualifier displays each section along with its offset from the beginning of the file and the size of each field. Use this information when there is a need to edit the instance file.

Example:

```
$ mupip replicate -editinstance -change -offset=0x00000410 -size=0x0008 -value=0x010 multisite.repl
```

This command sets the value of the field having the specified offset and size to 16. Note that `mupip replicate -editinstance -show -detail` command displays the offset and size of all fields in an instance file.

## Starting the Source Server

Command syntax:

```
mupip replicate -source -start
{-secondary=<hostname:port>|-passive}
[-buffsize=<Journal Pool size in bytes>]
[-filter=<filter command>]
[-freeze[=on|off] -[no]comment[="<string>"]]
[-connectparams=<hard tries>,<hard tries period>,
<soft tries period>, <alert time>, <heartbeat period>,
<max heartbeat wait>]
-instsecondary=<replicating instance name>
-log=<log file name> [-log_interval=<integer>]
{-rootprimary|-propagateprimary} [{-updok|-updnotok}]
[-cmlvl=<compression level>]
[-tlsid=<label>]
[-[no]plaintextfallback]
[-renegotiate_interval=<minutes>]
```

Qualifiers:

*-replicate*

Use this qualifier to access the replication subsystem.

*-source*

Identifies the Source Server.

*-start*

Starts the Source Server.

*-secondary=<hostname:port>*

## Database Replication

Identifies the replicating instance. <hostname:port> specifies an IPv4 or IPv6 address optionally encapsulated by square-brackets ([]) like "127.0.0.1", "::1", "[127.0.0.1]", or "[::1]" or a hostname that resolves to an IPv4 or IPv6 address and the port at which the Receiver Server is waiting for a connection.

If, in your environment, the same hostname is used for both IPv4 and IPv6, GT.M defaults to IPv6. If you wish to use IPv4, perhaps because you have a Receiver Server running a pre-V6.0-003 version of GT.M that does not support IPv6, set the environment variable `gtm_ipv4_only` to "TRUE", "YES", or a non-zero integer in order to force GT.M to use IPv4.

*-passive*

Starts the Source Server in passive mode.

*-log=<log file name>*

Specifies the location of the log file. The Source Server logs its unsuccessful connection attempts starting frequently and slowing to approximately once every five minutes. This interval does not affect the rate of connection attempts.

*-log\_interval=<integer>*

Specifies the number of transactions for which the Source Server should wait before writing to the log file. The default logging interval is 1000 transactions.

`-log_interval=0` sets the logging interval to the default value.

*-buffsize=<Journal Pool size in bytes>*

Specifies the size of the Journal Pool. The server rounds the size up or down to suit its needs. Any size less than 1 MB is rounded up to 1 MB. If you do not specify a qualifier, the size defaults to the GT.M default value of 64 MB. Remember that you cannot exceed the system-provided maximum shared memory. For systems with high update rates, specify a larger buffer size to avoid the overflows and file I/O that occur when the Source Server reads journal records from journal files.

*-filter=<filter command>*

Specifies the complete path of the filter program and any associated arguments. If you specify arguments, then enclose the command string in quotation marks. If a filter is active, the Source Server passes the entire output stream to the filter as input. Then, the output from the filter stream passes to the replicating instance. If the filter program is an M program with entry-ref `OLD2NEW^FILTER`, specify the following path:

```
filter='"$gtm_dist/mumps -run OLD2NEW^FILTER"'
```

Write the filter as a UNIX process that takes its input from STDIN and writes its output to STDOUT.

The format of the input and output data is the MUPIP journal file extract format. The filter must maintain a strict 1:1 relationship between transactions on the input stream and transactions on the output stream. If a transaction on the input results in no sets and kills in the output, the filter must still write an empty transaction to the output stream.

Example:

```
extfilter
; A command like mupip replic -source -start -buffsize=$gtm_buffsize
;-instsecondary=$secondary_instance -secondary=$IP_Address:$portno
;-filter='"$gtm_exe/mumps -run ^extfilter"' -log=$SRC_LOG_FILE
;deploys this filter on the Source Server.
set $ztrap="goto err"
```

## Database Replication

```
set TSTART="08"
set TCOMMIT="09"
set EOT="99"
set log=$ztrnlm("filterlog") ; use the environment variable filterlog" (if defined)
;to specify which logfile to use
if logersion="" set log="logcharout" char
if $zv["VMS" sechar EOL=$C(13)_$C(10)
else set EOL=$C(10)
open log:newve:sion
use $principal:nowrap
for do
. use $principal
. read extrRec
. if $zeof halt
. set rectype=$piece(extrRec,"\\",1)
. if rectype'=EOT do
.. if rectype'=TSTART set filtrOut=extrRec_EOL
.. else do
... set filtrOut=extrRec_EOL
... for read extrRec set filtrOut=filtrOut_extrRec_EOL quit:$zextract(extrRec,1,2)=TCOMMIT
... if $zeof halt
.. ; set $x=0 is needed so every write starts at beginning of record position
.. ; do not write more than width characters in one output operation to avoid "chopping".
.. ; and/or eol in the middle of output stream
.. ; default width=32K-1
.. ; use $zsubstr to chop at valid character boundary (single or multi byte character)
.. set cntr=0,tmp=filtrOut
.. for quit:tmp="" do
... set cntr=cntr+1,$x=0,record(cntr)=$zsubstr(tmp,1,32767),tmp=$zextract(tmp,$zlength(record(cntr))+1,
$zlength(tmp))
... write record(cntr)
. use log
. write "Received: ",EOL,$s(rectype'=TSTART:extrRec_EOL,1:filtrOut)
. if rectype'=EOT write "Sent: ",EOL,filtrOut
. else write "EOT received, halting..." halt
quit
err
set $ztrap=""
use log
write !!!,"**** ERROR ENCOUNTERED ****",!!!
zshow "x"
halt
```

This example reads logical database updates associated with a transaction from STDIN and writes them to log.out and STDOUT just like the UNIX tee command. It runs on GT.M V5.5-000 where it is no longer required to treat filter output as a transaction. To run this example on a pre-GT.M V5.5-000 version, replace the line:

```
.. if rectype'=TSTART set filtrOut=extrRec_EOL
```

with

```
.. if rectype'=TSTART set filtrOut=TSTART_EOL_extrRec_EOL_TCOMMIT_EOL
```

to wrap mini-transactions in 08 09.

```
-freeze[=on/off] -[no]comment[="<string>"]
```

## Database Replication

Promptly sets or clears an Instance Freeze on an instance irrespective of whether a region is enabled for an Instance Freeze. -freeze with no arguments displays the current state of the Instance Freeze on the instance.

-[no]comment[="*<string>*"] allows specifying a comment/reason associated with an Instance Freeze. Specify -nocomment if you do not wish to specify a comment/reason.

For more information on enabling a region to invoke an Instance Freeze on custom errors, refer to the -INST\_FREEZE\_ON\_ERROR section of "SET" (page 109).

For more information on Instance Freeze, refer to "Instance Freeze" (page 193).

*-connectparams=<hard tries>,<hard tries period>,<soft tries period>,<alert time>,<heartbeat period><max heartbeat wait>*

Specifies the connection retry parameters. If the connection between the Source and Receiver Servers is broken or the Source Server fails to connect to the Receiver Server at startup, the Source Server applies these parameters to the reconnection attempts.

First, the Source Server makes the number of reconnection attempts specified by the *<hard tries>* value every *<hard tries period>* milliseconds. Then, the Source Server attempts reconnection every *<soft tries period>* seconds and logs an alert message every *<alert time>* seconds. If the specified *<alert time>* value is less than  $\text{<hard tries> * <hard tries period> / 1000 + <soft tries period>}$ , it is set to the larger value. The Source Server sends a heartbeat message to the Receiver Server every *<heartbeat period>* seconds and expects a response back from the Receiver Server within *<max heartbeat wait>* seconds.

*-instsecondary*

Identifies the replicating instance to which the Source Server replicates data.

With no *-instsecondary* specified, the Source Server uses the environment variable *gtm\_repl\_instsecondary* for the name of the replicating instance.

With no *-instsecondary* specified and environment variable *gtm\_repl\_instsecondary* not set, *mupip replicate -source -checkhealth* looks at all the Source Servers (Active or Passive) that are alive and running and those that were abnormally shutdown (kill -9ed). Any Source Server that was kill -15ed or MUIPIP STOPped is ignored because GT.M considers those Source Server shut down in the normal way. This command reports something only if it finds at least one Source Server that is alive and running or was abnormally shutdown (kill -9ed). Otherwise it returns a zero (0) status without anything to report even when the Journal Pool exists and GT.M processes (Update Process or Receiver Server on the replicating instance) are up and running.

You can start multiple Source Servers from the same originating instance as long as each of them specifies a different name for *-instsecondary*.

Specify *-instsecondary* explicitly (by providing a value) or implicitly (through the environment variable *gtm\_repl\_instsecondary*) even for starting a Passive Source Server. Whenever it activates, a Passive Source Server connects to this replicating instance.

Example:

```
$ mupip replicate -source -start -buffsize=$gtm_buffsize -secondary=localhost:1234 -log=A2B.log -instsecondary=B
```

This command starts the Source Server for the originating instance with instance B as its replicating instance.

*-rootprimary*

Assign the current instance as the originating instance. You can specify *-rootprimary* either explicitly or implicitly to start an instance as an originating instance.

*-updok*

Instructs the Source Server to allow local updates on this instance. This is a synonym for `-rootprimary` but is named so it better conveys its purpose.

*-propagate[primary]*

Use this optional qualifier to assign the current instance as a propagating instance. Specifying `-propagateprimary` disables updates on the current instance.

Note that it is not possible to transition an originating instance to a propagating instance without bringing down the Journal Pool. However, it is possible to transition a propagating instance to an originating instance without bringing down the Journal Pool for an already running passive Source Server (start one with `-propagateprimary` if none is running).

Both `-rootprimary` and `-propagateprimary` are optional and mutually exclusive. However, FIS recommends you to specify both `-rootprimary` and `-propagateprimary` explicitly in the script for clarity.

Example:

```
$ mupip replicate -source -activate -rootprimary
```

This command transitions a propagating originating instance to an originating instance without bringing down the Journal Pool.

With neither `-rootprimary` nor `-propagateprimary` specified, GT.M uses a default value of `-propagateprimary` for the passive Source Server startup command (`mupip replic -source -start -passive`) and the deactivate qualifier (`mupip replicate -source -deactivate`). GT.M uses a default value of `-rootprimary` for the `mupip replicate -source -start -secondary=...` and the `mupip replic -source -activate` commands. These default values make the replication script simpler for users who are planning to limit themselves to one originating instance and multiple replicating instance (without any further replicating instances downstream).

```
$ export gtm_repl_instance=multisite.repl
$ mupip set -journal="enable,before,on" -replication=on -region "*"
$ mupip replicate -instance_create -name=America
$ mupip replicate -source -start -buffsize=$jnlpool_size -secondary=localhost:1234 -log=A2B.log -instsecondary=Brazil
```

This example starts the Source Server at port 1234 for the replicating instance Brazil. The Source Server creates a Journal Pool. A GT.M Process writes the updated journal records to the Journal Pool. Then, the Source Server process transports each record from the Journal Pool to Brazil via a TCP/IP connection.



### Note

Before starting replication, always remember to rundown every replicated database region then start the Source Server.



### Important

GT.M updates to replicated regions are permitted only on the originating instance and disabled on ALL other replicating instances.

The Source Server records actions and errors in `A2B.log`. It also periodically record statistics such as the current backlog, the number of journal records sent since the last log, the rate of transmission, the starting and current `JNL_SEQNO`, and the path of the filter program, if any.

*-updnok*

Instructs the Source Server to not allow local updates on this instance. This is a synonym for `-propagateprimary` but is named so it better conveys its purpose.

*-cmplvl=n*

Specifies the desired compression level for the replication stream. *n* is a positive integer value indicating the level of compression desired. Level 0 offers no compression. Level 1 offers the least compression while Level 9 (as of version 1.2.3.3 of the zlib library) offers the most compression (at the cost of the most CPU usage). Specifying `-cmplvl` without an accompanying `-start` produces an error. In the case of the source server, if *N* specifies any value outside of the range accepted by the zlib library or if `-cmplvl` is not specified, the compression level defaults to zero (0). In the case of the receiver server, as long as *N* is non-zero the decompression feature is enabled; since the source server setting determines the actual level, any legal non-zero value enables compressed operation at the receiver.

Alternatively, the environment variable `gtm_zlib_cmp_level` can specify the desired compression level (in the same value range as *N* above) and the source server can then be started without `-cmplvl`. This has the same effect as starting it with `-cmplvl` specified. An explicitly specified value on the command line overrides any value specified by the environment variable.

Whenever the source and receiver server connect with each other, if the source server was started with a valid non-zero compression level, they first determine whether the receiver server is running a version of GT.M which handles compressed records and has been started with a non-zero compression level. Only if this is true, do they agree to use compressed journal records. They also verify with a test message that compression/decompression works correctly before sending any compressed journal data across. They automatically fall back to uncompressed mode of transmission if this test fails or if, at any point, either side detects that compression or decompression has failed. That is, any runtime error in the compression/decompression logic results in uncompressed replication (thereby reducing replication throughput) but never jeopardizes the functional health of replication.

The Source and Receiver Servers log all compression related events and/or messages in their respective logs. The source server also logs the length of the compressed data (in addition to the uncompressed data length) in its logfile.



## Note

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 provided by zlib. Simple instructions for compiling zlib on a number of platforms follow. Although GT.M uses zlib, zlib is not FIS software and FIS does not support zlib. These instructions are merely provided as a convenience to you.

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

Solaris/cc compiler from Sun Studio:

```
./configure --shared
make CFLAGS="-KPIC -m64"
```

HP-UX(IA64)/HP C compiler:

```
./configure --shared
make CFLAGS="+DD64"
```

AIX/XL compiler:

```
./configure --shared
Add -q64 to the LDFLAGS line of the Makefile
make CFLAGS="-q64"
```

Linux/gcc:

```
./configure --shared
make CFLAGS="-m64"
```

z/OS:

Download the zlib 1.1.4 from the libpng project's download page on SourceForge.com. Use a transfer mechanism that does not perform automatic conversion to download a source tarball. If pax cannot read the archive, it is a sign that the download mangled the archive. Use pax to unpack the tarball, converting files from ASCII to EBCDIC.

```
pax -r -o setfiletag -ofrom=ISO8859-1,to=IBM-1047 -f zlib-1.1.4.tar.gz
```

Apply the following patch to the zlib 1.1.4 sources:

```
----- zlib_1.1.4_zos.patch -----diff -purN downloads/
zlib/src.orig/configure downloads/zlib/src/configure--- downloads/zlib/src.orig/configure Tue Dec 16 14:09:57
2008+++ downloads/zlib/src/configure Mon Feb 9 14:30:49 2009@@ -116,6 +116,11 @@ else SFLAGS=${CFLAGS-"-
Kconform_pic -O"} CFLAGS=${CFLAGS-"-O"} LDSHARED=${LDSHARED-"cc -G"};;+ OS/390*)+ CC=xlc+
SFLAGS=${CFLAGS-"-qascii -q64 -Wc,DLL,LP64,XPLINK,EXPORTALL -D_ALL_SOURCE_NOTHREADS"}+
CFLAGS=${CFLAGS-"-qascii -q64 -Wc,DLL,LP64,XPLINK,EXPORTALL -D_ALL_SOURCE_NOTHREADS"}+
LDSHARED=${LDSHARED-"xlc -qascii -q64 -Wl,dll,LP64,XPLINK "};; # send working options for other systems to
support@gzip.org *) SFLAGS=${CFLAGS-"-O"} CFLAGS=${CFLAGS-"-O"}
```

Build and install the zlib DLL, placing the xlc compilers in compatibility to mode by setting the environment variable C89\_CCMODE to 1. When not in compatibility mode, xlc follows strict placement of command line options. Configure and build the zlib software with `./configure --shared && make`. By default, the configure script places zlib in `/usr/local/lib`. Install the software with `make install`. To ensure that GT.M finds zlib, include `/usr/local/lib` in `LIBPATH`, the environment variable that provides a search path for processes to use when they link DLLs.

By default, GT.M searches for the `libz.so` shared library (`libz.sl` on HP-UX PA-RISC) 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 `$LIBPATH` (AIX and z/OS) or `$LD_LIBRARY_PATH` (other UNIX platforms) 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.

`-tlsid=<label>`

Instructs the Source or Receiver Server to use the TLS certificate and private key pairs having `<label>` as the `TLSID` in the configuration file pointed to by the `gtmencrypt_config` environment variable. `TLSID` is a required parameter if TLS/SSL is to be used to secure replication connection between instances. If private keys are encrypted, an environment variable of the form `gtmtls_passwd_<label>` specifies their obfuscated password. You can obfuscate passwords using the `'maskpass'` utility provided along with the encryption plugin. If you use unencrypted private keys, set the `gtmtls_passwd_<label>` environment variable to a non-null dummy value; this prevents inappropriate prompting for a password.

`-[NO]PLAINtextfallback`

Specifies whether the replication server is permitted to fallback to plaintext communication. The default is `-NOPLAINtextfallback`. If `NOPLAINTEXTFALLBACK` is in effect, GT.M issues a `REPLNOTLS` error in the event it is unable to

## Database Replication

establish a TLS connection. [Note: GT.M versions prior to V6.1-000 did not support TLS for replication - if needed it could be implemented with an external application such as stunnel (<http://stunnel.org>.)] If PLAINTEXTFALLBACK is in effect, in the event of a failure to establish a TLS connection, GT.M issues REPLNOTLS as a warning. Once a permitted plaintext replication connection is established for a connection session, GT.M never attempts to switch that connection session to TLS connection.

*-RENEgotiate\_interval=<minutes >*

Specifies the time in minutes to wait before attempting to perform a TLS renegotiation. The default -RENEGOTIATE\_INTERVAL is a little over 120 minutes. A value of zero causes GT.M never to attempt a renegotiation. The MUIP REPLIC -SOURCE -JNLPOOL -SHOW [-DETAIL] command shows the time at which the next TLS renegotiation is scheduled, and how many such renegotiations have occurred thus far for a given secondary instance connection. As renegotiation requires the replication pipeline to be temporarily flushed, followed by the actual renegotiation, TLS renegotiation can cause momentary spikes in replication backlog.

## Shutting down the Source Server

Command syntax:

```
mupip replicate -source -shutdown [-timeout=<timeout in seconds>]
```

Qualifiers:

*-shutdown*

Shuts down the Source Server.

*-timeout=<timeout in seconds>*

Specifies the time (in seconds) the Source Server should wait before shutting down. If you do not specify *-timeout*, the default timeout period is 120 seconds. If you specify *-timeout=0*, shutdown occurs immediately.

## Activating a Passive Source Server

Command syntax:

```
mupip replicate -source -activate
  -secondary=<hostname:port>
  -log=<log file name>
  -connectparams=<hard tries>,<hard tries period>,
  <soft tries period>,<alert time>,<heartbeat period>,
  <max heartbeat wait>]
  -instsecondary=<instance_name>
  {-rootprimary|-propagateprimary}
```

Qualifiers:

*-activate*

Activates a passive Source Server. Once activated, the Source Server reads journal records from the Journal Pool and transports them to the system specified by *-secondary*.

Before activation, *-activate* sets the Source Server to ACTIVE\_REQUESTED mode. On successful activation, GT.M sets the Source Server mode to ACTIVE. GT.M produces an error when there is an attempt to activate a Source Server in ACTIVE\_REQUESTED mode.

## Database Replication

*-instsecondary=<instance\_name>*

Identifies the replicating instance to which the passive Source Server connects after activation.

With no *-instsecondary* specified, the passive Source Server uses the environment variable `gtm_repl_instsecondary` as the value of *-instsecondary*.

*-rootprimary*

Specifies that the passive Source Server activation occurs on an originating instance.

*-propagateprimary*

Specifies that the passive Source Server activation occurs on a propagating instance.

If neither *-rootprimary* nor *-propagateprimary* are specified, this command assumes *-propagateprimary*.

Example:

```
$ mupip replicate -source -activate -secondary=localhost:8998 -log=A2B.log -instsecondary=America
```

This example activates a Source Server from passive mode.

## Deactivating an Active Source Server

Command syntax:

```
mupip replicate -source -deactivate -instsecondary=<instance_name>
```

Qualifiers:

*-deactivate*

Makes an active Source Server passive. To change the replicating instance with which the Source Server is communicating, deactivate the Source Server and then activate it with a different replicating instance.

Before deactivation, *-deactivate* sets the Source Server to `PASSIVE_REQUESTED` mode. On successful deactivation, GT.M sets the Source Server mode to `PASSIVE`. GT.M produces an error when there is an attempt to deactivate a Source Server in `PASSIVE_REQUESTED` mode.

*-instsecondary=<instance\_name>*

Identifies the active Source Server to transition to the passive (standby) state.

With no *-instsecondary* specified, `$gtm_repl_instsecondary` determines the active Source Server.

*-rootprimary*

Specifies that the active Source Server is on originating instance.

*-propagateprimary*

Specifies that the active Source Server is on a propagating instance.

If neither *-rootprimary* nor *-propagateprimary* are specified, this command assumes *-propagateprimary*.

## Stopping the Source Filter

There are two ways to stop an active filter on the Source Server.

- Execute `mupip replicate -source -stopsourcefilter` on the originating Source Server.
- Specify `-stopsourcefilter` as an option for starting the Receiver Server.



If a filter fails to respond within just over a minute to delivery of a mini-transaction or TP transaction, a Source or Receiver Server issues a `FILTERTIMEDOUT` error, stops the filter, and exits.

## Checking Server Health

Use the following command and qualifier to determine whether the Source Server is running.

Command syntax:

```
mupip replicate -source -checkhealth
[-instsecondary=<instance_instance>] [-he[elpers]]
```

Qualifiers:

*-checkhealth*

Determine whether the Source Server is running. If the Source Server is running, the exit code is 0 (zero). If the Source Server is not running or an error exists, the exit code is not 0.

With helpers specified, `-checkhealth` displays the status of Helper Processes in addition to the status of Receiver Server and Update Process.

*-instsecondary=<instance\_name>*

Identifies a Source Server process.

If `-instsecondary` is not specified, `-checkhealth` checks all Source Server processes.

Example:

```
$ mupip replic -source -checkhealth -inst=INSTB
Fri May 21 15:26:18 2010 : Initiating CHECKHEALTH operation on source server pid [15511] for secondary
instance name [INSTB]
PID 15511 Source server is alive in ACTIVE mode

$ mupip replic -source -checkhealth -inst=INSTB
Fri May 21 15:29:52 2010 : Initiating CHECKHEALTH operation on source server pid [0] for secondary
instance name [INSTB]
PID 0 Source server is NOT alive
%GTM-E-SRCSRVNOTEXIST, Source server for secondary instance INSTB is not alive
```

## Changing the Log File

Command syntax:

```
mupip replicate -source -changelog -log=<log file name> [-log_interval=<integer>]  
▶ -instsecondary=<instance_name>
```



Qualifiers:

*-changelog*

Instructs the Source Server to change its log file.

*-instsecondary=<instance\_name>*

Identifies a Source Server process.

*-log=<log file name>*

Use this mandatory qualifier to specify the name of the new log file. If you specify the name of the current log file, no change occurs.

Example:

```
$ mupip replicate -source -changelog -log=/more_disk_space/newA2B.log -instsecondary=Brazil
```

*-log\_interval=<integer>*

Specifies the number of transactions for which the Source Server should wait before writing to the log file. The default logging interval is 1000 transactions.

*-log\_interval=0* reverts the logging interval to the prior value.

## Enabling/Disabling Detailed Logging

Command syntax:

```
mupip replicate -source -statslog={ON|OFF}  
[-log_interval=<integer>
```

Qualifiers:

*-statslog={ON | OFF}*

Enables or disables detailed logging. When ON, the system logs current-state information of the Source Server and messages exchanged between the Source and Receiver Servers. By default, detailed logging is OFF. Once you enable it (ON), changing *-statslog* to OFF can stop detailed logging.

*-log\_interval=<integer>*

Specifies the number of transactions for which the Source Server should wait before writing to the log file. The default logging interval is 1000 transactions.

*-log\_interval=0* reverts the logging interval to the prior value.

## Stopping a Source Server

Command Syntax:

```
mupip replic -source -shutdown [-instsecondary=<instance_name>] [-timeout=<seconds>]
```

Qualifiers:

*-instsecondary=<instance\_name>*

Identifies a Source Server process.

If *-instsecondary* is not specified, *-shutdown* stops all Source Server processes.

*-timeout=<seconds>*

Specifies the period of time (in seconds) a Source Server should wait before shutting down. If you do not specify *-timeout*, the default timeout period is 30 seconds. If you specify *-timeout=0*, shutdown occurs immediately.

## Reporting the Current Backlog of Journal Records

Command syntax:

```
mupip replicate -source -showbacklog
```

Qualifiers:

*-showbacklog*

Reports the current backlog of journal records (in terms of JNL\_SEQNO) on the output device (normally the standard output device). This qualifier does not affect the statistics logged in the log file. The backlog is the difference between the last JNL\_SEQNO written to the Journal Pool and the last JNL\_SEQNO sent by the Source Server to the Receiver Server. In the WAS\_ON state, *-showbacklog* reports the backlog information even if the Source Server is shut down.

Example:

```
$ mupip replic -source -showbacklog -inst=INSTB
Wed May 19 18:58:29 2010 : Initiating SHOWBACKLOG operation on source server pid [0] for secondary
instance [INSTB]
101 : backlog number of transactions written to journal pool and yet to be sent by the source server
102 : sequence number of last transaction written to journal pool
1 : sequence number of last transaction sent by source server
%GTM-E-SRCSRVTOTEXIST, Source server for secondary instance INSTB is not alive
```

## Processing Lost Transactions File

Except following a failover when the backlog is zero, whenever a former originating instance comes up as a new replicating instance, there is always a lost transaction file. Apply this lost transaction file at the new originating instance as soon as practicable because there could be additional lost transaction files in the event of other failovers. For example, failure of the new originating instance before the lost transaction file is processed. These additional lost transactions files can complicate the logic needed for lost transaction processing.

Apply the lost transactions on the new originating instance either manually or in a semi-automated fashion using the M-intrinsic function \$ZQGBLMOD(). If you use \$ZQGBLMOD(), two perform 2 additional steps ( mupip replicate -source -

## Database Replication

needrestart and mupip replicate -source -losttncomplete ) as part of lost transaction processing. Failure to run these steps can cause \$ZQGBLMOD() to return false negatives that in turn can result in application data consistency issues.

Command Syntax:

```
mupip replicate -source  
{-losttncomplete | -needrestart}  
-instsecondary=<replicating instance name>
```

### *-losttncomplete*

Indicate to GT.M that all lost transaction processing using \$ZQGBLMOD() is complete. Use this qualifier either explicitly or implicitly to prevent a future \$ZQGBLMOD() on an instance from returning false positives when applying future lost transactions. This ensures accuracy of future \$ZQGBLMOD() results.

Always use this qualifier when an originating instance comes up as a replicating instance.

Always run MUPIP REPLICATE -SOURCE -LOSTTNCOMPLETE on each of the replicating instances after applying all lost transaction files except on the following occasions:

- The replicating instance is connected to the originating instance at the time the command is run on the originating instance.
- The replicating instance is not connected at the time the command is run on the originating instance but connects to the originating instance, before the originating instance is brought down.

### *-needrestart*

Checks whether the originating instance ever communicated with the specified replicating instance (if the receiver server or a fetchresync rollback on the replicating instance communicated with the Source Server) since the originating instance was brought up. If so, this command displays the message SECONDARY INSTANCE xxxx DOES NOT NEED TO BE RESTARTED indicating that the replicating instance communicated with the originating instance and hence does not need to be restarted. If not, this command displays the message SECONDARY INSTANCE xxxx NEEDS TO BE RESTARTED FIRST. In this case, bring up the specified instance as a replicating instance before the lost transactions from this instance are applied. Failure to do so before applying the corresponding lost transactions causes \$ZQGBLMOD() to return false negatives which can result in application data inconsistencies.

The mupip replic -source -needrestart command should be invoked once for each lost transaction file that needs to be applied. It should be invoked on the new originating instance before applying lost transactions. Specify -instsecondary to provide the instance name of the replicating instance where the lost transaction file was generated. If not, the environment variable gtm\_repl\_instsecondary is implicitly assumed to hold the name of the replicating instance.

If the lost transaction file was generated from the same instance to which it is to be applied, a mupip replicate -source -needrestart command is not required.

Always remember to bring the replicating instance (specified in the -needrestart command) as an immediate replicating instance of the current originating instance. If it is brought up as a replicating instance through a different intermediate replicating instance, the -needrestart command unconditionally considers the instance as not having communicated with the originating instance even though it might be up and running.

The Source Server on the originating instance and/or Receiver Server or fetchresync rollback on the replicating instance need not be up and running at the time you run this command.

However, it is adequate if they were up at some point in time after the originating instance was brought up.

## Database Replication

This command protects against a scenario where the originating instance when the lost transaction file is generated is different from the primary instance when the lost transactions are applied (note that even though they can never be different in case of a dual-site configuration, use of this command is nevertheless still required).

`$ZQGBLMOD()` relies on two fields in the database file header of the originating instance to be set appropriately. `Zqgblmod Trans` and `Zqgblmod Seqno`. In an LMS configuration, if there are more than two instances, and no instances other than the originating and replicating instances are involved in the rollback `-fetchresync` participate in the sequence number determination. Hence, they do not have their `Zqgblmod Seqno` (and hence `Zqgblmod Trans`) set when that particular lost transaction file is generated. If any of the non-participating instances is brought up as the new originating instance and that particular lost transaction file is applied on the originating instance, the return values of `$ZQGBLMOD()` will be unreliable since the reference point (`Zqgblmod Trans`) was not set appropriately. Hence, this command checks whether the replicating instance where the lost transaction was previously generated has communicated with the current originating instance after it came up as the originating instance. If it is affirmative, the `Zqgblmod Seqno` and `Zqgblmod Trans` fields would have been appropriately set and hence `$ZQGBLMOD()` values will be correct.

Example:

```
$ mupip replic -source -losttncomplete
```

This command updates the `Zqgblmod Seqno` and `Zqgblmod Trans` fields (displayed by a `dse dump -fileheader` command) in the database file headers of all regions in the global directory to the value 0. Doing so causes a subsequent `$ZQGBLMOD()` to return the safe value of one unconditionally until the next lost transaction file is created.

## Lost Transaction File format

The first line of the lost transaction file include up to four fields. The first field displays `GDSJEX04` signifying the file format version. The second field indicates whether the lost transaction file is a result of a rollback or recover. If the second field is `ROLLBACK`, a third field indicates whether the instance was a `PRIMARY` or `SECONDARY` immediately prior to the lost transaction file being generated. If it was a `PRIMARY`, the lost transaction file is termed a primary lost transaction file and it needs to be applied on the new originating instance. A fourth field holds the name of the replicating instance on which the lost transactions were generated. This instance name should be used as the `-instsecondary` qualifier in the `mupip replic -source -needrestart` command when the lost transactions are applied at the new originating instance.

The first line of a lost transaction file looks like the following:

```
GDSJEX04 ROLLBACK SECONDARY Perth
```

## Starting the Receiver Server

Command syntax:

```
mupip replicate -receiver -start
  -listenport=<port number>
  -log=<log file name> [-log_interval="[integer1],[integer2]"]
  [-autorollback[=verbose]]
  [-buffsize=<Receive Pool size in bytes>]
  [-filter=<filter command>]
  [-noresync]
  [-stopsourcefilter]
  [-updateresync=</path/to/bkup-orig-repl-inst-file>]
  {[-resume=<strm_num>|-reuse=<instname>]}
  [-initialize] [-cmlvl=n]
  [-tlsid=<label>]
```

Qualifiers:

*-receiver*

Identifies the Receiver Server.

*-start*

Starts the Receiver Server and Update Process.

*-listenport=<port number>*

Specifies the TCP port number the Receiver Server will listen to for incoming connections from a Source Server. Note that the current implementation of the Receiver Server does not support machines with multiple IP addresses.

*-autorollback[=verbose]*

Allows the receiving instance of SI or BC replication to roll back as required when the source instance rolls back with a mupip journal *-rollback -backward* command. The optional value keyword **VERBOSE** allows roll back to provide more diagnostic information.



### Caution

As *autorollback* uses mupip online rollback under the covers, it should be considered field test grade functionality as long as that function is considered field test grade functionality.

*-log=<recsrv\_log\_file\_name >*

Specifies the location of the log file of the Receiver Server. When *-log* is specified, the Update Process writes to a log file named *<recsrv\_log\_file\_name>.updproc*. Note that the name of the Update Process log file name has *.updproc* at the end to distinguish it from the Receiver Server's log file name.

*-log\_interval="[integer1],[integer2]"*

*integer1* specifies the number of transactions for which the Receiver Server should wait before writing to its log file. *integer2* specifies the number of transactions for which the Update Process should wait before writing to its log file. The default logging interval is 1000 transactions.

If *integer1* or *integer2* is 0, the logging interval is set to the default value.

*-stopsourcefilter*

Starting the Receiver Server with *-stopsourcefilter* turns off any active filter on the originating Source Server. Use this option at the time of restarting the Receiver Server after a rolling upgrade is complete.

*-updateresync=</path/to/bkup-orig-repl-inst-file>*

*-updateresync* guarantees GT.M that the replicating instance was, or is, in sync with the originating instance and it is now safe to resume replication. Use *-updateresync* only in the following situations:

- To replace an existing replication instance files when an upgrade to a GT.M version changes the instance file format. Consult the release notes to determine whether this applies to your upgrade.
- When an existing replication instance file is unusable because it was damaged or deleted, and is replaced by a new replication instance file.

## Database Replication

- Setting up an A->P configuration for the first time if P is an existing instance with existing updates that are not, and not expected to be, in the originating instance.
- Setting up a new replicating instance from a backup of the originating instance (A->P only) or one of its replicating secondary instances.
- If you are running a GT.M version prior to V5.5-000 and you have to set up a replicating instance from a backup of an originating instance.

-updateresync uses the journal sequence number stored in the replicating instance's database and the history record available in the backup copy of the replication instance file of the originating instance (</path/to/bkup-orig-repl-inst-file>) to determine the journal sequence number at which to start replication.

When replication resumes after a suspension (due to network or maintenance issues), GT.M compares the history records stored in the replication instance file of the replicating instance with the history records stored in the replication instance file of the originating instance to determine the point at which to resume replication. This mechanism ensures that two instances always remain in sync when a replication connection resumes after an interruption. -updateresync bypasses this mechanism by ignoring the replication history of the replicating instance and relying solely on the current journal sequence number and its history record in the originating instance's history to determine the point for resuming replication. As it overrides a safety check, use -updateresync only after careful consideration. You can check with your GT.M support channel as to whether -updateresync is appropriate in your situation.

To perform an updateresync, the originating instance must have at least one history record. You need to take a backup (BACKUP -REPLINST) of the replication instance file of the originating instance while the Source Server is running. This ensures that the instance file has at least one history record. Even though it is safe to use a copy (for example, an scp) of the replication instance file of the originating instance taken after shutting down its Source Server, BACKUP -REPLINST is recommended because it does not require Source Server shutdown. You also need an empty instance file (-INSTANCE\_CREATE) of the replicating instance to ensure that it bypasses the history information of the current and prior states.

You also need use -updateresync to replace your existing replication instance files if a GT.M version upgrade changes the instance file format. The instance file format was changed in V5.5-000. Therefore, upgrading a replicating instance from a version prior to GT.M V5.5-000 up to V5.5-000 or higher requires replacing its instance file.

Prior to V5.5-000, -updateresync did not require the argument (<bkup\_orig\_repl\_inst\_file>). The syntax for -updateresync depends on the GT.M version.

For information on the procedures that use -updateresync, refer to “Setting up a new replicating instance of an originating instance (A->B, P->Q, or A->P)” (page 221), “Replacing the replication instance file of a replicating instance (A->B and P->Q)” (page 221), “Replacing the replication instance file of a replicating instance (A->P)” (page 221), and “Setting up a new replicating instance from a backup of the originating instance (A->P)” (page 222).

### *-initialize*

Used when starting a Receiver Server of an SI replication stream with -updateresync to specify that this is the first connection between the instances. MUPIP ignores these qualifiers when starting BC replication (that is, no updates permitted on the instance with the Receiver Server). This qualifier provides additional protection against inadvertent errors.

### *-resume=<strm\_num>*

Used when starting a Receiver Server of an SI replication stream with -updateresync in case the receiver instance has previously received from the same source but had only its instance file (not database files) recreated in between (thereby erasing all information about the source instance and the stream number it corresponds to recorded in the receiver instance

## Database Replication

file). In this case, the command `mupip replic -receiv -start -updateresync=<instfile> -resume=<strm_num>`, where *strm\_num* is a number from 1 to 15, instructs the receiver server to use the database file headers to find out the current stream sequence number of the receiver instance for the stream number specified as `<strm_num>`, but uses the input instance file (specified with `-updateresync`) to locate the history record corresponding to this stream sequence number and then exchange history with the source to verify the two instances are in sync before resuming replication. Note that in case `-resume` is not specified and only `-updateresync` is specified for a SI replication stream, it uses the input instance file name specified with `-updateresync` to determine the stream sequence number as well as provide history records to exchange with the source instance (and verify the two are in sync). Assuming that instance files are never recreated (unless they are also accompanied by a database recreate), this qualifier should not be required in normal usage situations.

`-reuse=<instname>`

Used when starting a Receiver Server of an SI replication stream with `-updateresync` in case the receiver instance has previously received from fifteen (all architecturally allowed) different externally sourced streams and is now starting to receive from yet another source stream. The command `mupip replic -receiv -start -updateresync=<instfile> -reuse=<instname>`, where *instname* is the name of a replication instance, instructs the receiver server to look for an existing stream in the replication instance file header whose *Group Instance Name* (displayed by a `mupip replic -editinstance -show` command on the receiver replication instance file) matches the instance name specified and if one does, reuse that stream number for the current source connection (erasing any record of the older Group using the same stream number).

`-noresync`

Instructs the Receiver Server to accept a SI replication stream even when the receiver is ahead of the source. In this case, the source and receiver servers exchange history records from the replication instance file to determine the common journal stream sequence number and replication resumes from that point onwards. Specifying `-noresync` on a BC replication stream is produces a `NORESYNCSUPPLONLY` error. Specifying `-noresync` on a SI replication stream receiver server where the receiving instance was started with `-UPDNOTOK` (updates are disabled) produces a `NORESYNCCUPDATERONLY` error. Note also that the `noresync` qualifier is not the opposite of the `resync` qualifier of `rollback` (`mupip journal -rollback -resync`), which is intended for use under the direction of FIS GT.M support.

`-tlsid=<label>`

Instructs the Source or Receiver Server to use the TLS certificate and private key pairs having `<label>` as the TLSID in the configuration file pointed to by the `gtmencrypt_config` environment variable. TLSID is a required parameter if TLS/SSL is to be used to secure replication connection between instances. If private keys are encrypted, an environment variable of the form `gtmtls_passwd_<label>` specifies their obfuscated password. You can obfuscate passwords using the 'maskpass' utility provided along with the encryption plugin. If you use unencrypted private keys, set the `gtmtls_passwd_<label>` environment variable to a non-null dummy value; this prevents inappropriate prompting for a password.

## Starting the Update Process

The following command starts the Update Process only, if it has been shutdown independent of the Receiver Server.

Command syntax:

```
mupip replicate -receiver -start {-updateonly|-helpers[=m[,n]]
```

Qualifiers:

`-updateonly`

If the Update Process has been shutdown independent of the Receiver Server, use this qualifier to restart the Update Process.

`-helpers[=m[,n]]`

Starts additional processes to help improve the rate at which updates from an incoming replication stream are applied on a replicating instance.

- `m` is the total number of helper processes and `n` is the number of reader helper processes, in other words  $m \geq n$ .
- Helper processes can start only on a receiver server.
- If helper processes are already running, specifying `-helpers[=m[,n]]` again starts additional helper processes. There can be a maximum of 128 helper processes for a receiver server.
- If `-helpers=0[,n]` is specified, GT.M starts no helper processes.
- With the `HELPERS` qualifier specified but neither `m` nor `n` specified, GT.M starts the default number of helper processes with the default proportion of roles. The default number of aggregate helper processes is 8, of which 5 are reader helpers and 3 writers.
- With only `m` specified, helper processes are started of which  $\text{floor}(5 * m / 8)$  processes are reader helpers.
- With both `m` and `n` specified, GT.M starts `m` helper processes of which `n` are reader helpers and `m-n` are writers. If  $m < n$ , mupip starts `m` readers, effectively reducing `n` to `m` and starting no writers.
- GT.M reports helper processes (for example, by the `ps` command and in `/proc/<pid>/cmdline` on platforms that implement a `/proc` filesystem) as `mupip replicate -updhelper -reader` and `mupip replicate -updhelper -writer`.

Example:

```
$ mupip replicate -receiver -start -listenport=1234 -helpers -log=B2C.log
▶ -buffsize=$recpool_size
```



This command starts the Receiver Server with Helper Processes. The following sample output from the `ps` command shows that there are 5 reader processes and 3 writer processes.

```
gtmuser1 11943 1      0 06:42 ? 00:00:00 /usr/library/GTM/mupip replicate -receiver -start
-listenport=1234 -helpers -log=B2C.log -buff=$rec_pool_size
gtmuser1 11944 11943 0 06:42 ? 00:00:00 /usr/library/GTM/mupip replicate -updateproc
gtmuser1 11945 11943 0 06:42 ? 00:00:00 /usr/library/GTM/mupip replicate -updhelper -reader
gtmuser1 11946 11943 0 06:42 ? 00:00:00 /usr/library/GTM/mupip replicate -updhelper -reader
gtmuser1 11947 11943 0 06:42 ? 00:00:00 /usr/library/GTM/mupip replicate -updhelper -reader
gtmuser1 11948 11943 0 06:42 ? 00:00:00 /usr/library/GTM/mupip replicate -updhelper -reader
gtmuser1 11949 11943 0 06:42 ? 00:00:00 /usr/library/GTM/mupip replicate -updhelper -reader
gtmuser1 11950 11943 0 06:42 ? 00:00:00 /usr/library/GTM/mupip replicate -updhelper -writer
gtmuser1 11951 11943 0 06:42 ? 00:00:00 /usr/library/GTM/mupip replicate -updhelper -writer
gtmuser1 11952 11943 0 06:42 ? 00:00:00 /usr/library/GTM/mupip replicate -updhelper -writer
```

## Stopping the Update Process

Command syntax:

```
mupip replicate -receiver -shutdown [-updateonly|-helpers] [-timeout=<timeout in seconds>]
```



Qualifiers:

*-updateonly*

Use this qualifier to stop only the Update Process. If neither *-updateonly* nor *-helper* are specified, the Update Process, all helper processes (if any), and Receiver Server shut down.

*-helper*

Shuts down only the Helper Processes and leaves the Receiver Server and Update Process to continue operating as before. All helpers processes shut down even if *-helper* values are specified.

*-timeout*

Specifies the period of time (in seconds) the Receiver Server should wait before shutting down. If you do not specify *-timeout*, the default timeout period is 30 seconds. If you specify *-timeout=0*, shutdown occurs immediately.

Example:

```
$ mupip replicate -receiver -shutdown -helper
```

This example shuts down only the helper processes of the current Receiver Server. Note that all helpers processes shut down even if HELPER values are specified.

## Checking Server Health

Use the following command to determine whether the Receiver Server is running.

Command syntax:

```
mupip replicate -receiver -checkhealth
```

## Changing the Log File

Command syntax:

```
mupip replicate -receiver -changelog -log=<log file name>
▶ [-log_interval="[integer1],[integer2]"]
```



*-log\_interval="[integer1],[integer2]"*

*integer1* specifies the number of transactions for which the Receiver Server should wait before writing to the log file. *integer2* specifies the number of transactions for which the Update Process should wait before writing to the log file. The default logging interval is 1000 transactions.

If *integer1* or *integer2* is 0, the logging interval reverts to the prior value.

## Enabling/Disabling Detailed Logging

Command syntax:

```
mupip replicate -receiver -statslog={ON|OFF}
```

```
[-log_interval="[integer1],[integer2]"]
```

```
-log_interval="[integer1],[integer2]"
```

*integer1* specifies the number of transactions for which the Receiver Server should wait before writing to the log file. *integer2* specifies the number of transactions for which the Update Process should wait before writing to the log file. The default logging interval is 1000 transactions.

If *integer1* or *integer2* is 0, the logging interval reverts to the prior value.

## Reporting the Current Backlog of Journal Records

Command syntax:

```
mupip replicate -receiver -showbacklog
```

Qualifiers:

```
-showbacklog
```

Use this qualifier to report the current backlog (that is, the difference between the last JNL\_SEQNO written to the Receive Pool and the last JNLSEQNO processed by the Update Process) of journal records on the Receiver Server.

## Rolling Back the Database After System Failures

Command syntax:

```
mupip journal -rollback
  {[-fetchresync=<port number>|-resync=<JNL_SEQNO>]
  [-rsync_strm=<strm_num>]}
  -losttrans=<extract file> -backward *
```

Qualifiers:

```
-rollback
```

Use this qualifier to rollback the database. If you do not use the *-fetchresync* qualifier, the database rolls back to the last consistent state.

```
-fetchresync=<port number>
```

The *<port number>* is the communication port number that the rollback command uses when fetching the reference point. Always use the same *<port number>* on the originating instance for rollback as the one used by the Receiver Server.



### Important

FIS recommends you to unconditionally script the `mupip journal -rollback -fetchresync` command prior to starting any Source Server on the replicating instance to avoid a possible out-of-sync situation.

The reference point sent by the originating instance is the RESYNC\_SEQNO (explained later) that the originating instance once maintained. The database/journal files are rolled back to the earlier RESYNC\_SEQNO (that is, the one received from originating instance or the one maintained locally). If you do not use *-fetchresync*, the database rolls back to the last consistent replicating instance state.

A FETCHRESYNC ROLLBACK operation sends its current working directory to the Source Server log.

The system stores extracted lost transactions in the file <extract file> specified by this mandatory qualifier. The starting point for the search for lost transactions is the JNL\_SEQNO obtained from the originating instance in the -fetchresync operation. If -fetchresync is not specified, <extract file> lists the post-consistent-state transactions that were undone by the rollback procedure to reach a consistent state.



### Note

The extracted lost transactions list may contain broken transactions due to system failures that occurred during processing. Do not resolve these transactions—they are not considered to be committed.



### Caution

The database header may get corrupted if you suspend an ongoing ROLLBACK -FETECHRESYNC operation or if the TCP connection between the two instances gets broken. The workaround is to restart the ROLLBACK -FETCHRESYNC operation or wait 60 seconds for the FETCHRESYNC operation to timeout.

Example:

```
$ mupip journal -rollback -fetchresync=2299 -losttrans="glo.lost" -backward *
```

This command performs a ROLLBACK -FETCHRESYNC operation on a replicating instance to bring it to a common synchronization point from where the originating instance can begin to transmit updates to allow it to catch up. It also generates a lost transaction file glo.lost of all those transactions that are present on the replicating instance but not on the originating instance at port 2299.

*-resync=<JNL\_SEQNO>*

Use this qualifier to roll back to the transaction identified by JNL\_SEQNO (in decimal) only when the database/ journal files need to be rolled back to a specific point. If you specify a JNL\_SEQNO that is greater than the last consistent state, the database/ journal files will be rolled back to the last consistent state. Under normal operating conditions, you would not need this qualifier.

*-losttrans=<extract file>*

If failover occurs (that is, originating instance fails and replicating instance assumes the originating instance role), some transactions committed to A's database may not be reflected in B's database. Before the former originating instance becomes the new replicating instance, these transactions must be rolled off before it can assume the role of an originating instance. These transactions are known as "lost transactions".

The system stores extracted lost transactions in the file <extract file> specified by this mandatory qualifier. The starting point for the search for lost transactions is the JNL\_SEQNO obtained from the originating instance in the -fetchresync operation. If -fetchresync is not specified, <extract file> lists the post-consistent-state transactions that were undone by the rollback procedure to reach a consistent state.



### Note

The extracted lost transactions list may contain broken transactions due to system failures that occurred during processing. Do not resolve these transactions—they are not considered to be committed.

*-rsync\_strm=<strm\_num>*

## Database Replication

Used when starting a rollback command with the `-resync` qualifier. The command `mupip journal -rollback -resync=<sequence_num> -rsync_strm=<strm_num>` instructs rollback to roll back the database to a sequence number specified with the `-resync=<sequence_num>` qualifier but that `<sequence_num>` is a journal stream sequence number (not a journal sequence number) corresponding to the stream number `<strm_num>` which can be any value from 0 to 15. Note that like the `-resync` qualifier, the `-rsync_strm` qualifier is also intended for use under the direction of your GT.M support channel.

---

## Chapter 8. M Lock Utility (LKE)

Revision History		
Revision V6.0-000/1	21 November 2012	In “SHow ” [257], added updated for V6.0-000.
Revision V5.5-000/4	6 June 2012	In “SHow ” [257], added the description of the LOCKSPACEUSE message.
Revision V5.4-002B	24 October 2011	Conversion to documentation revision history reflecting GT.M releases with revision history for each chapter.

---

### Introduction

The M Lock Utility (LKE) is a tool for examining and changing the GT.M LOCK environment. For a description of M LOCKs, refer to the LOCKs section in the General Language Features of M chapter and the description of the LOCK command in the Commands chapter of the *GT.M Programmer's Guide*.

The two primary functions of the M Lock Utility (LKE) are:

1. **SHOW** all or specified LOCKs currently active
2. **CLEAR** all or specified LOCKs currently active

When debugging an M application, you may use LKE to identify a possible deadlock situation, that is, two or more processes have LOCKs and are waiting to add resource names LOCKed by the other(s).

```
Process 1   Process 2
LOCK A
            LOCK B
            LOCK +A
LOCK +B
```

Process 1 has A LOCKed and attempts to LOCK B. Process 2 has B LOCKed and attempts to LOCK A. Because these processes do not release their current LOCKs before adding additional LOCKs, nor do they provide a timeout to detect the problem, they are deadlocked. Neither process can proceed normally. You can use LKE to release one of the LOCKs so both processes may execute. However, because releasing a LOCK may cause the process to violate its design assumptions, terminating one process is generally a safer way to break the deadlock.



#### Note

When a process leaves M, GT.M normally releases any LOCKs or ZALLOCATEs held by that process. If a GT.M process terminates abnormally, or if the system "crashes" while a GT.M process is active, GT.M cannot perform normal clean-up. However, as soon as any other process waits several seconds for a LOCK owned by a process that no longer exists, GT.M automatically clears the "orphaned" LOCK.

### To Invoke and Exit LKE

GT.M installation procedure places the LKE utility package in a directory specified by the environment variable `gtm_dist`.

## M Lock Utility (LKE)

LKE requires that the environment variable `gtmgbldir` be defined.

Invoke LKE using the following command at the shell prompt. If this does not work, consult your system manager to investigate setup and file access issues.

```
$gtm_dist/lke LKE>
```



### Important

Always run LKE on the node where the lock is held.

When LKE is ready to accept commands, it displays the `LKE>` prompt. To leave LKE, enter the `EXIT` command at the `LKE>` prompt.

When additional information is entered on the command line after the LKE command, LKE processes the additional information as its command.

```
$gtm_dist/lke show -all
```

This command displays all current LOCKs and then returns to the shell prompt.

If your LKE argument contains quotes, precede each quote in the argument by a back-slash (`\`) or enclose the entire argument in a set of quotes (matching single or double). Apply this convention only for those LKE commands that you run from the shell.

```
$gtm_dist/lke show -lock="^Account(\\"Name\\")"  
$gtm_dist/lke show -lock='^Account("Name")'
```

Both these commands display the status of LOCK `^Account("Name")` in the default region.

## To establish a Global Directory

LKE uses the environment variable `gtmgbldir` to identify the active global directory. `gtmgbldir` should be defined by individual users in their login files.

```
$ gtmgbldir=prod.gld  
$ export gtmgbldir
```

## LKE Commands and Qualifiers

The general format of LKE commands is:

```
command [-qualifier[=qualifier-value]]
```

LKE accepts command and qualifier abbreviations. The section describing each command provides the minimal abbreviation that can be used for that command, and the command qualifiers, if any. FIS recommends the use of a minimum of four characters for key words in scripts to ensure new keywords do not conflict with older scripts.

## Clear

Use the **CLEAR** command to remove active LOCKs.



## Caution

FIS recommends restricting the use of the LKE CLEAR facility to debugging environments; removing LOCKs in a production environment typically violates application design assumptions and can cause aberrant process behavior. GT.M automatically removes abandoned LOCKs so it is typically safer to MUPIP STOP a process that is inappropriately hanging on to a LOCK.

The format of the **CLEAR** command is:

```
C[LEAR] [-qualifier...]
```

The optional qualifiers are:

```
-A[LL]
-L[OCK]
-[NO]C[RIT]
-[NO]EXACT
-[NO]I[NTERACTIVE]
-O[UTPUT]="file-name"
-P[ID]=pid
-R[EGION]=region-name
```

By default, CLEAR operates interactively (**-INTERACTIVE**).

Qualifiers for CLEAR

```
-A[LL]
```

Specifies all current LOCKs.

- -ALL removes all current LOCKs.
- If used, **CLEAR** and **-REGION** qualifier, **-ALL** removes all LOCKs in that region.
- Issue a **CLEAR - ALL** only when there are no active GT.M processes using LOCKs, or when you can predict the effect on the application.
- By default, **CLEAR -ALL** operates interactively (**-INTERACTIVE**).

```
-[NO]C[RIT]
```

Allows **LKE CLEAR** to work even if another process is holding a critical section.



## Caution

This can damage current LOCKs and the LOCK mechanism. It is intended for use only under the direction of FIS.

By default LKE operates in CRIT mode and ensures a consistent view of LOCKs by using the database critical section(s).

```
-[NO]EXACT
```

Limits the CLEAR command to the exact resource name specified with **-LOCK=resource\_name**. NOEXACT (the default) treats the specified resource name as a prefix and works not only on it, but also on any of its descendants, since their existence effectively LOCK their parent tree.

## M Lock Utility (LKE)

```
-L[OCK]="resource_name"
```

Unless used with **-EXACT**, specifies the leading prefix for an implicit wild card search of all locks that start with the **resource\_name**.

- The **resource\_name** is enclosed in **two** double quotation marks (" "). Because M resource names are formatted the same as global nodes with punctuation characters, in this context they are usually enclosed in sets of double quotation marks with string subscripts enclosed in sets of two double quotations.
- When used with **CLEAR**, **-LOCK** removes the locks that start with **resource\_name**.
- When used with **SHOW**, **-LOCK** provides a precise way to examine the specified lock.

```
-[NO]I[NTERACTIVE]
```

Interactively clears one **LOCK** at a time. LKE displays each current **LOCK** with the **PID** of the owner process and prompts for verification that the **LOCK** should be cleared. LKE retains the **LOCK** for any response other than **Y[ES]**.

- By default, **CLEAR** operates interactively (**-INTERACTIVE**).
- To avoid holding a lock resource too long, LKE skips to the next matching **LOCK** if there is no operator response for several seconds.
- **-NOINTERACTIVE** forces the action to take place without user confirmation of each change. Using **-NOINTERACTIVE** prevents the LKE operator from controlling the **LOCK** subsystem for potentially long periods of time when many locks are held. To do this, it limits the amount of time it waits for each response.

```
-O[UTPUT]="file-name"
```

Directs the reporting of all specified **LOCKS** to a file.

- If you specify an existing file, LKE creates a new version and overwrites that file.
- If **file-name** has permission issues, **OUTPUT** reports the cause of the error.
- The **-OUTPUT** qualifier is compatible with all other qualifiers.
- By default, **CLEAR** sends output messages to **stdout**.

```
-P[ID]=pid
```

Specifies the process identification number that holds a **LOCK** on a resource name.

- LKE interprets **pid** as a decimal number.
- **PID** clears **LOCKS** held by the process with the specified process identification number.
- Provides a means for directing **CLEAR** to **LOCKS** held by a process that is behaving abnormally.
- The **-PID** qualifier is compatible with all other qualifiers.

```
-R[EGION]=region-name
```

## M Lock Utility (LKE)

**region-name** specifies the region that holds the locked resource names.

- **REGION** clears **LOCKS** mapped by the current global directory to a region specified by the region-name.
- The **-REGION** qualifier is compatible with all other qualifiers.
- By default, **CLEAR -REGION=** operates interactively (**-INTERACTIVE**).

Example:

```
LKE>CLEAR -ALL
```

This command clears all current **LOCKS**.

Example:

```
LKE>clear -pid=2325 -interactive
```

This command presents all **LOCKS** held by the process with **PID** equal to 2325. You can choose whether or not to clear each **LOCK**.

```
LKE>clear -reg=areg -interactive
```

This command produces an output like the following:

```
AREG ^a Owned by PID= 2083 which is an existing  
process Clear lock ?
```

Type **Yes** or **Y** in response to the prompt.

LKE responds with an informational message:

```
%GTM-S-LCKGONE, Lock removed : ^a
```

Type **Yes** or **N** or **No** or **N** until all **LOCKS** are displayed and acted upon.

```
LKE> clear -pid=4208 -nointeractive
```

This command clears the lock held by a process with **PID** 4208. This command produces an output like the following:

```
DEFAULT Lock removed : ^A
```

Note that **-NOINTERACTIVE** forced the action without asking for a confirmation.

Example:

```
LKE>clear -lock="^a("b")  
Clear lock ? y  
Lock removed : ^a("b")  
LKE>
```

This command clears **lock** **^a("b")** in the default region.

Example:

```
LKE>clear -lock="^a" -nointeractive
```

## M Lock Utility (LKE)

This command clears all the locks that start with "^a" in the default region. **-NOINTERACTIVE** qualifier instructs LKE to clear these locks without further user intervention.

Example:

```
LKE>clear -lock="^a" -exact -nointeractive
```

This command clears **lock ^a** in the default region. **-NOINTERACTIVE** instructs LKE to clear **lock ^a** without further user intervention.

Example:

```
LKE>CLEAR -PID=4109 -LOCK=""^A""
Clear lock ? Y
Lock removed : ^A
LKE>
```

This command clears **LOCK ^A** held by process with PID 4109.

## SHow

Use the **SHOW** command to get status of the LOCK mechanism and the LOCK database. The format of the SHOW command is:

```
SH[OW] [-qualifier...]
```

The optional qualifiers are:

```
-A[LL]
-L[OCK]
-[NO]C[RIT]
-O[UTPUT]="file-name"
-P[ID]=pid
-R[EGION]=region-name
-W[AIT]
```

- By default, **SHOW** displays **-A[LL]**.
- The **SHOW** command reports active LOCKs. Information includes the LOCK resource name and the process identification (PID) of the LOCK owner.
- LKE SHOW displays lock space usage with a message in the form of: "%GTM-I-LOCKSPACEUSE, Estimated free lock space: xxx% of pppp pages." If the lock space is full, it also displays a LOCKSPACEFULL error.
- A LOCK command which finds no room in LOCK\_SPACE to queue a waiting LOCK, does a slow poll waiting for LOCK\_SPACE to become available. If LOCK does not acquire the ownership of the named resource with the specified timeout, it returns control to the application with \$TEST=0. If timeout is not specified, the LOCK command continues to do a slow poll till the space becomes available.
- LOCK commands which find no available lock space send a LOCKSPACEFULL message to the operator log. To prevent flooding the operator log, GT.M suppresses further such messages until the lock space usage drops below 75% full.
- The results of a **SHOW** may be immediately "outdated" by **M LOCK** activity.

## M Lock Utility (LKE)

- If the LOCK is owned by a GT.CM server on behalf of a client GT.M process, then **LKE SHOW** displays the client **NODENAME** (limited to the first 15 characters) and client **PID**. The client **PID (CLNTPID)** is a decimal value in UNIX.



### Note

GT.CM is an RPC-like way of remotely accessing a GT.M database.

**-ALL**

Specifies all current LOCKs.

- **-ALL** displays all current LOCKs in all regions and information about the state of processes owning these LOCKs.
- The **-ALL** qualifier is compatible with all other qualifiers.
- When **-ALL** is combined with **-PID** or **-REGION**, the most restrictive qualifier prevails.
- **SHOW -ALL** and **-WAIT** displays both **-ALL** and **-WAIT** information.

**-L[OCK]=resource\_name**

*resource\_name* specifies a single lock.

- The *resource\_name* is enclosed in double quotation marks (" "). Because M resource names are formatted the same as global nodes with punctuation characters, in this context they are usually enclosed in sets of double quotation marks with string subscripts enclosed in sets of two double quotations.
- When used with the **CLEAR** command, the **LOCK** qualifier removes the specified lock.
- When used with the **SHOW** command, the **LOCK** qualifier provides a precise way to examine the specified lock and any descendant LOCKed resources.

**-[NO]C[RIT]**

Allows the **SHOW** command to work even if another process is holding a critical section.

- By default LKE operates in **CRIT** mode and ensures a consistent view of LOCKs by using the database critical section(s).
- Use **NOCRIT** with **SHOW** only when normal operation is unsuccessful, as **NOCRIT** may cause LKE to report incomplete or inconsistent information.

**-O[UTPUT]="file-name"**

Directs the reporting of all specified LOCKs to a file.

- If you specify an existing file, LKE creates a new version and overwrites that file.
- The **-OUTPUT** qualifier is compatible with all other qualifiers.
- By default, the **SHOW** command send output messages to stdout.

## M Lock Utility (LKE)

```
-P[ID]=pid
```

Specifies the process identification number that holds a LOCK on a resource name.

- LKE interprets *pid* as a decimal number.
- **PID** displays all LOCKs owned by the specified process identification number.
- The **-PID** qualifier is compatible with all other qualifiers; the most restrictive of the qualifiers prevails.
- By default, **SHOW** displays the LOCKs for all PIDs.

```
-R[EGION]=region-name
```

Specifies the region that holds the locked resource names.

- The **REGION** qualifier displays LOCKs of that specified region.
- The **REGION** qualifier is compatible with all other qualifiers; the most restrictive of the qualifiers prevails.
- By default, **SHOW** displays the LOCKs for all regions.

```
-W[AIT]
```

Displays the LOCK resource name and the process state information of all processes waiting for the LOCK to be granted.

- **SHOW -WAIT** does not display the owner of the LOCK.
- **SHOW -ALL -WAIT** displays both **-ALL** and **-WAIT** information.
- When a process abandons a "wait" request, that request may continue to appear in **LKE SHOW -WAIT** displays. This appearance is harmless, and is automatically eliminated if the GT.M lock management requires the space which it occupies.

Use the following procedure to display all LOCKs active in the database(s) defined by the current global directory.

```
LKE> SHOW -ALL -WAIT
```

This produces an output like the following:

```
No locks were found in DEFAULT
AREG
^a Owned by PID=2080 which is an existing process
BREG
^b(2) Owned by PID= 2089 which is a nonexistent process
No locks were found in CREG
```

Example:

```
LKE>SHOW -ALL
```

This command displays all LOCKs mapped to all regions of the current global directory. It produces an output like the following:

```
DEFAULT
^A Owned by PID= 5052 which is an existing process
```

## M Lock Utility (LKE)

```
^B Owned by PID= 5052 which is an existing process
%GTM-I-LOCKSPACEUSE, Estimated free lock space: 99% of 40 pages
```

Example:

```
LKE>show -lock="^a"("^"b"")"
```

This command shows **lock ^a("b")** in the default region.

Example:

```
LKE>SHOW -CRIT
```

This command displays all the applicable locks held by a process that is holding a critical section.

Example:

```
LKE>show -all -output="abc.lk"
```

This command create a new file called abc.lk that contains the output of the **SHOW -ALL** command.

Example:

```
LKE>show -pid=4109
```

This command displays all locks held by process with PID 4109 and the total lock space usage.

Example:

```
LKE>show -region=DEFAULT -lock="^^A"
```

This command displays the lock on **^A** in the region **DEFAULT**. It produces an output like the following:

```
DEFAULT
^A Owned by PID= 5052 which is an existing process
%GTM-I-LOCKSPACEUSE, Estimated free lock space: 99% of 40 pages
```

## Exit

The **EXIT** command ends an LKE session. The format of the **EXIT** command is:

```
E[EXIT]
```

## Help

The **HELP** command explains LKE commands. The format of the **HELP** command is:

```
H[ELP] [options...]
```

Enter the LKE command for which you want information at the Topic prompt(s) and then press **RETURN** or **CTRL-Z** to return to the LKE prompt.

Example:

```
LKE> HELP SHOW
```

This command displays help for the **SHOW** command.

## SPawn

Use the **SPAWN** command to create a sub-process for access to the shell without terminating the current LKE environment. Use the **SPAWN** command to suspend a session and issue shell commands such as **ls** or **printenv**.

The format of the **SPAWN** command is:

```
SP[AWN]
```

The SPAWN command has no qualifiers.

Example:

```
LKE>spawn
```

This command creates a sub-process for access to the current shell without terminating the current LKE environment. Type **exit** to return to LKE.

## Summary

COMMAND	QUALIFIER	COMMENTS
C[LEAR]	-ALL -L[OCK] -[NO]CRIT -[NO]EXACT -[NO]I[NTERACTIVE] -O[UTPUT]=file-name -P[ID]=pid -R[EGION]=name	Use CLEAR with care and planning.
E[XIT]	none	-
H[ELP]	[option]	-
SH[OW]	-ALL -L[OCK] -[NO]CRIT -N[OINTERACTIVE] -O[UTPUT]=file-name -P[ID]=pid	-

## M Lock Utility (LKE)

COMMAND	QUALIFIER	COMMENTS
	-R[EGION]=name -W[AIT]	
SP[AWN]	none	shellcommand

## LKE Exercises

When using M Locks, you must use a well designed and defined locking protocol. Your locking protocol must specify guidelines for acquiring LOCKs, selecting and using timeout, releasing M Locks, defining a lock strategy according the given situation, identifying potential deadlock situations, and providing ways to avoid or recover from them. This section contains two exercises. The first exercise reinforces the concepts of GT.M LOCKs previously explained in this chapter. The second exercise describes a deadlock situation and demonstrates how one can use LKE to identify and resolve it.

### Exercise 1: Preventing concurrent updates using M Locks

Consider a situation when two users (John and Tom) have to exclusively update a global variable `^ABC`.



#### Note

Transaction Processing may offer a more efficient and more easily managed solution to the issue of potentially conflicting updates. For more information, see General Language Features of M chapter of the *GT.M Programmer's Guide*.

At the GT.M prompt of John, execute the following commands:

```
GTM>lock +^ABC
```

This command places a GT.M LOCK on "`^ABC`" (not the global variable `^ABC`). Note: LOCKs without the +/- always release all LOCKs held by the process, so they implicitly avoid dead locks. With **LOCK +**, a protocol must accumulate LOCKs in the same order (to avoid deadlocks).

Then execute the following command to display the status of the LOCK database.

```
GTM>zsystem "lke show -all"
```

This command produces an output like the following:

```
DEFAULT ^ABC Owned by PID= 3657 which is an existing  
process
```

Now, without releasing lock `^ABC`, execute the following commands at the GT.M prompt of Tom.

```
GTM>lock +^ABC
```

This command wait for the lock on resource "`^ABC`" to be released. Note that that the LOCK command does not block global variable `^ABC` in any way. This command queues the request for locking resource "`^ABC`" in the LOCK database. Note that you can still modify the value of global variable `^ABC` even if it is locked by John.

Now, at the GT.M prompt of John, execute the following command:

```
GTM>zsystem "LKE -show -all -wait"
```

This command produces an output like the following:

```
DEFAULT ^ABC Owned by PID= 3657 which is an existing process
Request PID= 3685 which is an existing process
```

This output shows that the process belonging to John with PID 3657 currently owns the lock for global variable `^ABC` and PID of Tom has requested the ownership of that lock. You can use this mechanism to create an application logic that adhere to your concurrent access protocols.

## Exercise 2: Rectifying a deadlock situation

Now, consider another situation when both these users (John and Tom) have to update two text files. While an update is in progress, a GT.M LOCK should prevent the other user from LOCKing that file. In some cases, a deadlock occurs when both users cannot move forward because they do not release their current LOCKs before adding additional LOCKs.

A deadlock situation can occur in the following situation:

John	Tom
LOCK +file_1	LOCK +file_2
LOCK +file_2	LOCK +file_1

Here both the users are deadlocked and neither can move forward. Note that a deadlock situation does not actually block the underlying resource.

Let us now create this situation.

At the GT.M prompt of John, execute the following commands:

```
GTM>set file1="file_1.txt"
GTM>lock +file1
GTM>open file1:APPEND
GTM>use file1
GTM>write "John",!
GTM>close file1
```

Note that John has not released the LOCK on resource "file1".

At the GT.M prompt of Tom, execute the following commands:

```
GTM> set file2="file_2.txt"
GTM> lock +file2
GTM> open file2:APPEND
GTM> use file2
GTM>write "Tom",!
GTM>close file2
```

Note that Tom has not released the LOCK on resource "file2".

Now, at the GT.M prompt of John, execute the following commands.

```
GTM>set file2="file_2.txt"
GTM>lock +file2
```

The latter command attempts to acquire a lock on resource *file2* that is already locked by Tom. Therefore, this results in a deadlock situation. Repeat the same process for Tom and attempt to lock resource *file1*.

## M Lock Utility (LKE)

Execute the following command at LKE prompt to view this deadlock situation.

```
LKE>show -all -wait
file1 Owned by PID= 2080 which is an existing process
Request PID= 2089 which is an existing process
file2 Owned by PID= 2089 which is an existing process
Request PID=2080 which is an existing process
```

This shows a deadlock situation where neither user can proceed forward because it is waiting for the other user to release the lock. You can resolve this situation by clearing the locks using the **LKE CLEAR -PID** command.



### Caution

Avoid using the **LKE CLEAR** command to clear a deadlock in a production environment as it may lead to unpredictable application behavior. Always use the **MUPIP STOP** command to clear a deadlock situation in your production environment. However, in a debugging environment, you can use LKE to debug LOCKs, analyze the status of the LOCK database and even experiment with **LKE CLEAR**.

---

## Chapter 9. GT.M Database Structure(GDS)

Revision History		
Revision V6.0-003/1	19 February 2014	In “File Header Data Elements ” [266], updated the descriptions for V6.0-003.
Revision V6.0-003	27 January 2014	In “GDS Blocks ” [273], corrected the description of block header fields.
Revision V5.5-000/10	28 September 2012	<ul style="list-style-type: none"><li>• In “Use of Keys ” [275], corrected the first sentence as follows:  GT.M locates records by finding the first key in a block lexically greater than, or equal to, the current key.</li><li>• Added the “Using GDS records to hold spanning nodes ” [274] section.</li></ul>
Revision V5.4-002B	24 October 2011	Conversion to documentation revision history reflecting GT.M releases with revision history for each chapter.

GDS is an FIS proprietary internal database structure used to store global variables and lock resource names. A high-level understanding of GDS can help database administrators correctly interpret GT.M logs/error messages, and maintain database metrics. You should always consult GT.M support ([gtmsupport@fisglobal.com](mailto:gtmsupport@fisglobal.com)) in the unlikely event of getting database integrity issues.



### Note

This chapter provides a high-level overview of GDS components. A comprehensive description of all the components of GDS is beyond the scope of this chapter.

---

## Database File Organization with GDS

GT.M processes a GDS file using predominantly low-level system services. The GDS file consists of two parts:

- The database file header
- The database itself

### Database File Header

The fields in the file header convey the following types of information:

- Data Elements
- Master Bitmap

## File Header Data Elements

All GT.M components, except GDE, (the run-time system, DSE, LKE, MUPIP) use the data elements of the file header for accounting, control, and logging purposes.

The current state of the file header always determines the characteristics of the database. The MUPIP CREATE command initializes the values of the file header data elements from the global directory and creates a new .DAT file.

The file header data elements are listed as follows in alphabetical order for easier access, rather than the order in which they appear in the file header.

Data Elements	Description
Access method	The buffering strategy of the database. Access Method can have 2 values - BG or MM. The default value is BG.  Buffered Global (BG) manages the buffers (the OS/file system may also buffer "in series"); MM - the OS/file system manages all the buffering.
Block size (in bytes)	The size (in bytes) of a GDS block. Block size can have values that are multiples of 512. The default value is 1024. Block size should be a multiple of the native block size for the OS file system chosen to accommodate all but outlying large records. For additional information, see Chapter 4: "Global Directory Editor" (page 32).
Blocks to Upgrade	The count of the blocks in the database that are still in prior major version format. GT.M uses this element during incremental upgrades.
Cache freeze id	The process identification number (PID) of a process which has suspended updates to the segment.
Certified for Upgrade to V5	Count of blocks "pre-certified" (with the dbcertify utility) for an incremental upgrade. GT.M uses this element during incremental upgrades.
Create in progress	Create in progress is TRUE only during the MUPIP CREATE operation. The normal value is FALSE.
Collation Version	The version of the collation sequence definition assigned to this database. DSE only reports this if an external collation algorithm is specified.
Commit Wait Spin Count	COMMITWAIT_SPIN_COUNT specifies the number of times a GT.M process waiting for control of a block to complete an update should spin before yielding the CPU when GT.M runs on SMP machines.
Current transaction	The 64-bit hexadecimal number of the most recent database transaction.
Default Collation	The collation sequence currently defined for this database. DSE only reports this if an external collation algorithm is defined.
Desired DB Format	The desired version format of database blocks. Desired DB Format can have 2 possible values- the major version for the current running GT.M distribution or the last prior major version. Newly created databases and converted databases have the current major version.
Endian Format	The Endian byte ordering of the platform.
Extension Count	The number of GDS blocks by which the database file extends when it becomes full. The default value is 100 and the maximum is 65535. In production, typically this value should reflect the amount of new space needed in a relatively long period (say a week or a month). UNIX file systems use lazy allocations so this value controls the frequency at which GT.M

### GT.M Database Structure(GDS)

Data Elements	Description
	checks the actual available space for database expansion in order to warn when space is low.
Flush timer	Indicates the time between completion of a database update and initiation of a timed flush of modified buffers. The default value is 1 second and the maximum value is 1 hour.
Flush trigger	The total number of modified buffers that trigger an updating process to initiate a flush. The maximum and default value is 93.75% of the global buffers; the minimum is 25% of the global buffers. For large numbers of global buffers, consider setting the value towards or at the minimum.
Free blocks	The number of GDS blocks in the data portion of the file that are not currently part of the indexed database (that is, not in use). MUPIP INTEG -NOONLINE (including -FAST) can rectify this value if it is incorrect.
Free space	The number of currently unused blocks in the fileheader (for use by enhancements).
Freeze match	The PID extension (image count) for Cache freeze id - OpenVMS only
Global Buffers	The number of BG buffers for the region. It can have values that are multiples of 512 (in bytes). The minimum value is 64 and the maximum is 2147483647 (may vary depending on your platform). The default value is 1024. In a production system, this value should typically be higher.
In critical section	The process identification number (PID) of the process in the write-critical section, or zero if no process holds the critical section.
Journal Alignsize	<p>Specifies the number of 512-byte-blocks in the alignsize of the journal file. DSE only reports this field if journaling is ENABLED (or ON).</p> <p>If the ALIGNSIZE is not a perfect power of 2, GT.M rounds it up to the nearest power of 2.</p> <p>The default and minimum value is 4096. The maximum value is 4194304 (=2 GigaBytes).</p> <p>A small alignsize can make for faster recover or rollback operations, but makes less efficient use of space in the journal file.</p>
Journal Allocation	The number of blocks at which GT.M starts testing the disk space remaining to support journal file extensions. DSE only reports this field if journaling is ENABLED or ON.
Journal AutoSwitchLimit	<p>The number of blocks after which GT.M automatically performs an implicit online switch to a new journal file. DSE only reports this field if journaling is ENABLED or ON.</p> <p>The default value for Journal AutoSwitchLimit is 8386560 &amp; the maximum value is 8388607 blocks (4GB-512 bytes). The minimum value is 16384. If the difference between the Journal AutoSwitchLimit and the allocation value is not a multiple of the extension value, GT.M rounds-down the value to make it a multiple of the extension value and displays an informational message.</p>
Journal Before imaging	<p>Indicates whether or not before image journaling is allowed; DSE only reports this field if journaling is ENABLED or ON.</p> <p>Journal Before imaging can either be TRUE or FALSE.</p>
Journal Buffer Size	The amount of memory allotted to buffer journal file updates. The default value is 2308. The minimum is 2307 and the maximum is 32K blocks which means that the maximum buffer you can set for your journal file output is 16MB. Larger journal buffers can improve

## GT.M Database Structure(GDS)

Data Elements	Description
	run-time performance, but they also increase the amount of information at risk in failure. Journal Buffer size must be large enough to hold the largest transaction.
Journal Epoch Interval	<p>The elapsed time interval between two successive EPOCHs in seconds. An EPOCH is a checkpoint, at which all updates to a database file are committed to disk. All journal files contain epoch records. DSE only reports this field if journaling is ENABLED or ON.</p> <p>The default value is 300 seconds (5 minutes). The minimum is 1 second and the maximum value is 32,767 (one less than 32K) seconds, or approximately 9.1 hours. Longer Epoch Intervals can increase run-time performance, but they can also cause longer recovery times.</p>
Journal Extension	<p>The number of blocks used by GT.M to determine whether sufficient space remains to support continuing journal file growth. DSE only reports this field if journaling is ENABLED or ON.</p> <p>The default value is 2048 blocks. The minimum is zero (0) blocks and the maximum is 1073741823 (one less than 1 giga) blocks. In production, this value should typically be either zero (0) to disable journal extensions and rely entirely on the Journal Allocation, or it should be large. In UNIX, this value serves largely to allow you to monitor the rate of journal file growth.</p> <p>UNIX file systems use lazy allocations so this value controls the frequency at which GT.M checks the actual available space for journal file expansion in order to warn when space is low.</p>
Journal File	The name of the journal file. DSE only reports this field if journaling is ENABLED or ON.
Journal State	Indicates whether journaling is ON, OFF, or DISABLED (not allowed).
Journal Sync IO	<p>Indicates whether WRITE operation to a journal file commits directly to disk. The default value is FALSE.</p> <p>DSE only reports this field if journaling is ENABLED (or ON).</p>
Journal Yield Limit	<p>The number of times a process needing to flush journal buffer contents to disk yields its timeslice and waits for additional journal buffer content to be filled-in by concurrently active processes, before initiating a less than optimal I/O operation.</p> <p>The minimum Journal Yield Limit is 0, the maximum Journal Yield Limit is 2048.</p> <p>The default value for Journal Yield Limit is 8. On a lightly loaded system, a small value can improve run-time performance, but on actively updating systems a higher level typically provides the best performance.</p>
KILLs in progress	<p>The sum of the number of processes currently cleaning up after multi-block KILLs and the number of Abandoned KILLs.</p> <div style="display: flex; align-items: center; margin-top: 10px;">  <div style="border: 1px solid #ccc; padding: 5px; background-color: #f9f9f9;"> <p><b>Note</b></p> <p>Abandoned KILLs are associated with blocks incorrectly marked busy errors.</p> </div> </div>
Last Bytestream Backup	The transaction number of the last transaction backed up with the MUPIP BACKUP - BYTESTREAM command.

**GT.M Database Structure(GDS)**

<b>Data Elements</b>	<b>Description</b>
Last Database Backup	The transaction number of the last transaction backed up with the MUPIP BACKUP - DATABASE command. (Note -DATABASE is the default BACKUP type.)
Last Record Backup	Transaction number of last MUPIP BACKUP -RECORD or FREEZE -RECORD command.
Lock space	A hexadecimal number indicating the 512 byte pages of space dedicated to LOCK information.  The minimum Lock space is 10 pages and the maximum is 65,536 pages. The default is 40 pages. In production with an application that makes heavy use of LOCKs, this value should be higher.
Master Bitmap Size	The size of the Master Bitmap. The current Master Bitmap Size of V6 format database is 496 (512 byte blocks).
Maximum key size	The minimum key size is 3 bytes and the maximum key size is 1019 bytes. For information on setting the maximum key size for your application design, refer to “ <i>Global Directory Editor</i> ” (page 32).
Maximum record size	The minimum record size is zero. A record size of zero only allows a global variable node that does not have a value. The maximum is 1,048,576 bytes (1MiB). The default value is 256 bytes.  GT.M an error if you decrease and then make an attempt to update nodes with existing longer records.
Maximum TN	The maximum number of TNs that the current database can hold. For a database in V6 format, the default value of Maximum TN is 18,446,744,071,629,176,83 or 0xFFFFFFFF83FFFFFF.
Maximum TN Warn	The transaction number after which GT.M generate a warning and update it to a new value. The default value of Maximum TN Warn is 0xFFFFFFFF93FFFFFF.
Modified cache blocks	The current number of modified blocks in the buffer pool waiting to be written to the database.
Mutex Hard Spin Count	The number of attempts to grab the mutex lock before initiating a less CPU-intensive wait period. The default value is 128.
Mutex Sleep Spin Count	The number of timed attempts to grab the mutex lock before initiating a wait based on interprocess wake-up signals. The default value is 128.
Mutex Spin Sleep Time	The number of milliseconds to sleep during a mutex sleep attempt. The default value is 2048.
No. of writes/flush	The number of blocks to write in each flush. The default value is 7.
Null subscripts	"ALWAYS" if null subscripts are legal. "NEVER" if they are not legal and "EXISTING" if they can be accessed and updated, but not created anew.
Number of local maps	(Total blocks + 511)\512.
Online Backup NBB	Block to which online backup has progressed. DSE displays this only when an online backup is currently in progress.
Reference count	The number of GT.M processes and utilities currently accessing that segment on a given node

**GT.M Database Structure(GDS)**

<b>Data Elements</b>	<b>Description</b>
	<p>Note: GT.M does not rely on this field. A database segment initially has a reference count of zero. When a GT.M process or utility accesses a segment, GT.M increments the reference count. GT.M decrements the reference count upon termination.</p> <p>GT.M counts DSE as a process. When examining this field with DSE, the reference count is always greater than zero. When DSE is the only process using a region, the reference count should be one.</p>
Region Seqno	The current replication relative time stamp for a region.
Replication State	Either On or OFF. [WAS ON] OFF means that replication is still working, but a problem with journaling has caused GT.M to turn it off, so GT.M is still replicating, but will turn replication OFF if it ever has to turn to the journal because the pool has lost data needed for replication.
Reserved Bytes	Number of bytes reserved in database blocks.
Starting VBN	Virtual Block Number of the first GDS block after the GDS file header; this is block 0 of the database and always holds the first local bitmap.
Timers pending	Number of processes considering a timed flush.
Total blocks	Total number of GDS blocks, including local bitmaps.
Wait Disk	Seconds that GT.M waits for disk space to become available before it ceases trying to flush a GDS block's content to disk. During the wait, it sends eight (8) approximately evenly spaced operator log messages before finally issuing a GTM-E-WAITDSKSPACE error. For example, if Wait Disk is 80 seconds and GT.M finds no disk space to flush a GDS block, it sends a GTM-E-WAITDSKSPACE syslog message about every 10 seconds, and after the eighth message issues a WAITDSKSPACE error. This field is only used in UNIX because of its reliance on lazy disk space allocation.
Zqgblmod Seqno	The replication sequence number associated with the \$Zqgblmod() Transaction number.
Zqgblmod Trans	Transaction number used by the \$ZQGBLMOD() function in testing whether a block was modified by overlapping transactions during a replication switchover.
Average Blocks Read per 100 Records	Acts as a clue for replication update helper processes as to how aggressively they should attempt to prefetch blocks. It's an estimate of the number of database blocks that GT.M reads for every 100 update records. The default value is 200. For very large databases, you can increase the value up to 400.
Update Process Reserved Area	An approximate percentage (integer value 0 to 100) of the number of global buffers reserved for the update process. The reader helper processes leaves at least this percentage of the global buffers for the update process. It can have any integer value between 0 to 100. The default value is 50.
Pre read trigger factor	The percentage of Update Process reserved area after which the update process processes signals the reader helper processes to resume processing journal records and reading global variables into the global buffer cache. It can have any integer value between 0 to 100. The default value is 50.
Update writer trigger factor	One of the parameters used by GT.M to manage the database is the flush trigger. One of several conditions that triggers normal GT.M processes to initiate flushing dirty buffers from the database global buffer cache is when the number of dirty buffers crosses the flush trigger. In an attempt to never require the update process itself to flush dirty buffers, when the number of dirty global buffers crosses the update writer trigger factor percentage of the

Data Elements	Description
	flush trigger value, writer helper processes start flushing dirty buffers to disk. It can have any integer value between 0 to 100. The default value is 33, that is, 33%.

## MLOCK Space

The file header also provides M lock information, specifically whether any resource names belonging to the database region are locked. M code commonly uses locks as flags or semaphores controlling access to global data. Generally the LOCK argument specifies the resource of the same name as the name of the global variable that requires protected access. For more information about LOCKs, see the "M LOCK Utility" chapter and the LOCK section in the "Commands" chapter in the *GT.M Programmer's Guide*.

## Local Bitmaps

GT.M partitions GDS blocks into 512-block groups. The first block of each group contains a local bitmap. A local bitmap reports whether each of the 512 blocks is currently busy or free and whether it ever contained valid data that has since been KILLED.

The two bits for each block have the following meanings:

- 00 - Busy
- 01 - Free and never used before
- 10 - Currently not a legal combination
- 11 - Free but previously used

These two bits are internally represented as:

- 'X' - BUSY
- '.' - FREE
- '?' - CORRUPT
- ':' - REUSABLE

The interpreted form of the local bitmap is like the following: >

Block 0	Size 90	Level -1	TN 1	V5	Master Status:	Free Space
		Low order			High order	
Block 0:		XXXXX...	.....	.....	.....	
Block 20:		.....	.....	.....	.....	
Block 40:		.....	.....	.....	.....	
Block 60:		.....	.....	.....	.....	
Block 80:		.....	.....	.....	.....	
Block A0:		.....	.....	.....	.....	
Block C0:		.....	.....	.....	.....	
Block E0:		.....	.....	.....	.....	
Block 100:		.....	.....	.....	.....	
Block 120:		.....	.....	.....	.....	
Block 140:		.....	.....	.....	.....	
Block 160:		.....	.....	.....	.....	
Block 180:		.....	.....	.....	.....	

```
Block 1A0: | ..... |
Block 1C0: | ..... |
Block 1E0: | ..... |
'X' == BUSY '.' == FREE ':' == REUSABLE '?' == CORRUPT
```



**Note**

The first block described by the bitmap is itself and is, therefore, always marked busy.

If bitmaps marked as "?", they denote that they are corrupted (not currently in a legal combination) bitmaps. The consequences of corrupted bitmaps are:

Possible loss of data when GT.M overwrites a block that is incorrectly marked as free (malignant).

Reduction in the effective size of the database by the number of blocks incorrectly marked as busy (benign).

**Master Bitmap**

Using bitmaps, GT.M efficiently locates free space in the database. A master bitmap has one bit per local bitmap which indicates whether the corresponding local bitmap is full or has free space. When there is no free space in a group of 512 blocks, GT.M clears the associated bit in the master map to show whether the local bitmap is completely busy. Otherwise, GT.M maintains the bit set.

There is only one Master Bitmap per database. You can neither see the contents of the master bitmap directly or can change the size of the master bitmap. The maximum size of a GT.M database is over 7TB (terabytes, assuming 32K blocks).

The size of the master bitmap constrains the size of the database. The size of the master maps reflects current expectations for the maximum operational size of a single database file. Note: In addition to the limit imposed by the size of the master map, GT.M currently limits a tree to a maximum number of 7 levels. This means if a database holds only one global, depending on the density and size of the data, it might reach the level limit before the master map limit.

**Database Structure**

The GT.M database structure is hierarchical, based on a form of balanced tree called a B-star tree (B\*-tree) structure. The B\*-tree contains blocks that are either index or data blocks. An index block contains pointers used to locate data in data blocks, while the data blocks actually store the data. Each block contains a header and records. Each record contains a key and data.

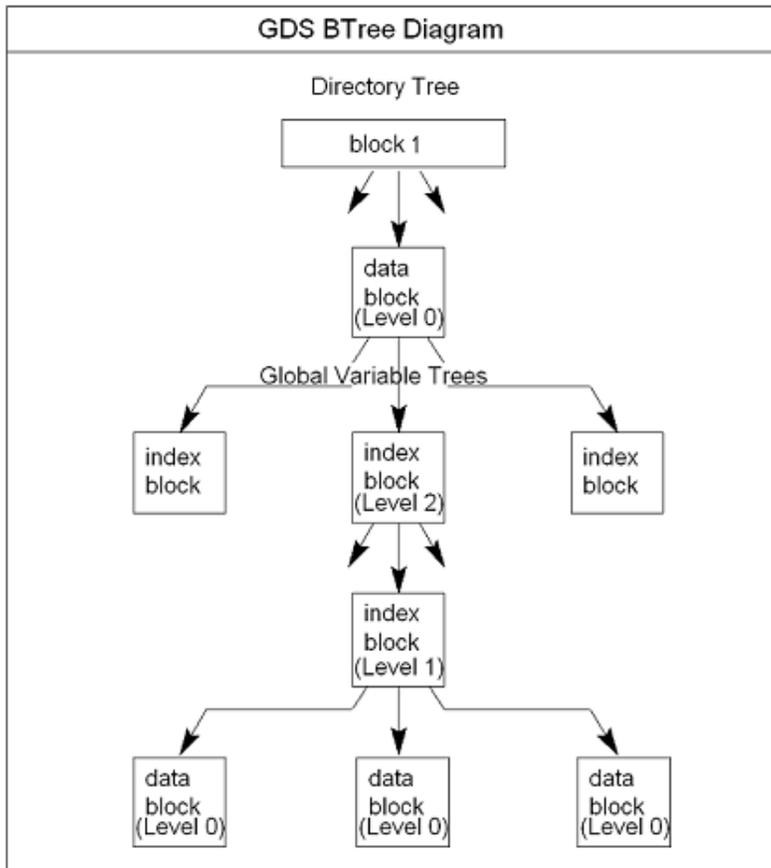
**Tree Organization**

GDS structures the data into multiple B\*-trees. GT.M creates a new B\*-tree, called a Global Variable Tree (GVT), each time the application defines a new named global variable. Each GVT stores the data for one named global, that is all global variables (gvn) that share the same unsubscripted global name. For example, global ^A, ^A(1), ^A(2), ^A("A"), and ^A("B") are stored in the same GVT. Note that each of these globals share the same unsubscripted global name, that is, ^A. A GVT contains both index and data blocks and can span several levels. The data blocks contain actual global variable values, while the index blocks point to the next level of block.

At the root of the B\*-tree structure is a special GDS tree called a Directory Tree (DT). DT contains pointers to the GVT. A data block in the DT contains an unsubscripted global variable name and a pointer to the root block of that global variable's GVT.

All GDS blocks in the trees have level numbers. Level zero (0) identifies the terminal nodes (that is, data blocks). Levels greater than zero (0) identify non-terminal nodes (that is, index blocks). The highest level of each tree identifies the root. All the B\*-trees have the same structure. Block one (1) of the database always holds the root block of the Directory Tree.

The following illustration describes the internal GDS B\*-tree framework GT.M uses to store globals.



GT.M creates a new GVT when a SET results in the first use of an unsubscripted global name by referring to a subscripted or unsubscripted global variable with a name prefix that has not previously appeared in the database.



### Important

GVTs continue to exist even after all nodes associated with their unsubscripted name are KILLED. An empty GVT occupies negligible space and does not affect GT.M performance. However, if you are facing performance issues because you have many empty GVTs, you need to reorganize your database file using MUPIP EXTRACT, followed by MUPIP CREATE, and the MUPIP LOAD to remove those empty GVTs.

The following sections describe the details of the database structures.

## GDS Blocks

Index and data blocks consist of a block header followed by a series of records. The block header has four fields that contain information. The first field, of two bytes, specifies the block version. The second field, of two bytes, specifies the number of bytes currently in use in the block. The third field, of one byte, specifies the block level. The last field of eight bytes represents the transaction number at which the block was last changed. An interpreted form of a block header looks like the following:

```
File      /home/jdoe/.fis-gtm/V6.0-000_x86_64/g/gtm.dat
Region   DEFAULT
```

Block 3   Size 262   Level 0   TN 3845EE V6

Depending on the platform, there may also be an empty field containing filler to produce proper alignment. The filler occurs between the second and third data field and causes the length of the header to increase from seven to eight bytes.

## GDS Records

Records consist of a record header, a key, and either a block pointer or the actual value of a global variable name (gvn). Records are also referred to as nodes.

The record header has two fields that contain information. The first field, of two bytes, specifies the record size. The second field, of one byte, specifies the compression count.



### Note

Like the GDS block headers, a filler byte may be added, depending on the platform.

The interpreted form of a block with global `^A("Name",1)="Brad"` looks like the following:

```
Rec:1 Blk 3 Off 10 Size 14 Cmpc 0 Key ^A("Name",1)
    10 : | 14 0 0 61 41 0 FF 4E 61 6D 65 0 BF 11 0 0 42 72 61 64|
        | . . . a A . . N a m e . . . . B r a d |
```

The data portion of a record in any index block consists of a four-byte block pointer. Level 0 data in the Directory Tree also consists of four-byte block pointers. Level 0 data in Global Variable Trees consists of the actual values for global variable names.

## Using GDS records to hold spanning nodes

A global variable node spans across multiple blocks if the size of its value exceeds one database block. Such a global variable node is called a "spanning node". For example, if `^a` holds a value that exceeds one database block, GT.M internally spans the value of `^a` in records with keys `^a(#SPAN1)`, `^a(#SPAN2)`, `^a(#SPAN3)`, `^a(#SPAN4)`, and so on. Note that `#SPAN1`, `#SPAN2`, `#SPAN3`, `#SPAN4`, and so on are special subscripts that are visible to the database but invisible at the M application level. GT.M uses these special subscripts to determine the sequence of the spanning nodes.

The first special subscript `#SPAN1` is called a "special index". A special index contains the details about the size of the spanning node's value and the number of additional records that are necessary to hold its value. `#SPAN2` and the rest of the records hold chunks of the value of the spanning node. During the load of a binary extract, GT.M uses these chunks to reconstitute the value of a global. This allows globals to be re-spanned if the block size of the source database is different from the block size of the destination database.



### Note

If the destination database's block size is large enough to hold the key and value, then the global is not a spanning node (because it can fit in one database block).

## GDS Keys

A key is an internal representation of a global variable name. A byte-by-byte comparison of two keys conforms to the collating sequence defined for global variable nodes. The default collating sequence is the one specified by the M standard. For more information on defining collating sequences, see the "Internationalization" chapter in the *GT.M Programmer's Guide*.

## Compression Count

The compression count specifies the number of bytes at the beginning of a key that are common to the previous key in the same block. The first key in each block has a compression count of zero. In a global variable tree, only the first record in a block can legitimately have a compression count of zero.

RECORD KEY	COMPRESSION COUNT	RESULTING KEY in Record
CUS(Jones, Tom)	0	CUS(Jones, Tom)
CUS(Jones, Vic)	10	Vic)
CUS(Jones, Sally)	10	Sally)
CUS(Smith, John)	4	Smith, John)

The previous table shows keys in M representation. For descriptions of the internal representations, refer to the section on keys.

The non-compressed part of the record key immediately follows the record header. The data portion of the record follows the key and is separated from the key by two null (ASCII 0) bytes.

## Use of Keys

GT.M locates records by finding the first key in a block lexically greater than, or equal to, the current key. If the block has a level of zero (0), the location is either that of the record in question, or, if the record in question does not exist, that of the (lexically) next record. If the block has a level greater than zero (0), the record contains a pointer to the next level to search.

GT.M does not require that the key in an index block correspond to an actual existing key at the next level.

The final record in each index block (the \*-record) contains a \*-key ("star-key"). The \*-key is a zero-length key representing the last possible value of the M collating sequence. The \*-key is the smallest possible record, consisting only of a record header and a block pointer, with a key size of zero (0).

The \*-key has the following characteristics:

- A record size of seven (7) or eight (8) bytes (depending on endian)
- A record header size of three (3) or four (4) bytes (depending on endian)
- A key size of zero (0) bytes
- A block pointer size of four (4) bytes

## Characteristics of Keys

Keys include a name portion and zero or more subscripts. GT.M formats subscripts differently for string and numeric values.

Keys in the Directory Tree represent unsubscripted global variable names. Unlike Global Variable Tree keys, Directory Tree keys never include subscripts.

Single null (ASCII 0) bytes separate the variable name and each of the subscripts. Two contiguous null bytes terminate keys. GT.M encodes string subscripts and numeric subscripts differently.

During a block split the system may generate index keys which include subscripts that are numeric in form but do not correspond to legal numeric values. These keys serve in index processing because they fall in an appropriate place in the collating sequence. When DSE represents these "illegal" numbers, it may display many zero digits for the subscript.

## Global Variable Names

The portion of the key corresponding to the name of the global variable holds an ASCII representation of the variable name excluding the caret symbol (^).

## String Subscripts

GT.M stores string subscripts as a variable length sequence of 8-bit codes ranging from 0 to 255. With UTF-8 specified at process startup, GT.M stores string subscripts as a variable length sequence of 8-bit codes with Unicode encoding.

To distinguish strings from numerics while preserving collation sequence, GT.M adds a byte containing hexadecimal **FF** to the front of all string subscripts. The interpreted form of the global variable **^A("Name",1)="Brad"** looks like the following:

```
Block 3   Size 24   Level 0   TN 1 V5
Rec:1 Blk 3 Off 10 Size 14 Cmpc 0 Key ^A("Name",1)
      10 : | 14 0 0 61 41 0 FF 4E 61 6D 65 0 BF 11 0 0 42 72 61 64|
          | . . . a A . . N a m e . . . . B r a d|
```

Note that hexadecimal **FF** is in front of the subscript "Name". GT.M permits the use of the full range of legal characters in keys. Therefore, a null (ASCII 0) is an acceptable character in a string. GT.M handles strings with embedded nulls by mapping **0x00** to **0x0101** and **0x01** to **0x0102**. GT.M treats **0x01** as an escape code. This resolves confusion when null is used in a key, and at the same time, maintains proper collating sequence. The following rules apply to character representation:

All codes except **00** and **01** represent the corresponding ASCII value.

**00** is a terminator.

**01** is an indicator to translate the next code using the following:

Code	Means	ASCII
01	00	<NUL>
02	01	<SOH>

With UTF-8 character-set specified, the interpreted output displays a dot character for all graphic characters and malformed characters. For example, the internal representation of the global variable **^DS=\$CHAR(\$\$FUNC^%HD("0905"))\_ \$ZCHAR(192)** looks like the following:

```
Rec:1 Blk 3 Off 10 Size C Cmpc 0 Key ^DS
      10 : | C 0 0 0 44 53 0 0 E0 A4 85 C0 |
          | . . . . D S . . ? . |
```

Note that DSE displays the wellformed character **?** for **\$CHAR(\$\$FUNC^%HD("0905"))** and a dot character for malformed character **\$ZCHAR(192)**.

With M character-set specified, the interpreted output displays a dot character for all non-ASCII characters and malformed characters.

## Numeric Subscripts

Numeric subscripts have the format:

```
[ sign bit ] [ biased exponent ] [ normalized mantissa ]
```

The sign bit and biased exponent together form the first byte of the numeric subscript. Bit seven (7) is the sign bit. Bits <6:0> comprise the exponent. The remaining bytes preceding the subscript terminator of one null (ASCII 0) byte represent the variable length mantissa. The following description shows a way of understanding how GT.M converts each numeric subscript type to its internal format:

Zero (0) subscript (special case)

- Represents zero as a single byte with the hexadecimal value 80 and requires no other conversion.

Mantissa

- Normalizes by adjusting the exponent.
- Creates packed-decimal representation.
- If number has an odd number of digits, appends zero (0) to mantissa.
- Adds one (1) to each byte in mantissa.

Exponent

- Stores exponent in first byte of subscript.
- Biases exponent by adding hexadecimal 3F.

The resulting exponent falls in the hexadecimal range 3F to 7D if positive, and zero (0) to 3E if negative.

Sign

- Sets exponent sign bit <7> in preparation for sign handling.
- If mantissa is negative: converts each byte of the subscript (including the exponent) to its one's-complement and appends a byte containing hexadecimal FF to the mantissa.

For example, the interpreted representation of the global `^NAME(.12,0,"STR",-34.56)` looks like the following:

```
Rec:1 Blk 5 Off 10 Size 1A Cmpc 0 Key ^NAME(.12,0,"STR",-34.56)
      10 : | 1A 0 0 61 4E 41 4D 45 0 BE 13 0 80 0 FF 53 54 52 0 3F|
          | . . . a N A M E . . . . . S T R . ?|
      24 : | CA A8 FF 0 0 31
          | . . . . . 1 |
```

Note that CA A8 ones complement representation is 35 57 and then when you subtract one (1) from each byte in the mantissa you get 34 56.

Similarly, the interpreted representation of `^NAME(.12,0,"STR",-34.567)` looks like the following:

```
Rec:1 Blk 5 Off 10 Size 1B Cmpc 0 Key ^NAME(.12,0,"STR",-34.567)
      10 : | 1B 0 0 9 4E 41 4D 45 0 BE 13 0 80 0 FF 53 54 52 0 3F|
```

### GT.M Database Structure(GDS)

```
      | . . . . N A M E . . . . . S T R . ?|
24 : | CA A8 8E FF 0 0 32 |
      | . . . . . 2 |
```

Note that since there are odd number of digits, GT.M appends zero (0) to mantissa and one (1) to each byte in mantissa.

---

## Chapter 10. Database Structure Editor

Revision History		
Revision V6.0-003	27 January 2014	Updated the output of DSE FUMP -FILEHEADER for V6.0-003.
Revision V6.0-001	27 February 2013	In CHange [287], added the description of -ABANDONED_KILLS, -STRM_NUM, -STRM_REG_SEQNO, and -KILL_IN_PROG.
Revision V6.0-000/1	21 November 2012	In CHange [287], added the description of -QBDRUNDOWN and updated the description of -CORRUPT_FILE qualifier for V6.0-000.
Revision V5.5-000/10	28 September 2012	In "ADD" [281], improved the description of -DATA.
Revision V5.5-000/8	03 August 2012	Added the description of the GVSTAT qualifier.
Revision V5.5-000/4	6 June 2012	In "Qualifiers of FIND" [306], corrected the description of -SIBLING and added an example.
Revision V5.5-000	27 February 2012	Added a caution note for changing the value of -CORRUPT_FILE fileheader element.
Revision 4	13 January 2012	In "Operating in DSE" [279], improved the description of the Change command.
Revision V5.4-002B	24 October 2011	Conversion to documentation revision history reflecting GT.M releases with revision history for each chapter.

---

### Operating in DSE

The GT.M Database Structure Editor, DSE, is primarily a tool for authorized GT.M consultants to examine and, under unusual circumstances, repair GT.M Database Structure (GDS) databases. With DSE, it is possible to see and change most of the attributes of a GT.M database.

DSE gives all possible control over a database and therefore, it may cause irreparable damage when used without knowing the consequences. Therefore, you unless you have extensive experience, you should always get guidance from FIS or an equivalently knowledgeable support resource before running any DSE command that changes any attribute of any production database or other database you value. However, you can use those DSE commands that let you see the attributes of your database for collecting database metrics and monitoring status.

GT.M installation procedure places the DSE utility program in a directory specified by the environment variable `gtm_dist`.

Invoke DSE using the "dse" command at the shell prompt. If this does not work, consult your system manager to investigate setup and file access issues.

Example:

```
$gtm_dist/dse
File/usr/name/mumps.dat
```

```
Region DEFAULT
DSE>
```

DSE displays the DSE> prompt.

You may also specify a command when entering DSE.

By default, DSE starts with the region that stands first in the list of regions arranged in alphabetical order. In the above example, the first region is DEFAULT.

You may also specify a command when entering DSE.

Example:

```
$gtm_dist/dse dump -fileheader
```

This command displays the fileheader of the region that stands first in the list of regions arranged in alphabetical order and then returns to the shell prompt. To look at other regions, at the DSE prompt you must first issue a FIND -REGION=<desired-region> command.

As previously mentioned, DSE provides control over most of the attributes of your database. With DSE, it is possible to examine them and, with a few exceptions, change them.

All DSE commands are divided into two categories—Change commands and Inquiry commands. Change commands allow you to modify the attribute of your database, in most cases without any warning or error. As the low level tool of last resort, Change commands allow you to take certain actions that can cause extensive damage when undertaken without an extensive understanding of the underlying data structures on disk and in memory and with an imperfect understanding of the commands issued. Do not use the Change commands unless you know exactly what you are doing and have taken steps to protect yourself against mistakes, both inadvertent and resulting from an incomplete understanding of the commands you issue. Change commands are not required for normal operation, and are usually only used under the direction of FIS support to recover from the unanticipated consequences of failures not adequately planned for (for example, you should configure GT.M applications such that you never need a Change command to recover from a system crash).

Inquiry commands let you see the attributes of your database. You may frequently use the inquiry commands for collecting your database metrics and status reporting.

The list of Change commands is as follows:

```
AD[D]
AL[L]
B[UFFER _FLUSH]
CH[ANGE]
CR[ITICAL]
REM[OVE]
RES[TORE]
SH[IFT]
W[CINIT]
OV[ERWRITE]
M[APS] -BU[SY] -F[REE] -M[ASTER] -R[ESTORE_ALL]
```

The list of Inquiry commands is as follows:

```
CL[OSE]
D[UMP]
EV[ALUATE]
EX[IT]
F[IND]
```

```
H[ELP]
I[NTEGRIT]
M[APS] -BL[OCK]
OP[EN]
P[AGE]
RA[NGE]
SA[VE]
SP[AWN]
```

Although DSE can operate concurrently with other processes that access the same database file, FIS strongly recommends using DSE in standalone mode when using Change commands. Some DSE operations can adversely impact the database when they occur during active use of the database. Other DSE operations may be difficult to perform in a logically sound fashion because a DSE operator works on a block at a time, while normal database operations update all related blocks almost simultaneously.



### Caution

When DSE attaches to a database with a version that does not match the DSE version, DSE issues an informational message and continues. At this point, you should exit DSE and find the version of DSE that matches the database. You should continue after this warning if and only if you are certain that the DSE is indeed from the GT.M version that has the database open (and hence the error results from a damaged database file header or shared memory that you intend to repair, following instructions from FIS).

Use the DSE EXIT, or QUIT command to leave DSE.

## DSE Commands and Qualifiers

The general format of DSE commands is:

```
command [-qualifier[...]] [object[,...]]
```

DSE interprets all numeric input as hexadecimal, except for time values, the values for the following qualifiers when used with CHANGE -FILEHEADER: -BLK\_SIZE=, DECLOCATION=, -KEY\_MAX\_SIZE=, -RECORD\_MAX\_SIZE, -REFERENCE\_COUNT=, -TIMERS\_PENDING and -WRITES\_PER\_FLUSH, and the value for -VERSION= when used with the REMOVE and RESTORE commands. These conventions correspond to the displays provided by DSE and by MUPIP INTEG.

## ADD

Adds a record to a block. The format of the ADD command for blocks with a level greater than zero (0) is:

```
ADD [-B[LOCK]=[block] {-OFFSET=offset|-RECORD=record} -STAR -POINTER=block
```

or

```
ADD [-B[LOCK]=[block] {-OFFSET=offset|-RECORD=record} -KEY=key -POINTER=pointer
```

The format of the ADD command for level 0 blocks is:

```
ADD [-B[LOCK]=[block] {-OFFSET=offset|-RECORD=record} -KEY=key -DATA=string
```

The ADD command requires either the -OFFSET or -RECORD qualifier to position the record in the block, and either the -KEY or the -STAR qualifier to define the key for the block.

The -STAR qualifier is invalid at level 0 (a data block). The ADD command requires the -DATA qualifier at level 0 or the -POINTER qualifier at any other level to provide record content.

## Qualifiers of ADD

*-B[LOCK]=block-number*

Specifies the block to receive the new record.

On commands with no `-BLOCK=` qualifier, DSE uses the last block handled by a DSE operation. When no block has been accessed, that is, on the first block-oriented command, DSE uses block one (1).

*-D[ATA]=string*

Specifies the data field for records added to a data block. Use quotation marks around the string and escape codes of the form `\a\b`, where "a" and "b" are hexadecimal digits representing non-printing characters. `\\` translates to a single backslash. `\'` translates to a NULL value.

Incompatible with: `-STAR`, `-POINTER`

*-K[EY]=key*

Specifies the key of the new record. Enclose M-style global references, including the leading caret symbol (^), in quotation marks (" ").

Incompatible with: `-STAR`

*-O[FFSET]=offset*

Adds the new record at the next record boundary after the specified offset.

Incompatible with: `-RECORD`, `-STAR`

*-P[OINTER]=pointer*

Specifies the block pointer field for records added to an index block. The `-POINTER` qualifier cannot be used at level 0. Note this means that to add pointers at level 0 of the Directory Tree you must specify a string of bytes or temporarily change the block level.

Incompatible with: `-DATA`

*-R[ECORD]=record-number*

Specifies a record number of the new record.

Incompatible with: `-OFFSET`, `-STAR`

*-S[TAR]*

Adds a star record (that is, a record that identifies the last record in an indexed block) at the end of the specified block. The `-STAR` qualifier cannot be used at level 0.

Incompatible with: `-DATA`, `-KEY`, `-OFFSET`, `-RECORD`

## Examples for ADD

```
DSE>add -block=6F -record=57 -key="^Capital("Mongolia")" -data="Ulan Bator"
```

## Database Structure Editor

This command adds a new record with key ^Capital("Mongolia") at the specified location. Note that this command is applicable to level 0 blocks only.

Example:

```
DSE>add -star -bl=59A3 -pointer=2
```

This command adds a star record in block 59A3. Note that this command is applicable to blocks > level 0.

Example:

```
DSE>add -block=3 -record=4 -key="^Fruits(4)" -data="Grapes"
```

Suppose your database has 3 global nodes -- ^Fruits(1)="Apple", ^Fruits(2)="Banana", and ^Fruits(3)="Cherry", then the above command adds a new node ^Fruits(4)="Grapes" at record 4. Note that this command is applicable to level 0 blocks only. The interpreted output as a result of the above command looks like the following:

```
Block 3   Size 4B   Level 0   TN 6 V5
Rec:1 Blk 3   Off 10   Size 14   Cmpc 0   Key ^Fruits(1)
      10 : | 14 0 0 0 46 72 75 69 74 73 0 BF 11 0 0 41 70 70 6C 65|
          | . . . . F r u i t s . . . . . A p p l e |
Rec:2 Blk 3   Off 24   Size D    Cmpc 8   Key ^Fruits(2)
      24 : | D 0 8 0 21 0 0 42 61 6E 61 6E 61 |
          | . . . . ! . . B a n a n a |
Rec:3 Blk 3   Off 31   Size D    Cmpc 8   Key ^Fruits(3)
      31 : | D 0 8 0 31 0 0 43 68 65 72 72 79 |
          | . . . . 1 . . C h e r r y |
Rec:4 Blk 3   Off 3E   Size D    Cmpc 8   Key ^Fruits(4)
      3E : | D 0 8 8 41 0 0 47 72 61 70 65 73 |
          | . . . . A . . G r a p e s |
```

Example:

```
$dse add -star -bl=1 -pointer=2
```

This command adds a star record in block 1. Note that this command is applicable to blocks > Level 0.

Example:

```
$ dse add -block=4 -key="^Vegetables" -pointer=7 -offset=10
```

This command creates a block with key ^Vegetables pointing to block 7.

Example:

```
DSE> add -record=2 -key="^foo" -data='\''
```

This example adds a new node (set ^foo="") as the second record of the current database block.

## ALL

Applies action(s) specified by a qualifier to all GDS regions defined by the current global directory.

The format of the ALL command is:

```
AL[L] -B[UFFER_FLUSH]
-C[RITINIT]
-[NO]F[REEZE]
-O[VERRIDE]]
-REF[ERENCE]
-REL[EASE]
-REN[EW]
-S[EIZE]
-W[CINIT]
```

- This is a very powerful command; use it with caution.
- Be especially careful if you have an overlapping database structure (for example, overlapping regions accessed from separate application global directories).
- If you use this type of database structure, you may need to construct special Global Directories that exclude overlapped regions to use with DSE.

## Qualifiers

*-BUFFER\_FLUSH*

Flushes to disk the file header and all pooled buffers for all regions of the current global directory.

Incompatible with: -RENEW

*-C[RITINIT]*

Initializes critical sections for all regions of the current directory.

Incompatible with: -RENEW, -RELEASE, -SIEZE



### Caution

Never use CRITINIT while concurrent updates are in progress as doing so may damage the database.

*-[NO]F[REEZE]*

Freezes or prevents updates all regions of the current global directory.

- The FREEZE qualifier freezes all GDS regions except those previously frozen by another process . Regions frozen by a particular process are associated with that process .
- A frozen region may be unfrozen for updates in one of two ways: The process which froze the region may unfreeze it with the -NOFREEZE qualifier; or another process may override the freeze in conjunction with the -OVERRIDE qualifier. For more information on a preferred method of manipulating FREEZE, refer to “FREEZE ” (page 86).
- By default, the -NOFREEZE qualifier unfreezes only those GDS regions that were previously frozen by a process . Once a region is unfrozen, it may be updated by any process . To unfreeze all GDS regions of the Global Directory, use the -OVERRIDE qualifier.

## Database Structure Editor

- DSE releases any FREEZE it holds when it exits, therefore, use the same DSE invocation or SPAWN to perform operations after executing the ALL -FREEZE command.

Incompatible with: -RENEW

*-O[VERRIDE]*

Overrides the ALL -FREEZE or ALL -NOFREEZE operation.

When used with -NOFREEZE, -OVERRIDE unfreezes all GDS regions, including those frozen by other users.

When used with -FREEZE, -OVERRIDE freezes all GDS regions, including those frozen by other processes associating all such freezes with the current process. The current process must then use -NOFREEZE to unfreeze the database; any other process attempting a -NOFREEZE should also have to include the -OVERRIDE qualifier.

Meaningful only with: [NO]FREEZE

*-REF[ERENCE]*

Resets the reference count field to 1 for all regions of the current global directory.

- A Reference count is a file header element field that tracks how many processes are accessing the database with read/write permissions.
- This qualifier is intended for use when DSE is the only process attached to the databases of the current global directory. Using it when there are other users attached produces an incorrect value.

Incompatible with: -RENEW

*-REL[EASE]*

Releases critical sections for all regions of the current global directory.

Incompatible with: -CRITINIT, -RENEW, -SEIZE

*-REN[EW]*

Reinitializes the critical sections (-CRITICAL) and buffers (-WCINIT), resets reference counts (-REFERENCE\_COUNT) to 1, and clears freeze (-NOFREEZE) for all regions of the current global directory .

- -RENEW requires confirmation.
- The RENEW action will cause all current accessors of the affected database regions to receive a fatal error on their next access attempt.
- This operation is dangerous, drastic, and a last resort if multiple database have hangs that have not yielded to other resolution attempts; there is almost never a good reason to use this option.

*-S[EIZE]*

Seizes the critical section for all regions of the current global directory. The -SEIZE qualifier is useful when you encounter a DSEBLKRDFAIL error, generated when DSE is unable to read a block from the database.

Incompatible with: -RENEW, -RELEASE, -CRITINIT

*-W[CINIT]*

Reinitializes the buffers for all regions of the current global directory.

-WCINIT requires confirmation.



### Caution

This operation is likely to cause database damage when used while concurrent updates are in progress.

Incompatible with: -RENEW

## Examples of ADD

Example:

```
DSE> all flush -buffer_flush
```

This command flushes the file header and cache buffers to disk for all regions.

Example:

```
DSE> ALL -CRITINIT
```

This command initializes critical sections for all regions of the current directory.

Example:

```
DSE> ALL -FREEZE
DSE> SPAWN "mumps -dir"
```

The first command freezes all regions of the current global directory. The second command creates an child (shell) process and executes the "mumps -dir" command. Then type S ^A=1 at GTM prompt. Notice that the command hangs because of the DSE FREEZE in place.

Example:

```
DSE> ALL -NOFREEZE -OVERRIDE
```

This command removes the FREEZE on all current region including the FREEZE placed by other users.

Example:

```
DSE> ALL -REFERENCE
```

This command sets the reference count field in the file header(s) to 1.

Example:

```
DSE> ALL -RELEASE
```

This command releases critical sections owned by the current process for all regions of the current global directory.

Example:

```
DSE> ALL -RENEW
```

This command reinitializes critical sections, buffers, resets the reference count to 1, and clears freeze for all regions of the current global directory.

Example:

```
DSE> ALL -SEIZE
```

This command seizes all critical sections for all regions of the current global directory.

Example:

```
DSE> WCINIT
```

This command reinitializes the buffers for all regions of the current global directory.

## Buffer\_flush

Flushes the file header and the current region's buffers to disk.

The format of the BUFFER\_FLUSH command is:

```
B[UFFER_FLUSH]
```

The BUFFER\_FLUSH command has no qualifiers.

## CHange

The CHANGE command changes fields of a block, file, or record header.

The format of the CHANGE command is:

```
CH[ANGE]
```

The CHANGE command either has a -FILEHEADER qualifier or an implicit or explicit -BLOCK qualifier, plus one or more of their associated qualifiers, to define the target of the change.

-BL[OCK]=block-number and one or more of the following qualifiers:

```
-BS[IZ]=block-size
-L[EVEL]=level
-TN[=transaction-number]
-OF[FSET]=offset
-RE[CORD]=record-number
-CM[PC]=compression-count
-RS[IZ]=record-size
```

or

-F[ILEHEADER] and one or more of the following qualifiers:

```
-AB[ANDONED_KILLS]=value
```

```

-AVG_BLK_READ=Average-blocks-read
-B_B[YTESTREAM]=transaction-number
-B_C[OMPREHENSIVE]=transaction-number
-B_D[ATABASE]=transaction-number
-B_I[NCREMENTAL]=transaction-number
-B_R[ECORD]=transaction-number
-BLK_SIZE=block-size
-BLO[CKS_FREE]=free-blocks
-CU[RRENT_TN]=transaction-number
-COM[MITWAIT_SPIN_COUNT]=boolean
-DEC[LOCATION]=value
-DEF[_COLLATION]=value
-ENCRYPTION_HASH
-FL[USH_TIME][=delta-time]
-FR[EEZE]=value
-FULLY_UPGRADED=boolean
-GV[STATSRESET]
-HARD_SPIN_CPUNT=Mutex-hard-spin-sount
-[HEXLOCATION]=value
-INT[ERRUPTED_RECOV]=boolean
-JNL_YIELD_LIMIT=journal-yeild-limit
-KEY_MAX_SIZE=key-max-size
-KILL_IN_PROG=value
-M[ACHINE_NAM]=value
-N[ULL_SUBSCRIPTS]=value
-NO[CRIT]
-OV[ERRIDE]
-Q[DBRUNDOWN]
-RC_SRV_COUNT
-RE_READ_TRIGGER=read-trigger
-REC[ORD_MAX_SIZE]=record-max-size
-REF[ERENCE_COUNT]=reference-count
-REG[_SEQNO]=sequence-number
-RESERVED_BYTES=reserved-bytes
-SLEEP_SPIN_COUNT=mutex-sleep-spin-count
-SPIN_SLEEP_TIME=mutex-sleep-time
-STRM_NUM=stream-number STRM_REG_SEQNO=hexa
-TIM[ERS_PENDING]=integer
-TOTAL_BLKs]=total-blocks
-TR[IGGER_FLUSH]=trigger-flus
-UPD_RESERVED_AREA=reserved-area
-UPD_WRITER_TRIGGER_FACTOR=trigger-factor
-W[RITES_PER_FLUSH]=writes-per-flush
-WAIT_DISK=wait-disk
-Zqgblmod_S[EQNO]=sequence-number
-Zqgblmod_T[rans]=sequence-number

```

## CHANGE -BLOCK Qualifiers

This section describes -BLOCK and all of its qualifiers.

*-BL[OCK]=block\_number*

Specifies the block to modify. The -BLOCK qualifier is incompatible with the -FILEHEADER qualifier and all qualifiers related to -FILEHEADER.

-BLOCK is the default qualifier. On commands with neither a -BLOCK nor a -FILEHEADER qualifier, DSE uses the last block handled by a DSE operation. When no block has been accessed, that is, on the first block-oriented command, DSE uses block one (1).

Incompatible with: -FILEHEADER and qualifiers used with -FILEHEADER

The following qualifiers operate on a block header.

*-BS[IZ]=block\_size*

Changes the block size field of the specified block.

- block\_size is in hexadecimal form.
- Decreasing the block size can result in loss of existing data.



**Note**

The block size must always be less than or equal to the block size in the file header.

Use only with: -BLOCK, -LEVEL, -TN

*-L[EVEL]=level*

Changes the level field for the specified block.



**Note**

DSE lets you change the level of a bitmap block to -1 (the value of the level for a bitmap block) when the bitmap level gets corrupted and takes on an arbitrary value. Note that you should specify -1 in hexadecimal form, that is, FF.

Use only with: -BLOCK, -BSIZ, -TN

Example:

```
DSE >change -level=FF
```

*-TN[=transaction\_number]*

Changes the transaction number for the current block.

- When a CHANGE command does not include a -TN=, DSE sets the transaction number to the current transaction number.
- Manipulation of the block transaction number affects MUPIP BACKUP -BYTESTREAM, and -ONLINE.

Use only with: -BLOCK, -BSIZ, -LEVEL

*-OF[FSET]=offset*

Specifies the offset, in bytes, of the target record within the block. If the offset does not point to the beginning of a record, DSE rounds down to the last valid record start (for example, CHANGE -OFFSET=10 starts at -OFFSET=A, if that was the last record).

Use only with: -BLOCK, -CMPC, and -RSIZ.

*-RE[CORD]=record\_number*

Specifies the record number of the target record.

Use only with: -BLOCK, -CMPC, and -RSIZ.

*-CM[PC]=compression\_count*

Change the compression count field of the specified record.

- The compression count specifies the number of bytes at the beginning of a key that are common to the previous key in the same block.
- Because compression counts propagate from the "front" of the block, this can potentially change the keys of all records following it in the block. If the goal is to change only a single record, it may be preferable to add a new record and remove the old one.

Use only with: -BLOCK, -RECORD, -OFFSET, -RSIZE

*-RS[IZ]=record\_size*

Changes the record size field of the specified record.



### Caution

Changing -RSIZ impacts all records following it in the block.

Use only with: -BLOCK, -RECORD, -CMPC, -OFFSET

Example:

```
DSE> change -record=3 -rsiz=3B -block=2
```

This command changes the record size of record 3 block 2 to 59 (Hex: 3B) bytes.

## CHANGE -Fileheader Qualifiers

This section describes the -FILEHEADER qualifier and the other qualifiers that operate on a file header.

*-FI[LEHEADER]*

Modifies a file header element that you specify with an associated qualifier.

Incompatible with: -BSIZ, -CMPC, -TN, -LEVEL, -OFFSET, -RECORD, -RSIZ

*-AB[ANDONED\_KILLS]=value*

Changes the value of the Abandoned Kills field. The value can be "NONE" or a positive integer.

Use only with: -FILEHEADER; decimal

*-BLK[\_SIZE]=block\_size*

Changes the decimal block size field of the current file.

## Database Structure Editor

- DSE does not allow you to change the block size to any arbitrary value. It always rounds the block size to the next higher multiple of 512.
- Use the CHANGE -BLK\_SIZE qualifier only upon receiving instructions from FIS and only in conjunction with the -FILEHEADER qualifier. This DSE command cannot change the working block size of a database and is useful only under very limited and extraordinary circumstances. If you need to change the block size on a database file, unload the data with MUPIP EXTRACT (or an appropriate alternative), change the global directory with GDE to specify the new block size, recreate the database with MUPIP CREATE and reload the data with MUPIP LOAD (or appropriate alternative).

Use only with: -FILEHEADER

*-BLO[CKS\_FREE]=free blocks*

Changes the free blocks field of the current file.

Use this to correct a value that MUPIP INTEG reports as needing a correction, but note that the "correct" value reported by INTEG may go out-of-date with the next update. It may be necessary to calculate a delta value from the INTEG report, FREEZE the region with DSE, DUMP the current -FILEHEADER value, then apply the delta and CHANGE the -BLOCKS\_FREE, and finally turn -OFF the FREEZE.

Use only with: -FILEHEADER

*-B[YTESTREAM]=transaction\_number*

Changes the transaction number in the file header of the last incremental backup to the value specified. Use this qualifier only in conjunction with the -FILEHEADER qualifier. For compatibility issues with prior versions, this can still be specified as -B\_COMPREHENSIVE.

*-D[ATABASE]=transaction\_number*

Changes the transaction number in the file header of the last comprehensive backup to the value specified. Use this qualifier only in conjunction with the -FILEHEADER qualifier. For compatibility issues with prior versions, this can still be specified as -B\_COMPREHENSIVE.

*-B\_R[ECORD]=transaction\_number*

Changes the transaction number in the file header field that maintains this information about the last -RECORD backup.

*-CO[RRUPT\_FILE]=boolean*

Indicates whether or not a region completed a successful recovery with the MUPIP JOURNAL -RECOVER command. Possible values are: T[RUE] or F[ALSE].

Changing this flag does not correct or cause database damage. When CORRUPT\_FILE is set to TRUE, the DSE DUMP command displays a message like the following:

```
%GTM-W-DBFLCORRP, /home/gtmnode1/mumps.dat Header indicates database file is corrupt
```



### Caution

After a CHANGE -FILEHEADER -CORRUPT=TRUE, the file is unavailable to future GT.M access other than DSE. Under normal conditions, there should never be a need to change this flag manually. A MUPIP SET -PARTIAL\_BYPASS\_RECOV sets this flag to false.

Use only with: -FILEHEADER

*-COM[MITWAIT\_SPIN\_COUNT]=value*

Specifies the number of times a GT.M process waiting for control of a block to complete a block update should spin before yielding the CPU when GT.M runs on SMP machines. When run on a uniprocessor system, GT.M ignores this parameter. On SMP systems, when a process needs a critical section that another process has, if critical sections are short (as they are by design in GT.M), spinning a little with the expectation that the process with the critical section will release it shortly provides a way to enhance performance at the cost of increased CPU usage. Eventually, a process awaiting a critical section yields the CPU if spinning for a little does not get it the needed critical section. Note that on heavily loaded systems, increasing COMMITWAIT\_SPIN\_COUNT may not trade off CPU for throughput, but may instead degrade both. If you set the COMMITWAIT\_SPIN\_COUNT to 0, the waiting process performs a sequence of small sleeps instead of the spins or yields.

The default value is 16.

Use only with: -FILEHEADER

*-CU[RRENT\_TN]=transaction\_number*

Changes the current transaction number for the current region.

- Raising the -CURRENT\_TN can correct "block transaction number too large" errors
- This qualifier has implications for MUPIP BACKUP -INCREMENTAL and -ONLINE.
- Used with the -BLOCK qualifier, CURRENT\_TN places a transaction number in a block header.

Use only with: -FILEHEADER

*-DECLOCATION*

Specifies an offset with the file header. If -VALUE is specified, GT.M puts it at that location.

Use only with: -FILEHEADER; decimal

*-E[NCRYPTION\_HASH]*

Changes the hash of the password stored in the database file header if and when you change the hash library. For more information on key management and reference implementation, refer to Chapter 12: "Database Encryption" (page 363).



### Caution

An incorrect hash renders the database useless.

Use only with: -FILEHEADER

*-FL[USH\_TIME][=delta\_time]*

Changes the flush\_time default interval (in delta\_time).

- The time entered must be between zero and one hour. Input is interpreted as decimal.
- A -FLUSH\_TIME with no value resets the -FLUSH\_TIME to the default value (one second for BG and 30 seconds for MM).

## Database Structure Editor

- The units of delta\_time are hours:minutes:seconds:centi-seconds (hundredths of a second). For example, to change the flush time interval to a second, delta\_time would be 00:00:01:00. To change it to 30 minutes, delta\_time would be 00:30:00:00. Valid values for the qualifier are one centi-second to one hour.

Use only with: -FILEHEADER

*-FR[EEZE]=value*

Sets availability of the region for update. Possible values are: T[RUE] or F[ALSE]. Use to "freeze" (disable database writes) or "unfreeze" the database.

Use only with: -FILEHEADER

For information about a preferred method of manipulating FREEZE, refer to "FREEZE" (page 86) of the General Database Management chapter.

DSE releases -FREEZE when it EXITS. To hold the database(s), CHANGE -FILEHEADER -FREEZE=TRUE and then SPAWN to perform other operations.

*-FU[LLY\_UPGRADED]=boolean*

Sets a flag that indicates whether or not the database was fully upgraded from V4 to V5 database format.. The value is either T[RUE] or F[ALSE].

Use only with: -FILEHEADER

*-GV[STATSRESET]*

Resets all the database file header global access statistics to 0. Note that this erases all statistics previously accumulated in the database file header.

Use only with: -FILEHEADER

*-HEXLOCATION*

Specifies an offset with the file header. If -VALUE is specified, GT.M puts it at that location.

Use only with: -FILEHEADER; hexadecimal

*-INT[ERRUPTED\_RECOV]=boolean*

Sets a flag that indicates whether or not a recovery with the MUPIP JOURNAL -RECOVER command was interrupted. The value is either T[RUE] or F[ALSE].

Use only with: -FILEHEADER

*-K[EY\_MAX\_SIZE]=key\_max\_size*

Changes the decimal value for the maximum allowable key size. Reducing KEY\_MAX\_SIZE can restrict access to existing data and cause GT.M to report errors. Do not create incompatible key and record sizes.

Before permanently changing the key size using DSE, use GDE to check that the appropriate Global Directory contains the same key size for the region. This prepares for future MUPIP CREATEs and performs a consistency check on the key and record size values. For more information on key and record sizes, refer to Chapter 4: "Global Directory Editor" (page 32).

Use only with: -FILEHEADER; decimal

*-KI[LL\_IN\_PROG]=value*

Changes the value of the KILLS in progress field. The value can be "NONE" or a positive integer.

Use only with: -FILEHEADER; decimal

*-N[ULL\_SUBSCRIPTS]=value*

Controls whether GT.M accepts null subscripts in database keys.

- value can either be T[RUE], F[ALSE], ALWAYS, NEVER, or EXISTING. See GDE chapter for more information on these values of null\_subscript.
- Prohibiting null subscripts can restrict access to existing data and cause GT.M to report errors.
- The default value is never.
- DSE cannot change the null subscript collation order. Instead, use GDE to change the null subscript collation order, MUPIP EXTRACT the current content, MUPIP CREATE the database file(s) with the updated collation and MUPIP LOAD the content.

Use only with: -FILEHEADER

*-OV[ERRIDE]*

Releases or "steals" a FREEZE owned by another process.

Use only with: -FREEZE

*-[NO]Q[DBRUNDOWN]*

Sets a flag that indicates whether or not the database is enabled for quick rundown. The default value is -NOQDBRUNDOWN.

For more information, refer to "Region Qualifiers" (page 54).

*-REC[ORD\_MAX\_SIZE]=record\_max\_size*

Changes the decimal value for the maximum allowable record size. Use the -RECORD\_MAX\_SIZE qualifier only in conjunction with the -FILEHEADER qualifier. Reducing RECORD\_MAX\_SIZE can restrict access to existing data and cause GT.M to report errors. Do not create incompatible key and record sizes.

Before making a permanent change to the records size using DSE, use GDE to check that the appropriate Global Directory contains the same record size for the region. This prepares for future MUPIP CREATEs and performs a consistency check on the key and record size values. For more information on key and record sizes, refer to Chapter 4: "Global Directory Editor" (page 32).

*-REF[ERENCE\_COUNT]=reference\_count*

Sets a field that tracks how many processes are accessing the database with read/write permissions. MUPIP INTEG and DSE use decimal numbers for -REFERENCE\_COUNT. To accurately determine the proper reference count, restrict CHANGE - FILEHEADER -REFERENCE\_COUNT to the case where the process running DSE has exclusive (standalone) access to the database file. When DSE has sole access to a database file the -REFERENCE\_COUNT should be one (1). This is an informational field and does not have any effect on processing.

*-REG[\_SEQNO]=sequence-number*

In an LMS environment, this sets the "Region Seqno" field. For more information, refer to Chapter 7: "Database Replication" (page 173).

*-RESYNC\_S[EQNO]=sequence-number*

In an LMS environment, this sets the "Resync Seqno" field. For more information, refer to Chapter 7: "Database Replication" (page 173).

*-RESYNC\_T[N]=sequence-number*

In an LMS environment, this sets the "Resync transaction" field. For more information, refer to Chapter 7: "Database Replication" (page 173).

*-STRM\_NUM=stream-number -STRM\_R[EG\_SEQNO]=str\_num's\_region\_sequence\_number*

Changes the Stream and its Reg Seqno. Use *-STRM\_NUM* and *-STRM\_REG\_SEQNO* together as part of the same *CHANGE -FILEHEADER* command.

Use only with: *-FILEHEADER*; hexa

*-TI[MERS\_PENDING]=timers\_pending*

*-TI[MERS\_PENDING]=timers\_pending*

Sets a field that tracks the number of processes considering a timed flush. Proper values are 0, 1, and 2.

Use the *CHANGE -TIMERS\_PENDING* qualifier only upon receiving instructions from FIS.

Use only with: *-FILEHEADER*; decimal

*-TO[TAL\_BLKs]=total\_blocks*

Changes the total blocks field of the current file. Use only with: *-FILEHEADER*; decimal



### Caution

The total blocks field should always reflect the actual size of the database. Change this field only if it no longer reflects the database size.

*-TR[IGGER\_FLUSH]=trigger\_flush*

Sets the decimal value for the triggering threshold, in buffers, for flushing the cache-modified queue.

Use the *CHANGE -TRIGGER\_FLUSH* qualifier only upon receiving instructions from FIS, and only in conjunction with the *-FILEHEADER* qualifier.

*-WR[ITES\_PER\_FLUSH]=writes\_per\_flush*

Set the decimal number of block to write in each flush. The default value is 7.

Use only with -FILEHEADER

## Examples for CHANGE

Example:

```
DSE> change -block=3 -bsiz=400
```

This command changes the size of block 3 to 1024 bytes.

Example:

```
DSE> change -block=4 -tn=10000
```

This command changes sets the transaction number to 65536 (Hex: 10000) for block 4.

Example:

```
DSE> change -block=2 -record=4 -CMPC=10 -key="^CUS("Jones,Vic")"
```

This command changes the compression count of the key ^CUS(Jones,Vic) to 10. It is assumed that the key CUS(Jones, Tom) already exists. The following table illustrates how GT.M calculates the value of CMPC in this case.

RECORD KEY	COMPRESSION COUNT	RESULTING KEY in Record
CUS(Jones, Tom)	0	CUS(Jones, Tom)
CUS(Jones, Vic)	10	Vic)
CUS(Jones, Sally)	10	Sally)
CUS(Smith, John)	4	Smith, John)

Example:

```
DSE> dump -fileheader
```

This command displays fields of the file header.

Example:

```
DSE> change -fileheader -blk_siz=2048
```

This command changes the block size field of the fileheader to 2048 bytes. The block field must always be a multiples of 512 bytes.

Example:

```
DSE> change -fileheader -blocks_free=5B
```

This command changes the blocks free fields of the file header to 91 (Hex: 5B). Example:

Example:

```
DSE> change -fileheader -b_record=FF
```

This command sets the RECORD backup transaction to FF.

Example:

```
DSE> change -fileheader corrupt_file=FALSE
```

This command sets the CORRUPT\_FILE field to false.

Example:

```
DSE> change -fileheader -current_tn=1001D1BF817
```

This command changes the current transaction number to 1100000000023 (Hex: 1001D1BF817). After you execute this command, subsequent transaction numbers will be greater than 1001D1BF817.

Example:

```
DSE> change -fileheader -flush_time=00:00:02:00
```

This command changes the flush time field of the file header to 2 seconds.

Example:

```
DSE> change -fileheader -freeze=true
```

This command makes the default region unavailable for updates.

Example:

```
DSE> change -fileheader -key_max_size=20
```

This command changes the maximum key size to 20. Note that the default max key size is 64.

Example:

```
DSE> CHANGE -FILEHEADER -NULL_SUBSCRIPTS="EXISTING"
```

This command changes the Null Subscripts field of the file header to EXISTING. Note that DSE cannot change the null subscript collation order. See GDE chapter for more information on changing the null subscript collation.

Example:

```
DSE> change -fileheader -reserved_bytes=8 -record_max_size=496
```

This command sets the maximum record size as 496 for the default region.

Example:

```
DSE> change -fileheader -reference_count=5
```

This command sets the reference count field of the file header to 5.

Example:

```
DSE> change -fileheader -timers_pending=2
```

This command sets the timers pending field of the file header to 2.

Example:

```
DSE> change -fileheader -TOTAL_BLKs=64
```

This command sets the total size of the database to 100 (Hex: 64) blocks.

Example:

```
DSE> change -fileheader -trigger_flush=1000
```

This command sets the Flush Trigger field of the file header to 1000. Note the default value of Flush Trigger is 960.

Example:

```
DSE> change -fileheader -writes_per_flush=10
```

This command changes the number of writes/flush field of the file header to 10. Note that the default value for the number of writes/flush is 7.

Example:

```
DSE> change -fileheader -zqblmod_seqno=FF
```

This command changes the ZGBLMOD\_SEQNO field to 255(Hex: FF).

## CAche

Operates on the cache of a database having BG access method. The format of the CACHE command is:

```
CA[CHE] -ALL  
-RE[COVER  
-VE[RIFY]
```

### Qualifiers of CACHE

*-RE[COVER] [-ALL]*

Resets the cache of a database having BG access method to a "clean" state.

- With -ALL specified, DSE includes all region of the current global directory for cache recovery.
- Attempt DSE -RECOVER only if a DSE CACHE -VERIFY commands reports the cache is "NOT clean".

*-VE[RIFY] [-ALL]*

Verifies the integrity of the cache data structures as well as the internal consistency of any GDS blocks in the global buffers of the current region.

- With -ALL specified, DSE performs cache verification on all regions of the current global directory.

- It reports the time, the region and a boolean result indicating whether the cache is clean or NOT clean. If you see "NOT clean" in report, execute DSE CACHE -RECOVER as soon as possible to reset the cache in a clean state.

## Examples for CACHE

Example:

```
DSE> CACHE -VERIFY
```

This command checks the integrity of the cache data structures as well as the internal consistency of GDS blocks in the global buffers of the current region.

Example:

```
DSE> CACHE -VERIFY -ALL  
Time 26-FEB-2011 14:31:30 : Region DEFAULT : Cache verification is clean
```

Execute CACHE recover command if Cache verification is "NOT" clean.

This command reports the state of database cache for all regions.

Example:

```
DSE> CACHE -RECOVER
```

This command reinitializes the cache data structures of the current region and reverts the cache of a database having BG access to "clean" state.

## CLOSE

The CLOSE command closes the currently open output file.

The format of the CLOSE command is:

```
CL[OSE]
```

The CLOSE command has no qualifiers.

## CRITICAL

Displays and/or modifies the status and contents of the critical section for the current region. The format of the CRITICAL command is:

```
CR[ITICAL] -I[NIT]  
-O[WNER]  
-REL[EASE]  
-REM[OVE]  
-RES[ET]  
-S[EIZE]
```

- The critical section field identifies, by its process identification number (PID), the process presently managing updates to database.

- Think of a critical section as a common segment of a train track. Just as a train moves through the common segment as quickly as possible, the same way a process moves as quickly as possible through any critical section so that other processes can use it.
- By default, the CRITICAL command assumes the -OWNER qualifier, which displays the status of the critical section.

## Qualifiers of CRITICAL

*-I[NIT]*

Reinitializes the critical section.

- The -INIT and -RESET qualifiers together cause all GT.M processes actively accessing that database file to signal an error.
- FIS recommends against using -INIT without the -RESET parameter when other processes are actively accessing the region because it risks damaging the database.

Use only with: -RESET

*-O[WNER]*

Displays the ID of the process at the head of the critical section. DSE displays a warning message when the current process owns the critical section.

Use alone

Example:

```
DSE> critical -OWNER
Write critical section is currently unowned
```

*-REL[EASE]*

Releases the critical section if the process running DSE owns the section.

Use alone.

*-REM[OVE]*

Terminates any write ownership of the critical section. Use this when the critical section is owned by a process that is nonexistent or is known to no longer be running a GT.M image.

Use alone.



### Caution

Using CRITICAL -REMOVE when the write owner of a critical section is an active GT.M process may cause structural database damage.

*-RES[ET]*

Displays the number of times the critical section has been through an online reinitialization.

Using -RESET with -INIT causes an error for processes that are attempting to get the critical section of the region. Under the guidance of FIS, use -RESET -INIT as a way to clear certain types of hangs.

Use only with: -INIT

*-S[EIZE]*

Seizes the critical section (if available).

- You can also use SEIZE to temporarily suspend database updates.
- Subsequently, execute CRITICAL -RELEASE command to restore normal operation.

## Examples for CRITICAL

Example:

```
DSE> critical -OWNER Write critical section owner is process id 4220
```

This command displays the ID of the process holding the critical section. Note that on catching a process ID on a lightly loaded (or unloaded) system (for example, text environment) is like catching lightning in a bottle. Therefore, you can artificially hold a critical section using the DSE CRIT -SEIZE command in one session and view the owner using a different session.

## Dump

Displays blocks, records, or file headers. DUMP is one of the primary DSE examination commands.

The format of the DUMP command is:

```
D[UMP] -A[LL]
-B[LOCK]=block_number
-C[OUNT]=count
-F[ILEHEADER]
-G[LO]
-G[VSTATS]
-[NO]C[RIT]
-[NO]H[EADER]
-O[FFSET]=offset
-R[ECORD]=record-number
-U[PDPROC]
-Z[WR]
```

Use the error messages reported by MUPIP INTEG to determine what to DUMP and examine in the database. DUMP also can transfer records to a sequential file for future study and/or for input to MUPIP LOAD (see the section on OPEN). The DUMP command requires specification of an object using either -BLOCK, -HEADER, -RECORD, or -FILEHEADER.

## Qualifiers of DUMP

*-A[LL]*

When used with -FILEHEADER, the -A[LL] qualifier displays additional information on the database most of which is useful for FIS in diagnosing issues. A complete description of all the elements that show up with the DSE DUMP -FILEHEADER -ALL

command are beyond the scope of this book. Use only with `-FILEHEADER` or `-UPDPROC` (which is actually redundant as `-ALL` displays the `UPDPROC` information).

`-B[LOCK]=block-number`

Specifies the starting block of the dump. For commands without an object qualifier, DSE uses the last block handled by a DSE operation. When no block has been accessed, (that is, on the first block-oriented command), DSE uses block one (1).

Incompatible with: `-ALL`, `-FILEHEADER` and `-UPDPROC`.

`-C[OUNT]=count`

Specifies the number of blocks, block headers, or records to DUMP.

Incompatible with: `-ALL`, `-FILEHEADER` and `-UPDPROC`.

`-F[ILEHEADER]`

Dumps file header information. A DSE dump of a database file header prints a 0x prefix for all fields printed in hexadecimal format. Refer to the "Introduction" section for a description of the file header fields.

Use only with `-ALL` or `-UPDPROC`

`-G[LO]`

Dumps the specified record or blocks into the current output file in Global Output (GO) format. FIS strongly suggests using `-ZWR` rather than `-GLO` as the `ZWR` format handles all possible content values, including some that are problematic with `-GLO`.

Incompatible with: `-ALL`, `-FILEHEADER`, `-UPDPROC` and `-ZWR`.

`-G[VSTATS]`

Displays the access statistics for global variables and database file(s).

`-NO[CRIT]`

Allows DSE DUMP to work even if another process is holding a critical section. Since results in this mode may be inconsistent, it should only be used if the critical section mechanism is not operating normally.

`-[NO]H[EADER]`

Specifies whether the dump of the specified blocks or records is restricted to, or excludes, headers. `-HEADER` displays only the header, `-NOHEADER` displays the block or record with the header suppressed. DUMP without the `-[NO]HEADER` qualifier dumps both the block/record and the header.

By default, DUMP displays all information in a block or record.

Incompatible with: `-ALL`, `-FILEHEADER`, `-GLO`, `-UPDPROC` and `-ZWR`.

`-O[FFSET]=offset`

Specifies the offset, in bytes, of the starting record for the dump. If the offset does not point to the beginning of a record, DSE rounds down to the last valid record start (e.g., DUMP `-OFF=10` starts at `-OFF=A` if that was the beginning of the record containing offset 10).

Incompatible with: -ALL, -FILEHEADER, and -RECORD.

*-R[ECORD]=record\_number*

Specifies the record number of the starting record of the dump. If you try to dump a record number that is larger than the last actual record in the block, a DSE error message provides the number of the last record in the block.

Incompatible with: -ALL, -FILEHEADER, and -OFFSET.

*-U[PDPROC]*

Displays the helper process parameters with the fileheader elements.

Use only with -FILEHEADER.

*-Z[WR]*

Dumps the specified record or blocks into the current output file in ZWRITE (ZWR) format.

Incompatible with: -ALL, -GLO, -HEADER and -FILEHEADER.

## Examples for DUMP

Example:

```
DSE> DUMP -FILEHEADER
```

This command displays an output like the following:

```
File /home/jdoe/.fis-gtm/V6.1-000_x86_64/g/gtm.dat
Region DEFAULT

File /home/jdoe/.fis-gtm/V6.1-000_x86_64/g/gtm.dat
Region DEFAULT
Date/Time 27-JAN-2014 03:13:40 [$H = 63214,11620]
Access method BG Global Buffers 1024
Reserved Bytes 0 Block size (in bytes) 1024
Maximum record size 256 Starting VBN 513
Maximum key size 64 Total blocks 0x000000C9
Null subscripts NEVER Free blocks 0x00000056
Standard Null Collation FALSE Free space 0x00000000
Last Record Backup 0x0000000000000001 Extension Count 100
Last Database Backup 0x0000000000000001 Number of local maps 1
Last Bytestream Backup 0x0000000000000001 Lock space 0x00000028
In critical section 0x00000000 Timers pending 0
Cache freeze id 0x00000000 Flush timer 00:00:01:00
Freeze match 0x00000000 Flush trigger 960
Current transaction 0x0000000000002712 No. of writes/flush 7
Maximum TN 0xFFFFFFFF83FFFFFF Certified for Upgrade to V6
Maximum TN Warn 0xFFFFFFFFD93FFFFFF Desired DB Format V6
Master Bitmap Size 496 Blocks to Upgrade 0x00000000
Create in progress FALSE Modified cache blocks 0
Reference count 1 Wait Disk 0
```

## Database Structure Editor

Journal State	[inactive] ON	Journal Before imaging	TRUE
Journal Allocation	2048	Journal Extension	2048
Journal Buffer Size	2308	Journal Alignsize	4096
Journal AutoSwitchLimit	8386560	Journal Epoch Interval	30
Journal Yield Limit	8	Journal Sync IO	FALSE
Journal File:	/home/jdoe/.fis-gtm/V6.1-000_x86_64/g/gtm.mjl		
Mutex Hard Spin Count	128	Mutex Sleep Spin Count	128
Mutex Queue Slots	1024	KILLS in progress	0
Replication State	OFF	Region Seqno	0x0000000000000001
Zqgblmod Seqno	0x0000000000000000	Zqgblmod Trans	0x0000000000000000
Endian Format	LITTLE	Commit Wait Spin Count	16
Database file encrypted	FALSE	Inst Freeze on Error	FALSE
Spanning Node Absent	TRUE	Maximum Key Size Assured	TRUE

Note that the certain fileheader elements appear depending on the current state of database. For example, if Journaling is not enabled in the database, DSE does not display Journal data element fields.

Example:

```
$ dse dump -fileheader -updproc
```

This command displays the fileheader elements along with the following helper process parameters:

```
Upd reserved area [% global buffers] 50 Avg blks read per 100 records 200
Pre read trigger factor [% upd rsrvd] 50 Upd writer trigger [%flshTrgr] 33
```

For more information, refer to the fileheader elements section in “*GT.M Database Structure(GDS)*” (page 265).

## Evaluate

Translates a hexadecimal number to decimal, and vice versa.

The format of the EVALUATE command is:

```
EV[EVALUATE] -D[DECIMAL]
-H[HEXADECIMAL]
-N[NUMBER]=number
```

The -DECIMAL and -HEXADECIMAL qualifiers specify the input base for the number. The -NUMBER qualifier is mandatory. By default, EVALUATE treats the number as having a hexadecimal base.

### Qualifiers of Evaluate

*-D[DECIMAL]*

Specifies that the input number has a decimal base.

Incompatible with: -HEXADECIMAL .

*-H[HEXADECIMAL]*

Specifies that the input number has a hexadecimal base.

Incompatible with: -DECIMAL

`-N[UMBER]=number`

Specifies the number to evaluate. Required.

## Examples for EVALUATE

Example:

```
DSE> evaluate -number=10 -decimal
```

```
Hex:  A   Dec:  10
```

This command displays the hexadecimal equivalent of decimal number 10.

Example:

```
DSE> evaluate -number=10 -hexadecimal
```

```
Hex:  10   Dec:  16
```

This command displays the decimal equivalent of hexadecimal 10.

Example:

```
$ dse evaluate -number=10
```

```
Hex:  10   Dec:  16
```

This command displays the decimal equivalent of Hexadecimal 10. Note that if you do not specify an qualifier with `-NAME`, then EVALUATE assumes Hexadecimal input.

## EXit

The EXIT command ends a DSE session.

The format of the EXIT command is:

```
EX[IT]
```

The EXIT command has no qualifiers.

## Find

Locates a given block or region. The format of the FIND command is:

```
F[IND]  -B[LOCK]=block-number
-E[XHAUSTIVE]
-F[REEBLOCK] /H[INT]
-K[EY]=key
-[NO]C[RIT]
-R[EGION][=region]
-S[IBLINGS]
```

- At the beginning of a DSE session, use the FIND `-REGION` command to select the target region.

- The FIND command, except when used with the -FREEBLOCK and -REGION qualifiers, uses the index tree to locate blocks. FIND can locate blocks only within the index tree structure. If you need to locate keys independent of their attachment to the tree, use the RANGE command.

## Qualifiers of FIND

*-B[LOCK]=block\_number*

Specifies the block to find.

On commands without the -BLOCK= qualifier, DSE uses the last block handled by a DSE operation. When no block has been accessed, that is, on the first block-oriented command, DSE uses block one (1).

Incompatible with: -KEY, -REGION

*-E[XHAUSTIVE]*

Searches the entire index structure for the desired path or siblings.

- FIND -EXHAUSTIVE locates blocks that are in the tree but not indexed correctly.
- FIND -EXHAUSTIVE locates all paths to a "doubly allocated" block.



### Note

A doubly allocated block may cause inappropriate mingling of data. As long as no KILLS occur, double allocation may not cause permanent loss of additional data. However, it may cause the application programs to generate errors and/or inappropriate results. When a block is doubly allocated, a KILL may remove data outside its proper scope. See "Maintaining Database Integrity Chapter" for more information on repairing doubly allocated blocks.

Incompatible with: -KEY, -REGION, -FREEBLOCK

*-F[REEBLOCK]*

Finds the nearest free block to the block specified by -HINT.

- The -FREEBLOCK qualifier is incompatible with all other qualifiers except -BLOCK and -HINT.
- The -HINT qualifier is required with the -FREEBLOCK qualifier.
- FIND -FREEBLOCK relies on the bitmaps to locate its target, so be sure to fix any blocks incorrectly marked "FREE" before using this command. See MAP -BUSY for more information on fixing incorrectly marked free errors.

Required with -HINT; compatible with -BLOCK and [NO]CRIT.

*-H[INT]=block\_number*

Designates the starting point of a -FREEBLOCK search.

FIND -FREE -HINT locates the "closest" free block to the hint. This provides a tool for locating blocks to add to the B-tree, or to hold block copies created with SAVE that would otherwise be lost when DSE exits. FIND -FREE relies on the bitmaps to locate its target, so be sure to fix any blocks incorrectly marked "FREE" before using this command.

Required with: -FREEBLOCK; compatible with -BLOCK and [NO]CRIT.

*-K[**EY**]=key*

Searches the database for the block containing the specified key or if the key does not exist, the block that would contain it, if it existed.

- Enclose an M-style key in quotation marks (" "). FIND -KEY is useful in locating properly indexed keys. The -KEY qualifier is incompatible with all other qualifiers.
- FIND -KEY= uses the index to locate the level zero (0) block , or data block, containing the key. If the key does not exist, it uses the index to locate the block in which it would reside. Note that FIND only works with the index as currently composed. In other words, it cannot FIND the "right" place, only the place pointed to by the index at the time the command is issued. These two locations should be, and may well be, the same; however, remind yourself to search for, understand and take into account all information describing any current database integrity issues.

Compatible only with [NO]CRIT.

*-[**NO**]C[**RIT**]*

Allows FIND to work even if another process is holding a critical section.

As results in this mode may be inconsistent, it should only be used if the critical section mechanism is not operating normally

*-R[**EGION**][=region]*

Switches to the named Global Directory region.

-REGION without a specified region, or -REGION="", displays all existing regions in the database.

Use Alone.

*-S[**IBLINGS**]*

Displays the block number of the specified block and its logical siblings in hexadecimal format.

The logical siblings are the blocks, if any, that logically exist to the right and left of the given block in the database tree structure.

Incompatible with: -FREEBLOCK, -HINT, -KEY, -REGION

## Examples for FIND

Example:

```
DSE> find -exhaustive -block=180
Directory path
Path--blk:off
1:10 2:1E

Global paths
Path--blk:off
6:51 1A4:249 180
```

## Database Structure Editor

This command locates block 180 by looking through the B-tree index for any pointer to the block. This command finds even those blocks that are connected to the tree but the first key in the block does not match the index path.

Example:

```
DSE> find -free -hint=180
```

```
Next free block is D8F.
```

This command locates the "closest" free block to block 180.

You can use this command as a tool for locating blocks to add to the B-tree, or to hold block copies created with SAVE that would otherwise be lost when DSE exits.

Example:

```
DSE>find -key="^biggbl(1)"
```

This command locates the key ^biggbl(1) in the database.

Example:

```
DSE> find -freeblock -hint=232
```

This command starts to search for free block after block 232.

Example:

```
DSE> FIND -FREEBLOCK -HINT=232 -NOCRIT
```

This command searches for freeblocks after block 232 even if another process is holding a critical section.

Example:

```
DSE> find -sibling -block=10
```

This command operates like FIND -BLOCK; however it reports the numbers of the blocks that logically fall before and after block 180 on the same level. This command produces an output like the following:

```
Left sibling      Current block    Right sibling
0x0000000F      0x00000010      0x00000011
```

## Help

The HELP command explains DSE commands. The format of the HELP command is:

```
-H[ELP] [help topic]
```

## Integrit

Checks the internal consistency of a single non-bitmap block. INTEGRIT reports errors in hexadecimal notation.

The format of the INTEGRIT command is:

```
I[INTEGRIT] -B[LOCK]=block-number
```



## Note

Unlike MUPIP INTEG, this command only detects errors internal to a block and cannot detect errors such as indices incorrectly pointing to another block. For information on the utility that checks multiple blocks, refer to the “INTEG ” (page 88) of the General Database Management chapter.

## Qualifiers of Integrit

*-B[LOCK]=block\_number*

Specifies the block for DSE to check. On commands with no -BLOCK qualifier, DSE uses the last block handled by a DSE operation. When no block has been accessed, that is, on the first block-oriented command, DSE uses block one (1).

*-NO[CRIT]*

Allows DSE INTEG to work even if another process is holding a critical section. Since results in this mode may be inconsistent, it should only be used if the critical section mechanism is not operating normally.

## Maps

Examines or updates bitmaps. The format of the MAPS command is:

```
M[APS] -BL[OCK]=block-number
-BU[SY]
-F[REE]
-M[ASTER]
-R[ESTORE_ALL]
```

MAPS can flag blocks as being either -BUSY or -FREE. The -MASTER qualifier reflects the current status of a local bitmap back into the master map. The -RESTORE\_ALL qualifier rebuilds all maps and should be used with caution since it can destroy important information.

By default, MAPS shows the status of the bitmap for the specified block.

## Qualifiers for MAP

*-BL[OCK]=block\_number*

Specifies the target block for MAPS. The -BLOCK qualifier is incompatible with the -RESTORE\_ALL qualifier.

On commands with no -BLOCK= or -RESTORE\_ALL qualifier, DSE uses the last block handled by a DSE operation. When no block has been accessed, that is, on the first block-oriented command, DSE uses block one (1).

Incompatible with: -RESTORE\_ALL

*-BU[SY]*

Marks the current block as busy in the block's local map and appropriately updates the master bitmap.

Compatible only with: -BLOCK

**-F[REE]**

Marks the current block as free in the block's local map and appropriately updates the master bitmap.

Compatible only with: -BLOCK

**-M[ASTER]**

Sets the bit in the master bitmap associated with the current block's local map according to whether or not that local map is full.

Use only with: -BLOCK.

**-R[ESTORE\_ALL]**

Sets all local bitmaps and the master bitmap to reflect the blocks used in the database file.

Use -RESTORE\_ALL only if the database contents are known to be correct, but a large number of the bitmaps require correction.



**Caution**

The -RESTORE\_ALL qualifier rebuilds all maps and should be used with a great deal of caution as it can destroy important information.

Use alone.

**Examples**

Example:

```
DSE> MAPS -BLOCK=20 -FREE
```

This command flags block 20 as free. A sample DSE DUMP output block 0 is as follows:

Block 0	Size 90	Level -1	TN 10B76A	V5	Master Status:	Free Space	
			Low order	High order			
Block 0:		XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX		
Block 20:		:XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX		
Block 40:		XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX		
Block 60:		XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX		
Block 80:		XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX		
Block A0:		XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX		
Block C0:		XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX		
Block E0:		XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX		
Block 100:		XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX		
Block 120:		XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX		
Block 140:		XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX		
Block 160:		XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX		
Block 180:		XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX		
Block 1A0:		XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX		
Block 1C0:		XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX		
Block 1E0:		XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX		

'X' == BUSY    '.' == FREE    ':' == REUSABLE    '?' == CORRUPT

Note that BLOCK 20 is marked as REUSABLE, which means FREE but in need of a before-image journal record.

Example:

```
DSE> maps -block=20 -busy
```

This command marks block 20 as busy. A sample DSE DUMP output of block 0 is as follows:

```
Block 0 Size 90 Level -1 TN 1 V5 Master Status: Free Space
```

	Low order	High order
Block 0:	XXX.....	.....
Block 20:	X.....	.....
Block 40:	.....	.....
Block 60:	.....	.....
Block 80:	.....	.....
Block A0:	.....	.....
Block C0:	.....	.....
Block E0:	.....	.....
Block 100:	.....	.....
Block 120:	.....	.....
Block 140:	.....	.....
Block 160:	.....	.....
Block 180:	.....	.....
Block 1A0:	.....	.....
Block 1C0:	.....	.....
Block 1E0:	.....	.....

'X' == BUSY '.' == FREE ':' == REUSABLE '?' == CORRUPT

Note that the BLOCK 20 is marked as BUSY.

## OPen

Use the OPEN command to open a file for sequential output of global variable data. The format of the OPEN command is:

```
OP[EN] F[ILE]=file
```

- OPEN a file to which you want to "dump" information.
- If an OPEN command does not have a -FILE qualifier, DSE reports the name of the current output file.

### Qualifiers for OPEN

*-F[ILE]=file-name*

Specifies the file to open.

### Examples for OPEN

Example:

```
DSE> OPEN
```

```
Current output file: var.out
```

This command displays the current output file. In this case, the output file is var.out.

Example:

```
DSE> OPEN -FILE=var1.out
```

The command OPEN -FILE=var1.out sets the output file to var1.out.

## Overwrite

Overwrites the specified string on the given offset in the current block. Use extreme caution when using this command.

The format of the OVERWRITE command is:

```
OV[ERWRITE] -D[ATA]=string
-O[FFSET]=offset
```

### Qualifiers for OVERWRITE

*-B[LOCK]=block number*

Directs DSE to OVERWRITE a specific block. If no block number is specified, the default is the current block.

*-D[ATA]=string*

Specifies the data to be written. Use quotation marks around the string and escape codes of the form \a or \ab, where "a" and "b" are hexadecimal digits representing non-printing characters. \\ translates to a single backslash.

*-O[FFSET]=offset*

Specifies the offset in the current block where the overwrite should begin.

### Examples for Overwrite

Example:

```
DSE>overwrite -block=31 -data="Malvern" -offset=CA
```

This command overwrites the data at the specified location.

## Page

Sends one form feed to the output device. Use PAGE to add form feeds to a dump file, making the hard copy file easier to read. If you plan to use the dump file with MUPIP LOAD, do not use PAGE.

The format of the PAGE command is:

```
P[AGE]
```

The PAGE command has no qualifiers.

## RANGE

The RANGE command finds all blocks in the database whose first key falls in the specified range of keys. The RANGE command may take a very long time unless the range specified by -FROM and -TO is small. Use FIND -KEY and/or FIND -KEY -EXHAUSTIVE first to quickly determine whether the key appears in the index tree.

The format of the RANGE command is:

```
RA[NGE] -F[ROM]=block-number
-T[O]=block-number
-I[NDEX]
-LOS[T]
-[NO]C[RIT]
-[NO]BU[SY]
-S[TAR]
-LOW[ER]=key
-U[PPER]=key
```

### Qualifiers of RANGE

*-F[ROM]=block\_number*

Specifies a starting block number for the range search.

By default, RANGE starts processing at the beginning of the file.

*-T[O]=block-number*

Specifies an ending block number for the range search. By default, RANGE stops processing at the end of the file.

*-I[NDEX]*

Restricts a search to index blocks.

*-LOS[T]=block\_number*

Restricts a search to blocks not found by a FIND -BLOCK.

*-LOW[ER]=key*

Specifies the lower bound for the key range.

*-[NO]BU[SY]=busy/free*

Restricts a search to either BUSY or FREE blocks.

*-[NO]C[RIT]*

Allows DSE RANGE to work even if another process is holding a critical section. Since results in this mode may be inconsistent, it should only be used if the critical section mechanism is not operating normally.

*-S[TAR]*

Includes index blocks that contain a single star key.

`-U[PPER]=key`

Specifies the upper bound for the key range.

## Examples for RANGE

Example:

```
DSE> range -lower="^abcdefgh" -upper="^abcdefghi" -from=A -to=CC
```

This command searches for a specified keys between block 10 and block 204. Note that the range (between FROM and TO) of blocks must be valid blocks specified in hexadecimal.

Example:

```
DSE> range -lower="^abcdefgh" -upper="^abcdefghi" -from=A -to=CC -noindex
```

This command searches only data blocks for the specified keys between block 10 and block 204.

Example:

```
DSE> range -lower="^abcdefgh" -upper="^abcdefghi" -from=A -to=CC -index
```

This command searches only index blocks for the specified keys between block 10 and block 204.

Example:

```
DSE> range -lower="^abcdefgh" -upper="^abcdefghi" -lost
```

This command includes lost blocks while searching for the specified keys and reports only blocks which are not currently indexed.

Example:

```
DSE> range -lower="^Fruits(15)" -upper="^Fruits(877)" -from=A -to=F
```

Blocks in the specified key range:

```
Block: 0000000A Level: 0
Block: 0000000B Level: 0
Block: 0000000C Level: 0
Block: 0000000D Level: 0
Block: 0000000E Level: 0
Block: 0000000F Level: 0
```

Found 6 blocks

This command search for keys between ^Fruits(15) and ^Fruits(877).

## REMove

Removes one or more records or a save buffer.

The format of the REMOVE command is:

```
REM[OVE] -B[LOCK]=block-number
```

```
-C[OUNT]=count
-O[FFSET]=offset
-R[ECORD]=record-number
-V[ERSION]=version-number
```

The version number is specified in decimal.

## Qualifiers of REMOVE

```
-B[LOCK]=block_number
```

Specifies the block associated with the record or buffer being deleted.

On commands with no -BLOCK= qualifier, DSE uses the last block handled by a DSE operation. When no block has been accessed, that is, on the first block-oriented command, DSE uses block one (1).

```
-C[OUNT]=count
```

Specifies the number of records to remove.

By default, REMOVE deletes a single record.

Incompatible with: -VERSION

```
-O[FFSET]=offset
```

Specifies the offset (in bytes) of the record to be removed. If the offset does not point to the beginning of a record, DSE rounds down to the beginning of the record containing the offset (for example, REMOVE -OFF=10 starts at OFF=A if that was the last prior record boundary).

Incompatible with: -VERSION, -RECORD

```
-R[ECORD]=record_number
```

Specifies the number that identifies the record to remove. The -RECORD qualifier is incompatible with the -OFFSET and -VERSION qualifiers.

Incompatible with: -VERSION, -OFFSET

```
-V[ERSION]=version_number
```

Specifies the version number, in decimal, of the save buffer to remove. -VERSION is required to REMOVE a SAVE buffer. -VERSION is incompatible with all qualifiers except -BLOCK.

Use only with: -BLOCK; decimal

## REStore

The RESTORE command restores saved versions of blocks.

```
RES[TORE] B[LOCK]=block-number
F[ROM]=from
```

```
R[EGION]=region
V[ERSION]=version-number
```

The version number is specified in decimal.

## Qualifiers of RESTORE

```
-B[LOCK]=block_number
```

Specifies the block to restore.

For commands with no -BLOCK= qualifier, DSE uses the last block handled by a DSE operation. When no block has been accessed, (i.e., on the first block-oriented command), DSE uses block one (1).

```
-F[ROM]=block_number
```

Specifies the block number of the SAVE buffer to restore.

DSE restores the block specified with -BLOCK qualifier with the block specified by the -FROM qualifier.

By default, RESTORE uses the target block number as the -FROM block number.

```
-R[EGION]=region
```

Specifies the region of the saved buffer to restore.

By default, RESTORE uses SAVE buffers from the current region.

```
-V[ERSION]=version_number
```

Specifies the decimal version number of the block to restore. The version number is required.

## SAve

The SAVE command preserves versions of blocks, or displays a listing of saved versions for the current DSE session. Saved information is lost when DSE EXITS.

Use with the RESTORE command to move SAVED blocks to a permanent location, and as a safety feature use SAVE to retain copies of database blocks before changing them.

The format of the SAVE command is:

```
SA[VE] -B[LOCK]=block-number
-C[OMMENT]=string
-L[IST]
-[NO]C[RIT]
```

## Qualifiers of SAVE

```
-B[LOCK]=block_number
```

Specifies the block to restore.

On commands with no -BLOCK= qualifier, DSE uses the last block handled by a DSE operation. When no block has been accessed, that is, on the first block-oriented command, DSE uses block one (1).

*-C[OMMENT]=string*

Specifies a comment to save with the block. Enclose the comment in quotation marks (" ").

Incompatible with: -LIST

*-L[IST]*

Lists saved versions of specified blocks. The -LIST qualifier is incompatible with the -COMMENT qualifier.

By default, SAVE -LIST provides a directory of all SAVED blocks.

Incompatible with: -COMMENT

*-[NO]C[MIT]*

Allows DSE SAVE to work even if another process is holding a critical section. Since results in this mode may be inconsistent, it should only be used if the critical section mechanism is not operating normally.

## SHift

Use the SHIFT command to shift data in a block, filling the block with zeros, or shortening the block. The format of the SHIFT command is:

```
SH[IFT] -B[ACKWARD]=b_shift  
-F[ORWARD]=f_shift  
-O[FFSET]=offset
```

b\_shift must always be less than or equal to offset. This means that DSE SHIFT in the backward direction is restricted to the maximum of OFFSET number of bytes. This ensures that the shift does not cross block boundaries, either intentionally or unintentionally.

### Qualifiers of SHIFT

*-B[ACKWARD]=shift*

Specifies the number of bytes to shift data in the direction of the block header.

Incompatible with: -FORWARD

*-F[ORWARD]=shift*

Specifies the number of bytes to shift data toward the end of the block.

Incompatible with: -BACKWARD

*-O[FFSET]=offset*

Specifies the starting offset, in bytes, of the portion of the block to shift.

*-SPawn*

## SPawn

Use the SPAWN command to fork a child process for access to the shell without terminating the current DSE environment.

The format of the SPAWN command is:

```
SP[AWN] [shell-command]
```

- The SPAWN command accepts an optional command string for execution by the spawned sub-process. If the SPAWN has no command string parameter, the created sub-process issues a shell prompt and accepts any legal shell command. To terminate the sub-process, use the shell logout command.
- The SPAWN command has no qualifiers.
- DSE SPAWN works with an argument. If the argument contains spaces, enclose it with quotes.

The SPAWN command has no qualifiers.

DSE SPAWN works with an argument. If the argument contains spaces, enclose it with quotes.

## Examples of SPAWN

Example:

```
DSE> SPAWN "mumps -run ^GDE"
```

This command suspends a DSE session and executes the shell command `mumps -run ^GDE`.

## Wcinit

Use the WCINIT command to reinitialize the global buffers of the current region. Because it cleans out the cache, the WCINIT command should not be used except under the guidance of FIS.



### Caution

A WCINIT command issued while normal database operations are in progress can cause catastrophic damage to the database.

The format of the WCINIT command is:

```
W[CINIT]
```

- The WCINIT command has no qualifiers.
- When you issue the WCINIT command, DSE issues the CONFIRMATION: prompt. You must verify the WCINIT command by responding with "YES."

If you do not confirm the WCINIT, DSE issues the message:

No action taken, enter yes at the CONFIRMATION prompt to initialize global buffers.

- WCINIT operations are more safely performed by MUPIP RUNDOWN. Use this command only under instructions from FIS.

## DSE CommandSummary

COMMAND	QUALIFIERS	COMMENTS
AD[D]	-B[LOCK]=block number	-
-	-D[ATA]=string	Incompatible with -POINTER, -STAR
-	-K[EY]=key	Incompatible with -STAR
-	-O[FFSET]=offset	Incompatible with -RECORD, -STAR
-	-P[OINTER]=pointer	Incompatible with -DATA
-	-R[ECORD]=record-number	Incompatible with -OFFSET, -STAR
-	-S[TAR]	Incompatible with -DATA,-KEY, -OFFSET, -RECORD
AL[L]	-B[UFFER_FLUSH]	Incompatible with -RENEW
-	-C[RITINIT]	Incompatible with -RENEW, -RELEASE, -SEIZE
-	-[NO]F[REEZE]	Incompatible with -RENEW
-	-O[VERRIDE]	Meaningful only with -[NO]FREEZE
-	-REF[ERENCE]	Incompatible with -RENEW
-	-REL[EASE]	Incompatible with -CRITINIT, -RENEW,-SEIZE
-	-REN[EW]	Use alone
-	-S[EIZE]	Incompatible with -RENEW, -RELEASE, -CRITINIT
-	-W[CINIT]	Incompatible with -RENEW
B[UFFER_FLUSH]	-	-
CH[ANGE]	-BL[OCK]=block number	Incompatible with -FILEHEADER and qualifiers used with -FILEHEADER
-	-BS[IZ]=block-size	Use only with -BLOCK, -LEVEL, -TN
-	-L[EVEL]=level	Use only with -BLOCK, -BSIZ, -TN
-	-TN [=transaction number]	Use only with -BLOCK, -BSIZ, -LEVEL
-	-OF[FSET]=offset	Use only with -BLOCK, -CMPC, -RSIZ
-	-RE[CORD]=record number	Use only with -BLOCK, -CMPC, -RSIZ

**Database Structure Editor**

COMMAND	QUALIFIERS	COMMENTS
-	-CM[PC]= compression count	Use only with -BLOCK, -RECORD, -OFFSET, -RSIZ
-	-RS[IZ]=record size	Use only with -CMPC -OFFSET, -RECORD, -BLOCK
-	-F[ILEHEADER]	Incompatible with -BSIZ, -CMPC, -TN, -LEVEL, -OFFSET, -RECORD, -RSIZ
-	AVG_BLK_READ=Average blocks read	-
-	B_B[YTESTREAM]=transaction number	-
-	-B_C[OMPREHENSIVE]=transaction number	Use only with -FILEHEADER; decimal
-	B_D[ATABASE] = transaction number	Use only with -FILEHEADER; decimal
-	-B_I[NCREMENTAL] = transaction number	Use only with -FILEHEADER; decimal
-	-BLK[_SIZE]=block size	Use only with -FILEHEADER; decimal
-	-BLO[CKS_FREE]=free blocks	Use only with -FILEHEADER; decimal
-	-B_R[ECORD]=transaction number	Use only with -FILEHEADER; decimal
-	-CO[RRUPT_FILE]=value	Use only with -FILEHEADER
-	-CU[RRENT_TN]=transaction number	Use only with -FILEHEADER
-	DECL[OCATION]=value	Use only with -FILHEADER; decimal
-	DEF[_COLLATION]=value	Use only with -FILEHEADER;
-	-ENCRYPTION_HASH	Use only with -FILEHEADER
-	-FL[USH_TIME][=delta time]	Use only with -FILEHEADER
-	-FR[EEZE]=value	Use only with -FILEHEADER
-	-FU[LLY_UPGRADED]=boolean	Use only with -FILEHEADER
-	-GV[STATSRESET]	Use only with -FILEHEADER
-	-HARD_SPIN_CPUNT=Mutex hard spin count	Use only with -FILEHEADER
-	-HEXL[OCATION]=value	Use only with -FILEHEADER;hexa
-	-INT[ERRUPTED_RECOV]=boolean	
-	-JNL_YIELD_LIMIT=journal yeild limit	
-	-K[EY_MAX_SIZE]=key_max_size	Use only with -FILEHEADER; decimal
-	-M[ACHINE_NAM]=value	
-	-N[ULL_SUBSCRIPTS]=value	Use only with -FILEHEADER
-	-NO[CRIT]	

**Database Structure Editor**

COMMAND	QUALIFIERS	COMMENTS
-	-OV[ERRIDE]	
-	-RC_SRV_COUNT	
-	-RE_READ_TRIGGER=read trigger	
-	-Q[UANTUM_INTERVAL] [=delta time]	Use only with -FILEHEADER; decimal
-	-REC[ORD_MAX_SIZE]=maximum record size	Use only with -FILEHEADER; decimal
-	-REF[ERENCE_COUNT]=reference count	Use only with -FILEHEADER; decimal
-	-REG[_SEQNO]=sequence number	Use only with -FILEHEADER; hexa
-	-RESERVED_BYTES=reserved bytes	Use only with -FILEHEADER;decimal
-	-[NO] RES[PONSE_INTERVAL] [=delta time]	Use only with -FILEHEADER; decimal
-	-SLEEP_SPIN_COUNT=mux sleep spin count	Use only with -FILEHEADER;
-	-SPIN_SLEEP_TIME=mux sleep time	
-	-[NO]S[TALENESS_TIMER] [=delta time]	Use only with -FILEHEADER; decimal
-	-TIC[K_INTERVAL] [=delta time]	Use only with -FILEHEADER; decimal
-	-TIM[ERS_PENDING]=timers pending	Use only with -FILEHEADER; decimal
-	-TO[TAL_BLKs]=total_blocks	Use only with -FILEHEADER
-	-TR[IGGER_FLUSH]=trigger flush	Use only with -FILEHEADER
-	-W[RITES_PER_FLUSH]=writes per flush	Use only with -FILEHEADER; decimal
-	-WAIT_DISK=wait disk	-
-	-Zqgblmod_S[EQNO] = sequence number	Use only with -FILEHEADER;hexa
-	-Zqgblmod_T[rans]=sequence_number	Use only with -FILEHEADER;hexa
CL[OSE]	-	-
CR[ITICAL]	-I[NIT]	Use only with -RESET
-	-O[WNER]	Use alone
-	-REL[EASE]	Use alone
-	-REM[OVE]	Use alone
-	-RES[ET]	Use only with -INIT
-	-S[EIZE]	Use alone
D[UMP]	-B[LOCK]=block_number	Incompatible with -FILEHEADER
-	-C[OUNT]=count	Incompatible with -FILEHEADER

**Database Structure Editor**

COMMAND	QUALIFIERS	COMMENTS
-	-F[ILEHEADER]	Use alone
-	-G[LO]	Incompatible with -FILEHEADER, -HEADER
-	-G[VSTATS]	Use only with -FILEHEADER
-	-[NO]H[EADER]	Incompatible with -FILEHEADER, -GLO
-	-O[FFSET]=offset	Incompatible with -FILEHEADER, -RECORD
-	-R[ECORD]=record_number	Incompatible with -FILEHEADER, -OFFSET
EV[ALUATE]	-D[ECIMAL]	Incompatible with -HEXADECIMAL
-	-H[EXADECIMAL]	Incompatible with -DECIMAL
-	-N[UMBER]=number	Required
EX[IT]		-
F[IND]	-B[LOCK]=block_number	Incompatible with -KEY, -REGION
-	-E[XHAUSTIVE]	Incompatible with -KEY, -REGION, -FREEBLOCK
-	-F[REEBLOCK]	Required with -HINT; compatible with -BLOCK
-	-H[INT]=block_number	Required with -FREEBLOCK
-	-K[EY]=key	Use alone
-	-R[EGION][=region]	Use alone
-	-S[BLINGS]	Incompatible with -FREEBLOCK, -HINT, -KEY, -REGION
H[ELP]	[help topic]	-
I[NTEGRIT]	-B[LOCK]=block_number	-
M[APS]	-BL[OCK]=block_number	Incompatible with -RESTORE_ALL
-	-BU[SY]	Compatible only with -BLOCK
-	-F[REE]	-
-	-M[ASTER]	-
-	-R[ESTORE_ALL]	Use alone
OP[EN]	-F[ILE]=file	-
OV[ERWRITE]	-B[LOCK]=block_number -D[ATA]=string	-
-	-O[FFSET]=offset	-
P[AGE]	-	-

**Database Structure Editor**

COMMAND	QUALIFIERS	COMMENTS
RA[NGE]	-F[ROM]=block_number	-
-	-T[O]=block_number	-
-	-I[NDEX]=block_number -L[OST]=block_number -[NOT]BUSY=busy/free -S[TAR]=block_number -L[OWER]=key	-
-	-U[PPER]=key	-
REM[OVE]	-B[LOCK]=block-number	-
-	-C[OUNT]=count	Incompatible with -VERSION
-	-O[FFSET]=offset	Incompatible with -VERSION, -RECORD
-	-R[ECORD]=record-number	Incompatible with -VERSION, -OFFSET
-	-V[ERSION]=version-number	Use only with -BLOCK; decimal
RES[TORE]	-B[LOCK]=block-number	-
-	-F[ROM]=block-number	-
-	-R[EGION]=region	-
-	-V[ERSION]=version-number	Required; decimal
SA[VE]	-B[LOCK]=block-number	-
-	-C[OMMENT]=string	Incompatible with -LIST
-	-L[IST]	Incompatible with -COMMENT
SH[IFT]	-B[ACKWARD]=shift	Incompatible with -FORWARD
-	-F[ORWARD]=shift	Incompatible with -BACKWARD
-	-O[FFSET]=offset	-
SP[AWN]	[CLI command]	-
W[CINIT]	-	-

\* Use these qualifiers only with instructions from FIS.

---

## Chapter 11. Maintaining Database Integrity

Revision History		
Revision V5.5-000/10	28 September 2012	Added two new sections—"O5—Salvage of a damaged spanning node" (page 354) and "Recovering data from damaged binary extracts" (page 331).
Revision V5.5-000/2	19 March 2012	In "O4—Salvage of Data Blocks with Lost Indices" (page 352), fixed syntax errors in the example.
Revision V5.4-002B	24 October 2011	Conversion to documentation revision history reflecting GT.M releases with revision history for each chapter.

This chapter discusses GT.M methods for maintaining data availability and integrity.

A database that has GDS integrity may not be consistent from the application data point of view. That is, certain types of failures that do not damage the GDS database structures may cause logical transactions (consisting of multiple database updates within an application) to stop in an "illogical" state with some, but not all, of the updates in place. Transaction processing and database journaling are good methods for maintaining application data consistency. For more information on transaction processing, refer to the "General Language Features of M" and "Commands" chapters of the *GT.M Programmer's Guide*. For more information on journaling, refer to the "GT.M Journaling" chapter of this manual.

Maintaining database integrity is integral to GT.M operation; you should seldom, if ever, need the material in this chapter, especially if you use journaling. However, databases can be corrupted by unusual events such as hardware failures, sudden loss of power, operating system failure, or improper operator action. All such events should be followed with database integrity checks.

The chapter describes the following:

- Suggested times to use MUPIP INTEG for verifying database integrity
- Recommended methods for handling database problems
- General techniques and strategies for using DSE
- Instructions for identifying and repairing errors with DSE

---

### Verifying Database Integrity

A consistent verification strategy expedites the process of rapid identification and correction of database damage, while minimizing the overhead of integrity checking. In GT.M, this strategy is logically developed around MUPIP INTEG and its numerous options for verifying GDS integrity. For detailed information on MUPIP INTEG, refer to the "MUPIP" chapter. The following sections describe situations when executing MUPIP INTEG is the appropriate action.

GTMASSERT sends an operator log message in addition to the usual user message. Because these are potentially dangerous conditions, all GTMASSERTs should be immediately reported to FIS. Check database integrity with the -FAST qualifier, if

appropriate, as soon as possible. GT.MCHECK is similar to GTMASSERT but less sophisticated. It does not send an operation log message; however, it sends a message to the Principal Device.

### Regularly Scheduled Verification

Schedule INTEGs at regular intervals to ensure that no unobserved or unreported events corrupt the database. These regular checks minimize the occurrence of damaged pointers, which may cause updates to incorrect places in the file, likely resulting in escalating damage.

### Before or After Major Transfers

Because of the time they require, and their relative value to the total database organization, operations that move large amounts of information into or out of a database should be accompanied by an INTEG. INTEG should precede output operations such as MUPIP EXTRACT, and follow input operations such as MUPIP LOAD, RESTORE, and JOURNAL RECOVER.

One consistent occurrence of large information transfers occurs during database backups. In many cases, successful recovery from catastrophic events depends on having a reliable backup copy of the database. Therefore, backup procedures should be designed to complement database integrity verification. When the backup is to disk, the fastest method may be to INTEG the backup copy immediately after making it. If the backup is not in GDS format, the INTEG should precede the backup.

### Immediately after Catastrophic Events

Any catastrophic event, such as hardware or operating system failure, should be immediately followed by an INTEG. To determine the cause of the failure, examine the system error messages, operator messages, and system log files, if available.

### Immediately after Run-Time Database Errors

Check database integrity when the GT.M run-time system reports database access errors. The table in section R1 lists all run-time errors that indicate system problems. Most of these errors should be followed by an INTEG, or by one of the appropriate alternatives discussed in the section identified by the table.

### Immediately After Database Repairs

Since the GT.M run-time system normally performs GDS maintenance, based on a fairly complex set of rules, DSE depends on its operator to determine whatever subset of those rules apply to the repair. Even when you have skill and confidence, FIS recommends you verify the result of all database repairs with a database integrity check.

---

## Approaches to Database Recovery

If you experience database integrity problems, there are three strategies to consider when approaching the recovery:

- Recover with journaling
- Restore from backup and redo any lost work
- Repair the database

To achieve the intended result, correction of database errors requires careful planning. Each strategy differs from the others in the scope of damage it can handle, in skills needed, and in database availability.

## Recover from Journals

Journaling is generally the most attractive approach to recovery from integrity problems. It allows management of recovery using logical rather than physical constructs, including suppression of updates based on time and/or source and preservation of application-level logical transactions. Backward journal recovery is generally the fastest means of repair. The cost of journaling is the added load it imposes on normal operation to make and store the journal files. For more information on journaling, refer to the "GT.M Journaling" chapter.

## Restore from Backup

Restoring the database from the backup is the least technically sophisticated approach to handling integrity problems. This strategy is most beneficial when the data in the database is static or can be recomputed. In other cases, it requires operational controls to identify, and people to reenter, the work performed between the backup and the failure. For more information on MUPIP BACKUP, RESTORE, EXTRACT, and LOAD, refer to the "MUPIP" chapter. You may also use UNIX utilities such as tar, dump, and restore.

Some database regions may be setup to hold only temporary data, typically only valid for the life of a GT.M process or even just during some operation performed by a process. Rather than restoring such a region, it is generally more appropriate to delete it and recreate it using MUPIP CREATE.

## Repair with DSE

Database repair with DSE requires more skill, and potentially more time than the other approaches. Using DSE requires vigilant attention to, and a clear understanding of, GDS. DSE can generally access and change almost any data in the database file. When using DSE, you assume the responsibility that GT.M normally carries for ensuring the integrity of the database structure. Because DSE may be used concurrently with other processes, updates by concurrent processes may interfere with repair actions. When possible, prevent other users from accessing the region during repairs.

If you elect to repair the database, you may want to seek assistance from an available source of expertise such as FIS or your GT.M Value Added Reseller (VAR). If your organization plans to perform repairs beyond straightforward corrections to the file header, FIS strongly recommends that the responsible person(s) familiarize themselves with the material in the INTEG section of the MUPIP chapter, the GDS and DSE chapters, and this chapter. FIS recommends using DSE on test files, in advance of any work on production files.

## Preventive Maintenance

Once you understand the cause of a database integrity problem, you can correct or improve the environment to prevent or minimize future damage. These changes may include hardware reconfiguration, such as improving the quality of power; changes to the operational procedures, such as implementing journaling; and/or changes to the Global Directories, such as balancing data assignment into files of more manageable sizes.

Use the following tools to help determine the cause of a database integrity problem.

- Knowledge of the application and how it is used
- Context dumps produced by application programs
- Core dumps produced by application programs
- Core dumps produced by GT.M
- Interviews with users to discover their actions

## Maintaining Database Integrity

- Review of all recent changes to hardware, UNIX, GT.M, the application, procedures, etc.
- Copies of damaged files
- The trail from DSE sessions in the form of notes, a script file recording the session, sequential files, and saved blocks.

## Determining the Cause of the Problem

The following questions may help you understand the type of information required to determine the nature of a database integrity problem.

- How seriously are operations affected?
- What level of urgency do you assign to getting the problem resolved?
- What were the circumstances under which the database became damaged or inaccessible?
- How was the problem first recognized?

Examine the accounting logs for information about recent process terminations. Capture information about what functions were in use. Look for any information which might be helpful in establishing patterns in case the problem is repetitive.

- Has the system crashed recently? If so, what caused the crash?
- Is there database damage?
  - What region(s) are affected? What globals?
  - What are the error messages?
  - What do you see when you examine the database?
  - Are you comfortable with fixing the problem?
- What version of GT.M are you using? What version of UNIX? What UNIX platform are you running?

## MUPIP Recovery

Bring down the damaged application using appropriate utilities, MUPIP RUNDOWN -REGION region or -FILE file-name naming the problem database. Restart the application. Consider writing programs or procedures to partially automate shutting down one or all applications; to reduce the chance of errors.

## Follow-up

Make sure to transfer any relevant files or reports to FIS. Please also communicate any information regarding the circumstances surrounding the problem, including the answers to the questions above. Consider the following:

- Has any hardware or software component of your system recently changed?
- Was anyone doing anything new or unusual?
- Was the problem preceded or followed by any other notable events?
- Did you have any unusual problems during the analysis or recovery?

- Do you have any suggestions about this procedure?

---

## Repairing the Database with DSE

When doing repairs with DSE, understanding the nature of the information in the database provides a significant advantage in choosing an appropriate and efficient repair design.

For example, if you know that certain data is purged weekly, and you find damage in some of this type of data that is already five or six days old, you may be able to discard rather than repair it. Similarly, you might find damage to a small cross-index global and have a program that can quickly rebuild it.

When you know what the data "looks" like, you are in a much better position to recognize anomalies and clues in both keys and data. For example, if you understand the format of a particular type of node, you might recognize a case where two pieces of data have been combined into a single GDS record.

## Using the Proper Database File

Because DSE lets you perform arbitrary actions without imposing any logical constraints, you must ensure that they are applied to the proper file.

First, verify that `gtmgbldirnames` an appropriate Global Directory. Check the definition with the `printenv` command. You may create or use Global Directories that differ from the "normal" Global Directory. For instance, you might create a Global Directory that mapped all global names except a normally unused name to a file with integrity problems, and map that unused name to a new file. Then you could use MUIP to CREATE the new file and use DSE to SAVE blocks from the damaged file and RESTORE them to the new file for later analysis.

When you initiate DSE, it operates on the default region specified by the Global Directory. Once DSE is invoked, use `FIND -REGION` to determine the available regions, and then to select the appropriate region. The technique of creating a temporary Global Directory, with the target region for the repair as the default region, prevents accidental changes to the wrong region.

## Locating Structures with DSE

DSE provides the `FIND` command and the `RANGE` command for locating information.

`FIND -REGION=` redirects DSE actions to a specified region.

`FIND -BLOCK=` locates a block by using the key in the first record of the block to try to look up that block through the B-tree index. If the block is not part of the tree, or the indexing of the block is damaged, DSE reports that the search failed.

`FIND -SIBLING -BLOCK=` operates like `FIND -BLOCK=`; however it reports the numbers of the blocks that logically fall before and after the specified block on the same level.

`FIND -EXHAUSTIVE -BLOCK=` locates a block by looking through the B-tree index for any pointer to the block. This should find the block in the case where the block is connected to the tree but the first key in the block does not match the index path. `FIND -EXHAUSTIVE` is useful in locating all paths to a "doubly allocated" block.

`FIND -KEY=` uses the index to locate the level zero (0) block, or data block, containing the key. If the key does not exist, it uses the index to locate the block in which it would reside. Note that `FIND` only works with the index as currently composed. In other words, it cannot `FIND` the "right" place, only the place pointed to by the index at the time the command is issued. These two locations should be, and may well be, the same; however, remind yourself to search for and take into account all information describing the failure.

FIND -FREE -HINT locates the "closest" free block to the hint. This provides a tool for locating blocks to add to the B-tree, or to hold block copies created with SAVE that would otherwise be lost when DSE exits. FIND -FREE relies on the bitmaps to locate its target, so be sure to fix any blocks incorrectly marked "FREE" before using this command.

The RANGE command sifts through blocks looking for keys. RANGE checks blocks without regard to whether they are in the B-tree, and without regard to whether they are marked free or busy in the bitmaps. RANGE provides a brute force way to find a key if it exists and can be very time consuming in a large database. Note that RANGE may report blocks that were previously used and were legitimately removed from the tree by an M KILL command.

## Safety in Repairs

DSE is a powerful tool with few restrictions that places great responsibility on the user. Establishing the following habits can greatly increase the safety margin.

- Plan your fallback strategy before starting repairs with DSE.  
This will enable you to make the best choice between repair and restore and/or recovery strategies as your analysis proceeds. In addition, you will be able to reasonably assess the potential risks of your decision.
- Determine, at least approximately, the extent of the damage, and how much work has been done since the last backup.
- Check the existence, dates, and sizes of all files; do not assume that everything is as it "should" be.
- Estimate the time required to restore and redo the work. Determine if there are special circumstances, such as imminent deadlines.
- Consider whether you have the disk space to pursue two courses in parallel.
- Consider whether you should back up the damaged database for additional protection or for later analysis.
- Before changing any block in the database, always use the DSE SAVE command to make an in-memory copy of that block.

If a modification fails to accomplish its intended goal, you can use the DSE RESTORE command to get the block back to its previous state. For instance, a CHANGE -BSIZ= that specifies a smaller block size causes DSE to discard all information falling beyond the new size.

An important aspect of this strategy is recognizing that testing some modifications requires using other tools such as MUPIP INTEG, but once you leave DSE to invoke MUPIP you lose anything saved in memory. To avoid this problem, use SPAWN to access those tools.

To save a copy of the block for further analysis, SAVE it, and then RESTORE it to an empty block. The best place to put such a copy, using RESTORE -REGION=, is in a special region created just to receive such blocks.

Alternatively, you can RESTORE it temporarily in a free block within the region, preferably near the end of the file. If you RESTORE the block to the original database, it may be overlaid when normal operation requires more blocks. You may prevent this overlay by using MAP -BUSY on the target block of the RESTORE. However this causes INTEG to report "incorrectly marked busy" errors.

- After changing a block, always check the quality of the result by using the DSE INTEG command.

DSE INTEG does not check the placement of the block in the tree. It checks only the single block specified explicitly with the -BLOCK= qualifier or implicitly (the current block) when -BLOCK= is omitted. If you need to verify the index structure related to a block, SPAWN and use MUPIP INTEG -REGION -FAST, possibly with the -BLOCK or -SUBSCRIPT qualifiers.

## Maintaining Database Integrity

Specifying `-BLOCK=` tends to avoid incorrect assumptions about which block DSE last handled. Not specifying `-BLOCK=` tends to minimize typographical errors in identifying the block.

## Discarding Data

When you must discard a block or a record, take steps to preserve or create structures that have integrity.

DSE has no single command that discards a block. You must locate the last block in its path with `FIND [-BLOCK]` or `FIND -EXHAUSTIVE` and `REMOVE` the record that points to the block being discarded. Then `MAP` the deleted block `-FREE`.

When you discard the only record in any block you must `MAP` that block `-FREE` and `REMOVE` the record (up one level) that points to the deleted block. The only exception is when it is the only block pointed to by the root block of the tree. Leaving empty blocks (except as the data level of empty or undefined globals) violates standard operating assumptions of GDS databases.

When you must discard the top block in a Global Variable Tree, you can alternatively use the method employed by `GT.M` when it processes a `KILL` command. This method maintains a record of the global variable name. To use this method, use `FIND -FREE` to locate a free block, and `MAP` the new block `-BUSY`. Next, `CHANGE` the new block `-BSIZ=header-size (7/8) -LEVEL=0`. Finally, `CHANGE` the top level block `-BSIZ=header-size (7/8) -LEVEL=1` and `ADD -STAR -POINTER=the-new-block`.

Never delete the only remaining record in block one (1). Block one (1) is the root block of the Directory Tree for the entire file.

## Concurrent Repairs

DSE can operate concurrently with normal access by the `GT.M` run-time system. This lets you perform an investigation and some types of repairs with minimal disruption.

Some repairs should only be undertaken by a process that has standalone access to the database, while other repairs present no danger when performed with other users accessing the file. However, there is still some risk with the latter type of repairs, depending on the "placement" of the error and the likelihood of concurrent access to that area of the database.

Unless availability is a critical problem, FIS recommends performing all repairs in standalone mode to ensure the safety of data. For environments where availability is an issue, your knowledge of the application and how it is used are the best guides in assessing the risk of performing concurrent repairs. To help you assess the amount of risk, the following sections identify repairs that should only be undertaken with standalone access.

If you attempt concurrent repairs, plan the order of your updates carefully. Always `REMOVE` the index record that points to a block before using `MAP -FREE` on that block. Always `MAP` a block `-BUSY` and assure that it meets GDS design criteria and accomplishes the repair goal before using `ADD` to create an index record that points to that block.

## Terminating Processes

In performing some types of repairs, you may have to stop one or more processes. You can choose from several methods.

- If the process' principal device is not available, or the process does not respond to pressing `<CTRL-C>`, use `MUPIP STOP`. This allows `GT.M` to disengage the process from all shared resources, such as I/O devices and open database files.
- The DSE command `CRITICAL -INITIALIZE -RESET` causes `GT.M` to terminate all images that are actively accessing the target database. This DSE command has a similar effect on processes to that of `MUPIP STOP`, except that it simultaneously terminates all processes actively using a database.

## Maintaining Database Integrity

- Finally, if the process does not respond to MUPIP STOP, use KILL-9. This terminates the process abruptly and may leave database files improperly closed and require a MUPIP RUNDOWN. Since KILL-9 may cause database damage, it should be followed by a MUPIP INTEG.

When processes have stopped or terminated abnormally, FIS recommends shutting down all GT.M processes, checking the integrity of the database, then restarting the processes. First, use `ps -af` to determine the process IDs. Then use MUPIP STOP or KILL-15 to terminate all the GT.M processes. Repeat the `ps -af` command to assure that all processes have terminated. If they have not, use KILL-9 instead of KILL-15.

When you have terminated all processes, do a MUPIP RUNDOWN on all database files:

```
mupip rundown -file <name of database>
```

Use the UNIX `ipcs` utility to examine the states of message queues, shared memory, and semaphores. If any of these resources are left from the processes that have just been killed, use the UNIX `ipcrm` utility to remove them. Refer to the "Appendix" for more information.



### Caution

Use `ipcrm` with extreme care, as removing the wrong resources can have disastrous results.

Example:

```
ipcs
IPC status from /dev/kmem as of Sat Feb 16 13:13:11 1999
T  ID  KEY      MODE      OWNER    GROUP
Shared Memory:
m  1800 0x01021233 --rw-rw-rw-   uuu     dev
m   91 0x01021232 --rw-rw-rw-   uuu     dev
Semaphores:
s  1360 0x01021233 --ra-ra-ra-   uuu     dev
s   61 0x01021232 --ra-ra-ra-   uuu     dev
```

This shows the state of these resources with a user `uuu` working on two databases `-m1800 -s1360` and `-m91 -s61`.

Check the integrity of the database:

```
mupip integ -file <name of database>
```

To preserve database integrity, always verify that all GT.M images have terminated and all GDS databases are RUNDOWN before shutting down your system.

Terminating GT.M abnormally with KILL-9 can leave the terminal parameters improperly adjusted, making them unsuited for interactive use. If you terminate GT.M with KILL-9 without terminating the job, `logout` to reset the terminal characteristics.

## Recovering data from damaged binary extracts

### CORRUPT Errors

You can recover the value of a corrupt global using the global variable name and the dump (in ZWRITE format) of rest of the block from the point of corruption and then insert it into the database.

## Maintaining Database Integrity

Because the ZWRITE format is used for reconstructing the value of the global, the part of the block after the point of corruption may contain internal structures, for example, a record header and other globals. Therefore, always take extra precautions while identifying the value portion of the global. In addition, ZWRITE format displays byte values as characters whenever it can. This may not reflect the actual usage of those bytes, for example, for internal structures. If the extract is damaged, you might need to do additional work to reconstruct the value.

After you reconstruct the value of a global, add it to the database using an M SET command. For very long values, build the value may using successive SETs with the concatenation operator or SET \$EXTRACT().

### LDSPANGLOINCMP Errors

To fix an LDSPANGLOINCMP error, use the following to reconstruct the value of the global and insert it into the database.

1. The global variable name of the spanning node which has the LDSPANGLOINCMP error.
2. The ZWRITE dump of the partial value corresponding to that global variable name, that is, whatever was accumulated.
3. The global variable name found in the record.
4. ZWRITE dump(s) of the errant chunk(s) from the point of corruption.

The conditions that lead to an LDSPANGLOINCMP error are as follows:

```
Case SN1 - While loading a spanning node the next record contained a non-spanning node:  
"Expected chunk number : ccccc but found a non-spanning node"
```

The partial value can be used as the basis for reconstructing the spanning node.

```
Case SN2 - While loading a spanning node the next record did contain the expected chunk:  
"Expected chunk number : ccccc but found chunk number : ddddd"
```

Use the partial value and the errant chunk as the basis for reconstructing the spanning node. After encountering this error, the binary load continues looking for the next global variable. If there are additional chunks from the damaged spanning node in the binary extract file, there is a case SN3 error for each of them. Use the errant chunk dumps from them as part of the reconstruction.

```
Case SN3 - Not loading a spanning node but found a record with a spanning node chunk:  
"Not expecting a spanning node chunk but found chunk : ccccc"
```

This can be the result of an immediately prior case SN2 error (as described in prior paragraphs) or an isolated errant chunk.

```
Case SN4 - While loading a spanning node adding the next chunk caused the value to go over expected size:  
"Global value too large: expected size : sssss actual size : ttttt chunk number : ccccc"
```

Adding the next chunk caused the value to go over the expected size. Examine the partial value and errant chunk dump.

Case SN5 - While loading a spanning node all of the chunks have been added but the value is not the expected size:

```
"Expected size : sssss actual size : ttttt"
```

All of the chunks were found but the size of the value is not what was expected.

### Example—Repairing an error in a binary extract

Here is an example for repairing an error in a binary extract.

## Maintaining Database Integrity

1. Assume that during the load of a binary extract, you get the following error:

```
%GTM-E-LDSPANGLOINCOMP, Incomplete spanning node found during load
  at File offset : [0x0000027E]
  Expected Spanning Global variable : ^mypoem
  Global variable from record: ^mypoem(#SPAN32)
  Expected chunk number : 3 but found chunk number : 32
  Partial Value :
"Half a league, half a league,Half a league onward,All in the valley of Death Rode the six hundred. Forward, the
Light
▶ Brigade! Charge for the guns he said: Into the valley of Death Rode the six hundred. Forward, the Light
Brigade! Was there a man dismayed? Not tho the soldiers knew Some one had blundered: Theirs not to make reply,
Theirs not to reason why, Theirs but to do and die: Into the valley of Death Rode the six hundred. Cannon to
right of them, Cannon to left of "
```

Errant Chunk :

```
"them, Cannon in front of them Volleyed and thundered; Stormed at with shot and shell, Boldly they rode and
well, Into
▶ the jaws of Death, Into the mouth of Hell Rode the six hundred. Flashed all their sabres bare, Flashed as
they turned in air Sabring the gunners there, Charging an army while All the world wondered: Plunged in the
battery-smoke Right thro the line they broke; Cossack and Russian Reeled from the sabre-stroke Shattered and
sundered. Then they rode back, but no"
```

```
%GTM-E-LDSPANGLOINCOMP, Incomplete spanning node found during load
  at File offset : [0x00000470]
  Global variable from record: ^mypoem(#SPAN4)
  Not expecting a spanning node chunk but found chunk : 4
  Errant Chunk :
"t Not the six hundred. Cannon to right of them, Cannon to left of them, Cannon behind them Volleyed and
thundered;
▶ Stormed at with shot and shell, While horse and hero fell, They that had fought so well Came thro the jaws of
Death, Back from the mouth of Hell, All that was left of them, Left of six hundred. When can their glory fade?
O the wild charge they made! All the world wondered. Honour the charge they made! Honour the Light Brigade,
Noble six hundred!"
```



Because the only issue in this case is that one of the chunk's keys has been damaged, put the value back together from the partial value and the contents of the errant chunks.

2. Execute:

```
$ $gtm_dist/mumps -direct
```

From the first error message pick :

```
Expected Spanning Global variable : ^mypoem
```

3. Use it together with the partial value:

```
GTM>set ^mypoem="Half a league, half a league,Half a league onward,All in the valley of Death Rode the six
▶ hundred. Forward, the Light Brigade! Charge for the guns he said: Into the valley of Death Rode the six
hundred. Forward, the Light Brigade! Was there a man dismayed? Not tho the soldiers knew Some one had
blundered: Theirs not to make reply, Theirs not to reason why, Theirs but to do and die: Into the valley of
Death Rode the six hundred. Cannon to right of them, Cannon to left of "
```



4. Add in the chunk that has the bad internal subscript:

```
GTM>set ^mypoeM=^mypoeM_"them, Cannon in front of them Volleyed and thundered; Stormed at with shot and shell,
▶ Boldly they rode and well, Into the jaws of Death, Into the mouth of Hell Rode the six hundred. Flashed all
their sabres bare, Flashed as they turned in air Sabring the gunners there, Charging an army while All the
world wondered: Plunged in the battery-smoke Right thro the line they broke; Cossack and Russian Reeled from
the sabre-stroke Shattered and sundered. Then they rode back, but no"
```



5. Finally, add the last chunk for that spanning node:

```
GTM>set ^mypoeM=^mypoeM_"t Not the six hundred. Cannon to right of them, Cannon to left of them, Cannon behind
▶ them Volleyed and thundered; Stormed at with shot and shell, While horse and hero fell, They that had fought
so well Came thro the jaws of Death, Back from the mouth of Hell, All that was left of them, Left of six
hundred. When can their glory fade? O the wild charge they made! All the world wondered. Honour the charge
they made! Honour the Light Brigade, Noble six hundred!"
```



You have successfully reconstructed the global from the damaged binary load:

```
GTM>w ^mypoeM
Half a league, half a league,Half a league onward,All in the valley of Death Rode the six hundred. Forward, the
Light
▶ Brigade! Charge for the guns he said: Into the valley of Death Rode the six hundred. Forward, the Light
Brigade! Was there a man dismayed? Not tho the soldiers knew Some one had blundered: Theirs not to make reply,
Theirs not to reason why, Theirs but to do and die: Into the valley of Death Rode the six hundred. Cannon to
right of them, Cannon to left of them, Cannon in front of them Volleyed and thundered; Stormed at with shot and
shell, Boldly they rode and well, Into the jaws of Death, Into the mouth of Hell Rode the six hundred. Flashed
all their sabres bare, Flashed as they turned in air Sabring the gunners there, Charging an army while All the
world wondered: Plunged in the battery-smoke Right thro the line they broke; Cossack and Russian Reeled from
the sabre-stroke Shattered and sundered. Then they rode back, but not Not the six hundred. Cannon to right of
them, Cannon to left of them, Cannon behind them Volleyed and thundered; Stormed at with shot and shell, While
horse and hero fell, They that had fought so well Came thro the jaws of Death, Back from the mouth of Hell,
All that was left of them, Left of six hundred. When can their glory fade? O the wild charge they made! All the
world wondered. Honour the charge they made! Honour the Light Brigade, Noble six hundred!
```



## Finding and Fixing Database Errors

The rest of this chapter is arranged loosely in the form of a decision tree. The material covers a wide range of scenarios and possible actions.

As you begin the decision-making process, follow these general guidelines from this point:

*IF THE SYMPTOM IS A FAILURE TO PROCESS*, refer to section H1.

*IF THE SYMPTOM IS A MUPIP INTEG ERROR REPORT*, refer to section I1. If you are investigating a particular error message, refer to the "MUPIP INTEG errors" table.

*IF THE SYMPTOM IS A GT.M RUN-TIME ERROR REPORT*, refer to section R1. If you are investigating a particular error message, refer to the "MUPIP INTEG errors" table.

To facilitate use of the material as a troubleshooting guide, the text in these sections refers to other sections with alphanumeric designators. Each alphanumeric section describes suggested actions to employ in handling a particular situation.

## C1–Possible Cache Control Problems

When a process detects that a normal cache operating principal has been violated, or that a cache operation is taking an unexpectedly long time, that process triggers a cache verification and rebuild. Such events can be caused by abnormal process termination, or by inappropriately configured or managed database storage subsystems.

When such an event occurs, GT.M sends a series of messages to the operator facility describing the results of the cache verification. If the cache rebuild is successful, no further immediate action is required. If the cache rebuild fails, the database administrator must close off access to the database and use DSE (CRIT and WCINIT) and MUPIP (INTEG) to reset the cache manually and verify that the database is not damaged.

If such events are delivered to the operator facility, you should investigate whether it is appropriate to modify your procedures to prevent abnormal termination, to reconfigure your disk subsystem, or to change the nature or schedule of disk activities so that database access is not disrupted during key periods of operation.

## H1–Process Hangs

The term "hang" refers to a failure to process. Processes may hang for a variety of reasons that have nothing to do with GT.M. However, hanging GT.M processes may indicate that a database has become inaccessible. When you suspect a hang, first determine the extent of the problem.

Your tools include:

- Knowledge of the application and how it is used
- Communication with users
- The ps command and other UNIX system utilities

*WHEN MANY PROCESSES ON A SYSTEM ARE HANGING*, determine if the hangs are confined to a particular application. If all applications are affected or if processes not using GT.M databases are affected, the problem is not a database-specific problem, but something more general, such as a UNIX problem. Refer to section H6.

*WHEN ONLY ONE PROCESS IS HANGING*, find out whether that process is the only one using a particular GT.M application. If it is the only process, start some appropriate second process and determine whether the second process is also affected.

*IF A PROCESS HANGS WHILE OTHER PROCESSES ACCESSING THE SAME DATABASE CONTINUE TO PROCESS*, the problem is not a database problem. Refer to section H2 and then to section H8.

*WHEN ONLY GT.M PROCESSES RUNNING A PARTICULAR APPLICATION HANG*, the problem may be a database problem. Refer to section H2.

Is the system "hung?" If so, consider the following additional questions:

- Does LKE work? If not, then a database has problems (see below).
  - Are there locks owned by a nonexistent process? Can they be cleared? What were the circumstances of a process leaving locks?

### Maintaining Database Integrity

- Are there locks which are not changing? What is the state of the owning process(es)? If not all processes are hung, can the stalled process(es) be MUPIP STOPped?
- Does some region have a "persistent" owner of the critical section (crit)? Which one(s)?
- If there is a crit owner, what is its state? If it is a nonexistent process can it be -REMOVED?
- Does a CRIT -INIT -RESET free the section or just change who owns it?
- If CRIT -INIT -RESET doesn't free the problem, the cache is damaged.

The following is another way of testing the cache: If CRIT is cleared and DSE BUFFER hangs, the cache is not working. Use MUPIP STOP and/or CRIT -INIT -RESET to get everyone out of the segment, then use DSE WCINIT. After a WCINIT, make sure that you can successfully exit from DSE. Use MUPIP INTEG (-FAST) to check for damage which can be induced by WCINIT.

## H3–Database Access Problems

Use the following diagnostic steps and references to determine an appropriate course of action for database access problems.

- Determine if the disk volume is inaccessible.

Use the UNIX ls utility to display information retrieved from the volume. If the volume is not accessible to UNIX, the problem is not a database problem. Refer to section H7.

- Determine whether UNIX can write to the disk.

Use a shell command such as mv or cp. If UNIX cannot write to the volume, the problem is not a database problem. Refer to section H7.

- Determine whether any database file used by the application has "Cache Freeze" set.

Use DSE FIND -REGION=region and DUMP -FILEHEADER to verify that CACHE FREEZE is zero (00000000) for any hung region(s).

If CACHE FREEZE shows a PID, that process used MUPIP or DSE to FREEZE the database. In this case, investigate whether the process is currently producing the desired results. If the FREEZE is legitimate, do whatever is appropriate to speed up the process using FREEZE. For example, use the NICE command. If the process still exists, but should not be running at this time, stop it. If CACHE FREEZE is non-zero but not in use to protect the database, use DSE FIND -REGION=region and CHANGE -FILEHEAD -FREEZE=FALSE to clear the FREEZE state.

Use the DSE commands FIND -REGION and DUMP -FILEHEADER. If any region is frozen, determine who initiated the freeze, and whether the process should be terminated or allowed to complete. The following actions freeze databases:

- DSE CHANGE -FILEHEADER -FREEZE=TRUE
- DSE ALL -FREEZE
- MUPIP BACKUP -NOONLINE
- MUPIP FREEZE
- MUPIP INTEG -REGION

- MUPIP EXTRACT -FREEZE

DSE CHANGE -FILEHEADER -FREEZE=FALSE and MUPIP FREEZE -OFF clear a freeze. However, when used with -OVERRIDE, these commands may cause damage to the results of the process that initiated the freeze. After the freeze is cleared, re-examine the entire situation.

- Determine whether the database files used by the application are accessible for reading.

Use an M function such as \$DATA() or \$ORDER().

- Determine whether the database files used by the application are accessible for writing.

SET a node in each database equal to itself.

*IF THE DATA CAN BE BOTH READ AND WRITTEN*, the problem is not a database problem. Refer to section H8.

*IF DATA CANNOT BE READ OR WRITTEN*, some process is unable to release full ownership of the database critical section. Determine the process identification number (PID) of the process using the DSE command CRITICAL. If the process exists, refer to section H4. If the process is non-existent, use DSE CRITICAL -REMOVE to emulate a release and re-examine the entire situation.

Example:

```
S reg=$V("GVNEXT",""),com="dbcheck.com" o m-* com:newv u com
W "$ DEFINE/USER SYS$OUTPUT dbcheck.lis",!,$ DSE",!
F Q:reg="" D
. W "FIND /REGION=",reg,!,"DUMP /FILEHEADER",!
. S reg(reg)="",reg=$V("GVNEXT",reg)
W "$ SEARCH dbcheck.lis ""Cache freeze""",!
; CAUTION: in the above line, "Cache freeze"
; MUST be mixed-case as shown
W "$ DELETE dbcheck.lis.",!,$ EXIT",!
C com ZSY "@dbcheck"
O com C com:delete
W !,"Attempting first access"
S g="^%" D:$D(^%) F S g=$O(@g) Q:g="" D
. S reg=$V("REGION",g) Q:$l(reg(reg))
. I $D(@g)'[0 S reg(reg)=g
. E S reg(reg)=$Q(@g)
. W !,"Successful Read in region: ",reg," of ",g
S reg="" F S reg=$O(reg(reg)) Q:reg="" D
W !,"Write to region: ",reg
S @(reg(reg)_"="_"_reg(reg)) W " OK"
Q
S reg=$V("GVFIRST"),com="dbcheck" o com:newv u com
W "dse <<y z > dbcheck.lis",!
F Q:reg="" D
. W "find -region=",reg,!,"dump -fileheader",!
. S reg(reg)="",reg=$V("GVNEXT",reg)
W "yz",!,"cat dbcheck.lis | grep 'Cache freeze'"
; CAUTION: in the above line, "Cache freeze"
; MUST be mixed-case as shown
W "|awk '{print $1, $2, $3}'"
C com ZSY "/bin/csh -c ""source dbcheck"""
O com,dbcheck.lis C com:delete,dbcheck.lis:delete
```

```

W !,"Attempting first access"
S g="^%" D:$D(^%) F S g=$0(@g) Q:g="" D
. S reg=$V("REGION",g) Q:$l(reg(reg))
. I $D(@g)'[0 S reg(reg)=g
. E S reg(reg)=$Q(@g)
. W !,"Successful Read in region: ",reg," of ",g
S reg="" F S reg=$0(reg(reg)) Q:reg="" D
. W !,"Write to region: ",reg
. S @(reg(reg)_"_"_reg(reg)) W " OK"
Q
    
```

This routine provides a generalized approach to automating some of the tasks described in this section. It contains argumentless DO commands primarily for typesetting reasons. The routine issues a report if any region is frozen, but does not report which regions are in that state. It may hang reading or writing a database. However, unless the region(s) holding ^% and the next global after ^% has a problem, it displays the name of the region that it is about to try. If this routine runs to completion, the databases in the current Global Directory are completely accessible. The limitations of this routine can be overcome by writing custom shell scripts and/or M programs that include embedded information about one or more Global Directories.



**Note**

If you have a Global Directory mapping globals to multiple files, you may create an alternative Global Directory using different mappings to those same files. Such mapping prevents the test program(s) from touching the "real" data.

Example:

Mapping	Production region	Test region
A to M		
\$DEFAULT	SCRATCH	
N to Z	SCRATCH	
\$DEFAULT		

**H4-Database Cache Problems**

To increase the access speed, GT.M buffers data exchanged between processes and database files in the shared memory cache. If information in the memory cache is damaged, it can block the transfer of data to the disk.

*IF A PROCESS HAS BEEN DETERMINED (FROM SECTION H3) TO NEVER RELEASE FULL OWNERSHIP OF THE DATABASE CRITICAL SECTION*, there may be a problem with the database cache. To determine where the problem is occurring terminate the process. If this clears the hang, the problem was not in the database but in the process, which was somehow damaged. Refer to section P1. Otherwise, another process showing the same symptoms takes the place of the terminated process. In this case, the cache is damaged.

*IF THE CACHE IS DAMAGED*, it must be reinitialized. It is crucial to stop all other database activity during cache initialization. Refer to section Q1 before continuing with this section.

To minimize database damage due to cache reinitialization, and to confirm that the problem is due to a damaged cache, use the DSE command CRITICAL SEIZE followed by BUFFER\_FLUSH. The DSE command BUFFER\_FLUSH attempts to flush the database cache which is a benign operation. Wait at least one minute for this operation to complete.

*IF THE BUFFER\_FLUSH DOES NOT HANG*, the cache is not damaged, and you should review all previous steps starting with section H1.

*IF THE BUFFER\_FLUSH DOES HANG*, use the DSE command WCINIT to reinitialize the cache. This command requires confirmation. Never use WCINIT on a properly operating database. After a WCINIT always perform at least a MUPIP INTEG FAST to detect any induced damage that has a danger of spreading. If the WCINIT command hangs, clear the critical section as described in section H5 and reissue the WCINIT.

## H5–Critical Section Problems

The concurrency control mechanism allows only one process at a time to execute code within a "critical section." To gain access to the database requires a process to first gain ownership of the critical section. The errors described in this section occur when a problem occurs in ownership control of the critical section.

*IF YOU HAVE DETERMINED WHICH PROCESS IS HOLDING THE CRITICAL SECTION* (from section H2 using system utilities), try terminating that process. If this corrects the problem, the damage was to the process, rather than the critical section. Refer to section P1.

*IF YOU CANNOT IDENTIFY THE PROCESS*, or if terminating such a process causes other processes to exhibit the same problem(s), the critical section is damaged and must be reinitialized. Restrict database activity during the reinitialization. Refer to section Q1 before continuing with this section.

*TO REINITIALIZE THE DATABASE CRITICAL SECTION*: Reinitializing a critical section on an active database file carries some risk of causing database damage. You can minimize this risk by restricting database activity during the reinitialization. Refer to section Q1 before continuing with this section.

The DSE command CRITICAL INITIALIZE RESET re-establishes the database-critical section and induces errors for all processes currently accessing the database in question. You can avoid the induced errors in other processes by dropping the RESET qualifier. However, this technique may result in other processes attempting to use partially created critical section structures, possibly corrupting them or the database contents.

After the CRITICAL INITIALIZE, use the DSE commands CRITICAL SEIZE and CRITICAL RELEASE to verify operation of the critical section. Actions such as those described in section H3 test more thoroughly for proper operation.

## H6–UNIX Problems

*IF YOU HAVE DETERMINED THAT MANY PROCESSES IN THE UNIX ENVIRONMENT ARE PERFORMING BADLY*, some processes may be using priorities to "hijack" the system. If this is the case, review why priorities are being adjusted and take appropriate action. Otherwise, you may have a UNIX-related problem.

## H7–Disk Hardware Problems

*IF YOU HAVE DETERMINED THAT A DISK VOLUME IS INACCESSIBLE TO OPENVMS FOR READ AND/OR WRITE*, use the DCL command SHOW DEVICE /FULL to check that the correct volume is properly mounted. If the volume cannot be written, examine the physical device to see whether write lock switches or plugs have been disturbed.

*IF YOU HAVE DETERMINED THAT A DISK VOLUME IS INACCESSIBLE TO UNIX FOR READ AND/OR WRITE*, use the df command to check that the correct volume is properly mounted. If the volume cannot be written, examine the physical device to see whether write lock switches or plugs have been disturbed.

*IF YOU CANNOT LOCATE THE PROBLEM*, run disk diagnostics. Be aware that many disk diagnostics are destructive (i.e., destroy your files). Avoid these diagnostics until you have exhausted all other avenues. If you have to run destructive disk diagnostics, or you determine that a disk spindle must be replaced, start planning for the recovery immediately.

## H8–Application Problems

Application problems may be caused by conflicting M LOCKs or OPEN commands in more than one process, or by a process waiting for completion of M READ or JOB command, which is dependent on an asynchronous event.

First, determine if processes are waiting, without relief, for M LOCKs using the LKE command SHOW ALL WAITING. M routines use LOCK commands to create mutual exclusion semaphores.

*IF THE SHOW COMMAND HANGS*, you have a cache or critical section problem. Restart your evaluation in section H5.

*IF THE SHOW COMMAND DISPLAYS NO LOCKS WAITING*, the problem is not a LOCK problem. If repeated use of SHOW does not display the one or more LOCKs that persist every time, the problem is not a LOCK problem. However, even if the problem is not a lock problem, continue with this section because it discusses the M commands JOB, OPEN, and READ, which may also produce hangs.

A LOCK identified as belonging to a non-existent process results from an abnormal process termination. GT.M automatically clears such LOCKs when some other process requests a conflicting LOCK.

### Persistent LOCKs

Persistent LOCKs belonging to currently existing processes are best released by terminating those processes. Using the LKE command CLEAR with various qualifiers can clear LOCKs, but may cause the routines using the LOCKs to produce inappropriate results. For more information on LKE, refer to the "M LOCK Utility" chapter.

The two most common reasons for persistent LOCKs are deadlocks and LOCKs held during operations that take indeterminate amounts of time.

### Deadlocks

Deadlocks occur when two or more processes own resources and are trying to add ownership of an additional resource already owned by another of the deadlocked processes.

Example:

Process 1	Process 2
-----	-----
LOCK ^A	LOCK ^B
LOCK +^B	LOCK +^A

This shows a sequence in which Process 1 owns ^A and Process 2 owns ^B. Each process is trying to get the resource owned by the other, while "refusing" to release the resource it owns.

Example:

Process 1	Process 2	Process 3
-----	-----	-----
LOCK ^A	LOCK ^B	LOCK ^C
LOCK +^B	LOCK +^C	LOCK +^A

This is similar to the previous example, except that it involves three processes. When an application uses LOCKs in a complex fashion, deadlocks may involve many processes.

## Preventing Deadlocks

You can prevent deadlocks by using timeouts on the LOCK commands. Timeouts allow the program to recognize a deadlock. Once a routine detects a deadlock, it should release its LOCKs and restart execution from the beginning of the code that accumulates LOCKs. Without timeouts, there is no way in M to break a deadlock. You must use outside intervention to terminate at least one deadlocked process, or use LKE to strip a LOCK from such a process.

Example:

```
for quit:$$NEW
quit
NEW() lock ^X(0)
set ^X(0)=^X(0)+1
quit $$STORE(^X(0))
STORE(x)
lock +^X(x):10 if set ^X(x)=name_"^"_bal
lock
quit $TEST
```

This uses a timeout on the LOCK of ^X(x) to cause a retry of NEW.

In addition to the LOCK command, the M JOB, OPEN, and READ commands can contribute to deadlocks.

Example:

```
Process 1          Process 2
-----          -----
LOCK ^A
                OPEN "MSA0:"
                OPEN "/dev/nrst0"
OPEN "MSA0:"
OPEN "/dev/nrst0"
                LOCK +^A
```

This shows a sequence in which Process 1 owns ^A and Process 2 owns device /dev/nrst0. Again, each is trying to get the resource held by the other. Notice that the LOCK commands could be replaced by OPEN commands specifying some non-shared device other than /dev/nrst0.

An application may combine the technique of timeouts on "long" commands to protect the current process, with the technique of minimizing LOCK and OPEN durations, to minimize conflicts with other processes.

Another type of application hanging occurs when a process acquires ownership of a resource and then starts an operation that does not complete for a long period of time. Other processes that need the unavailable resource(s) then hang.

Example:

```
Process 1          Process 2
-----          -----
LOCK ^A
READ x
                LOCK ^A
```

If the READ by Process 1 is to an interactive terminal, and the operator has abandoned that device, the READ may take what seems, at least to Process 2, forever. The M commands OPEN and JOB, as well as READ, can produce this problem. When this situation arises, take action to get long-running commands completed or to terminate the process performing those commands.

## Maintaining Database Integrity

There are two programming solutions that help avoid these situations. You can either limit the duration of those commands with timeouts, or defer resource ownership until any long operations are complete.

Example:

```
for quit:$$UPD
quit
UPD() set x:^ACCT(acct)
do EDITACCT
lock ^ACCT(acct)
if x:^ACCT(acct) set ^ACCT(acct)=y
else write !,"Update conflict Please Reenter"
lock
QUIT $TEST
```

This stores the contents of ^ACCT(acct) in local variable x, before the interactive editing performed by sub-routine EDITACCT (not shown). When the interaction is complete, it LOCKs the resource name and tests whether ^ACCT(acct) has been changed by some other process. If not, it updates the global variable. Otherwise, it informs the user and restarts UPD. This technique eliminates the "open update" problem, but it introduces the possibility the user may have to re-enter work. An application that needs to minimize the possibility of re-entry may extend this technique by testing individual fields (pieces) for conflicting changes.

## I1-MUPIP INTEG Errors

Database errors reported by MUPIP INTEG differ in impact and severity. Some require an immediate action to prevent extending the damage. Action on other less severe errors may be delayed.

The next section provides general guidelines for determining your next course of action and a table with information related to the error messages you may encounter.

## Evaluating the Danger Level of a Database Problem

If you encounter an anomaly in your database or its operations, the following list may offer some help in determining your next course of action. The heading of each section indicates the level of urgency FIS attributes to those items listed below it.

### Requires Immediate Attention

- Block incorrectly marked free errors are very serious and lead to accelerating damage. They degenerate into block doubly-allocated errors, which are also very dangerous. A database with these errors should be closed immediately for repairs.
- Any (structural) error in an index block is dangerous and should be repaired as soon as possible.

Repairs for such errors should also be performed on a database that has been closed to normal activity. The need for both of these actions occurring quickly arises from the likelihood of the bad index being used. Only if your knowledge of the application allows you to predict that a damaged area is used exclusively by restricted functions which are not active (e.g., monthly processing or purges) should you defer repairs.

### Can Be Deferred

- Any (structural) error in a data block (level 0) does not pose a threat of accelerating damage. However, level 0 errors may cause errors or unreliable behavior in the application.

## Maintaining Database Integrity

- Block "incorrectly marked busy" errors only result in database space becoming unavailable until the errors are corrected. An index block error generates incorrectly marked busy errors, because INTEG cannot process the descendants of the damaged index. Therefore, incorrectly marked busy errors should be corrected only after all other errors, except for bitmap errors, are corrected.
- Any bitmap errors flag not only the incorrectly marked block, but also the associated bitmap, and sometimes the master map. Therefore, local and master map errors should be corrected only after all bitmap marked busy or free errors are corrected.
- Transaction number errors usually impact only incremental and online backups.
- File size errors can misdirect MUPIP but do not cause the GT.M run-time system to generate further errors. An exception is auto-extend, which may not work properly if there are file size errors.
- Reference count errors and free block errors are informational only.

The following list of INTEG messages classifies error severity using the following codes, and refers you to a section identifying appropriate follow-up action.

A Access: prevents database access

B Benign: presents no risk of additional damage and has little or no effect on database performance

D Dangerous: presents a high risk that continuing updates may cause significant additional damage

I Index: if the block is an index block, continuing updates will be quite dangerous: treat as a D; if the block is a data block, continuing updates can only cause limited additional damage

T Transient: usually cleared by an update to the database

Repair Dangerous and Access errors immediately. You may assess the benefits of deferring correction of less severe errors until normally scheduled down-time.

MUPIP INTEG Error Messages		
SEVERITY	ERROR MESSAGE	SECTION
I	Bad key name.	K1
I	Bad numeric subscript.	K1
D	Bad pointer value in directory.	K4
D	Bitmap block number as pointer.	K4
D	Block at incorrect level.	O1
D	Block busy/free status unknown (local bitmap corrupted).	M1
D	Block doubly allocated.	K3
B	Block incorrectly marked busy.	M1
D	Block incorrectly marked free.	M1
I	Block larger than file block size.	O1
D	Block pointer larger than file maximum.	K4
D	Block pointer negative.	K4
A	Block size equals zero.	I3

### Maintaining Database Integrity

MUPIP INTEG Error Messages		
SEVERITY	ERROR MESSAGE	SECTION
A	Block size is greater than 64K.	I3
A	Block size not a multiple of 512 bytes.	I3
I	Block too small.	01
T	Block transaction number too large.	I6
D	Blocks per local map is less than 512.	I3
D	Blocks per local map is greater than 2K.	I3
D	Blocks per local map is not a multiple of 512.	I3
B	Cannot INTEG region across network.	I5
T	Cannot determine access method;trying with BG.	I6
I	Compression count not maximal.	K6
T	Current tn and early tn are not equal.	I6
A	Database for region rrr is already frozen, not INTEGing	I6
T	Database requires flushing.	I7
B	File size larger than block count would indicate.	I4
D	File size smaller than block count would indicate.	I4
A	File smaller than database header.	I3
I	First record of block has nonzero compression count.	O1
B	Free blocks counter in file header: nnn is incorrect, should be mmm.	I3
A	Header indicates file creation did not complete.	I3
A	Header indicates file is corrupt.	I8
A	Header size not valid for database.	I3
D	Block xxxx doubly allocated in index block.	K3
A	Incorrect version of GT.M database.	I2
D	Invalid mixing of global names.	K3
I	Key greater than index key.	K2
I	Key larger than database maximum.	K7
I	Key larger than maximum allowed length.	K1
I	Key too long.	K1
I	Key too short.	K1

### Maintaining Database Integrity

MUPIP INTEG Error Messages		
SEVERITY	ERROR MESSAGE	SECTION
I	Keys less than sibling's index key.	K2
I	Keys out of order.	K2
I	Last record of block has invalid size.	K5
D	Last record of block has nonzero compression count.	K5
B	Local bitmap incorrect.	M1
B	Local map block level incorrect.	M2
B	Map block too large.	M2
B	Map block too small.	M2
T	Map block transaction number too large.	I6
B	Master bitmap incorrectly asserts this local map has free space.	M1
B	Master bitmap incorrectly marks this local map full.	M1
B	Master bitmap shows this map full, agreeing with disk local map.	M1
B	Master bitmap shows this map full, agreeing with MUPIP INTEG.	M1
B	Master bitmap shows this map full, in disagreement with both disk and mu_int result.	M1
B	Master bitmap shows this map has space, agreeing with disk local map.	M1
B	Master bitmap shows this map has space, agreeing with MUPIP INTEG.	M1
D	Read error on bitmap.	H7
I	Record has too large compression count.	O2
..	Record too large.	O2
I	Record too small.	O2
D	Reference count should be zero, is nnn.	I6
D	Root block number greater than last block number in file.	K4
D	Root block number is a local bitmap number.	K4
D	Root block number negative.	K4
D	Root level higher than maximum.	O1
D	Root level less than one.	O1
A	Start VBN smaller than possible.	I3
A	Total blocks equals zero.	I4

MUPIP INTEG Error Messages		
SEVERITY	ERROR MESSAGE	SECTION
A	Unable to verify that this is a database file.	I3

## I2–GT.M Version Mismatch

GT.M databases and Global Directories may change with new releases of the product.

*IF YOU GET AN ERROR INDICATING A VERSION MISMATCH*, first identify the GT.M version using the M command `WRITE $ZVERSION` from Direct Mode.

Then refer to the installation procedures for your new release. If you are running more than one release of GT.M investigate the environment variables that define the environments, and take appropriate action.

## I3–File Header Errors

These errors indicate damage to the control or reference information in the file header.

"Start VBN smaller than possible" indicates that INTEG cannot locate the database structure. "Header indicates that file creation did not complete" indicates a MUPIP CREATE problem. In these cases, the database has effectively been lost. DSE cannot correct these problems. If you determine that the costs of recovering from a backup, hopefully with journal files, are prohibitive, consider consulting with FIS.

To correct the other errors of this type use the DSE CHANGE FILEHEADER command with the `BLK_SIZE=`, `BLOCKS_FREE=`, and `TOTAL_BLKS` qualifiers.

"Free blocks counter ..." indicates that the count of free blocks in the file header is not correct. This error only affects `$VIEW("FREECNT",region)` and `DUMP FILEHEADER` which return the information.

## I4–File Size Errors

File size errors can misdirect MUPIP, but do not cause the GT.M run-time system to generate further errors. Auto-extend is the exception and may not function properly if there are file size errors. One possible symptom of an auto-extend problem would be incorrectly marked busy errors from a partial bitmap at the "old" end of the database which had previously been incorrectly initialized.

These errors indicate that the total blocks count does not agree with the file size. Get the starting VBN and the block size for the file by using `DSE DUMP FILEHEADER`. Then calculate the correct total blocks value with the following formula:

$$((\text{file size} - \text{starting VBN} + 1) / (\text{block size} / 512))$$

A decimal number results from this formula. Convert this decimal to a hexadecimal number, then change the total block count to this hexadecimal value using `DSE CHANGE FILEHEADER TOTAL_BLKS=` . You may also need to adjust the free blocks count with `BLOCKS_FREE=`. MUPIP INTEG informs you if this is necessary and gives the correct values.

## I5–More Database Access Problems

These error messages reflect failures to find, open, or access a database file. Examine any secondary error messages to obtain additional information about the problem.

## Maintaining Database Integrity

Use `printenv` to check `gtmgbldir` or use the M command `WRITE $ZGBLDIR` to verify that the "pointer" identifies the proper Global Directory. If the pointer is not appropriate, reset `gtmgbldir` or use the M command `SET $ZGBLDIR=` to name the proper file.

Examine the Global Directory using GDE. If the Global Directory is not appropriate, correct or recreate it with GDE. For more information on the use of GDE, refer to the "Global Directory Editor" chapter.

IF THE GLOBAL DIRECTORY IS DAMAGED BUT ACCESSIBLE WITH GDE, investigate who may have used GDE to perform the modifications. If the Global Directory is damaged and not accessible with GDE, investigate what program, other than GT.M and its utilities, might have written to the file. Except for GDE, all GT.M components treat the Global Directory as static and read-only.

IF THE GLOBAL DIRECTORY APPEARS CORRECT, use the DCL command `SHOW LOGICAL` to verify that any logical names it uses are properly defined for the process experiencing the problem. If the process has an environment to which you do not have access, you may have to carefully read the command procedures used to establish that environment.

IF THE GLOBAL DIRECTORY APPEARS CORRECT, use `printenv` to verify that any environment variables that it uses are properly defined for the process experiencing the problem. If the process has an environment to which you do not have access, you may have to carefully read the shell scripts used to establish that environment.

IF THE ENVIRONMENT VARIABLES APPEAR CORRECT, use the `ls -l` to examine the file protection. Remember to examine not only the file, but also all directories accessed in locating the file.

IF THE FILES APPEAR TO BE PROPERLY MAPPED by the Global Directory, correctly placed given all logical names, and correctly protected to permit appropriate access, use one of the DCL commands `TYPE` or `DUMP` to verify access to the files, independent of GT.M.

IF THE FILES APPEAR TO BE PROPERLY MAPPED by the Global Directory, properly placed given all environment variables, and properly protected to permit appropriate access, use the `od` or `cat` utility to verify access to the files, independent of GT.M.

IF YOU SUSPECT A VERSION MISMATCH PROBLEM, refer to section I2.

IF YOU SUSPECT A DISK HARDWARE PROBLEM, refer to section H7.

## I6–Transient Errors

GT.M corrects certain errors automatically. If you find that any of these errors persist, contact your GT.M support channel.

"Block transaction number too large" indicates that the file header has a smaller transaction number than the database block.

If you are not running TP or incremental backup this is a benign error (from the database's point of view; application data consistency should be verified). GT.M automatically self-corrects these errors as soon as it performs sufficient updates to get the current transaction number of the database higher than any block's transaction number. If this error persists, perform the following steps:

- Run the `MUPIP INTEG` command on your database and look for the following output:

**"Largest transaction number found in database was HHHHHHHH"**

- Run the following command:

**`dse change -fileheader -current_tn=<HHHHHHHH+1>`**

## Maintaining Database Integrity

Where <HHHHHHH+1> is the largest transaction number + 1. This command sets the current transaction number to one more than the largest transaction number found in the database. Note that HHHHHHHH is in hexadecimal form.

"Current tn and early tn are not equal" indicates that the critical section has been damaged. "Reference count is not zero" indicates an improper file close. The first access that references a questionable database should correct these errors. Generally, these errors indicate that the file was not closed normally. This problem is typically caused by an unscheduled shutdown of the system. Review your institution's shutdown procedures to ensure a controlled shutdown.

"Cannot determine access method..." indicates that the fileheader has been damaged. When INTEG detects this error, it forces the access method to BG and continues. If there is no other damage to the file header, no other action may be required.

However, if the access method should be MM, use MUPIP SET ACCESS\_METHOD= to correct the database.

## 17-Database Rundown Problem

A MUPIP INTEG may be performed without write access to the file. However, in the case where the file was improperly closed, it must be RUNDOWN prior to being INTEGed. To do this, MUPIP requires write access to the file, so either increase the privileges for the process, change the protection on the file, or use a more privileged process and repeat the MUPIP INTEG.

## 18-Repair-Induced Problems

These error messages are created by operator actions performed with DSE.

The DSE commands CRITICAL INITIALIZE RESET, ALL RESET, and ALL RENEW induce CRITRESET errors in all processes attempting to access the target database(s).

Any process attempting to access a database that has its "corrupt" flag set to TRUE receives a DBCRPT error.



### Caution

Using the DSE command CHANGE FILEHEADER CORRUPT=TRUE is very dangerous. If the DSE session EXITs before issuing a CHANGE FILEHEADER CORRUPT=FALSE, the database becomes entirely useless.

## K1-Bad Key

This section describes appropriate actions when the error message indicates a damaged key. GDS transforms subscripted or unsubscripted global variable names into keys, which are part of the database record used to index the corresponding global variable data values. The keys are stored in a compressed form which omits that part of the prefix held in common with the previous key in the block. The compression count is the number of common characters. Except in the Directory Tree, all records after the first one have a non-zero count. The first record in a block always has a compression count of zero (0).

IF THE BLOCK IS A DATA BLOCK, that is, level zero (0), refer to section O3.

IF THE BLOCK HAS A LEVEL GREATER THAN ZERO (0), examine the record with the DSE command DUMP BLOCK=OFFSET where the block and offset values are provided by the INTEG error report. If the record appears to have a valid block pointer, note the pointer. Otherwise, refer to section O2.

After noting the pointer, SPAWN and use MUPIP INTEG BLOCK=pointer (if you have time constraints, you may use the FAST qualifier) to check the structure.

IF THE SUB-TREE IS INVALID, according to the MUPIP INTEG, DSE REMOVE the record containing the reported bad key, INTEG, and refer to section O4.

Otherwise use the DSE command DUMP BLOCK= RECORD=9999 to find the last record in the block and examine it using the DUMP RECORD= command. Continue using DSE to follow the pointer(s) down to level 0, always choosing the right-hand branch. Note the largest key at the data level. REMOVE the record containing the reported bad key. Determine the proper placement for the noted key using FIND KEY= and ADD KEY= POINTER where the key and the pointer are those noted in the preceding actions.

### **K2–Keys Misplaced**

When the error is a misplaced key, the keys are not in proper collating sequence.

IF THE BLOCK IS A DATA BLOCK, that is, level zero (0), DUMP it GLO, REMOVE the records that point to it, MAP it FREE, and MUPIP LOAD the output of the DUMP GLO.

IF THE BLOCK HAS A LEVEL GREATER THAN ZERO (0), you may choose to reposition the record in its proper place or use the salvage strategy discussed in section O4. In general, the salvage strategy is less demanding and less dangerous. However, it may be time consuming if the index block holding the record has a level much greater than one (1). If you decide against the salvage strategy, note the contents of the damaged record. In either case, REMOVE the record. If using salvage, refer to section O4. If not, determine the proper location for the record using FIND KEY= to display the closest existing path, then follow the procedure outlined in the last paragraph of K1.

### **K3–Block Doubly Allocated**

A doubly allocated block is dangerous because it causes data to be inappropriately mingled. As long as no KILLS occur, double allocation does not cause permanent loss of additional data. However, it may cause the application programs to generate errors and/or inappropriate results. When a block is doubly allocated, a KILL may remove data outside its proper scope.

A doubly allocated index block may also cause increasing numbers of blocks to become corrupted. Use the following process to correct the problem.

First, identify all pointers to the block, using FIND EXHAUSTIVE and/or information reported by MUPIP INTEG. If the error report identifies the block as containing inappropriate keys or a bad level, INTEG has identified all paths that include the block. In that case, INTEG reports all paths after the first with the doubly allocated error, and the first path with some other, for example, "Keys out of order" error.

IF THE INTEG REPORT DOES NOT MENTION THE BLOCK PRIOR TO THE DOUBLY ALLOCATED ERROR, use FIND EXHAUSTIVE to identify all pointers to that block.

IF THE BLOCK IS A DATA BLOCK, that is, level zero (0), DUMP it GLO, REMOVE the records that point to it, MAP it FREE, and MUPIP LOAD the output of the DUMP GLO.

IF THE BLOCK HAS A LEVEL GREATER THAN ZERO (0), you may sort through the block and its descendants to disentangle intermixed data. If the block has a level of more than one (1), this may be worth a try. The salvage strategy (discussed in section O4) may be time consuming and there may be only one misplaced node. However, in general, the salvage strategy is less demanding and less dangerous.

IF YOU CHOOSE THE SALVAGE STRATEGY, REMOVE the records that point to the block, MAP it FREE, and refer to section O4.

IF YOU DECIDE TO WORK WITH THE BLOCK, choose the path to retain, REMOVE the other pointer record, and relocate any misplaced descendants with DSE ADD and REMOVE.

## K4–Pointer Problems

Each index block is made up of records that contain keys and corresponding pointers. In the case where database damage is a symptom of an incorrect key paired with a valid pointer, the repair strategy, which may be implemented with a number of tactics, is to use the pointer to locate the data and reconstruct the key.

While they occur very infrequently, invalid pointers do not permit the same strategy. If there is an invalid pointer, always eliminate the record containing the bad pointer using the DSE REMOVE command. Since no data can be stored under an invalid pointer, either the pointer error was discovered on the first attempt to use it and no data has been lost, or the pointer was damaged during use. If the pointer was damaged during use, the lost data should be located by examining "Block incorrectly marked busy" errors and generally be recovered as described in section O4.

IF MUCH DATA IS LOST, it may be worthwhile attempting to reconstruct the bad record as follows. Before removing the record containing the bad pointer, use the DUMP command to note the key in the record. Using the error reports and/or the DSE RANGE command, locate the block to which the key should point. Then use DSE ADD to replace the previously deleted record with a new record that has the correct key and pointer in place.

## K5–Star Key Problems

The last record in every index block must be a star-key record that points to a block that continues the path to all data not covered by the preceding records in the block. Star-key records have a unique format with a size of seven (7), or eight (8), depending on the platform, and a compression count of zero (0). The errors discussed in this section indicate a missing or damaged star-key and may be attacked with two strategies.

In general, you should turn the last existing record into a star-key. This works well as long as the block holds at least one valid record. If you choose this strategy, locate the last record using DUMP RECORD=9999. Then DUMP the last record and note its pointer. Next, REMOVE the last record. Finally, ADD STAR POINTER= to the key you noted.

If the star-key is the only record in a root block, you should add a new empty level 0 descendent. If you choose this strategy, add a new star-key using FIND FREEBLOCK HINT=this-block to locate a nearby block. Next, MAP the new block BUSY and CHANGE LEVEL= 0 and BSIZ=7(or 8, if your platform dictates). If the new block has a level of zero (0), return to the damaged block and ADD STAR POINTER=the-first-new-block.

## K6–Compression Count Error

"Compression count not maximal" indicates that the compression count that is used to save space in key storage is not correct.

IF THE BLOCK IS A DATA BLOCK, that is, level zero (0), DUMP it GLO, REMOVE the records that point to it, MAP it FREE, and MUPIP LOAD the output of the DUMP GLO.

IF THE BLOCK HAS A LEVEL GREATER THAN ZERO (0), REMOVE the record and ADD it back in the same location with the same KEY=, and POINTER= or STAR.

You may also adjust the compression count using CHANGE CMPC=. Because this changes the value of all subsequent keys in the block (except the star-key), you should try this alternative only if those keys also appear incorrect.

## K7–Key Warning

"Key too large for database maximum" indicates that the database holds a key that is legal to GT.M but exceeds the KEY\_MAX\_SIZE for the database.

Use the DSE command `CHANGE FILEHEADER KEY_MAX_SIZE=` to adjust the file limitation. Alternatively, you may remove the record, using the M command `KILL` on an ancestor node. If any user attempts to modify or replace the record in the database while the key is over-length, GT.M will reject the SET with an error.

## M1–Bitmap Errors

Every block in the file has a corresponding bit in a bitmap. All blocks with valid data are marked busy in their maps; all blocks that are unused or no longer hold data are marked free. GDS uses bitmaps to locate free blocks efficiently. The errors discussed in this section indicate problems with bitmaps.

"Block incorrectly marked free" is the only potentially dangerous bitmap error. This error means that the block is within the B-tree structure, but that the bitmap shows it available for use (i.e., it is a "Block doubly allocated" waiting to happen). Immediately use DSE to MAP such blocks BUSY.

Bitmap information is redundant (i.e., bitmaps can be recreated by scanning the B-tree); however, the majority of bitmap errors reflect secondary errors emanating from flaws in the B-tree, which are often reported as key or data errors by MUPIP INTEG.

When INTEG encounters an error, it stops processing that leaf of the tree. When it subsequently compares its generated bitmaps to those in the database, it reports the blocks belonging in the tree that it could not find as "Block incorrectly marked busy." This error type can be viewed as a flag, marking the location of a block of lost data whose index is disrupted.

INTEG reports each block that it concludes is incorrectly marked, and also the local map that holds the "bad" bits. Furthermore, if the local map "errors" affect whether the local map should be marked full or not full in the master map, INTEG also reports the (potential) problem with the master map. Therefore, a single error in a level one (1) index block will generate, in addition to itself, one or more "Block incorrectly marked busy", one or more "Local bitmap incorrect", and possibly one or more "Master bitmap shows..." Errors in higher level index blocks can induce very large numbers of bitmap error reports.

Because bitmap errors are typically secondary to other errors, correcting the primary errors usually also cures the bitmap errors. For this reason and, more importantly, because bitmap errors tend to locate "lost" data, they should always be corrected at, or close to, the end of a repair session.

The DSE command `MAP` provides a way to switch bits in local maps with `FREE` and `BUSY`, propagate the status of a local map to the master map with `MASTER`, and completely rebuild all maps from the B-tree with `RESTORE`. Never use `MAP MASTER` until all non-bitmap errors have been resolved.

## M2–Bitmap Header Problems

Bitmaps are stored in blocks that have a unique header format with a level of minus one (-1) and a block size of 87 or 88 depending on the Euclidian ordering of the platform. The errors discussed in this section indicate a bitmap block header that violates that format.

Use the DSE command `CHANGE` with the `BSIZ=87` or `88` (depending on platform) and `LEVEL=-1FF` qualifiers to correct the problem. If the block size is too small, the bitmap will have to be reconstructed using `MAP RESTORE` or manually from INTEG error reports using `MAP FREE`. If there are other errors, defer any `MAP RESTORE` until after they have been repaired.

## O1–Bad Block

GDS organizes the B-tree into logical blocks, each of which GT.M handles discretely. A block consists of a block header and a lexically increasing sequence of records. Blocks starting with the root block up to the data blocks are index blocks. The last block in any complete path is a data block. The errors discussed in this section indicate a damaged block.

Determine if the block has other problems by using the DSE command INTEGRIT. Examine the contents of the block using the DSE command DUMP. You may also examine the block preceding this block in the path and/or blocks pointed to by records in this block. If you can determine an appropriate action, use CHANGE with the BSIZ= and/or LEVEL= qualifiers. If you cannot quickly repair the block, examine its level with DUMP HEADER. If the block is a data block, that is, level zero (0), refer to section O3. If the block has a level greater than zero (0), REMOVE the record that points to the block and refer to section O4.

### O2–Record Errors

GDS organizes keys with pointers or data to form records. A record has a header, which holds the record size, and a compression count, which identifies how much of the preceding key is held in common by this record. Records in the block are ordered by the values of their keys. The errors discussed in this section indicate damage to a record. Record errors present an added challenge, in that they potentially prevent GT.M from correctly interpreting subsequent records in the same block.

IF THE BLOCK IS A DATA BLOCK, that is, level zero (0), refer to section O3.

IF THE BLOCK IS AN INDEX BLOCK, that is, has a level greater than zero (0), the best option is generally to use the salvage strategy discussed in section O4. REMOVE the damaged record and INTEG the block. If the block is still corrupt, repeat the last step, REMOVE the pointer to it, and MAP it FREE. In any case, refer to section O4.

### O3–Data Block Errors

The errors described in this section include damage to the header, the records, or the keys.

IF THE BLOCK IS LEVEL ZERO (0), use DSE DUMP to examine the contents of the block. Note any information that might allow you to correct the problem or might help to identify and recreate the endangered data. If you are familiar with GDS and hexadecimal representations, you may be able to recognize data that DSE cannot recognize because of misalignment.

IF THE BEGINNING OF THE BLOCK IS VALID, DUMP GLO may be able to capture its contents up to the point where it is damaged. In the worst case, REMOVE the record that points to the block, MAP it FREE, and lose its entire contents. The extent and importance of the damage depends on the size of the block and what it should be holding. In a similar but not quite as drastic case, REMOVE the record with the problem and lose the contents of that record.

### O4–Salvage of Data Blocks with Lost Indices

This strategy uses bitmap errors to locate data blocks containing information that belongs in the B-tree, but are no longer indexed because of errors and/or repairs to defective indices.

The algorithm is based on the fact that most bitmap errors are secondary to index errors. Therefore, it is optimistic about bitmaps and pessimistic about indices, and tends to error on the side of restoring more rather than less data to the B-tree. After using this technique, you should always check to see if obsolete, deleted data was restored. If data was restored, and GDS integrity has been restored, you can safely KILL the "extra" data.

IF THE INDICES HAVE BEEN DAMAGED FOR SOME TIME AND THE DAMAGE CAUSED DUPLICATE KEYS TO BE CREATED, this strategy raises the issue of which value is the "correct" value. Because most applications either form new nodes or update existing nodes rather than simply overlaying them, this issue seldom arises. Usually the application will fail in an attempt to update any "misplaced" node. If the problem does arise, the issue may not be determining the "correct" value, but the best available value.

IF THE DUPLICATE NODE PROBLEM COULD BE AN APPLICATION ISSUE, you can load the sequential file produced in DSE with an M program that detects and reports duplicate nodes. You can also use the block transaction numbers as clues to the

## Maintaining Database Integrity

order in which blocks were updated. However, remember that you generally cannot know which record was modified on the last update, and that DSE repair actions modify the block transaction number.

If the duplicate node problem poses a significant problem, you should probably not use DSE to repair the database, but instead, use journals to recover or restore from backups.

This strategy works well when the missing indices are level one (1). However, the time required increases dramatically as the level of the missing index increases. If you have a problem with a level four (4) or level five (5) index, and you have developed skill with DSE, you may wish to try the more technically demanding approach of repairing the indices.

Once you have corrected all errors except bitmap errors, SPAWN and use MUPIP INTEG FAST REGION NOMAP to get a list of all remaining bitmap errors. If the report includes any "Blocks incorrectly marked free", MAP them BUSY. Then use DUMP HEADER BLOCK= to examine each "Block incorrectly marked busy." If the level is one (1), DUMP the block GLO. In any case, MAP it FREE. Once all blocks have been collected in a sequential file in this fashion, use MUPIP LOAD to reclaim the data from the sequential file.

Example:

```
salvage;
  read !,"SET REGION to <DEFAULT>: ",r s:r="" r="DEFAULT"
  write !
  set in="db_integ.log",x="mupip integ -fast -nomap -region"
  set teg="/bin/csh -c ""_x_ "_r_ > "_in_""""
  zsystem teg
  set out="db_drive",skip=$char(32,10,13)
  set prefix="map -bl=",old="",lenold=0,blk=0
  open in:(read:exc="goto done"),out:newv
  use out
  write "dse <<yz",!,"find -region=",r,!,"open -file=", "db.go",!
  use in
  for read x if x["marked" use out do out use in
  ; CAUTION: in the above line, "marked" MUST be in lower-case
  ;
done
  use out
  write "close",!,"exit",!
  write "yz",!
  write "mupip load db.go",!
; comment out the line above if you wish to examine
; db.go and initiate the load separately
  close in,out
  zsystem "/usr/local/bin/tcsh -c ""source db_drive""""
  quit
out
  for j=1:1:$length(x) quit:skip'[$extract(x,j)
  set blk=$piece($piece($extract(x,j,999)," ",1),":",1)
  set state=$select($extract(x,42)="f":" -busy",1:" -free")
  ; CAUTION: in the above line, "f" MUST be in lower-case
  if state=" -free" write "dump -glo -bl=",blk,!
  ; CAUTION: in the above line " -free" MUST match the
  ; case in the $select above
  ; comment out the above line (starting with "i state")
  ; if you wish to eliminate, rather than save,
  ; the contents of loose busy blocks
  write prefix,blk,state,!
```

quit

This routine provides a basic example of automating the technique described in this section. It must be run from an appropriate directory with a properly defined `gtmgbldir`, but can be extended to be more user friendly.

## O5–Salvage of a damaged spanning node

The following example shows how to salvage a damaged spanning node in `^mypoem`.

1. Run MUPIP INTEG to find the location of the damaged spanning node. A MUPIP INTEG report of a region that has damaged spanning nodes might look something like the following:

```
Integ of region DEFAULT

Block:Offset Level
%GTM-E-DBSPANGLOINCMP,
    7:10      0  Spanning node is missing. Block no 3 of spanning node is missing
                Directory Path:  1:10, 2:10
                Path: 4:31, 7:10
Spanning Node ^mypoem(#SPAN1) is suspect.
%GTM-E-DBKEYGTIND,
    7:10      0  Key greater than index key
                Directory Path:  1:10, 2:10
                Path: 4:31, 7:10
Keys from ^mypoem(#SPAN48) to ^mypoem(#SPAN3*) are suspect.
%GTM-E-DBSPANCHUNKORD,
    3:10      0  Chunk of 1 blocks is out of order
                Directory Path:  1:10, 2:10
                Path: 4:3D, 3:10
Spanning Node Chunk ^mypoem(#SPAN4) is suspect.

Total error count from integ:          3

Type           Blocks      Records      % Used      Adjacent
Directory       2           2           5.468       NA
Index           1           4           13.476      1
Data            4           5           76.562      4
Free           93          NA           NA          NA
Total          100          11          NA          5
[Spanning Nodes:2 ; Blocks:3]
%GTM-E-INTEGERRS, Database integrity errors
```

Notice the lines that contain: "Block no 3 of spanning node is missing", "Key greater than index key", and `^mypoem(#SPAN48)` and there is an extra chunk that is not connected to `^mypoem(#SPAN4)`.

2. Confirm whether you have determined the spanning range of the node:

- Is `^mypoem(#SPAN48)` the last node (block number 3)?
- Is `^mypoem(#SPAN4)` the last node?

Clearly, GT.M did not find block 3 and `^mypoem(#SPAN4)` terminated the spanning node, so `^mypoem(#SPAN4)` might be the last node. So, the parts of a spanning node that contain the value are `^mypoem(#SPAN2)` through `^mypoem(#SPAN4)`.

3. Use DSE to find the spanned nodes:

```
DSE> find -key=^mypoe(#SPAN2)
```

```
Key found in block 6.
```

```
Directory path
```

```
Path--blk:off
```

```
1:10, 2:10,
```

```
Global tree path
```

```
Path--blk:off
```

```
4:25, 6:10,
```

```
DSE> find -key=^mypoe(#SPAN3)
```

```
Key not found, would be in block 7.
```

```
Directory path
```

```
Path--blk:off
```

```
1:10, 2:10,
```

```
Global tree path
```

```
Path--blk:off
```

```
4:31, 7:10,
```

```
DSE> find -key=^mypoe(#SPAN4)
```

```
Key found in block 3.
```

```
Directory path
```

```
Path--blk:off
```

```
1:10, 2:10,
```

```
Global tree path
```

```
Path--blk:off
```

```
4:3D, 3:10,
```

```
DSE> f -k=^mypoe(#SPAN5)
```

```
Key not found, would be in block 3.
```

```
Directory path
```

```
Path--blk:off
```

```
1:10, 2:10,
```

```
Global tree path
```

```
Path--blk:off
```

```
4:3D, 3:10,
```

Notice that there is #SPAN2 and #SPAN4 but no #SPAN5. Therefore, #SPAN4 is the last piece. #SPAN3 was not found and is most likely the damaged node.

4. Dump all the blocks in ZWRITE format to see what can be salvaged.

```
DSE> open -file=mypoe.txt
```

```
DSE> dump -block=6 -zwr
```

```
1 ZWR records written.
```

```
DSE> dump -block=7 -zwr
```

```
1 ZWR records written.
```

```
DSE> dump -block=3 -zwr
```

```
1 ZWR records written.
```

```
DSE> close
Closing output file: mypoem.txt

$ cat mypoem.txt
; DSE EXTRACT
; ZWR
$ze(^mypoem,0,480)="Half a league, half a league,Half a league onward,All in the valley of Death Rode the six
▶ hundred. Forward, the Light Brigade! Charge for the guns he said: Into the valley of Death Rode the six
hundred. Forward, the Light Brigade! Was there a man dismayed? Not tho the soldiers knew Some one had
blundered: Theirs not to make reply, Theirs not to reason why, Theirs but to do and die: Into the valley of
Death Rode the six hundred. Cannon to right of them, Cannon to left of "
$ze(^mypoem,22080,480)="them, Cannon in front of them Volleyed and thundered; Stormed at with shot and shell,
Boldly
▶ they rode and well, Into the jaws of Death, Into the mouth of Hell Rode the six hundred. Flashed all their
sabres bare, Flashed as they turned in air Sabring the gunners there, Charging an army while All the world
wondered: Plunged in the battery-smoke Right thro the line they broke; Cossack and Russian Reeled from the
sabre-stroke Shattered and sundered. Then they rode back, but no"
$ze(^mypoem,960,468)="t Not the six hundred. Cannon to right of them, Cannon to left of them, Cannon behind
them
▶ Volleyed and thundered; Stormed at with shot and shell, While horse and hero fell, They that had fought so
well Came thro the jaws of Death, Back from the mouth of Hell, All that was left of them, Left of six hundred.
When can their glory fade? O the wild charge they made! All the world wondered. Honour the charge they made!
Honour the Light Brigade, Noble six hundred!"
```



Notice that block 3 (which is the second block above (because you started with block 2)) has the correct value but its internal subscript must have been damaged.

5. Fix the starting position in the \$ZEXTRACT statement:

```
$ze(^mypoem,480,480)="them, Cannon in front of them Volleyed and thundered; Stormed at with shot and shell,
Boldly they
▶ rode and well, Into the jaws of Death, Into the mouth of Hell Rode the six hundred. Flashed all their sabres
bare, Flashed as they turned in air Sabring the gunners there, Charging an army while All the world wondered:
Plunged in the battery-smoke Right thro the line they broke; Cossack and Russian Reeled from the sabre-stroke
Shattered and sundered. Then they rode back, but no"
```



Verify the value for correctness if you have the knowledge of the type of data in this global. This completes data recovery (whatever was possible).

6. Kill the existing global:

```
GTM>kill ^mypoem

GTM>write ^mypoem
%GTM-E-GVUNDEF, Global variable undefined: ^mypoem
```

7. Load the salvaged global:

```
$ mupip load -format=zwr mypoem.txt
```

```
; DSE EXTRACT
; ZWR
Beginning LOAD at record number: 3

LOAD TOTAL          Key Cnt: 3  Max Subsc Len: 8  Max Data Len: 480
Last LOAD record number: 5

$ gtm

GTM>w ^mypoem
Half a league, half a league,Half a league onward,All in the valley of Death Rode the six hundred.  Forward, the
Light
▶ Brigade! Charge for the guns he said: Into the valley of Death Rode the six hundred.  Forward, the Light
Brigade! Was there a man dismayed? Not tho the soldiers knew Some one had blundered: Theirs not to make reply,
Theirs not to reason why, Theirs but to do and die: Into the valley of Death Rode the six hundred.  Cannon to
right of them, Cannon to left of them, Cannon in front of them Volleyed and thundered; Stormed at with shot and
shell, Boldly they rode and well, Into the jaws of Death, Into the mouth of Hell Rode the six hundred.  Flashed
all their sabres bare, Flashed as they turned in air Sabring the gunners there, Charging an army while All the
world wondered: Plunged in the battery-smoke Right thro the line they broke; Cossack and Russian Reeled from
the sabre-stroke Shattered and sundered.  Then they rode back, but not Not the six hundred.  Cannon to right of
them, Cannon to left of them, Cannon behind them Volleyed and thundered; Stormed at with shot and shell, While
horse and hero fell, They that had fought so well Came thro the jaws of Death, Back from the mouth of Hell, All
that was left of them, Left of six hundred.  When can their glory fade? O the wild charge they made! All the
world wondered.  Honour the charge they made! Honour the Light Brigade, Noble six hundred!
```



## P1–Process Damage

A damaged process is one that has become internally "confused" and is executing in a pathological way not caused by circumstances in the external environment.

IF YOU HAVE DISCOVERED THAT A PROCESS WAS DAMAGED, carefully review all events related to that process leading to the discovery of the problem. It may be possible that the process had an elevated priority and was not hanging, but rather was "hogging" system resources. It is also possible that the problem is an application loop problem, missed by not performing the steps in section H3 with enough rigor.

Check for evidence of any hardware problem that might damage a process.

## Q1–Restricting Database Access

Prevent new users from attempting to access the database by taking steps such as bringing the system to the single-user state or removing execute access to GT.M components for an appropriate class or users. Also, terminate or suspend all processes accessing the database in question, using the UNIX `ps -af` utility to find such processes. Because the DSE command `CRITICAL -INITIALIZE -RESET` generates errors for all processes accessing a database file, it provides a quick way to stop such processes.

## R1–GT.M Run-Time Errors

GT.M processes may detect errors at run-time. These errors trigger the GT.M error handling mechanism, which generally places the process in direct mode, or triggers the application programs to transcribe an error context to a sequential file or to a global. For more information on error handling, refer to the "Error Processing" chapter of the *GT.M Programmer's Guide*.

## Maintaining Database Integrity

Most run-time errors are related to the application and its environment. However, some errors reflect the inability of a process to properly deal with a database. Some errors of this type are also, or only, generated by the GT.M utility programs.

For descriptions of individual errors, refer to the GT.M Message and Recovery Procedure Reference Manual.

IF YOU CANNOT REPRODUCE SUCH ERRORS WITH ANOTHER PROCESS PERFORMING THE SAME TASK, or with an appropriately directed MUPIP INTEG, they were most likely reported by a damaged process. In this case, refer to section P1.

The following table lists run-time errors, alphabetically by mnemonic, each with a section reference for further information.

<b>Run-Time Error Messages Identifying Potential System Problems</b>		
<b>ERROR MNEMONIC</b>	<b>ERROR MESSAGE TEXT</b>	<b>SECTION</b>
BADDVER	Incorrect database version vvv	I2
BITMAPSBAD	Database bitmaps are incorrect	M1
BTFAIL	The database block table is corrupt	R3
CCPINTQUE	Interlock failure accessing Cluster Control Program queue	R7
CRITRESET	The critical section crash count for rrr region has been incremented	I8
DBCCERR	Interlock instruction failure in critical mechanism for region rrr	R7
DBCRPT	Database is flagged corrupt	I8
DBFILERR	Error with database file	I5
DBNOFILEP	No database file has been successfully opened	I5
DBNOTGDS	Unrecognized database file format	I5
DBOPNERR	Error opening database file	I5
DBRDERR	Cannot read database file after opening	I5
FORCEDHALT	Image HALTed by MUPIP STOP	R4
GBLDIRACC	Global Directory access failed, cannot perform database functions	I5
GBLOFLOW	Database segment is full	R5
GVKILLFAIL	Global variable KILL failed. Failure code: cccc	R2
GVORDERFAIL	Global variable \$ORDER or \$NEXT function failed. Failure code: cccc	R2
GVPUTFAIL	Global variable put failed. Failure code: cccc	R2
GVQUERYFAIL	Global variable \$QUERY function failed. Failure code: cccc	R2
GVRUNDOWN	Error during global database rundown	I5
GDINVALID	Unrecognized Global Directory format: fff	I5
GTMCHECK	Internal GT.M error-report to FIS	R6
GVDATAFAIL	Global variable \$DATA function failed. Failure code: cccc	R2

## Maintaining Database Integrity

Run-Time Error Messages Identifying Potential System Problems		
ERROR MNEMONIC	ERROR MESSAGE TEXT	SECTION
GVDIRECT	Global variable name could not be found in global directory	I5
GVGETFAIL	Global variable retrieval failed. Failure code: cccc	R2
GVZPREVFAIL	Global variable \$ZPREVIOUS function failed. Failure code: cccc	R2
MUFILRNDWNFL	File rundown failed	I5
UNKNOWNFOREX	Process halted by a forced exit from a source other than MUPIP	R4
TOTALBLKMAX	Extension exceeds maximum total blocks, not extending	R5
WCFAIL	The database cache is corrupt	R3

## R2–Structural Database Integrity Errors

These run-time errors indicate that the process detected an integrity error within the body of the database.

Verify the error using the MUPIP command `INTEG SUBSCRIPT=`, specifying an immediate ancestor node of the global variable displayed in the error message. Alternatively, you may try the access by running the same routine in another process or by using Direct Mode to perform the same actions as those performed by the M line that triggered the error. If you cannot reproduce the error, refer to section P1.

Most of these errors terminate with a four-character failure code. Each character in the code represents the failure type for an attempt to perform the database access. In cases where the letters are not all the same, the last code is the most critical, because it reflects what happened when the process made its last retry. Earlier tries show error codes that are important to establishing the context of the last error.

The following table lists the failure codes, whether or not they require a MUPIP `INTEG`, a brief description of the code's meaning, and a section reference for locating more information.

Run-Time Database Failure Codes			
FAIL CODE	RUn INTEG	DESCRIPTION	SECTION
A	x	Special case of code C.	O2
B	x	Key too large to be correct.	K1
C	x	Record unaligned: properly formatted record header did not appear where expected.	O2
D	x	Record too small to be correct.	O2
E		History overrun prevents validation of a block.	R3
G	-	Cache record modified while in use by the transaction.	R3
H*	x	Development of a new version of a block encountered an out of design condition.	P1
* Indicates a process problem			

## Maintaining Database Integrity

Run-Time Database Failure Codes			
FAIL CODE	RUn INTEG	DESCRIPTION	SECTION
J		Level on a child does not show it to be a direct descendent of its parent.	O1
K		Cache control problem encountered or suspected.	C1
L		Conflicting update of a block took priority.	R3
M	x	Error during commit that the database logic does not handle.	P1
N	x	A primitive such as a file or queue operation failed.	R7
O		Before image was lost prior to its transfer to the journal buffer.	R3
P		Multi-block update aborted - database damage likely.	I5
Q		Shared memory interlock failed.	R7
R	x	Critical section reset (probably by DSE).	R5
S		Attempt to increase the level beyond current maximum.	R8
U		Cache record unstable while in use by the transaction.	R3
V		Read-only process could not find room to work.	R9
X		Bitmap block header invalid.	M2
Y	x	Record offset outside of block bounds.	02
Z	x	Block did not contain record predicted by the index.	02
a		Predicted bitmap preempted by another update.	R3
b		History overrun prevents validation of a bitmap.	R3
c		Bitmap cache record modified while in use by the transaction.	R3
d	x	Not currently used.	-
e		Attempt to read a block outside the bounds of the database.	02
f		Conflicting update took priority on a non-isolated global and a block split requires a TP_RESTART.	R3
g		The number of conflicting updates on non-isolated global nodes exceed an acceptable level and requires a TP_RESTART.	R3
* Indicates a process problem			

### R3–Run-time Database Cache Problems

These messages indicate probable process damage or database cache corruption. Retry the action with another process. If the second process also fails, refer to section H4; otherwise, refer to section P1.

## R4–Stopped Processes

These errors indicate the process received a message from a kill system service requesting that the image terminate.

The MUPIP STOP command uses kill with a distinguished code. The code provided by MUPIP STOP allows the process to include the source of the stop directive in the error message.

## R5–No More Room in the File

IF THE DATABASE FILLS UP AND CANNOT EXPAND, processes that try to add new information to the database experience run-time errors. The following conditions prevent automatic database expansion.

- Using the MM access method
- Using a file extension of zero (0)
- Inadequate free blocks available on the volume to handle the specified extension

You can handle the first two cases by using the MUPIP EXTEND command. MUPIP EXTEND may also help in dealing with the third case by permitting an extension smaller than that specified in the file header. Note that the extension size in the file header, or /BLOCKS= qualifier to MUPIP EXTEND, is in GDS blocks and does not include overhead for bitmaps.

IF THERE IS NO MORE SPACE ON A VOLUME, you may use the M command KILL to delete data from the database. To KILL an entire global, the database file must contain one free GDS block. You may acquire these by KILLing a series of subscripted nodes or by doing a small extension.

You may also use UNIX utilities such as tar, cp, and lprm to remove files from the volume and place them on another volume.

Finally, you may create or add to a bound volume set with the MOUNT utility invoked by the DCL command MOUNT. If you change the RMS placement of the files, be sure to adjust the Global Directory and/or the logical names to match the new environment.

You can also add a new disk. If you change the placement of the files, be sure to also adjust the Global Directory and/or the environment variables to match the new environment.

## R6–GTMASSERT and GTMCHECK Errors

GTMASSERT and GTMCHECK errors indicate that a process has detected some sort of logical inconsistency. Consult with FIS after gathering all information about the circumstances surrounding the error.

## R7–Interlocked Queue Hardware Problems

These messages indicate possible problems with multiple processor synchronization. Initiate running of hardware diagnostics. If the diagnostics do not locate a problem, consider consulting with FIS after gathering all information about the circumstances of the error.

## R8–Database Tree Maximum Level Exceeded

An attempt has been made to create a tree in the database that contains seven or more levels. The legal levels for a tree are zero to seven. You can add new levels to the global either by killing some of the existing subscripts, or by extracting the global and reloading it into a database with a larger block size, so it does not require as large a tree.

## **R9–Read-only Process Blocked**

While it is unlikely in normal operation, there is a possibility that a process that has read-only access to a database file may fail because it cannot acquire enough cache space to do its work. Because it does not have authority to write to the database, such a process cannot flush modified cache records to disk: it must rely on updating processes to keep the number of modified records down to a point that permits read-only access to the database to proceed successfully. However, if updating processes exit in a fashion that does not permit them to flush out modified records, the read-only process (particularly one doing a large transaction) may fail because the cache cannot supply enough blocks. This condition can be cleared by a DSE BUFFER command in the affected region(s).

---

## Chapter 12. Database Encryption

Revision History		
Revision V990	D990	In “Special note - Gnu Privacy Guard version 2” (page 389), specified that GT.M versions prior to V5.4-001 are not compatible with GPG 2.x.
Revision V6.0-001	27 February 2013	Updated “Tested Reference Implementations ” (page 387), “Installation ” (page 390), and “Plugin Architecture & Interface ” (page 394) for V6.0-001.
Revision V5.5-000/11	05 October 2012	In “Special note - Gnu Privacy Guard version 2” (page 389), added information about setting the environment variable GTMXC_gpgagent.
Revision V5.4-002B	24 October 2011	Conversion to documentation revision history reflecting GT.M releases with revision history for each chapter.

---

### Introduction

### Overview

GT.M on selected platforms can encrypt data in database and journal files. Encryption protects data at rest (DAR), that is it protects against unauthorized access to data by an unauthorized process that is able to access disk files.

A plug-in architecture allows you to use your choice of encryption package. The characteristics of encryption are entirely determined by the package you choose - for example, GT.M implements no "back doors" or "key recovery", and if you want such functionality, you need to choose or construct an encryption package that provides the features you want.

FIS distributes the source and object code for the reference implementation of a plug-in that uses popular, widely available, encryption libraries. If the reference implementation plug-in meets your needs, you are welcome to use it as distributed, but please read and understand the section “Disclaimer ” (page 363). You can also use it as a base to implement your own plug-in.

In the reference implementation, GT.M uses a symmetric cipher to encrypt data. The reference implementation encrypts the key for the symmetric cipher with an asymmetric cipher using public and private keys. The private keys are stored in a key ring on disk locked with a password (or passphrase - the terms are used interchangeably).

### Disclaimer

Database encryption is only useful as one component of a comprehensive security plan and is insufficient as the sole means of securing data. The use of database encryption should follow from a good security plan. This document describes implementing encrypted GT.M databases; it does not discuss security plans.

Proper security protocol never places an unencrypted password, even in obfuscated form and/or in an obscure location, on disk. With GT.M database encryption, unencrypted passwords exist in the address space of processes accessing the database, which

## Database Encryption

means that unencrypted passwords can theoretically be written to swap files when process memory is paged out. To be secure, an installation must handle this by means such as: using encrypted swap devices or files, ensuring that GT.M processes are not paged out, or some other means to ensure that information in swap files is available only to the running process. In other words, even with respect to encryption, GT.M database encryption is only part of a complete security infrastructure.

Our expertise is in GT.M, not in encryption. Encryption needs vary. Furthermore, the use of encryption may be restricted - or required - by regulations applicable to your location and circumstances. Therefore, our approach is to create a plug-in architecture where you can choose your preferred encryption software. In the course of development, we tested it primarily with GNU Privacy Guard, the widely available implementation of Pretty Good Privacy (see "PGP: Pretty Good Privacy" by Simson Garfinkel). Ensure that you have confidence in (and confidence in the support for) whichever encryption software you choose, because failure of the encryption software is likely to leave your data unrecoverable. GT.M itself performs no encryption, and encryption is performed exclusively by software that you install and configure. FIS neither endorses nor supports any specific encryption algorithm or library.

Furthermore, just as GT.M allows for the use of your choice of encryption libraries, encryption libraries in turn require keys that must be managed. In its simplest form, key management requires both that only those who need a key have that key, and also that keys are not lost. Key management is two steps removed from GT.M's implementation of database encryption, but is important to the successful use of encrypted databases. It must be part of your operations policies and procedures. FIS strongly recommends that you understand in detail how to implement the infrastructure for whichever specific encryption you choose.

## Limitations of GT.M Database Encryption

Elements of your security infrastructure and management processes outside of GT.M database encryption need to manage issues discussed in the following sections.

### Data Not At Rest Not Protected

GT.M database encryption is designed to protect data at rest. Applications execute business logic to manipulate and produce unencrypted data. Unencrypted data must exist within application processes, and can be accessed by any process with access rights to the virtual address space of a process containing unencrypted data. Also, data in transit between systems and between processes is not protected by GT.M database encryption.

1. Before creating a core dump, GT.M attempts to clear any keys that it is aware of within the process address space. The reference implementation also uses the encryption libraries so as to minimize the probability of keys appearing in core dumps. Since it is not possible to guarantee that keys will not appear in a process core dump, depending on your security policy, FIS recommends that you consider whether to disable the creation of core dumps by GT.M processes accessing encrypted databases, or use other means to limit access to core dumps. Note also that the use of random byte sequences as keys makes it harder to discern them in a core dump<sup>1</sup>.

### Keys in the Process Address Space / Environment

This is a corollary of the fact that data not at rest is not protected by GT.M database encryption.

In order to encrypt and decrypt databases, keys must exist in the address space / environment of GT.M processes. Furthermore, with the reference implementation, processes also need to have access to the user's private key, and to get access to the private key, they need access to the passphrase of the user's GPG keyring. In order to pass encryption to child processes, the passphrase also exists in the process environment, even if obfuscated. This means that any process that can access the address space or environment of a GT.M process accessing encrypted databases has access to the passphrases and keys.

1. If an application provides some or all users access to a shell prompt or a GT.M direct mode prompt, or allows that user to specify arbitrary code that can be EXECUTE'd, those users can find ways to view and capture keys and passphrases. Note

## Database Encryption

that, if a key or passphrase can be captured, it can be misused - for example, a captured GPG keyring passphrase is captured, it can be used to change the passphrase. You must therefore ensure that your application does not provide such access to users who should not view keys and passphrases.

2. This limitation makes it all the more important that those who have access to shell prompts, GT.M direct mode prompts, etc. not leave sessions unlocked, even briefly, if it is at all possible for someone who should not have knowledge of keys and passphrases to access the sessions during that time.

## Long Lived Keys

A database file has an extended life. In typical operation, only a minuscule fraction of the data within a database changes each day. As changing an encryption key requires re-encrypting all the data, this means encryption keys for files have long lives. Since long-lived keys are security risks - for example, they cannot be changed when an employee leaves - key management must therefore be part of the overall security plan. At a minimum, long lived keys require two stage key management - a database key with a long life, not normally accessed or viewed by a human, stored in a form encrypted by another key that can be changed more easily.

Furthermore, a key must be retained at least as long as any backup encrypted with that key; otherwise the backup becomes useless. You must have appropriate procedures to retain and manage old keys. Since successful data recovery requires both keys and algorithms, the retention processes must also preserve the encryption algorithm.

## Voluminous Samples of Encrypted Data

Database and journal files are large (GB to hundreds of GB). This large volume makes database encryption more amenable to attack than a small encrypted message because having many samples of encrypted material makes it easier to break a key.

## Encryption Algorithms Neither Endorsed Nor Supported by FIS

FIS neither endorses nor supports any specific encryption algorithm.

The selection of an encryption algorithm is determined by many factors, including but not limited to, organizational preferences, legal requirements, industry standards, computational performance, robustness, the availability of encryption hardware, etc. No algorithm meets all needs.

Therefore, GT.M provides a "plug-in" architecture for encryption algorithms, which allows you to integrate your preferred encryption software with GT.M. In the GT.M development environment, we created variations on a reference implementation using popular encryption packages for our validation. We tested each reference implementation variation on at least one computing platform, and one reference implementation variation on each computing platform. This document lists which encryption package we tested on which platform.

You take all responsibility for the selection and use of a specific encryption package. Please be aware that:

1. All encryption libraries that run within the address space of a GT.M process must conform to the rules of any functions for GT.M, as documented, including but not limited to being single threaded, not altering GT.M's signal handlers, restricting the use of timers to the API provided by GT.M, etc.<sup>2</sup>
2. Malfunction of encryption software or hardware can render your data irrecoverable. As part of your comprehensive organizational risk management strategy, please consider the use of logical multi-site application configurations, possibly with different encryption packages and certainly with different encryption keys.
3. The cipher used for database encryption must not change the length of the encrypted sequence of bytes. In other words, if the cleartext string is n bytes, the encrypted string must also be n bytes.

## No Key Recovery

The reference implementation of GT.M database encryption has no "back door" or other means to recover lost keys. We are also not aware of back doors in any of the packages used by the reference implementation.

Lost keys make your data indistinguishable from random ones and zeros. While FIS recommends implementing a documented key management process including techniques such as key escrow, ultimately, you take all responsibility for managing your keys.

## Human Intervention Required

At some point in the process invocation chain, the reference implementation requires a human being to provide a password that is placed (in obfuscated form) in the process environment where child processes can inherit it. If you want to be able to access encrypted databases without any human interaction, you must modify the reference implementation, or create your own implementation.

For example, if you have a GT.M based application server process that is started by xinetd in response to an incoming connection request from a client, you may want to consider an approach where the client sends in a key that is used to extract an encrypted password for the master key ring from the local disk, obfuscates it, and places it in the environment of the server process started by xinetd. If the application protocol cannot be modified to allow the client to provide an additional password, xinetd can be started with the `$gtm_passwd` obfuscated password in its environment, and the `xinetd passenv` parameter used to pass `$gtm_passwd` from the xinetd process to the spawned server process.

## MM Databases

GT.M database encryption is only supported for the Buffered Global (BG) access method. It is not supported for the Mapped Memory (MM) access method. See "Alternatives to Database Encryption" [366], for other options.

## Alternatives to Database Encryption

On some platforms, you may be able to use disk drives with built-in encryption, or encrypted file systems to protect data at rest. These may or may not be as secure as GT.M database encryption: for example, once an encrypted file system is mounted, files thereon can be accessed by any process that has appropriate permissions; with GT.M database encryption each process accessing a database file must individually have access to the keys for that database file.

## Device IO

The built-in interface to encryption is implemented only for data in database, journal, backup and certain formats of extract files. To encrypt IO (say for sequential disk files), you can use IO to PIPE devices. Alternatively, you can call encryption routines from GT.M using the external call interface.

## Replication and GT.CM

GT.M encrypts neither the replication stream nor GT.CM (GNP/OMI) network traffic. When needed, there are excellent third party products for implementing secure TCP/IP connections: software solutions as well as hardware solutions such as encrypting routers.

As with any GT.M process that accesses databases, the Update Process, helper processes and GT.CM server all require provisioning with keys to enable their access to encrypted databases.

## Database Encryption

When a GT.M server has a key for an encrypted database, any client connecting to the server can access encrypted records in that database.

### FIPS Mode

For database encryption, the plugin reference implementation also provides an option to use libgcrypt (from GnuPG) and libcrypto (OpenSSL) in "FIPS mode" removing a need to modify the plugin for sites that require certification for compliance with FIPS 140-2. When the environment variable `$gtmencrypt_FIPS` is set to 1 (or evaluates to a non-zero integer, or any case-independent string or leading substring of "TRUE" or "YES"), the plugin reference implementation attempts to use either OpenSSL or Libgcrypt to provide database encryption that complies with FIPS 140-2. The supported platforms are as follows:

Platform	Libgcrypt	OpenSSL	OpenSSL FIPS
Linux x86_64	1.4.5	1.0.0	1.0.1e
Linux x86	1.4.5	1.0.0	1.0.1e
AIX RS600	1.5.1	1.0.0e	1.0.1e
SunOS SPARC	1.5.1	1.0.0j	1.0.1e
HP-UX IA64	1.4.1	1.0.1e	1.0.1e

Before using FIPS mode on these platforms, ensure that you have a configuration a valid FIPS 140-2 implementation an OpenSSL or Libgcrypt installation.



#### Note

Achieving FIPS 140-2 certification requires actions and controls well beyond the purview of GT.M, including underlying cryptographic libraries that are certifiably FIPS compliant, administrative controls, and so on. FIS neither provides cryptographic libraries with GT.M nor recommends the use of any specific library.

## Theory of Operation

This section describes the operation of GT.M database encryption with the reference implementation. A subsequent section describes the functions of the reference implementation which can be reworked or rewritten to use different encryption packages.

## Definition of Terms

Terms	Description
Cipher	An encryption algorithm or the implementation of an encryption algorithm, for example, the symmetric cipher AES 256 CFB.
Hash (or Fingerprint)	A signature algorithmically derived from an object which is certain to a very impressive probability that uniquely identifies an object within a set of similar objects.
Key length	The number of bits comprising a key. Longer key lengths may result in stronger encryption (more difficult to break) but require more computation.

## Database Encryption

Terms	Description
Key management	<p>The generation, distribution, and access of keys. The reference implementation of database encryption uses:</p> <ol style="list-style-type: none"> <li>1. symmetric keys to encrypt data and index records.</li> <li>2. public keys to encrypt symmetric keys (so they can be placed on disk).</li> <li>3. private keys to decrypt symmetric keys.</li> <li>4. passwords to encrypt private keys (so they can be placed on disk).</li> </ol>
Master key file	<p>This file contains pairs of entries indicating which symmetric key is used to encrypt/decrypt database records. Database records can be found in database, journal, extract and backup files.</p>
Obfuscation	<p>A technique used to make data difficult to discern on casual observation. A common example is "pig Latin". Since the password used for the GPG keyring exists in the process' environment with the reference implementation, GT.M obfuscates it to reduce the chance that visual access to process information (say during debugging) inadvertently exposes the password.</p>
Password (or Passphrase)	<p>A secret word or phrase used in the reference implementation to protect a private key on disk (a password should never be on disk in the clear, which is the electronic equivalent of taping it to your monitor with a sticky note).</p>
Public key / Private key (or Asymmetric keys)	<p>A pair of keys used so what one key encrypts the other can decrypt. The private key is sometimes referred to as the "secret" key (because it is not shared as opposed to the public key which is; the private key should never be on disk in the clear). In the reference implementation, asymmetric keys are used to encrypt the symmetric database key. This allows a master to encrypt a symmetric database key with a user's public key (so only the user can decrypt it with their private key).</p> <p>Encryption using a public key / private key pair is referred to as "public key encryption". The reference implementation uses GNU Privacy Guard with associated libraries libpgpme and libpgp-error for asymmetric key encryption.</p>
Symmetric key	<p>The same key used to both encrypt and decrypt. Symmetric ciphers are faster than asymmetric ciphers. Encryption using a symmetric key is referred to as "symmetric key encryption". Depending on the platform, the reference implementation uses either GNU Privacy Guard's libgcrypt, or libcrypto from OpenSSL, for symmetric key encryption.</p>

## Overview

### Warning

GT.M implements database encryption with a plug-in architecture that allows for your choice of cipher. Any code statically or dynamically linked in to a GT.M process must meet the requirements of code used for external calls. The GT.M distribution includes a reference implementation that interfaces to several common packages and libraries. You are free to use the reference implementations as is, but remember that the choice of cipher and package is yours, and FIS neither recommends nor supports any specific package.



### Note

In any given instance, you must use the same encryption libraries for all databases accessed by the processes of an application instance, but each database file can have its own key. Of course, all processes accessing a database or journal file must use the same encryption algorithm and key.

## Data in Database and Journal Files

A GT.M database file contains several parts:

1. A file header containing information pertaining to the database file itself.
2. Global and local bit maps, which together specify which blocks in the file are in use and which blocks are free.
3. Data blocks containing the actual data, as well as index blocks containing structural information providing paths to the actual data (there is a directory tree, and one or more global variable trees). Each data or index block consists of a block header, and one or more data records.

In an encrypted database, GT.M encrypts only the index and data records in a database. The file header, bit maps, and block headers are not encrypted, i.e., information relating to database structure is not encrypted. This means some system administration operations such as turning journaling on and off, do not require the encryption key for a database file. Others, such as MUPIP EXTRACT, do.

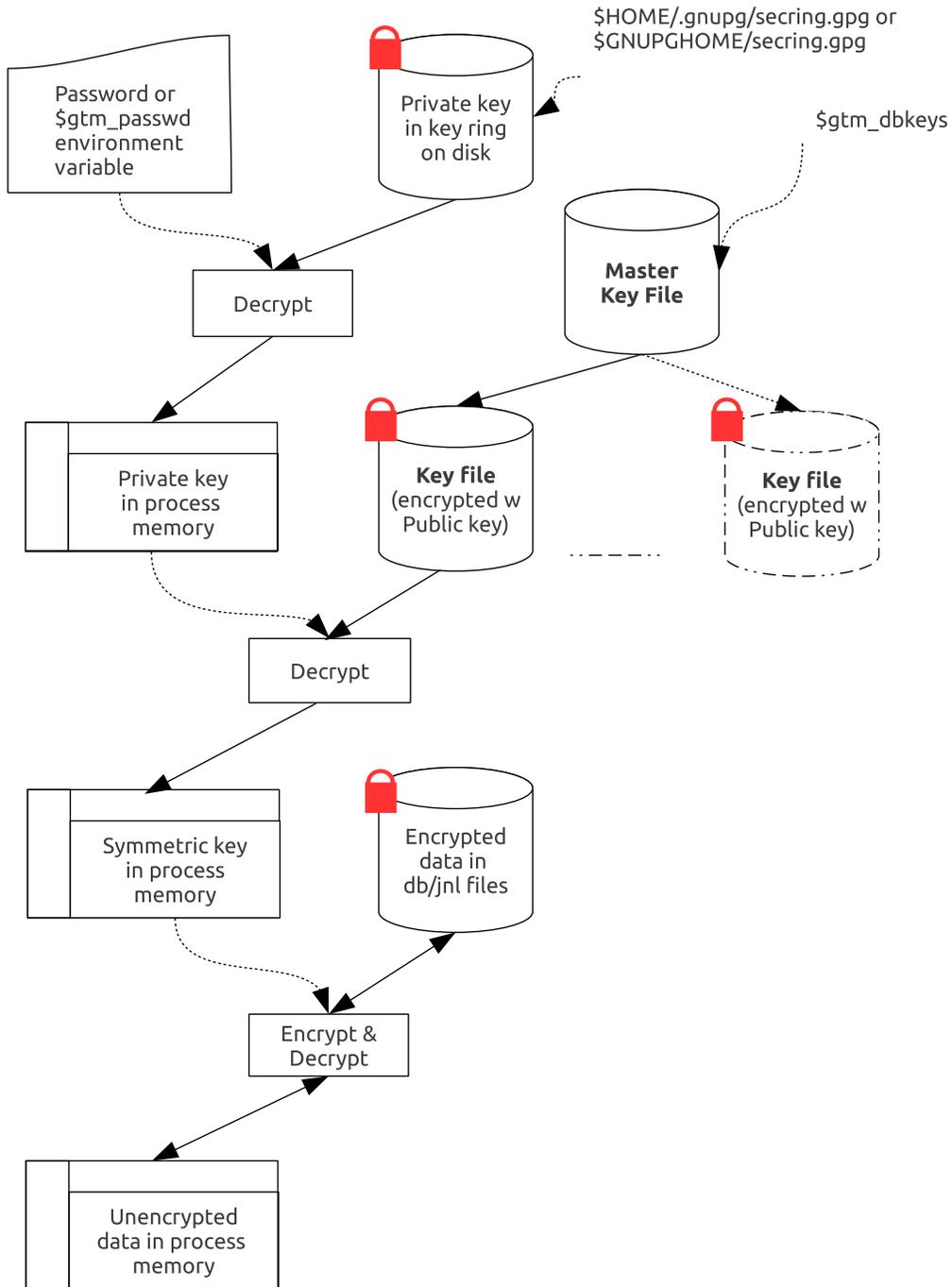
Journal files contain data records, such as before image records, update records, and after image records, as well as structural information such as transaction markers, process records, etc. Again, only records that contain data - before image records, update records and after image records - are encrypted. Records that contain structural information remain in cleartext.

Records subject to encryption are collectively referred to in the document as data records.

## Symmetric and Asymmetric Ciphers

For performance, a symmetric cipher is used to encrypt and decrypt data records. Asymmetric ciphers are used by the reference implementation to secure the symmetric cipher keys stored on disk. A password is used to secure the private key which is stored on a key ring on disk. The following illustration is an overview of GT.M database encryption in the reference implementation using GNU Privacy Guard (GPG) to provide the ciphers.

## Database Encryption



### Key Ring on Disk

In the reference implementation, a password protected key ring on disk contains the private key of the asymmetric cipher. A password is required to access the key ring on disk and obtain the private key. Password acquisition happens in one of three ways:

## Database Encryption

1. When the environment variable `$gtm_passwd` is not set, before a GT.M mumps process needs to open an encrypted database file, the application calls a program such as `GETPASS.m` to prompt for and obtain a password for the key ring on disk.
2. When the environment variable `$gtm_passwd` is set to the null string, at process startup, GT.M implicitly calls the program `GETPASS.m` to prompt for and obtain a password. The environment variable, `$gtm_passwd` is then set to an obfuscated version of the password required to unlock the key ring on disk.
3. The environment variable `$gtm_passwd` contains an obfuscated version of the password required to unlock the key ring on disk to obtain the private key. The environment variable can be passed in to GT.M, or it can be prompted for and set, as described below.

Some graphical user interfaces, e.g., GNOME or KDE, may detect when you are being prompted for the GPG keyring password and use a graphical interface instead of the terminal interface. You may be able to disable this behavior if you unset the `$DISPLAY` environment variable, or use an ssh connection to localhost that disables X forwarding. Consult your graphical user interface documentation.

In order to enable the Job command, the password for the key ring on disk exists in the environment of the process in environment variable `$gtm_passwd` where it can be passed from a parent process to a child. In order to prevent inadvertent disclosure of the password, for example, in a dump of the environment submitted to FIS for product support purposes, the password in the environment is obfuscated using information available to processes on the system on which the process is running, but not available on other systems.

`$gtm_passwd` is the only way for a child process to receive a password from a parent. In the event that the parent process does not pass `$gtm_passwd` to the child, or passes an incorrect password, there is little a child without access to an input device can do except log an error and terminate.

An obfuscated password in the environment is the only way that other GT.M processes (MUPIP and DSE) can be provided with a password. If they encounter an encrypted database or journal file, and do not have an obfuscated password to the key ring on disk in the environment, they terminate with the error message "GTM-E-CRYPTINIT, Error initializing encryption library. Environment variable `gtm_passwd` set to empty string. Password prompting not allowed for utilities". There are (at least) two ways to provide MUPIP and DSE processes with obfuscated passwords in `$gtm_passwd`:

1. `maskpass` is a stand-alone program that prompts the user for the password to the key ring on disk, and returns an obfuscated password to which `$gtm_passwd` can be set. The environment variable `$gtm_passwd` should be not set, set to a null value, or set to a value produced by `maskpass`. Setting `$gtm_passwd` to an incorrect non-null value without using `maskpass` could result in undefined behavior of the encryption library. You can use `maskpass` in shell scripts. For example:

```
$ echo -n "Enter Password: ";export gtm_passwd=`$gtm_dist/plugin/gtmcrypt/maskpass|cut -f 3 -d " "`
Enter Password:
$
```

2. Create a one line GT.M program as follows:

```
zcmd ZSystem $ZCmDline Quit
```

and use it invoke the MUPIP or DSE command. For example:

```
$ gtm_passwd="" mumps -run zcmd mupip backup -region \"*\\"
```

The empty string value of `$gtm_passwd` causes the MUMPS process to prompt for and set an obfuscated password in its environment which it then passes to the MUPIP program. Shell quote processing requires the use of escapes to pass the quotes from the `ZSystem` command to the shell.

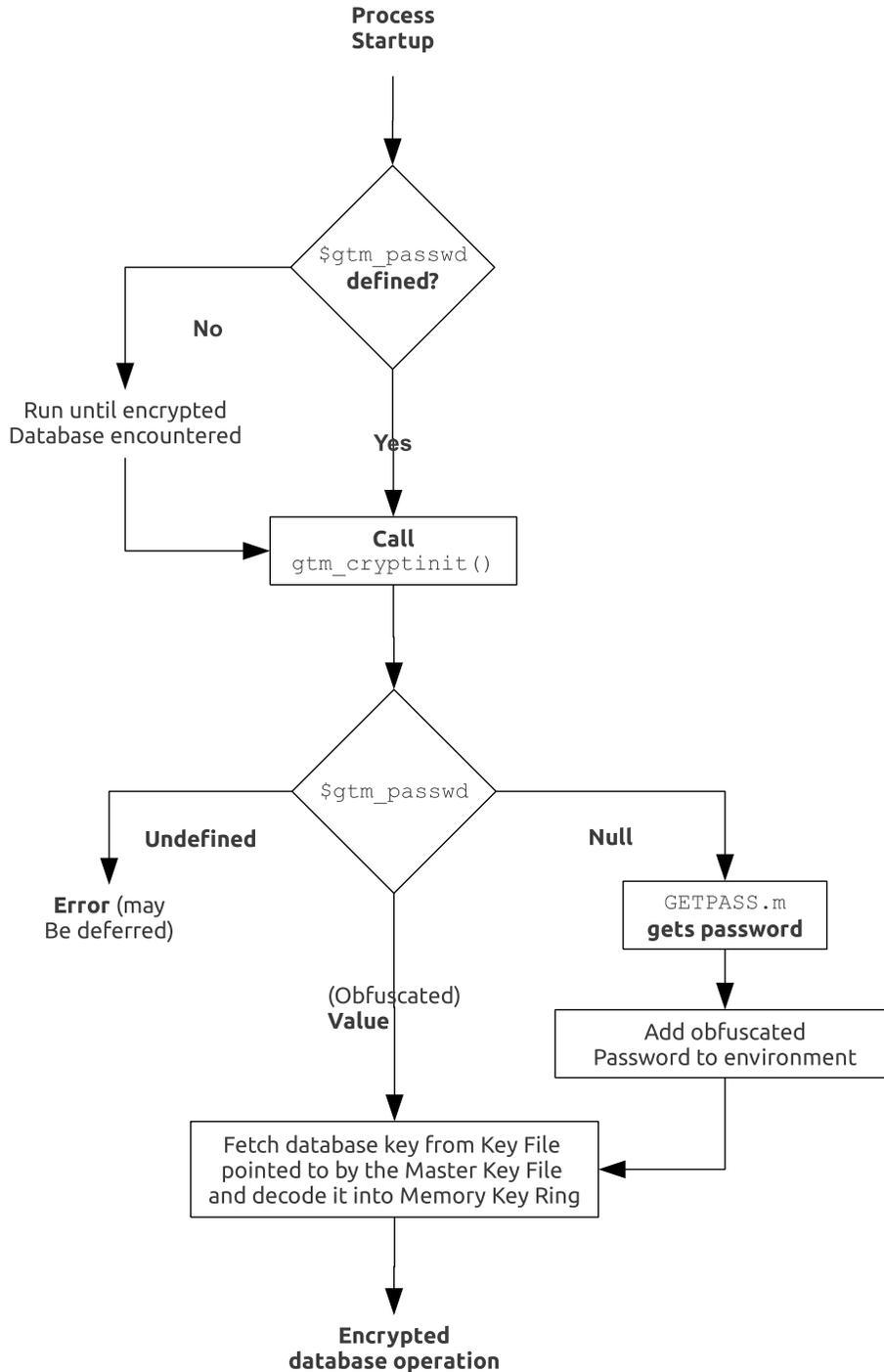
## Database Encryption

The environment variable `$gtm_passwd` should be one of the following:

- not set
- set to a null value
- set to a value corresponding to an obfuscated password (e.g., produced by `maskpass`)

The following schematic illustrates acquisition of the password for the key ring on disk. Note that an error (for example from the entry of an incorrect password) may not be triggered immediately - for example, DSE does not need an encryption key until you attempt to access data (since the file header is not encrypted, access to it does not require a key).

## Database Encryption



## Master Key File and Key Files

The reference implementation uses a master key file for each user to obtain the symmetric keys for each database or journal file. The environment variable `$gtm_dbkeys` specifies the master key file. If `$gtm_dbkeys` points to a file, it is the master key file. If it points to a directory, the file `.gtm_dbkeys` in that directory is the master key file (that is: `$gtm_dbkeys/.gtm_dbkeys`).

## Database Encryption

If the environment variable is not defined, the functions look for a key file `~/.gtm_dbkeys` (i.e., in the home directory of the process' userid). The master key file contains sections as follows:

**dat** *database\_filename*

**key** *key\_filename*

where *database\_filename* is the name of a database file, for example, `/var/xyzapp/gbls/accounts.dat` and *key\_filename* is the name of a key file containing a symmetric key encrypted with a public key, for example: `/home/sylvia/dbkeys/accounts.key`.

Key files are text files which can even be faxed or e-mailed: since they are secured with asymmetric encryption, you can transmit them over an insecure channel. As discussed below, the same *database\_filename* can occur multiple times in a master key file.

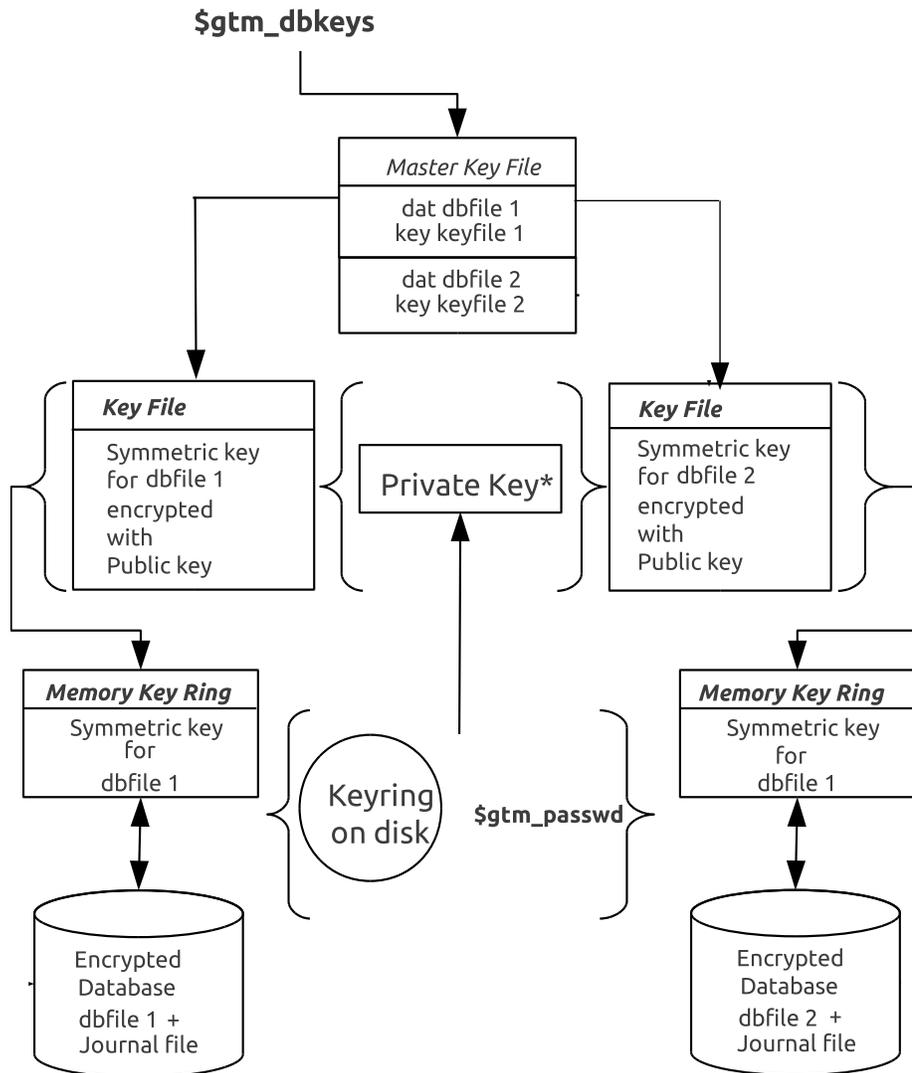
### Memory Key Ring

For each *key\_filename*, the GT.M process (MUMPS, MUPIP or DSE) builds a memory key ring from the key ring on disk and the master key file. The memory key ring contains a list of elements where each element consists of a filename, a symmetric cipher key, and a cryptographic hash of that symmetric cipher key. Using the private key obtained from the key ring on disk, GT.M obtains the symmetric keys from key files pointed to by the master key file.

Database and journal file headers include a cryptographic hash of the encryption key and algorithm used for that file. When opening a file, GT.M uses the key in the memory key ring whose hash matches that in the header - the *database\_filename* in the key ring is ignored. Older keys need not be deleted until they are no longer required (for example, an older key may be required to access a restored backup copy of a database). Permitting the same *database\_filename* to occur multiple times in a master key file also enables one master key file to be used for multiple instances of an application. This ensures that the correct key for a file is always used, even if the file has been renamed, copied from another location, etc. - the correct key must of course be available in the memory key ring; if no such key exists, GT.M triggers a `CRYPTKEYFETCHFAILED` error.

Only for MUPIP CREATE does GT.M rely on the *database\_filename* in the key ring. MUPIP CREATE computes the cryptographic hash for the correct key to place in the database file header. If the same *database\_filename* occurs more than once in the master key file (and hence in the memory key ring), MUPIP CREATE uses the *key\_filename* associated with the last occurrence of that *database\_filename* in the master key file.

This is illustrated by the following illustration:



\* Private Key allows the encrypted symmetric key in the Key File to be unencrypted in the Memory Key Ring.

## Key Validation and Hashing

As discussed earlier, a process uses that key in its memory key ring whose hash matches the hash in the database or journal file header; the file name is not checked. MUPIP CREATE computes the hash value for the key at database creation time, and writes it to the database file header. When GT.M creates a new journal file for an encrypted database file, it copies the hash from the

database file header into the journal file header. Similarly, `MUIP EXTRACT -FORMAT=BINARY`, places the database file hash in the extract, which is encrypted; indeed, since an extract can come from multiple database files, extract places the hash from the file header of each encrypted database in the extract. When processing each section in the extract, `MUIP LOAD` uses that key in its memory key ring that matches the hash for each section of the extract.

## Database Operation

On disk, database and journal files are always encrypted - `GT.M` never writes unencrypted data to an encrypted database or journal file. `GT.M` uses decryption when reading data records from disk, and encryption when it writes data records to disk.

With encrypted databases, the number of global buffers allocated is automatically doubled, for example, if the database file header specifies 2000 global buffers, when the file is opened, `GT.M` automatically allocates 4000 global buffers. Global buffers are used in pairs: one global buffer has a copy of the encrypted database block as it exists on disk and the other has a copy of the unencrypted version. There is no change to the size of the control structures (including lock space and journal buffers) in shared memory. So, when using encrypted databases, you need to adjust your calculations of memory and shared memory usage accordingly: for each open database file, the shared memory usage will increase by the number of global buffers times the block size. For example, if the block size of a database file is 4KB, with 2048 global buffers, and the shared memory segment for that database file occupies 9MB when unencrypted, it occupies 17MB when the file is encrypted. Depending on your operating system you may need to change system configuration and tuning parameters. Other than global buffers, there is no change to memory usage with encryption.

Encrypted databases consume additional CPU resources for encryption and decryption. Without detailed knowledge of the chosen algorithms, the application patterns and hardware configuration, it is not possible to predict whether this will be appreciable, and whether application throughput will be affected. As far as possible, FIS has attempted to engineer `GT.M` database encryption so that the additional CPU resources are consumed outside software critical sections. The intention is to minimize the impact of encryption on application throughput, at least on computer systems that are not starved of CPU resources. You should determine the actual impact of encryption on your application when it runs on your system, preferably using a test environment that exactly reflects your production environment.

---

## Examples of use

The commands here are all line oriented to illustrate that they can be automated by being called from `GT.M` or from a shell script. For interactive use, there are many graphical user interfaces (GUIs) usable with GPG. Although these examples were generated on Linux, usage on other UNIX systems should be virtually identical.

## Key Management

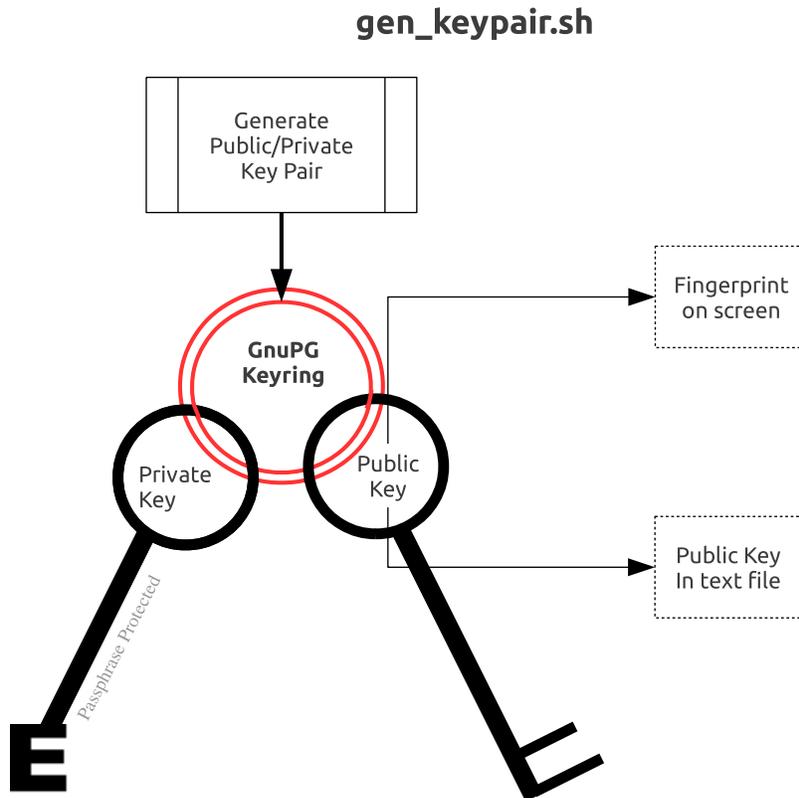
This is an example of key management using GPG and the reference implementation.

Helen Keymaster (`helen@gt.m`) is the master of keys, and provides a database key to Phil Keyuser (`phil@gt.m`). Helen does not manage the database. Phil is the database manager, but he is not the master of keys. In order to communicate securely, Helen and Phil each set up a GPG keyring, generate a public / private key pair, and exchange & authenticate each other's public keys. This permits a secure transfer of the key for the symmetric cipher used for the database. **Warning:** If you attempt key generation on a virtual machine, or other computer system that does not have a good supply of entropy, the `gen_key_pair.sh` script could take a very, very long time. Similarly, a key quality of 2 for the `gen_sym_key.sh` script on a machine without a plentiful supply of entropy can also tax your patience. Use a physical computer system with a lot of entropy. If you are able to, use an entropy gathering daemon such as `egd` (<http://egd.sourceforge.net>), or consider acquiring an entropy source such as the Entropy Key (<http://www.entropykey.co.uk>) that you can use to distribute entropy to your virtual machines.

The workflow is as follows:

## Database Encryption

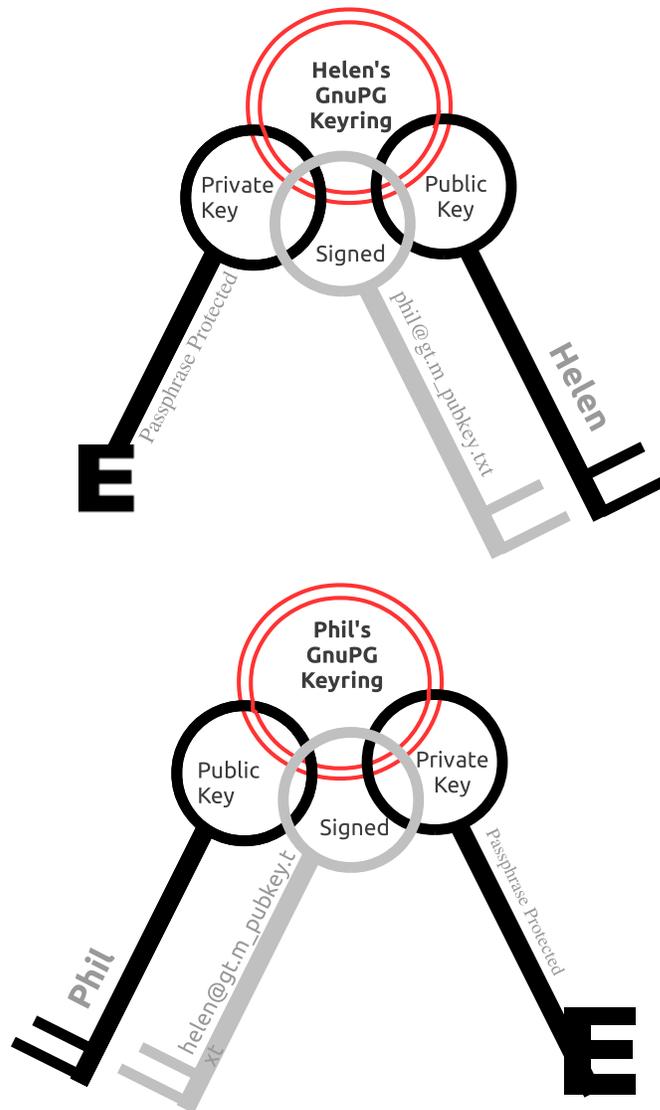
1. Helen and Phil each create a new GPG keyring and a new public-private key pair<sup>3</sup>. In the `gen_keypair.sh` script GPG generates the key pair<sup>4</sup>, putting public and private keys in the key ring; the latter locked with a passphrase. The public key is also exported to a text file, and its fingerprint is displayed in the terminal session. Each of them e-mails (or otherwise sends) her/his public key text file to the other<sup>5</sup>. This is illustrated below; first Helen, then Phil (if the `GNUPGHOME` environment variable is not set, it will default to `$HOME/.gnupg`).



## Database Encryption

- Helen e-mails `helen@gt.m_pubkey.txt` the file containing her exported public key to Phil, and Phil sends `phil@gt.m_pubkey.txt`, his exported public key to Helen. To protect against "man in the middle" attacks, they speak on the phone to exchange keyfingerprints, or send each other the fingerprints by text message, or facsimile - a different communication channel than that used to exchange the keys. Phil does likewise with Helen's key. They use the `import_and_sign_key.sh` shell script. After importing and signing each other's public keys, Phil and Helen can communicate securely with each other, even in the presence of eavesdroppers. Helen's keyring with Phil's imported key is shown below:

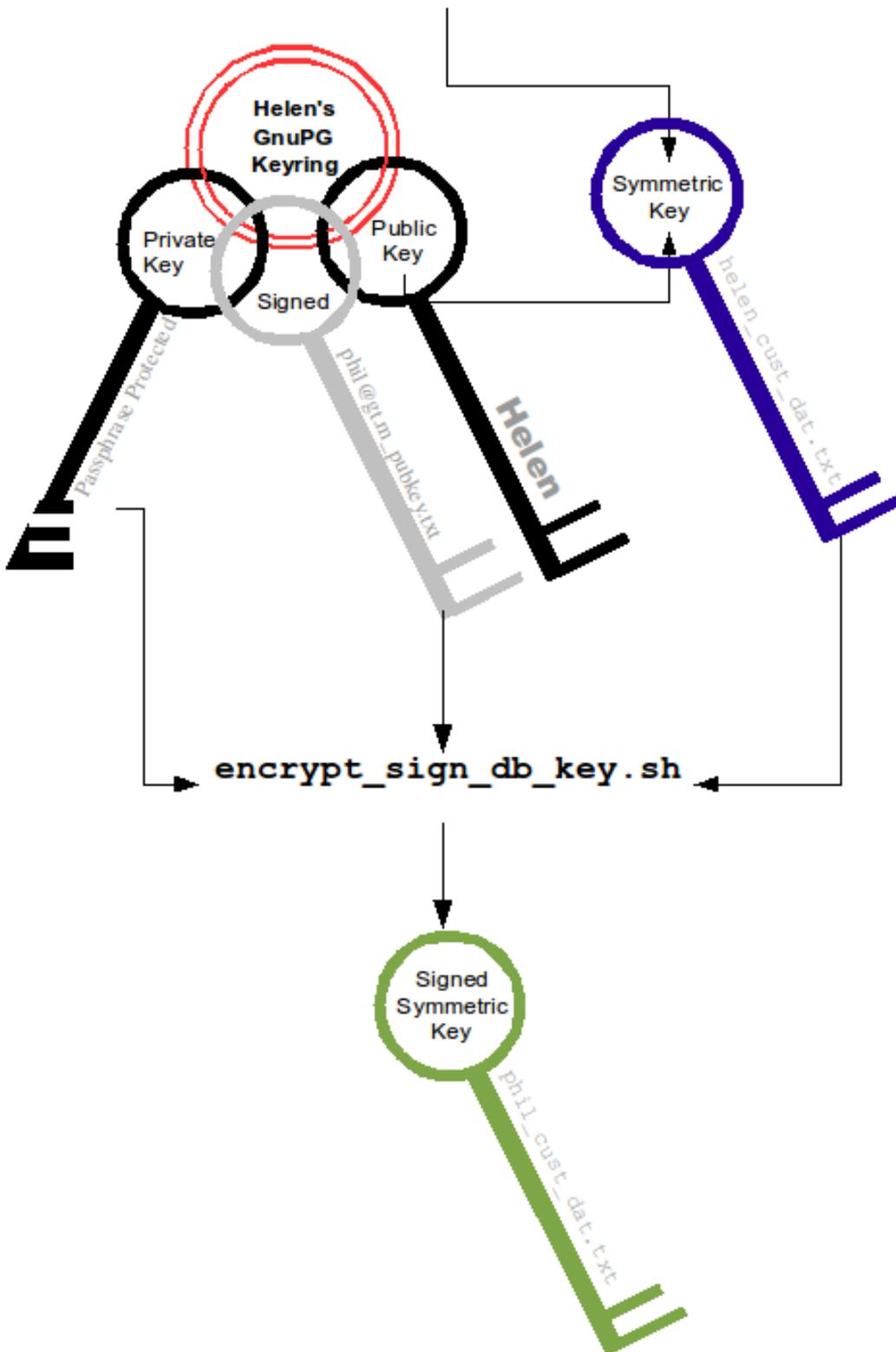
### import\_and\_sign\_key.sh



### Database Encryption

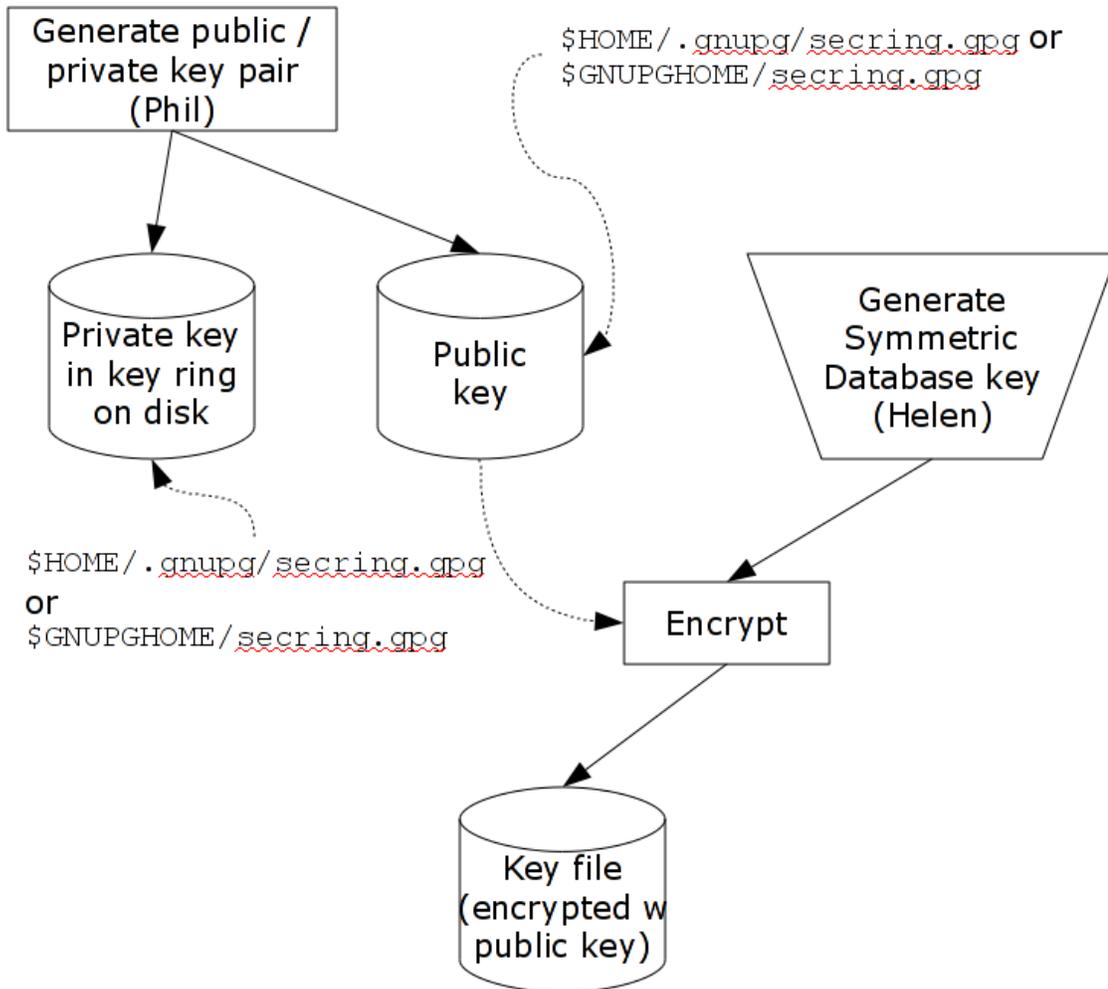
3. Using the `gen_sym_key.sh` script, Helen generates a symmetric cipher key for Phil to use in encrypting a new database file `cust.dat`. With a key strength of 2, a symmetric key is suitable for use in production and in the example is stored in file `helen_cust_dat.txt` encrypted with Helen's public key so that only she can decrypt it. The `gen_sym_key.sh` script never displays the symmetric cipher key; the key in the text file on disk can only be decrypted with Helen's private key.
4. With the `encrypt_sign_db_key.sh` script, Helen uses her private key to decrypt the symmetric cipher key in `helen_cust_dat.txt`, encrypts it with Phil's public key, and signs it with her private key, creating a file called `phil_cust_dat.txt`. She sends this file to Phil, either as an e-mail attachment, or putting it in a mutually agreed upon location on disk. As before, even though the key is on disk, it can be decrypted only with Phil's private key. Note that from this point on, even if Helen is hit by a truck, or resigns, Phil has access to the key and can use the same `encrypt_sign_db_key.sh` script to provide the key to, say, Xavier, Helen's successor. Helen preparing the key for Phil is shown below.

### gen\_sym\_key.sh



## Database Encryption

- With the `add_db_key.sh` script, Phil now adds the key to his GT.M master key file. He can then create the encrypted database file with `mupip create`, load it with data and use it. Until the database is created and loaded with data, the key has no value and can be discarded at will. Once the database is created and loaded with the data, the key must be retained as long as access to the database - or even a backup thereof - is ever required. The entire process is illustrated below:



- As a final check to make sure that the database was created with the correct symmetric cipher key and the correct cipher, Helen can use the `gen_sym_hash.sh` script to compute a hash from the key in `helen_cust_dat.txt` while Phil uses GT.M's `dse dump -fileheader -all` command to print the key from the file header of the database file he creates. If the hashes match, the database file has been correctly created.

Below are scripts of the key management example above.

Helen creates a new GPG keyring with a public and private key pair:

```
helen$ export GNUPGHOME=$PWD/.helengnupg
helen$ $gtm_dist/plugin/gtmcrypt/gen_keypair.sh helen@gt.m Helen Keymaster
Passphrase for new keyring:
Verify passphrase:
Key ring will be created in /home/helen/.helengnupg
```

## Database Encryption

```
Key generation might take some time. Do something that will create entropy, like moving the mouse or typing in another session.
gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0 valid: 1 signed: 0 trust: 0-, 0q, 0n, 0m, 0f, 1u
/home/helen/.helengnupg/pubring.gpg
-----
pub 1024D/BC4D0739 2010-05-07
Key fingerprint = B38B 2427 5921 FFFA 5278 8A91 1F90 4A46 BC4D 0739
uid Helen Keymaster <helen@gt.m>
sub 2048R/A2E8A8E8 2010-05-07
Key pair created and public key exported in ASCII to helen@gt.m_pubkey.txt
helen$
```



Phil creates a new GPG keyring with a public and private key pair:

```
phil$ export GNUPGHOME=$PWD/.philgnupg
phil$ $gtm_dist/plugin/gtmcrypt/gen_keypair.sh phil@gt.m Phil Keyuser
Passphrase for new keyring:
Verify passphrase:
Key ring will be created in /home/phil/.philgnupg
Key generation might take some time. Do something that will create entropy, like moving the mouse or typing in another session.
gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0 valid: 1 signed: 0 trust: 0-, 0q, 0n, 0m, 0f, 1u
/home/phil/.philgnupg/pubring.gpg
-----
pub 1024D/A5719A99 2010-05-07
Key fingerprint = 886A BAFC E156 A9AD 7EA9 06EA 8B8B 9FAC A571 9A99
uid Phil Keyuser <phil@gt.m>
sub 2048R/AD37D5A0 2010-05-07
Key pair created and public key exported in ASCII to phil@gt.m_pubkey.txt
phil$
```



Then Helen sends Phil the file helen@gt.m\_pubkey.txt and Phil sends Helen the file phil@gt.m\_pubkey.txt.

Helen imports Phil's public key into her keyring, verifying the fingerprint when she imports it, and signing it to confirm that she has verified the fingerprint:

```
helen$ $gtm_dist/plugin/gtmcrypt/import_and_sign_key.sh phil@gt.m_pubkey.txt phil@gt.m
gpg: key A5719A99: public key "Phil Keyuser <phil@gt.m>" imported
gpg: Total number processed: 1
gpg: imported: 1
#####
pub 1024D/A5719A99 2010-05-07
Key fingerprint = 886A BAFC E156 A9AD 7EA9 06EA 8B8B 9FAC A571 9A99
uid Phil Keyuser <phil@gt.m>
sub 2048R/AD37D5A0 2010-05-07
#####
```

## Database Encryption

```
Please confirm validity of the fingerprint above (y/n/[?]): y
Passphrase for keyring:
Successfully signed public key for phil@gt.m received in phil@gt.m_pubkey.txt
helen$
```

Phil likewise imports, verifies and sign's Helen's public key:

```
phil$ $gtm_dist/plugin/gtmcrypt/import_and_sign_key.sh helen@gt.m_pubkey.txt helen@gt.m
gpg: key BC4D0739: public key "Helen Keymaster <helen@gt.m>" imported
gpg: Total number processed: 1
gpg: imported: 1
#####
pub 1024D/BC4D0739 2010-05-07
Key fingerprint = B38B 2427 5921 FFFA 5278 8A91 1F90 4A46 BC4D 0739 uid Helen Keymaster <helen@gt.m>
sub 2048R/A2E8A8E8 2010-05-07
#####
Please confirm validity of the fingerprint above (y/n/[?]): y
Passphrase for keyring:
Successfully signed public key for helen@gt.m received in helen@gt.m_pubkey.txt
phil$
```

Helen and Phil can now securely exchange information.

Helen generates a symmetric cipher key for the new database file cust.dat:

```
helen$ $gtm_dist/plugin/gtmcrypt/gen_sym_key.sh 2 helen_cust_dat.txt
helen$
```

Then she encrypts the symmetric cipher key with Phil's public key, signs it, and produces a file phil\_cust\_dat.txt that she can send Phil:

```
helen$ $gtm_dist/plugin/gtmcrypt/encrypt_sign_db_key.sh helen_cust_dat.txt phil_cust_dat.txt phil@gt.m
Passphrase for keyring:
gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0 valid: 1 signed: 1 trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: depth: 1 valid: 1 signed: 0 trust: 1-, 0q, 0n, 0m, 0f, 0u
helen$
```

Phil adds the key in phil\_cust\_dat.txt to his master key file \$HOME/.gtm\_dbkeys:

```
phil$ export gtm_dbkeys=$HOME/.gtm_dbkeysphil$ $gtm_dist/plugin/gtmcrypt/add_db_key.sh $PWD/gtm.dat
phil_cust_dat.txt $gtm_dbkeys
phil$
```

Phil creates a global directory, where he changes the configuration parameter for the database file cust.dat specifying that it be encrypted the next time it is created. (Remember that except for mapping from global variable names to database file names, configuration parameters in the global directory are used only when MUIPIP creates new database files.) He then creates the database file, runs a DSE dump fileheader to extract the hash (highlighted in the output), and sends it to Helen for verification (notice that MUIPIP CREATE generates an error for the mumps.dat file that exists already, but creates a new encrypted cust.dat file):

```
phil$ export gtmgbldir=gtm.gld
phil$ export gtm_passwd=""
phil$ $gtm_dist/mumps -dir
Enter Passphrase:
GTM>zsystem "$gtm_dist/mumps -run GDE"
%GDE-I-LOADGD, Loading Global Directory file
```

## Database Encryption

```
/var/myApp/databases/gtm.gld
%GDE-I-VERIFY, Verification OK
GDE> change -segment DEFAULT -encryption
GDE> exit
%GDE-I-VERIFY, Verification OK
%GDE-I-GDUPDATE, Updating Global Directory file
/var/myApp/databases/gtm.gld
```

```
GTM>zsystem "$gtm_dist/mupip create"
Created file /var/myApp/databases/gtm.dat
Error opening file /var/myMpp/databases/mumps.dat
```

: File exists

%GTM-F-DBNOCRE, Not all specified database files, or their associated journal files were created

```
GTM>zsystem "dse"
```

```
File /var/myApp/databases/cust.dat
Region CUST
```

```
DSE> dump -fileheader -all
```

```
File /var/myApp/databases/cust.dat
Region CUST
Date/Time 04-MAY-2010 11:24:10 [$H = 61850,41050]
Access method BG Global Buffers 1024
Reserved Bytes 0 Block size (in bytes) 1024
Maximum record size 256 Starting VBN 129
Maximum key size 64 Total blocks 0x00000065
Null subscripts NEVER Free blocks 0x00000062
Standard Null Collation FALSE Free space 0x00000000
Last Record Backup 0x0000000000000001 Extension Count 100
Last Database Backup 0x0000000000000001 Number of local maps 1
Last Bytestream Backup 0x0000000000000001 Lock space 0x00000028
In critical section 0x00000000 Timers pending 0
Cache freeze id 0x00000000 Flush timer 00:00:01:00
Freeze match 0x00000000 Flush trigger 960
Current transaction 0x0000000000000001 No. of writes/flush 7
Maximum TN 0xFFFFFFFFE3FFFFFF Certified for Upgrade to V5
Maximum TN Warn 0xFFFFFFFF73FFFFFF Desired DB Format V5
Master Bitmap Size 112 Blocks to Upgrade 0x00000000
Create in progress FALSE Modified cache blocks 0
Reference count 1 Wait Disk 0
Journal State DISABLED
Mutex Hard Spin Count 128 Mutex Sleep Spin Count 128
Mutex Spin Sleep Time 2048 KILLS in progress 0
Replication State OFF Region Seqno 0x0000000000000001
Zqgblmod Seqno 0x0000000000000000 Zqgblmod Trans 0x0000000000000000
Endian Format LITTLE Commit Wait Spin Count 16
Database file encrypted TRUE

Dualsite Resync Seqno 0x0000000000000001 DB Current Minor Version 8
Blks Last Record Backup 0x00000000 Last GT.M Minor Version 8
Blks Last Stream Backup 0x00000000 DB Creation Version V5
Blks Last Comprehensive Backup 0x00000000 DB Creation Minor Version 8

Total Global Buffers 0x00000400 Phase2 commit pid count 0x00000000
Dirty Global Buffers 0x00000000 Write cache timer count 0xFFFFFFFF
Free Global Buffers 0x00000400 wcs_wtstart pid count 0x00000000
Write Cache is Blocked FALSE wcs_wtstart intent cnt 0x00000000
```

## Database Encryption

```

Actual kills in progress          0 Abandoned Kills          0
Process(es) inhibiting KILLS     0

DB Trigger cycle of ^#t         0

MM defer_time                    0
Database file encryption hash 12D119C93E28BBA9389C6A7FD53C2373CFF7181DF48FEF
213523B7B38199EF18B4BADB232D30CBDA2DBFC5F85D97D7A5C4A3E3D13276DCBB63B30EBDAA6B5
DD7

Full Block Writes                OFF Full Block Write Len      0

TP blkmod nomod                  0
TP blkmod gvcst_srch             0
TP blkmod t_qread                0
TP blkmod tp_tend                0
TP blkmod tp_hist                0

Free blocks                       992 Backup blocks              0
Reformat blocks                   0 Total blocks                992
Shmpool blocked                   FALSE File Offset                0x0000000000000000
Shmpool crit holder               0 Backup_errno                0
Backup Process ID                 0 Backup TN                   0x0000000000000000
Inc Backup TN 0x0000000000000000 Process Failed                0
Allocs since check                0 Backup Image Count          0
Temp File:

Database is Fully Upgraded        : TRUE
Database WAS ONCE Fully Upgraded from V4 : TRUE
Blocks to Upgrade subzero(negative) error : 0x00000000
TN when Blocks to Upgrade last became 0 : 0x0000000000000000
TN when Desired DB Format last changed : 0x0000000000000000
TN when REORG upgrd/dwngrd changed dbfmt : 0x0000000000000000

Block Number REORG upgrd/dwngrd will restart from : 0x00000000

Upd reserved area [% global buffers] 50 Avg blks read per 100 records 200
Pre read trigger factor [% upd rsrvd] 50 Upd writer trigger [%flshTrgr] 33

Snapshot in progress              FALSE Number of active snapshots      0
Snapshot cycle                    0 Active snapshot PID              0
Snapshot TN                       0 Total blocks                     0
Free blocks                        0 Process failed                    0
Failure errno                      0 Snapshot shared memory identifier -1
Snapshot file name

DSE> exit

GTM>halt
$

```

Phil calls Helen with the hash, texts her mobile, or sends e-mail. Helen ensures that the hash of the key she generated matches the hash of the database file created by Phil, and communicates her approval to Phil. Phil can now use the database. Either Phil or Helen can provide the key to other users who are authorized to access the database and with whom they have securely exchanged keys.

```

helen$ $gtm_dist/plugin/gtmcrypt/gen_sym_hash.sh helen_cust_dat.txt

Passphrase for keyring:gpg: encrypted with 2048-bit RSA key, ID A2E8A8E8, created 2010-05-07"

```

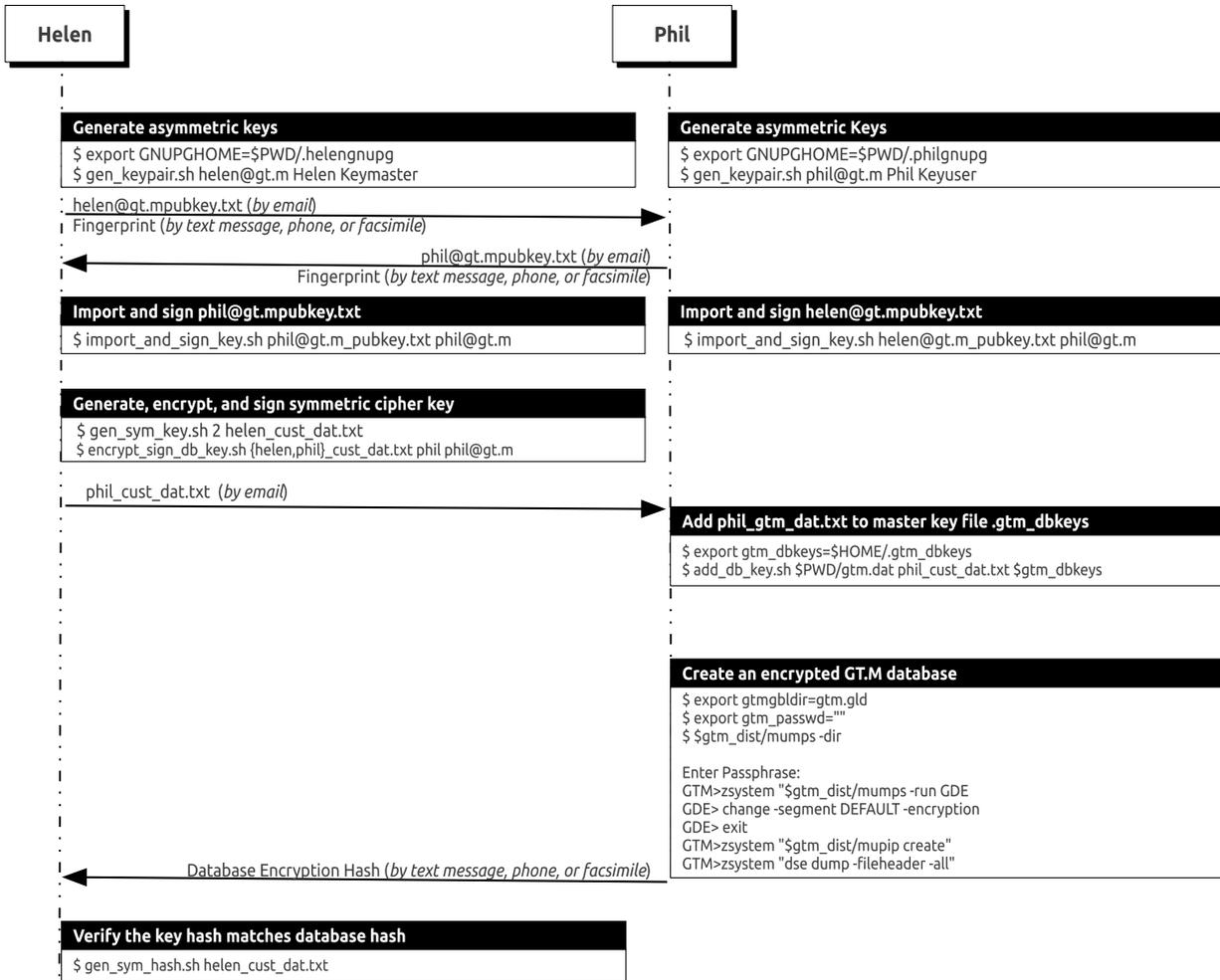
## Database Encryption

```
Helen Keymaster <helen@gt.m>"178E55E32DAD6BFF761BF917412EF31904C...  
helen$
```

The encrypted database file `cust.dat` is now ready for use. That file, all journal files, backups, and binary extracts will all have the same symmetric encryption cipher and key, which means that software libraries that provide that cipher and copies of the key (encrypted with the public keys of all those who are authorized to access them) must be retained as long as there may be any need to access data in that database file, its journal files, extracts and backups.

The following command sequence diagram illustrates how Helen and Phil operate with one another.

## Database Encryption



## Tested Reference Implementations

GT.M database encryption comes with a reference implementation that should work "out of the box" with selected encryption packages. You can use this for your initial development and testing with GT.M database encryption. There are many encryption packages. As discussed earlier, FIS neither endorses nor supports any specific cipher or package. For your production use, you

## Database Encryption

take responsibility for choosing, implementing and procuring support for your preferred package. Please remember that a malfunction in your chosen encryption package may result in unrecoverable data and FIS will be unable to help you.

The Plugin Architecture and Interface section below details the reference implementation, which is provided with full source code that you can freely modify for your own use.

For each platform on which GT.M supports encryption, the following table lists the encryption packages and versions against which FIS tested GT.M. Note that FIS tested GT.M for operation against these packages; FIS did not test the robustness of the encryption packages themselves.

OS (HW)	libgpgme	libgpg-error	libgcrypt / libcrypto	GPG
Ubuntu 12.04 LTS (x86_64)	1.2.0-1.2	1.6-1	libgcrypt 1.4.4-5	
RHEL 6 (x86_64)	1.1.6	1.4-2	libgcrypt 1.4.4-5	
RHEL 6 (x86)	1.1.6	1.4-2	libgcrypt 1.4.4-5	
AIX 6.1 and 7.1	1.1.8 + fix	1.7	libcrypto from OpenSSL - (version >= 1.5)  AES256CFB as implemented by OpenSSL - (version >= 0.9.8)	
Sun SPARC Solaris 10 (Update 6 and above)	1.1.4 + fix	1.6	libgcrypt 1.4.1	
HP-UX 11V3 (11.31)	1.1.8 + fix	1.7	libgcrypt 1.4.1	
SLES 11 (x86_64)	1.1.6-25.30	1.6-8.6	libgcrypt 1.4.4-2.2	

Where the table lists a package version number followed by "+ fix" it means that in the process of testing, we identified issues with the package that we fixed. We have provided the source code for our fixes to the upstream package maintainers. If you have a support agreement with FIS, we will share that source code with you, upon request.

The reference implementation uses:

- The key ring on disk implemented by GPG.
- For public key encryption including the generation of public/private key pairs: RSA as implemented by GPG.
- For the cryptographic hash: SHA-512.
- For a programmatic interface to GPG: libgpgme.
- To provide error messages for GPG: libgpg-error.
- For symmetric encryption: AES256CFB implemented by libgcrypt on all platforms.

When GT.M database encryption was first released with V5.3-004, the reference implementation for AIX was Blowfish CFB. At that time, there were certain limitations in libgcrypt as a consequence of the port of libgcrypt to the 64-bit environment being less mature than its port to the 32-bit environment (GT.M on AIX is a 64-bit application). Also, Blowfish was used because the implementation of AES on libcrypto from OpenSSL at that time required data to be in chunks that are multiples of 16 bytes. If you wish to continue using Blowfish CFB with V6.0-001 via the reference implementation of the plugin, you need to change a symbolic link post-installation, or define the environment variable `gtm_crypt_plugin` as follows:

## Database Encryption

- If the environment variable `gtm_crypt_plugin` is defined and provides the path to a shared library relative to `$gtm_dist/plugin`, GT.M uses `$gtm_dist/plugin/$gtm_crypt_plugin` as the shared library providing the plugin.
- If `$gtm_crypt_plugin` is not defined, GT.M expects `$gtm_dist/plugin/libgtmencrypt.so` to be a symbolic link to a shared library providing the plugin. The expected name of the actual shared library is `libgtmencrypt_cryptlib_CIPHER.so` (depending on your platform, the actual extension may differ from `.so`), for example, `libgtmencrypt_openssl_AESCFB`. GT.M cannot and does not ensure that the cipher is actually AES CFB as implemented by OpenSSL - GT.M uses CIPHER as salt for the hashed key in the database file header, and `cryptlib` is for your convenience, for example, for troubleshooting. Installing the GT.M distribution creates a default symbolic link.



### Note

To migrate databases from Blowfish CFB to AES CFB requires that the data be extracted and loaded into newly created database files. To minimize the time your application is unavailable, you can deploy your application in a Logical Multi-Site (LMS) configuration, and migrate using a rolling upgrade technique. Refer to the Chapter 7: “*Database Replication*” (page 173) for more complete documentation.

When a GT.M process first opens a shared library providing an encryption plugin, it ensures that the library resides in `$gtm_dist/plugin` or a subdirectory thereof. This ensures that any library implementing an encryption plugin requires the same permissions to install, and is protected by the same access controls, as the GT.M installation itself.

On all platforms on which GT.M supports encryption, the distribution includes three reference implementations of the plugin, provided by the shared libraries `libgtmencrypt_gcrypt_AES256CFB.so`, `libgtmencrypt_openssl_AES256CFB.so` and `libgtmencrypt_openssl_BLOWFISHCFB.so`. On installation, platforms other than AIX, `libgtmencrypt.so` is a symbolic link to `libgtmencrypt_gcrypt_AES256CFB.so`; on AIX symbolic link is to `libgtmencrypt_openssl_AESCFB.so`. To use Blowfish CFB as the default, you should change the symbolic link after installing GT.M. To use AES CFB as the default, but retain Blowfish CFB for databases migrated from V6.0-000, you can set `gtm_crypt_plugin` when using those databases. For scripts intended to be portable between V6.0-000 and V6.0-001, you can safely set a value for `gtm_crypt_plugin`, which V6.0-000 ignores.



### Note

- Encrypted database files are compatible between different endian platforms as long as they use the same key and the same cipher.
- The sample shell scripts in the reference implementation use the standard shell (`/bin/sh`).



### Caution

During development, in a core dump, FIS noticed a decrypted symmetric database key in buffer released by `libgpgme` despite the fact that GT.M made an appropriate call to the library to destroy the key. We have communicated this to the upstream developers. This emphasizes again the desirability of strong random numbers as database keys as well as the disabling of core dumps except when required. These strong keys can be created using the `gen_sym_key.sh` script described in the “Key Management” (page 376) section.

---

## Special note - Gnu Privacy Guard version 2

For the most part GPG version 2 is upward compatible with GPG version 1. An important difference between the two is that when GPG v2 invoked via GPGME (such as for example with a `JOB` command or a command such as `ZSYSTEM "mupip create"`),

## Database Encryption

there is not a convenient way to avoid invoking an "agent" (by default in `/usr/bin/pinentry`) that obtains the passphrase for the keyring from the user. When the reference implementation has placed an obfuscated password in the environment, the password should be derived from that obfuscated password, and the user should not be prompted for the password. The solution is to create a GT.M pinentry function (packaged in `pinentry-gtm.sh` and `pinentry.m`).

If you are using Gnu Privacy Guard version 2, you need to set the environment variable `GTMXC_gpgagent` to point to the location of `gpgagent.tab`. By default, GT.M places `gpgagent.tab` in the `$gtm_dist/plugin/` directory. `gpgagent.tab` is an external call table that `pinentry-gtm.sh` uses to create a GT.M pinentry function.

When the `gen_keypair.sh` script is executed, it creates a file `gpg-agent.conf` in the GnuPG directory (`~/.gnupg` or specified by the environment variable `$GNUPGHOME`) with the line such as `pinentry-program /usr/lib/fis-gtm/V5.4-001_x86/plugin/gtmcrypt/pinentry-gtm.sh` which causes `/usr/lib/fis-gtm/V5.4-001_x86/plugin/gtmcrypt/pinentry-gtm.sh` to be invoked by GnuPG v2 as the pinentry program. If the script finds the environment variable `$gtm_passwd` to be set, and as well as an executable GT.M, it runs the `pinentry.m` program which provides GnuPG v2 with the keyring password from the obfuscated password. Otherwise, it calls `/usr/bin/pinentry`.

If you are using GnuPG v2 with a `.gnupg` directory not created by `gen_keypair.sh`, you should create a `gpg-agent.conf` as described here, substituting the directory where GT.M is actually installed.

The GT.M pinentry function should not be used while changing the keyring passphrase, e.g., the `passwd` subcommand of the `gpg --edit-key` command. One way to do this is to temporarily comment out the `pinentry-program` line in `gpg-agent.conf` by placing a `"#"` in front of the line, e.g. `#!/usr/lib/fis-gtm/V5.4-001_x86/plugin/gtmcrypt/pinentry-gtm.sh`

GT.M versions prior to V5.4-001 are not compatible with GPG 2.x?.

---

## Installation

The normal GT.M installation script (invoked by `sh ./configure` executed as root or with `sudo sh ./configure` in the directory in which you have unpacked the GT.M distribution) will automatically install GT.M with the reference implementation plug-in.

If the encryption libraries are not part of the automatic search path on your system, you will need to take action specific to your operating system and directory structure to make them accessible. For example, you may need to set one of the environment variables `$LD_LIBRARY_PATH` or `$LIBPATH`, for example: `export LIBPATH="/lib:/usr/lib:/usr/local/lib"` and/or run the `ldconfig` command.

You must also implement appropriate key management, including ensuring that users have appropriate values for `$gtm_dbkeys`.

The `$gtm_dist/plugin` directory is configured such that you can copy it to a location of your choice to customize your own encryption plug-in. After testing, simply replace the directory as distributed with your customized directory.

The structure of the `$gtm_dist/plugin` directory on Linux x86 is as follows:

```
plugin/
|-- gpgagent.tab
|-- gtmcrypt
|   |-- add_db_key.sh
|   |-- encrypt_sign_db_key.sh
|   |-- gen_keypair.sh
|   |-- gen_sym_hash.sh
|   |-- gen_sym_key.sh
|   |-- gtmcrypt.tab
```

```

| |-- import_and_sign_key.sh
| |-- maskpass
| |-- pinentry-gtm.sh
| |-- pinentry.m
| |-- show_install_config.sh
| `-- source.tar
|-- libgtmencrypt_gcrypt_AES256CFB.so
|-- libgtmencrypt_openssl_AES256CFB.so
|-- libgtmencrypt_openssl_BLOWFISHCFB.so
|-- libgtmencrypt.so -> ./libgtmencrypt_gcrypt_AES256CFB.so
|-- o
`-- r

```

## Administration and Operation of Encrypted Databases

Utility programs written in M (such as %GO) run within mumps processes and behave like any other code written in M. Encryption keys are required if the mumps process accesses encrypted databases. A process running a utility program written in M that does not access encrypted databases (such as %RSEL) does not need encryption keys just to run the utility program.

Utility programs not written in M (e.g., MUPIP) that need access to encryption keys do not prompt for the password to the key ring on disk. They require the obfuscated password to be available in the environment. You can use the maskpass program to set the password in the environment or a mumps wrapper process as discussed earlier to set the obfuscated password in the environment. In some cases, if a required key is not supplied, or if an incorrect key is specified, the utility program defers reporting the error at process start up in case subsequent actions don't require access to encrypted data, and instead reports it when first attempting an encryption or decryption operation.

Since they do not access application data at rest, the GDE and LKE utilities do not need access to encryption keys to operate with encrypted databases.

MUPIP and DSE use the same plug-in architecture as mumps processes - gtmcrypt\_init() to acquire keys, gtmcrypt\_encrypt() to encrypt, etc.

## Utility Programs

### GDE

Since the global directory file is never encrypted, GDE does not need access to encryption keys.

### Format / Upgrade

The need to support encryption brings an upgrade to the global directory format, whether or not you use encryption. Simply opening an existing global directory with GDE and closing the program with an EXIT command upgrades the global directory.



### Important

FIS strongly recommends you make a copy of any global directory before upgrading it. There is no way to downgrade a global directory - you need to recreate it.

If you inadvertently upgrade a global directory to the new format and wish to recreate the old global directory, execute the SHOW ALL command with the new GT.M release and capture the output. Use the information in the SHOW ALL command to

create a new global directory file with the prior GT.M release, or better yet, create a script that you can feed to GDE to create a new global directory.

### -[NO]ENcryption

-[NO]ENcryption is a SEGMENT qualifier. When creating the database file for a segment that is flagged as encrypted, MUPIP CREATE acquires an encryption key for that file, and puts a cryptographic hash of the key in the database file header.

## MUPIP

Except for the following commands where it does not need encryption keys to operate on encrypted databases, MUPIP needs access to encryption keys to operate on encrypted databases: BACKUP -BYTESTREAM, EXIT, EXTEND, FTOK, HELP, INTRPT, REPLICATE, RUNDOWN, STOP. MUPIP looks for the password for the key ring on disk in the environment variable \$gtm\_passwd, terminating with an error if it is unable to get a matching key for any database, journal, backup or extract file that contains encrypted data.



### Note

MUPIP JOURNAL operations that only operate on the journal file without requiring access to the database - EXTRACT and SHOW - require only the key for the journal file, not the current database file key.

MUPIP SET operations that require stand-alone access to the database do not need encryption keys; any command that can operate with concurrent access to the database requires encryption keys.

All other MUPIP operations require access to database encryption keys.

MUPIP EXTRACT -FORMAT=ZWRITE or -FORMAT=GLO and MUPIP JOURNAL -EXTRACT are intended to produce readable database content, and produce cleartext output even when database and journal files are encrypted.

Since a MUPIP EXTRACT -FORMAT=BINARY extract file can contain encrypted data from multiple database files, the extract file contains the hashes for all database files from which extracted data was obtained.

An encrypted database cannot be downgraded to GT.M version 4 (V4) format.

## MUPIP CREATE

MUPIP CREATE is the only command that uses the database\_filename in the master key file to obtain the key from the corresponding key\_filename. As discussed elsewhere, all other commands use the key from the key ring in memory that matches the cryptographic hash for the encrypted data. If there are multiple files with the same file name, MUPIP CREATE uses the key specified in the last database\_filename entry with that name in the master key file.

## MUPIP JOURNAL

The MUPIP JOURNAL -SHOW command now displays the cryptographic hash of the symmetric key stored in the journal file header (the output is one long line):

```
$ mupip journal -show -backward mumps.mjl 2>&1 | grep hash
Journal file hash F226703EC502E9757848 ...
$
```

## MUPIP LOAD

Since an extract may contain the cryptographic hashes of multiple database files from which the data has been extracted, MUPIP LOAD may require multiple keys even to load one database file. Additionally, the database file into which the data is being loaded may have a different key from any data in the extract.

## DSE

Unless you are acting under the specific instructions of FIS GT.M support, please provide DSE with access to encryption keys by setting the value of `$gtm_passwd` in the environment.

DSE operations that operate on the file header (such as `CHANGE -FILEHEADER`) do not need access to database encryption keys, whereas DSE operations that access data blocks (such as `DUMP -BLOCK`) usually require access to encryption keys. However, all DSE operations potentially require access to encryption keys because if DSE is the last process to exit a database, it will need to flush dirty global buffers, for which it will need the encryption keys. DSE does not encrypt block dumps. There is a current misfeature, that access to the database key is needed to look at block 0 (a bitmap). In practical usage this is not a severe restriction since typically when a bitmap is examined data records are also examined (which require the key anyway).

Please remember that DSE is a low level utility for use by knowledgeable users, and does not check for reasonableness of commands and values.

The DSE `DUMP -FILEHEADER -ALL` command shows the database file header, including the encryption hash (the hash is a very long line):

```
$ dse dump -fileheader -all 2>&1 | grep hash
Database file encryption hash F226703EC502E9757848EEC733E1C3CABE5AC...
$
```

## Changing the hash in the database file header

Under normal operating conditions, you should not need to change the cryptographic hash of the symmetric key. However, since there are theoretical attacks against hashes, and because a new cryptographic hash standard (SHA-3) is under development as of this date, DSE provides the ability to change the hash of the password stored in the database file header if and when you change the hash library.

The DSE `CHANGE -FILEHEADER -ENCRYPTION_HASH` function hashes the symmetric key in the key file and replaces the hash in the database file header with this new value. The procedure to change the hash is:

- With the old hash function linked to your plug-in, ensure that the database is structurally sound with a `MUPIP INTEG`. Although changing the hash in the file header makes no change to any data block, you will have more confidence in your work, and easier troubleshooting in the event of subsequent problems, if you verify database wholesomeness before proceeding.
- Switch the plug-in to use the new hash function.
- Execute the DSE `CHANGE -FILEHEADER -ENCRYPTION_HASH` operation.
- Since recovery is not possible with a prior generation journal file with a different hash, if the database is journaled, create a new journal file without a back-pointer using the `MUPIP SET -JOURNAL -NOPREVJNL` command. FIS suggests backing up the database at this time.
- Verify the correctness of the new hash function by reading a global node or with a `DSE DUMP -BLOCK` command.

## Database Encryption

As there is no way to change the hash in a journal file header, make sure that you retain access to the hash packages used for any journal file as long as you want the data in old journal files to be accessible. These old journal files with different hashes cannot be used for database recovery. The data in them can, however, be accessed with a MUPIP JOURNAL -EXTRACT command by a MUPIP process using the old hash function.

## Changing Encryption Keys

The only way to change the encryption key of a database file is to extract the data and load it into a new database file created with a different key. Use a logical multi site (LMS) application configuration to change keys while keeping the application available. For example, if A is initially the initiating (primary) instance and B the replicating (secondary) instance:

1. Bring down instance B and change the database keys with EXTRACT and LOAD. Remember to save the journal sequence numbers in the original database files, and to set the journal sequence number in all the newly created database files to the largest number in any original database file.
2. Bring up instance B and let it catch up with A.
3. At a convenient time, switchover. Now application logic executes on B and A is the replicating instance.
4. Bring down instance A and change the database keys with either EXTRACT / LOAD or using a backup from B. Then bring it back up and let it catch up.
5. To restore the original operating configuration, switchover at a convenient time. Now A again executes application logic which is replicated to B.

FIS suggests using different encryption keys for different instances, so that if the keys for one instance are compromised, the application can be kept available from another instance whose keys are not compromised, while changing the encryption keys on the instance with compromised keys.

## Database Creation

Just as there is no way to change the encryption key of a database file, it is not possible to turn on encryption for an unencrypted database file, or to turn it off for an encrypted database file. Once a database file is created, its encryption attributes are immutable. To create an encrypted database, use GDE to specify encryption in the global directory file. Then use MUPIP CREATE to create an encrypted database and MUPIP LOAD to load data into it.

---

## Plugin Architecture & Interface

As noted in the Tested Reference Implementations, GT.M includes a reference implementation that uses widely available encryption packages. It is your choice: you can decide to use the packages that FIS tested GT.M against, or you can choose to interface GT.M to another package of your choice. As noted earlier, FIS neither recommends nor supports any specific package (not even those that we test against) and you should ensure that you have confidence in and support for whichever package that you intend to use in production. The reference implementation is provided compiled as ready to run object code as well as source code that you can customize to meet your needs.

Building the reference implementation from source code requires standard development tools for your platform, including C compiler, make, ld, standard header files, header files for encryption libraries, etc.

This section discusses the architecture of and interface between GT.M and the plugin. You must ensure that any plugin you provide presents the same interface to GT.M as the reference implementation.

## Packaging

The reference implementation by default resides in `$gtm_dist/plugin/gtmcrypt`.

The reference implementation includes:

1. A `$gtm_dist/plugin/gtmcrypt` sub-directory with all source files and scripts. The scripts include those needed to build/install `libgtmcrypt.so` and "helper" scripts, for example, `add_db_key.sh`. A brief description of these scripts is as follows:

<code>show_install_config.sh</code>	Reports the cryptographic library and cipher that a GT.M process would use, from <code>\$gtm_crypt_plugin</code> , if it has a value and otherwise from the name of the library linked to by <code>libgtmcrypt.so</code> .
<code>gen_sym_hash.sh</code>	Uses <code>show_install_config.sh</code> to identify the currently installed encryption configuration so that it can generate the appropriate cryptographic hash for the provided symmetric key.
<code>import_and_sign_key.sh</code>	Imports and signs one another's public keys.
<code>gen_sym_key.sh</code>	Generates a symmetric cipher key for others to use in encrypting a database file.
<code>encrypt_sign_db_key.sh</code>	Uses a private key to decrypt the symmetric cipher key, encrypts it with other's public key, and signs it with the private key.
<code>add_db_key.sh</code>	Adds a key to the master key file.

2. The plugin interface that GT.M expects is defined in `gtmcrypt_interface.h`. Never modify this file - it defines the interface that the plugin must provide.
3. `$gtm_dist/plugin/libgtmcrypt.so6` is the shared library containing the executables which is dynamically linked by GT.M and which in turn calls the encryption packages. If the `$gtm_dist/utf8` directory exists, then it should contain a symbolic link to `../plugin`.
4. Source code is provided in the file `$gtm_dist/plugin/gtmcrypt/source.tar` which includes the following scripts to compile and install `libgtmcrypt.so`.

<code>build.sh</code>	Compiles <code>libgtmcrypt.so</code> from the source code and has the following parameters: <ul style="list-style-type: none"> <li>• <code>gcrypt</code> or <code>openssl</code>: specifies the cryptographic library</li> <li>• <code>d</code> or <code>p</code>: Use <code>p</code> for production and <code>d</code> for debugging</li> <li>• <code>AES256CFB</code> or <code>BLOWFISHCFB</code> (optional): This optional parameter allows you to specify the cipher with which to build the encryption plugin. The default is <code>AES256CFB</code>.</li> </ul>
<code>install.sh</code>	Installs <code>libgtmcrypt.so</code> and has the following parameters: <ul style="list-style-type: none"> <li>• <code>gcrypt</code> or <code>openssl</code>: specifies the cryptographic library</li> <li>• <code>AES256CFB</code> or <code>BLOWFISHCFB</code> (optional): This optional parameter allows you to specify the cipher with which to build the encryption plugin. The default is <code>AES256CFB</code>.</li> </ul>

## Extensions to the GT.M External Interface

GT.M provides additional C structure types (in the `gtmxc_types.h` file):

## Database Encryption

1. `gtmencrypt_key_t` - a datatype that is a handle to a key. The GT.M database engine itself does not manipulate keys. The plugin keeps the keys, and provides handles to keys that the GT.M database engine uses to refer to keys.
2. `xc_fileid_ptr_t` - a pointer to a structure maintained by GT.M to uniquely identify a file. Note that a file may have multiple names - not only as a consequence of absolute and relative path names, but also because of symbolic links and also because a file system can be mounted at more than one place in the file name hierarchy. GT.M needs to be able to uniquely identify files.

Although not required to be used by a customized plugin implementation, GT.M provides (and the reference implementation uses) the following functions for uniquely identifying files:

1. `xc_status_t gtm_filename_to_id(xc_string_t *filename, xc_fileid_ptr_t *fileid)` - function that takes a file name and provides the file id structure for that file.
2. `xc_status_t gtm_is_file_identical(xc_fileid_ptr_t fileid1, xc_fileid_ptr_t fileid2)` - function that determines whether two file ids map to the same file.
3. `gtm_xcfileid_free(xc_fileid_ptr_t fileid)` - function to release a file id structure.

## Operation

Mumps, MUPIP and DSE processes dynamically link to the plugin interface functions that reside in the shared library. The functions serve as software "shims" to interface with an encryption library such as `libmcrypt` or `libpgpme / libgpcrypt`.

The plugin interface functions are:

1. `gtmencrypt_init()`
2. `gtmencrypt_getkey_by_name()`
3. `gtmencrypt_getkey_by_hash()`
4. `gtmencrypt_hash_gen()`
5. `gtmencrypt_encrypt()`
6. `gtmencrypt_decrypt()`
7. `gtmencrypt_close()`
8. and `gtmencrypt_strerror()`

A GT.M database consists of multiple database files, each of which has its own encryption key, although you can use the same key for multiple files. Thus, the `gtmencrypt*` functions are capable of managing multiple keys for multiple database files. Prototypes for these functions are in `gtmencrypt_interface.h`.

The core plugin interface functions, all of which return a value of type `xc_status_t` are:

- `gtmencrypt_init()` performs initialization. If the environment variable `$gtm_passwd` exists and has an empty string value, GT.M calls `gtmencrypt_init()` before the first M program is loaded; otherwise it calls `gtmencrypt_init()` when it attempts the first operation on an encrypted database file.
- Generally, `gtmencrypt_getkey_by_hash` or, for MUPIP CREATE, `gtmencrypt_getkey_by_name` perform key acquisition, and place the keys where `gtmencrypt_decrypt()` and `gtmencrypt_encrypt()` can find them when they are called.

## Database Encryption

- Whenever GT.M needs to decode a block of bytes, it calls `gtmdecrypt_decrypt()` to decode the encrypted data. At the level at which GT.M database encryption operates, it does not matter what the data is - numeric data, string data whether in M or UTF-8 mode and whether or not modified by a collation algorithm. Encryption and decryption simply operate on a series of bytes.
- Whenever GT.M needs to encrypt a block of bytes, it calls `gtmencrypt_encrypt()` to encrypt the data.
- If encryption has been used (if `gtmencrypt_init()` was previously called and returned success), GT.M calls `gtmencrypt_close()` at process exit and before generating a core file. `gtmencrypt_close()` must erase keys in memory to ensure that no cleartext keys are visible in the core file.

More detailed descriptions follow.

- `gtmencrypt_key_t *gtmencrypt_getkey_by_name(xc_string_t *filename)` - MUPIP CREATE uses this function to get the key for a database file. This function searches for the given filename in the memory key ring and returns a handle to its symmetric cipher key. If there is more than one entry for the given filename, the reference implementation returns the entry matching the last occurrence of that filename in the master key file.
- `xc_status_t gtmencrypt_hash_gen(gtmencrypt_key_t *key, xc_string_t *hash)` - MUPIP CREATE uses this function to generate a hash from the key then copies that hash into the database file header. The first parameter is a handle to the key and the second parameter points to 256 byte buffer. In the event the hash algorithm used provides hashes smaller than 256 bytes, `gtmencrypt_hash_gen()` must fill any unused space in the 256 byte buffer with zeros.
- `gtmencrypt_key_t *gtmencrypt_getkey_by_hash(xc_string_t *hash)` - GT.M uses this function at database file open time to obtain the correct key using its hash from the database file header. This function searches for the given hash in the memory key ring and returns a handle to the matching symmetric cipher key. MUPIP LOAD, MUPIP RESTORE, MUPIP EXTRACT, MUPIP JOURNAL and MUPIP BACKUP -BYTESTREAM all use this to find keys corresponding to the current or prior databases from which the files they use for input were derived.
- `xc_status_t gtmencrypt_encrypt(gtmencrypt_key_t *key, xc_string_t *inbuf, xc_string_t *outbuf)` and `xc_status_t gtmencrypt_decrypt(gtmencrypt_key_t *key, xc_string_t *inbuf, xc_string_t *outbuf)`- GT.M uses these functions to encrypt and decrypt data. The first parameter is a handle to the symmetric cipher key, the second a pointer to the block of data to encrypt or decrypt, and the third a pointer to the resulting block of encrypted or decrypted data. Using the appropriate key (same key for a symmetric cipher), `gtmencrypt_decrypt()` must be able to decrypt any data buffer encrypted by `gtmencrypt_encrypt()`, otherwise the encrypted data is rendered unrecoverable<sup>7</sup>. As discussed earlier, GT.M requires the encrypted and cleartext versions of a string to have the same length.
- `char *gtmencrypt_strerror()` - GT.M uses this function to retrieve additional error context from the plug-in after the plug-in returns an error status. This function returns a pointer to additional text related to the last error that occurred. GT.M displays this text as part of an error report. In a case where an error has no additional context or description, this function returns a null string.

The complete source code for reference implementations of these functions is provided, licensed under the same terms as GT.M. You are at liberty to modify them to suit your specific GT.M database encryption needs. Check your GT.M license if you wish to consider redistributing your changes to others.

---

## Using the V5.4-001 Reference Implementation with Older Releases

Since the interface between GT.M and the encryption libraries is stable, encrypted databases require no special upgrade procedures beyond any that may be required for ordinary databases when upgrading GT.M versions. Beyond an occasional need to recompile the plugin and rebuild the shared library, a plugin other than the reference implementation should also continue to operate with new GT.M versions. This stable interface also means that the encryption plugins from newer GT.M

versions can be retrofitted to older GT.M versions. This section describes interfacing the reference plugin bundled with V5.4-001 to an older GT.M version that supports encryption, such as V5.4-000A; adapt it to your needs should you wish to use a plugin from one version of GT.M with another version.



### Note

Since the reference plugin bundled with V5.4-001 supports GPG v2 as well as v1, and since its key management is more robust from that bundled with older versions of GT.M that support encryption, FIS suggests that you consider such a retrofit if you have chosen to deploy encrypted databases on older versions of GT.M with the reference implementation.

The instructions provided in this section are expert friendly and intended for someone familiar with UNIX/Linux and GT.M. All commands are examples, rather than intended to be executed as written. Adapt them to your specific needs and **DO NOT BLINDLY EXECUTE THEM VERBATIM**.

## Installation

To obtain the V5.4-001 reference implementation:

1. To go the GT.M project website on Source Forge.
2. Download the `gtmencrypt_V54001.tar.gz` file. from the V5.4-001 release of GT.M-x86-Linux-src.

`gtmencrypt_V54001.tar.gz` contains the source files for the reference plugin distributed with GT.M V5.4-001. (Depending on when you downloaded it, it may have a pre-release version.) Before you begin:

- Ensure that you have a working C compiler (`cc`).
- Ensure that you have root access.
- The instructions provided in this section are for installing the V5.4-001 prerelease reference implementation on Ubuntu Linux on an `x86_64` system. Other platform should be similar.

To install the new database encryption plug-in on an older GT.M release, perform the following steps:

1. Unpack the distribution files in a temporary directory, for example, `/tmp/tmp3` and change to the `src` sub-directory within.

```
mkdir /tmp/tmp3tar zxvf gtmencrypt_V54001.tar.gz -C /tmp/tmp3cd /tmp/tmp3/src
```

2. Run the `build.sh` script specifying the encryption library (`gcrypt` or `openssl`) and the build type (`d` for debug; `p` for production). The script needs the environment variable `$gtm_dist` to be defined, and, depending on the way ICU is compiled on your system, may well need `$gtm_icu_version` to be defined as well.

```
export gtm_dist=/usr/lib/fis-gtm/V5.4-000A_x86_64
export gtm_icu_version=4.2./build.sh gcrypt p
```

3. As root, make a backup copy of the existing plugin directory.

```
sudo cp -a $gtm_dist/plugin{,_sav_`date +%Y%m%d%H%M%S`}
```

4. As root, delete the Korn shell scripts and old source tarball in the existing plugin directory.

```
sudo rm -f $gtm_dist/plugin/gtmencrypt/{*.ksh,source.tar}
```

## Database Encryption

5. As root, run the install script (executing via sudo will not work here, since it does not pass the \$gtm\_dist environment variable to the shell script).

```
export gtm_dist=/usr/lib/fis-gtm/V5.4-000A_x86_64
./install.sh
```

6. Edit the gpgagent.tab file to specify the correct pathname to the libgtmdecrypt.so shared library, and as root, copy the distributed shell scripts and gpgagent.tab file.

```
cd ..
sudo cp *.sh gpgagent.tab $gtm_dist/plugin/gtmcrypt/
```

7. Set ownership and permissions appropriate to your installation.

```
sudo chown -R bin.gtm $gtm_dist/plugin
sudo chmod -R o-rwx,a-w $gtm_dist/plugin
```

8. Your new database encryption plugin should now be ready for use!

---

## Chapter 13. GT.CM Client/Server

Revision History		
Revision V5.5-000/6	19 July 2012	Corrected the example of starting a GT.CM server.
Revision V5.5-000/4	6 June 2012	First published version.

---

### Introduction

GT.CM is the network client/server database software for GT.M. GT.CM on UNIX allows access to GT.M databases residing on a server, by client processes running on multiple nodes on a network.

GT.CM consists of a Server and a Client. The Server is a network object that performs database operations on behalf of GT.M Client processes, running on other nodes of the network. GT.CM uses TCP task-to-task communication facilities for the link between a client GT.M process and a server. GT.CM on UNIX operates properly between supported platforms independent of the native byte ordering. The GT.CM client is packaged within the GT.M run-time system.

Since GT.CM on UNIX uses TCP whereas GT.CM on OpenVMS uses DECnet for communication, GT.CM operability between UNIX and OpenVMS is not supported.

When a GT.M process requires access to a database on another node, it sends a request across the network to a GT.CM Server process running on that node, and the Server accesses the database. The process requesting access to a database through GT.CM is referred to as a Client process. The node from which the data is requested is referred to as the Server node.

The use of GT.CM is largely transparent to the GT.M application code. The only visible change in the application's environment is the addition of error messages delivered in case of problems in network operations.



#### Note

GT.M transaction processing (TP) is not supported via GT.CM, and accessing GT.CM served databases within an M TRANSACTION may cause application level inconsistency in the data.



#### Note

GT.CM servers do not invoke triggers. This means that the client processes must restrict themselves to updates which don't require triggers, or explicitly call for the actions that triggers would otherwise perform. Because GT.CM bypasses triggers, it may provide a mechanism to bypass triggers for debugging or complex corrections to repair data placed in an inconsistent state by a bug in trigger logic.

In the event of recovery from system crashes, application level database consistency cannot be guaranteed for data residing in databases (M global variable namespaces) accessed via different GT.CM servers or distributed between GT.CM and local access.

---

### Overview

A GT.M program uses Global Directory to reference a global variable (gvn) or resource name for the object of a database lock operation (nref) residing on a remote node. When a file in the Global Directory specifies a remote node name that does not

match the name of the node on which the process is running, GT.M maps the segment to a database file on the remote node using the GT.CM client. The two main components of GT.CM are:

1. GT.CM Server and
2. GT.CM Client

## GT.CM Server

The GT.CM server accepts requests from GT.CM clients, processes the operation requested by the clients on the server database and sends messages back to the clients with a status and if appropriate, along with the results of the requested operation.

## GT.CM Client

The GT.CM client sends messages containing the operation type (SET, KILL, \$ORDER, etc), and operation specific data (eg. gvn to be SET, or KILLed) to the GT.CM server through a communication channel over a network. The client generally waits for response from the server and initiates error processing based on the status contained in the response. The format of the messages exchanged between the server and client is as defined by the FIS-developed GNP protocol.

If a client process needs to specify the port, and, optionally the TCP address to be used for the server side of the communication with a particular node, it must have an environment variable of the form `GTCM_ [<node-name>]` which defines the information in the form `[<IP address>]:<port number>`. For a server running on a machine with multiple IP addresses, if the IP address is not specified, GT.M uses the system default.



### Note

GT.CM can communicate between systems having different endian architectures.

## GT.CM Server Startup and Shutdown

This section describes the starting up and shutting down procedure of GT.CM server.

### GT.CM Server Startup

A GT.CM server must be operating on every node of a network from which data is requested during distributed database operation, including server nodes and nodes with both client and server processes. There are two ways by which the GT.CM server can be invoked.

1. By explicitly starting the GT.CM server to listen to a specified port number, or by defaulting the port number.
2. Invoking the GT.CM server to listen at a standard port number assigned to the GNP protocol (e.g., in `/etc/services` file).

The GT.CM server executable (`gtcm_gnp_server`) should be placed in the directory referenced by the environment variable `$gtm_dist`.

A process starting the GT.CM server must have the environment variables required to run GT.M.

Here is an example on how to start a GT.CM server:

```
$gtm_dist/gtcm_gnp_server -log=GTCM.log -service=6789
```

## GT.CM Client/Server

This starts the GT.CM server in the background so that it listens at port 6789 for requests from GT.CM clients. The detailed log information of the server is written in the GTCM.log file. If -log is not specified, log information is written in \$gtm\_log/gtm\_gnp\_server.log file. On nodes with multiple IP addresses issue the following command to configure the GT.CM server to listen at a port specific to an IP address:

```
-service=192.160.105.212:6789
```

## GT.CM Server Shutdown

To shutdown the GT.CM server, identify the process id of the GT.CM server to be shutdown and issue the following command:

```
$gtm_dist/mupip stop <GT.CM server PID>
```

This causes the GT.CM server to shutdown normally.

## Types of Operations

The GT.CM client sends messages to the GT.CM server requesting the type of operation to be performed.

GT.CM server can recognize the following types of operations and process the specified operations on the "local" database.

- SET
- KILL
- GET
- DATA
- ORDER
- REVERSE ORDER
- QUERY
- LOCK
- UNLOCK
- ZALLOCATE
- ZDEALLOCATE

The MERGE, SET \$PIECE() and SET \$EXTRACT() facilities are currently implemented by the client using the operations from the above set.

## Error Messages

Errors can be classified into the following categories:

- Database Errors
- Protocol Errors

- Session Establishment Errors

Each type of valid operation may issue an error from any of the above categories in case of a failure. Database errors include application errors and database integrity errors; both types of errors are detected by the GT.M runtime system. The GT.CM server does not deal with database errors directly, but passes them back to the client requesting the operation that detected the error. GT.M handles any database errors delivered through the network by GT.CM in a way similar to the way it treats errors detected when GT.CM is not involved.

When GT.CM is in use, GT.M may deliver errors resulting from network problems. Errors detected by the network interface are passed to the component accessing the interface at the time of error. In recovering from a network related error, GT.CM sacrifices all LOCKs owned by the client process that receives a network error. This should be taken into account if such a process attempts to resume operations involving a database served through the lost connection.

Examples of Database Errors:

```
Undefined global, Global reference content not valid.
```

Examples of Protocol Errors:

```
Message format not valid, Operation type not valid.
```

Examples of Session Establishment Errors:

```
GNP version not supported, GNP session not established.
```

## Examples

The following is an example illustrating the transparency of the GT.CM Client/Server Architecture while using GT.M.

On NODE1:

Map the local segment to remote file.

When the file specification in the Global Directory on the local node specifies a remote node name, GT.M maps the segment to a database on the remote node using GT.CM.

To specify a node name in a Global Directory file specification, use the format on NODE1:

```
$ GDE
GDE> ch -seg DEFAULT -file=NODE2:/testarea/gtm/database/data.dat
GDE> exit
```

This example creates a local Global Directory, mapping all global names to the database file /testarea/gtm/database/data.dat. Note that some of the key-words have been truncated to permit the example to appear on a single line.

On NODE2:

Create a database file on server Node2:

1. Change directory (cd) to the specified location (that is /testarea/gtm/database)
2. Create a global directory

```
$ GDE
```

```
GDE> change -segment DEFAULT -file=data.dat  
GDE> exit
```

3. Create the database file (data.dat).

```
$ mupip create
```

4. Start the GT.CM server.

Note that the global directory created on the remote node in this example is only used by mupip create, and never used by either the client or the server.

On NODE1:

On NODE1, invoke GT.M and perform the following operations:

```
$setenv GTCM_NODE2 6789  
$GTM  
GTM> s ^x=1  
GTM> k ^x  
GTM> s ^y=10  
GTM> h
```

All these updates should be reported in the NODE2:/testarea/gtm/database/data.dat file.

---

## Appendix A. GT.M's IPC Resource Usage

Revision History		
Revision V5.4-002B	24 October 2011	Conversion to documentation revision history reflecting GT.M releases with revision history for each chapter.

In GT.M's database engine, processes cooperate with one another to manage the database. To effect this cooperation, GT.M processes use UNIX Interprocess Communication (IPC) resources to coordinate and control access to the database and journal files, implement M LOCKs, and for database replication. This appendix helps you understand GT.M's IPC resource utilization.

This appendix includes two sections.

- The first section contains an exercise to identify "orphan" shared memory segments (those with no attached processes) in a multi-site replication configuration. Usually orphan segments appear due to certain abnormal terminations (for example, **kill -KILL** or a process crash situation) and create an out-of-design state for GT.M IPC resources where the last process may not be able to clean up the shared memory segment. If you know the IPC resources GT.M uses, then you can easily find the abnormal IPC resources when something does not seem right.
- The second section describes role of the **gtmsecshr** daemon process GT.M uses to manage M locks and clean up IPC resources. This section also includes guidelines to fine-tune **gtmsecshr** for smooth operation.

---

### Examining GT.M's IPC Resources

GT.M uses UNIX IPC resources as follows:

- For each database region, GT.M uses a shared memory segment (allocated with **shmat()**) for control structures and to implement M Locks. For journaled databases, the journal buffers reside in that shared memory segment. With the BG database access method, global buffers for the database also reside there. Note that use of the online help system by a process opens a database file with the BG access method. The first process to open a database file creates and initializes the shared memory segment, and the last process to exit normally cleans up and deletes the shared memory segment. However, under certain abnormal terminations of the last process (for example, if it is terminated with a **kill -KILL**), that last process may not be able to clean up the shared memory segment, resulting in "orphan" shared memory segments (those with no attached processes).
- For database regions which use the MM access method, the file system manages an additional shared memory segment (allocated with **mmap()**) to memory map the database file. GT.M does not explicitly allocate this shared memory. Because UNIX allocates shared memory segment when GT.M opens a database file, and releases it when the process terminates, such shared memory segments allocated by **mmap()** are never orphaned.
- When replicating, GT.M implements the journal pool on the primary in a shared memory segment. On the secondary, GT.M uses a shared memory segment for the receive pool.
- GT.M operations such as creating a shared memory segment for a given database file should be performed only by one process even if more than one process opens the database file at the same time. GT.M uses sets of public UNIX semaphores to insure these operations are single-threaded. GT.M uses other sets of public semaphores to setup and tear down shared memory segments allocated with **shmat()**.

## GT.M's IPC Resource Usage

- Public semaphore ids may be non-unique. Private semaphore ids are always unique for each database.
- The semaphore with keys starting **0x2b** and **0x2c** are startup and rundown semaphores. A GT.M process uses them only while attaching to or detaching from a database.
- The number of processes and the number of semaphore attached to an IPC resource may vary according to the state of your database. Some shared memory regions have **0** processes attached to them (the **nattch** column). If these correspond to GT.M database regions or to global directories, they are most likely from improper process termination of GT.M (GT.M processes show up as "**mumps**" in a **ps** command) and GT.M utility processes; source server, receiver server, or update processes (which appear as "**mupip**"); or other GT.M utilities ("**mupip**", "**dse**", or "**lke**").
- An instance has one journal pool, and, if a replicating instance, one receiver pool too. Note that you might run multiple instances on the same computer system.
- For simple GT.M operation (that is, no multisite replication), there is no journal pool or receive pool.

The following exercise demonstrates how GT.M utilizes IPC resources in a multisite database replication configuration. The task is to set up a replicated GT.M database configuration on two servers at 2 different locations.

The steps of this task are as follows:

1. Create 2 databases-**America** and **Brazil**-on 2 different servers ( **Server\_A** and **Server\_B**) and deploy them in a multisite database replication configuration so that **America** is the primary site and **Brazil** is the secondary site. Ensure that no GT.M processes exist on either server.
2. In **Server\_A** and in the directory holding database files for **America** give the following commands (note that because the default journal pool size is 64MB, a value of 1048576 bytes - GT.M's minimum size of 1MB for this exercise):

```
$ export gtm_repl_instance=multisite.repl
$ mupip replicate -instance_create -name=America
$ mupip set -replication-on -region "*"
$ mupip replicate -source -start -buf=1048576 -secondary=Server_B:1234 -log=A2B.log -instsecondary=Brazil
```

3. Now execute the following command:

```
$ gtm_dist/ftok mumps.dat multisite.repl
```

4. This command produces the "public" (system generated) IPC Keys (essentially hash values) for mumps.dat and its replication instance **multisite.repl**. It produces a sample output like the following:

```
mumps.dat :: 721434869 [ 0x2b0038f5 ]
multisite.repl :: 721434871 [ 0x2b0038f7 ]
```

5. The keys starting with **0x2b** (Hexadecimal form) are the keys for the semaphores used by replication instance **America** with the high order hexadecimal **0x2b** replaced by **0x2c** for the replication instance file (GT.M's standard prefix for semaphores for journal pools is **0x2c** and that for database files is **0x2b**). You can observe this with the **ipcs** command:

```
----- Semaphore Arrays -----
key      semid  owner   perms nsems
0xd74e4524 196608 welsley 660   1
0x2c0038f7 983041 welsley 777   3
0x00000000 1015810 welsley 777   5
0x2b0038f5 1048579 welsley 777   3
0x00000000 1081348 welsley 777   3
```



**Note**

You can expect files in separate file systems to share the same public **ftok**. This is a normal behavior for large systems with multiple installations and does not affect GT.M operations in any way. This is because GT.M does not assume the semaphore has a one-to-one relationship with the resource and startup/shutdown operations are relatively rare, so the interference among resources have a minimal or no impact. However, the private semaphore (with the **0** key) is unique for a database and is used while a process is actively using the resource.

- Execute the following command and note down the **shared memory id** and **private semaphore id** on instance America.

```
$ mupip ftok mumps.dat
```

This command identifies the "private" (GT.M generated) semaphores that a process uses for all "normal" access. The sample output of this command looks like the following:

File	Semaphore Id	Shared Memory Id	FileId
mumps.dat	1081348 [0x00108004]	2490370 [0x0026002]	0xf538030000000000fe00000000000fffffd2

- Now, execute the following command and note down the **shared memory** and **private semaphore id** for journal pool.

```
$ mupip ftok -jnl multisite.repl
```

The sample output of this command looks like the following:

File	Semaphore Id	Shared Memory Id	FileId
multisite.repl	1015810 [0x000f8002]	2457601 [0x00258001]	0xf738030000000000fe00000000000fffffd2

Note that the **Semaphore id 1015810** and **Shared Memory ID 2457601** are in the sample output of the **ipcs -a** command below.

- Now execute the command **ipcs -a** to view the current IPC resources. This command produces an output like the following:

```
----- Shared Memory Segments -----
key      shmid   owner    perms  bytes  nattch status
0x00000000 0        root     777   122880  1
0x00000000 2457601 welsley  777   1048576 1
0x00000000 2490370 welsley  777   2633728 1
0x00000000 2523139 welsley  600   393216  2      dest
0x00000000 2555908 welsley  600   393216  2      dest
0x00000000 1048583 welsley  600   393216  2      dest
0x00000000 1081352 welsley  600   393216  2      dest
0x00000000 1114121 welsley  666   376320  2
0xd74e4524 1146890 welsley  660   64528   0
0x00000000 1933323 welsley  666   62500   2
0x00000000 1966092 welsley  666   1960000 2

----- Semaphore Arrays -----
key      semid   owner    perms  nsems
0xd74e4524 196608  welsley  660   1
0x2c0038f7 983041  welsley  777   3
0x00000000 1015810 welsley  777   5
0x2b0038f5 1048579 welsley  777   3
```



```
$ mupip ftok -jnl multisite1.repl
```

The sample output of this command looks like the following:

```
File      :: Semaphore Id      :: Shared Memory Id      :: FileId
-----
multisite1.repl :: 262145 [0x00040001] :: 11632641[0x00b18001]:: 0xd1e6340000000000a00000000000000000
```

Note that the **Semaphore id 262145** and **Shared Memory ID 11632641** are in the sample output of the **ipcs -a** command below.

- Now, execute the command **ipcs -a** to view the IPC resources of Brazil.

This command produces a sample output like the following:

```
----- Shared Memory Segments -----
key      shmid   owner   perms bytes  nattch status
0x00000000 11632641 gtmuser 777   1048576 3
0x00000000 11665410 gtmuser 777   2629632 2
0x00000000 11698179 gtmuser 777   1048576 2

----- Semaphore Arrays -----
key      semid   owner   perms nsems
0x2c0ae6d1 229376  gtmuser 777   3
0x00000000 262145  gtmuser 777   5
0x2b0ae6cf 294914  gtmuser 777   3
0x00000000 327683  gtmuser 777   3
0x00000000 360452  gtmuser 777   5

----- Message Queues -----
key      msqid   owner   perms used-bytes messages
```

**Brazil** has 1 region and its receiver server is listening to **America**, therefore as per the formula for calculating GT.M IPC resources, the total IPCs utilized by GT.M is:  $5 [1 * 3 + 1 + 1]$ .

## gmtsecshr

The GT.M installation script installs **gmtsecshr** as owned by root and with the **setuid bit on**. **gmtsecshr** is a helper program that enables GT.M to manage interprocess communication and clean up interprocess resources. It resides in the **\$gtm\_dist/gtmsecshrdir** subdirectory which is readable and executable only by root. **gmtsecshr** is guarded by a wrapper program. The wrapper program protects **gmtsecshr** in the following ways:

- It restricts access to **gmtsecshr** in such a way that processes that do not operate as root cannot access it except through the mechanism used by the wrapper.
- Environment variables are user-controlled input to **gmtsecshr** and setting them inappropriately can affect system operation and cause security vulnerabilities. While **gmtsecshr** itself guards against this, the wrapper program provides double protection by clearing the environment of all variables except **gtm\_dist**, **gtmdbglvl**, **gtm\_log**, and **gtm\_tmp** and truncating those when they exceed the maximum allowed length for the platform.
- gmtsecshr** logs its messages in the system log. These messages can be identified with the GTMSECSHR facility name as part of the message. GT.M processes communicate with **gmtsecshr** through socket files in a directory specified by the environment variable **gtm\_tmp**.

**gtmsecs**hr automatically shuts down after 60 minutes of inactivity. Normally, there is no need to shut it down, even when a system is making the transition between a secondary and a primary. The only occasions when **gtmsecs**hr must be explicitly shut down are when a GT.M version is being removed - either a directory containing the GT.M version the running **gtmsecs**hr process belongs to is being deleted, or when a new GT.M version is being installed in the same directory as an existing one.



### Note

FIS strongly recommends against installing a new GT.M version on top of an existing GT.M version.

To terminate a **gtmsecs**hr process, use a **KILL-15** *after* shutting down *all* GT.M processes *and* running down *all* database regions in use by GT.M in that directory.



### Important

FIS strongly recommends that all GT.M processes that use a given version of GT.M use the same settings for the **gtm\_log** and **gtm\_tmp** environment variables. **gtmsecs**hr inherits these values from the GT.M process that starts it. Not having common values for **gtm\_tmp** and **gtm\_log** for all processes that use a given version of GT.M can have an adverse impact on performance.

If there are multiple GT.M versions active on a system, FIS recommends different values of **gtm\_tmp** and **gtm\_log** be used for each version. This makes system administration easier.



### Caution

A given database file can only be open by processes of a single version of GT.M at any given time. Contemporary releases of GT.M protect against concurrent access to GT.M files by processes executing different versions of GT.M. Since historical versions of GT.M did not protect against this condition, FIS recommends procedural safeguards against inadvertent concurrent access by processes of multiple versions on systems on which old versions of GT.M are installed and active, since such concurrent usage can cause structural damage to the database.

---

## Appendix B. Monitoring GT.M Messages

Revision History		
Revision V6.0-000/1	21 November 2012	Added a table of replication message identifiers that are logged in the operator log.
Revision V5.4-002B	24 October 2011	Conversion to documentation revision history reflecting GT.M releases with revision history for each chapter.

---

### Monitoring GT.M Messages

This section covers information on monitoring GT.M messages. There are several types of messages.

- The GT.M run-time system sends messages (such as when a database file extends) to the system log. These are not trapped by the application error trap.
- Compilation errors generated by GT.M are directed to STDERR. These are not trapped by the application error trap. You can avoid them by compiling application code before deploying it in production, or log them by running mumps processes with STDERR directed to a file.
- Errors trapped by the application and logged by the application. These are outside the purview of this discussion.

A system management tool will help you automate monitoring messages.

GT.M sends messages to the system log at the LOG\_INFO level of the LOG\_USER facility. GT.M messages are identified by a signature of the form **GTM-s-abcdef** where **-s-** is a severity indicator and **abcdef** is an identifier. The severity indicators are: **-I-** for informational messages, **-W-** for warnings, **-E-** for errors and **-F-** for events that cause a GT.M process to terminate abnormally. Your monitoring should recognize the important events in real time, and the warning events within an appropriate time. All messages have diagnostic value. It is important to create a baseline pattern of messages as a *signature* of normal operation of your system so that a deviation from this baseline - the presence of unexpected messages, an usual number of expected messages (such as file extension) or the absence of expected messages - allows you to recognize abnormal behavior when it happens. In addition to responding to important events in real time, you should regularly review information and warning messages and ensure that deviations from the baseline can be explained.

Some message identifiers are described in the following table:

Component	Instance File or Replication Journal Pool	Receiver Pool	Identifier
Source Server	Y	N/A	SRCSRVR
	N	N/A	MUPIP
Receiver Server	Y	N/A	RCVSRVR

## Monitoring GT.M Messages

Component	Instance File or Replication Journal Pool	Receiver Pool	Identifier
	N	N/A	MUPIP
Update Process	Y	N/A	UPD
	N	N/A	MUPIP
Reader Helper	N/A	Y	UPDREAD
	N/A	N	UPDHELP
Writer Helper	N/A	Y	UPDWRITE
	N/A	N	UPDHELP

In addition to messages in the system log, and apart from database files and files created by application programs, GT.M creates several types of files: journal files, replication log files, gtmsecshr log files, inter-process communication socket files, files from recovery / rollback and output and error files from JOB'd processes. You should develop a review and retention policy. Journal files and files from recovery / rollback are likely to contain sensitive information that may require special handling to meet business or legal requirements. Monitor all of these files for growth in file numbers or size that is materially different than expectations set by the baseline. In particular, monitoring file sizes is computationally inexpensive and regular monitoring - once an hour, for example - is easily accomplished with the system crontab.

While journal files automatically switch to new files when the limit is reached, log files can grow unchecked. You should periodically check the sizes of log files and switch them when they get large - or simply switch them on a regular schedule.

1. **gtmsecshr** log file - **gtm\_secshr\_log** in the directory **\$gtm\_log** (send a SIGHUP to the **gtmsecshr** process to create a new log file).



### Important

Starting with V6.0-000, GT.M logs gtmsecshr messages in the system log and ignores the environment variable `gtm_log`.

2. Source Server, Receive Server, and Update Process log files. For more information, refer to Section : “Changing the Log File” (page 239) in the Database Replication chapter.

Since database health is critical, database growth warrants special attention. Ensure every file system holding a database file has sufficient space to handle the anticipated growth of the files it holds. Remember that with the lazy allocation used by UNIX file systems, all files in a system compete for its space. GT.M issues an informational message each time it extends a database file. When extending a file, it also issues a warning if the remaining space is less than three times the extension size. You can use the `$VIEW()` function to find out the total number of blocks in a database as well as the number of free blocks.

As journal files grow with every update they use up disk faster than database files do. GT.M issues messages when a journal file reaches within three, two and one extension size number of blocks from the automatic journal file switch limit. GT.M also issues messages when a journal file reaches its specified maximum size, at which time GT.M closes it, renames it and creates a new journal file. Journal files covering time periods prior to the last database backup (or prior to the backup of replicating secondary instances) are not needed for continuity of business, and can be deleted or archived, depending on your retention

policy. Check the amount of free space in file systems at least hourly and perhaps more often, especially file systems used for journaling, and take action if it falls below a threshold.

GT.M uses monotonically increasing relative time stamps called transaction numbers. You can monitor growth in the database transaction number with `DSE DUMP -FILEHEADER`. Investigate and obtain satisfactory explanations for deviations from the baseline rate of growth.

After a `MUPIP JOURNAL -ROLLBACK` (non replicated application configuration) or `MUPIP JOURNAL -RECOVER -FETCHRESYNC` (replicated application configuration), you should review and process or reconcile updates in the broken and unreplicated (lost) transaction files.

In a replicated environment, frequently (at least hourly; more often suggested since it takes virtually no system resources), check the state of replication and the backlog with `MUPIP REPLICATE -CHECKHEALTH` and `-SHOWBACKLOG`. Establish a baseline for the backlog, and take action if the backlog exceeds a threshold.

When a GT.M process terminates abnormally, it attempts to create a `GTM_FATAL_ERROR.ZSHOW_DMP_*` file containing a dump of the M execution context and a core file containing a dump of the native process execution context. The M execution context dump is created in the current working directory of the process. Your operating system may offer a means to control the naming and placement of core files; by default they are created the current working directory of the process with a name of `core.*`. The process context information may be useful to you in understanding the circumstances under which the problem occurred and/or how to deal with the consequences of the failure on the application state. The core files are likely to be useful primarily to your GT.M support channel. If you experience process failures but do not find the expected files, check file permissions and quotas. You can simulate an abnormal process termination by sending the process a SIGILL (with `kill -ILL` or `kill -4` on most UNIX/Linux systems).



### Caution

Dumps of process state files are likely to contain confidential information, including database encryption keys. Please ensure that you have appropriate confidentiality procedures as mandated by applicable law and corporate policy.

GT.M processes issued with the `JOB` command create `.mje` and `.mjo` files for their `STDERR` and `STDOUT` respectively. Analyze non-empty `.mje` files. Design your application and/or operational processes to remove or archive `.mjo` files once they are no longer needed.

Use the environment variable `gtm_procstuckexec` to trigger monitoring for processes holding a resource for an unexpectedly long time. `$gtm_procstuckexec` specifies a shell command or a script to execute when any of the following conditions occur:

- An explicit `MUPIP FREEZE` or an implicit freeze, such as for a `BACKUP` or `INTEG -ONLINE` that lasts longer than one minute.
- `MUPIP` actions find `kill_in_prog` (KILLs in progress) to be non-zero after a one minute wait on a region.
- `BUFOWNERSTUCK`, `INTERLOCK_FAIL`, `JNLPROCSTUCK`, `SHUTDOWN`, `WRITERSTUCK`, `MAXJNLQIOLOCKWAIT`, `MUTEXLCKALERT`, `SEMWT2LONG`, and `COMMITWAITPID` operator messages are being logged.

The shell script or command pointed to by `gtm_procstuckexec` can send an alert, take corrective actions, and log information.



### Note

Make sure user processes have sufficient space and permissions to run the shell command / script.

---

## Appendix C. Compiling ICU on GT.M supported platforms

Revision History		
Revision 2	2 December 2011	Changed the instructions for compiling ICU 4.4 on AIX 6.1
Revision V5.4-002B	24 October 2011	Conversion to documentation revision history reflecting GT.M releases with revision history for each chapter.

---

### Compiling ICU

This appendix includes sample instructions to download ICU, configure it not to use multi-threading, and compile it for various platforms. Note that download sites, versions of compilers, and milli and micro releases of ICU may have changed ICU since the embedded dates when these instructions were tested making them out-of-date. Therefore, these instructions must be considered examples, not a cookbook.

Although GT.M uses ICU, ICU is not FIS software and FIS does not support ICU. The sample instructions for installing and configuring ICU are merely provided as a convenience to you.

All GT.M versions prior to V5.3-004 require exactly ICU 3.6, however, V5.3-004 (or later) accept ICU 3.6 or later.

#### Compiling ICU 4.4 on AIX 6.1

As of November 2011, ICU version 4.4 can be compiled on AIX with the following configuration:

Operating System: *AIX*

Version: *AIX 6.1 64-bit*

Compilers: *IBM XL C/C++ 10, GNU make 3.80*

#### Installation Instructions

1. Ensure that system environment variable PATH includes the location of all the compilers mentioned above.
2. Set the environment variable OBJECT\_MODE to 64.
3. Download the source code of ICU version 4.4 for C from <http://site.icu-project.org/download>
4. At the shell prompt, execute the following commands:

```
gunzip -d < icu4c-4_4-src.tgz | tar -xf -
cd icu/source
./runConfigureICU AIX --disable-threads --enable-renaming=no --with-library-bits=64 CC=xlc
gmake
gmake -k check
gmake install
```

## Compiling ICU on GT.M supported platforms

5. Set the environment variable `LIBPATH` to point to the location of ICU. AIX uses the environment variable `LIBPATH` to search for dynamically linked libraries.

This installs ICU in the `/usr/local` directory.



### Note

By default, ICU is installed in the `/usr/local` directory. To install ICU in a different directory, use `--prefix=<install_path>` with the `runConfigureICU` command.

## Compiling ICU 4.2 on AIX 6.1

As of December 2009, ICU version 4.2 can be compiled on AIX with the following configuration:

Operating System: *UNIX*

Version: *AIX 6.1 64-bit*

Compilers: *IBM XL C/C++ 10, GNU make 3.80*

## Installation Instructions

1. Ensure that system environment variable `PATH` includes the location of all the compilers mentioned above.
2. Set the environment variable `OBJECT_MODE` to 64.
3. Download the source code of ICU version 4.2 for C from <http://site.icu-project.org/download>
4. At the shell prompt, execute the following commands:

```
gunzip -d < icu4c-4_2_1-AIX6_1-VA9.tgz | tar -xf -
cd icu/source
./configure CC=/usr/vac/bin/cc CXX=/usr/vacpp/bin/xlc++ --disable-threads --disable-renaming
gmake
gmake check
gmake install
```

5. Set the environment variable `LIBPATH` to point to the location of ICU. AIX uses the environment variable `LIBPATH` to search for dynamically linked libraries.

This installs ICU in the `/usr/local` directory.



### Note

By default, ICU is installed on `/usr/local` directory.

## Compiling ICU 4.2 on AIX 5.3

As of December 2009, ICU version 4.2 could be compiled on AIX with the following configuration:

Operating System: *AIX*

Version: *AIX 5.3 (PowerPC 64-bit)*

Compilers: *VisualAge 6, GNU make (3.77+), ANSI C compiler*

## Installation Instructions

1. Ensure that system environment variable PATH includes the location of all the compilers mentioned above.
2. Download the source code of ICU version 3.6 for C from <http://site.icu-project.org/download>
3. At the shell prompt, execute the following commands:

```
gunzip -d < icu4c-3_6-src.tgz | tar -xf -
cd icu/source/
chmod +x runConfigureICU configure install-sh
./configure --disable-renaming --disable-threads --enable-debug CC=cc CFLAGS=-q64
gmake
gmake check
gmake install
```

4. Set the environment variable LIBPATH to point to the location of ICU. AIX uses the environment variable LIBPATH to search for dynamically linked libraries.

This installs ICU in the /usr/local directory.



### Note

By default, ICU is installed in the /usr/local directory. To install ICU in a different directory, use --prefix=<install\_path> with the runConfigureICU command.

## Compiling ICU 3.6 on AIX 5.2

As of January 2007, ICU version 3.6 could be compiled on AIX with the following configuration:

Operating System: *AIX*

Version: *AIX 5.2 (PowerPC 64-bit)*

Compilers: *VisualAge 6, GNU make (3.77+), ANSI C compiler*

## Installation Instructions

1. Ensure that system environment variable PATH includes the location of all the compilers mentioned above.
2. Download the source code of ICU version 3.6 for C from <http://icu.sourceforge.net/download/3.6.html#ICU4C>
3. At the shell prompt, execute the following commands:

```
gunzip -d < icu4c-3_6-src.tgz | tar -xf -
cd icu/source/
chmod +x runConfigureICU configure install-sh
runConfigureICU AIX --disable-64bit-libs --disable-threads
gmake
gmake check
```

```
gmake install
```

4. Set the environment variable LIBPATH to point to the location of ICU. AIX uses the environment variable LIBPATH to search for dynamically linked libraries.

This installs ICU in the **/usr/local** directory.



### Note

By default, ICU is installed on **/usr/local** directory. If you need to install ICU on a different directory type:

- **runConfigureICU AIX --prefix=<install\_path> --disable-64bit-libs --disable-threads**
- Then, execute the gmake commands, and set the environment variable LIBPATH to point to the appropriate location.

## Compiling ICU 4.2 on HP Integrity IA64 HP-UX 11.31

As of November 2009, ICU version 4.2 could be compiled on HP Integrity IA64 HP-UX with the following configuration:

Operating System: **UNIX**

Version: **IA64 HP-UX 11.31**

Compilers: **HP C/aC++ B3910B A.06.15, GNU make (3.81)**

### Installation Instructions

1. Ensure that system environment variable PATH includes the location of all the compilers mentioned above.
2. Download the source code of ICU (in this example version 3.6 for C from <http://icu.sourceforge.net/>).
3. At the shell prompt, run the following commands:

```
gunzip -d < icu4c-4_2_1-src.tgz | tar -xf -
cd icu/source/
chmod +x runConfigureICU configure install-sh
./runConfigureICU HP-UX/ACC --disable-renaming --disable-threads --with-library-bits=64
gmake
gmake check
gmake install
```

4. Set the environment variable LD\_LIBRARY\_PATH to point to the location of ICU. HP-UX uses the environment variable LD\_LIBRARY\_PATH to search for dynamically linked libraries.

This installs ICU in the **/usr/local** directory.



### Note

By default, ICU is installed in the **/usr/local** directory. To install ICU in a different directory, use **--prefix=<install\_path>** with the **runConfigureICU** command.

## Compiling ICU 3.6 on HP Integrity IA64 HP-UX 11.31

As of November 2009, ICU version 3.6 could be compiled on HP Integrity IA64 HP-UX with the following configuration:

Operating System: *UNIX*

Version: *IA64 HP-UX 11.31*

Compilers: *HP C/aC++ B3910B A.06.15, GNU make (3.81)*

### Installation Instructions

1. Ensure that system environment variable PATH includes the location of all the compilers mentioned above.
2. Download the source code of ICU (in this example version 3.6 for C from <http://icu.sourceforge.net/download/3.6.html#ICU4C>).
3. At the shell prompt, run the following commands:

```
gunzip -d < icu4c-3_6-src.tgz | tar -xf -
cd icu/source/
chmod +x runConfigureICU configure install-sh
runConfigureICU HP-UX/ACC --disable-threads
gmake
gmake check
gmake install
```

4. Set the environment variable LD\_LIBRARY\_PATH to point to the location of ICU. HP-UX uses the environment variable LD\_LIBRARY\_PATH to search for dynamically linked libraries.

This installs ICU in the **/usr/local** directory.



#### Note

By default, ICU is installed in the **/usr/local** directory. If you install ICU in a different directory, type:

- **runConfigureICU HP-UX/ACC --prefix=<install\_path> --disable-threads**
- Then run the gmake commands, and set the environment variable LD\_LIBRARY\_PATH to point to the appropriate location.

## Compiling ICU 3.6 on HP PA-RISC HP-UX 11.31 (11iv3)

As of November 2009, ICU version 3.6 could be compiled on HP PA-RISC HP-UX with the following configuration:

Operating System: *UNIX*

Version: *HP-UX 11.31 (11iv3)*

Compiler: *cc HP C/aC++ B3910B A.06.12, aCC HP C/aC++ B3910B A.06.15, GNU Make 3.81*

## Installation Instructions

1. Ensure that system environment variable `PATH` includes the location of all the compilers mentioned above.
2. Download the source code of ICU (in this example, version 3.6 for C from <http://icu.sourceforge.net/download/3.6.html#ICU4C>)
3. At the shell prompt, execute the following commands:

```
gunzip -d < icu4c-3_6-src.tgz | tar -xf -
cd icu/source/
chmod +x runConfigureICU configure install-sh
runConfigureICU HP-UX/ACC --disable-64bit-libs? --enable-rpath --disable-threads
gmake
gmake check
gmake install
```

4. Set the environment variable `LD_LIBRARY_PATH` to point to the location of ICU. HP-UX uses the environment variable `LD_LIBRARY_PATH` to search for dynamically linked libraries.

This installs ICU in the `/usr/local` directory.



### Note

By default, ICU is installed in `/usr/local`. If you install ICU in a different directory, type:

- **`runConfigureICU HP-UX/ACC --prefix=<install_path> --enable-64bit-libs? --enable-rpath --disable-threads`**
- Then execute the `gmake` commands, and set the environment variable `LD_LIBRARY_PATH` to point to the appropriate location.

## Compiling ICU 3.6 on HP PA-RISC HP-UX 11.11

As of January 2007, ICU version 3.6 could be compiled on PA-RISC HP-UX with the following configuration:

Operating System: ***UNIX***

Version: ***HP-UX 11.11***

Compilers: ***aCC A.03.50, cc B.11.11.08, GNU make (3.77+)***

## Installation Instructions

1. Ensure that system environment variable `PATH` includes the location of all the compilers mentioned above.
2. Download the source code of ICU version 3.6 for C from <http://icu.sourceforge.net/download/3.6.html#ICU4C>
3. Add the following line in the configuration file `source/config/mh-hpux-acc` to include the appropriate C++ runtime libraries:

```
DEFAULT_LIBS = -lstd_v2 -lCsup_v2 -lc1
```

4. At the shell prompt, execute the following commands:

```
gunzip -d < icu4c-3_6-src.tgz | tar -xf -
```

## Compiling ICU on GT.M supported platforms

```
cd icu/source/  
chmod +x runConfigureICU configure install-sh  
runConfigureICU HP-UX/ACC --disable-64bit-libs --disable-threads  
gmake  
gmake check  
gmake install
```

5. Set the environment variable **LD\_LIBRARY\_PATH** to point to the location of ICU. HP-UX uses the environment variable **LD\_LIBRARY\_PATH** to search for dynamically linked libraries.

This installs ICU in the **/usr/local** directory.



### Note

By default, ICU is installed in the **/usr/local** directory. If you need to install ICU on a different directory type:

- **runConfigureICU HP-UX/ACC --prefix=<install\_path> --disable-64bit-libs --disable-threads**
- Then execute the gmake commands, and set the environment variable **LD\_LIBRARY\_PATH** to point to the appropriate location.

## Compiling ICU 4.2 on Red Hat Enterprise Linux 4 Update 2

As of December 2009, ICU version 4.2 could be compiled on x86\_64 Linux with the following configuration:

Operating System: x86\_64 Linux

Version: Red Hat Enterprise Linux 4 Update 2

Compilers: gcc 3.4.4, GNU make (3.77+), ANSI C compiler

### Installation Instructions

1. Ensure that system environment variable **PATH** includes the location of all the compilers mentioned above.
2. Download the source code of ICU version 4.2 for C from <http://site.icu-project.org/download>.
3. At the shell prompt, execute the following commands:

```
gunzip -d < icu4c-3_6-src.tgz | tar -xf -  
cd icu/source/  
chmod +x runConfigureICU configure install-sh  
./runConfigureICU Linux --disable-renaming --disable-threads --with-library-bits=64  
gmake  
gmake check  
gmake install
```

4. Set the environment variable **LD\_LIBRARY\_PATH** to point to the location of ICU. Linux uses the environment variable **LD\_LIBRARY\_PATH** to search for dynamically linked libraries to be loaded.

This installs ICU in the **/usr/local** directory.

**Note**

By default, ICU is installed in the `/usr/local` directory. To install ICU in a different directory, use `--prefix=<install_path>` with the `runConfigureICU` command.

**Compiling ICU 3.6 on Red Hat Enterprise Linux 4 Update 2**

As of January 2007, ICU version 3.6 could be compiled on x86 Linux with the following configuration:

Operating System: *x86 Linux*

Version: *Red Hat Enterprise Linux 4 Update 2*

Compilers: *gcc 3.4.4, GNU make (3.77+), ANSI C compiler*

**Installation Instructions**

1. Ensure that system environment variable `PATH` includes the location of all the compilers mentioned above.
2. Download the source code of ICU version 3.6 for C from <http://icu.sourceforge.net/download/3.6.html#ICU4C>
3. At the shell prompt, execute the following commands:

```
gunzip -d < icu4c-3_6-src.tgz | tar -xf -
cd icu/source/
chmod +x runConfigureICU configure install-sh
runConfigureICU Linux --disable-64bit-libs --disable-threads
gmake
gmake check
gmake install
```

4. Set the environment variable `LD_LIBRARY_PATH` to point to the location of ICU. Linux uses the environment variable `LD_LIBRARY_PATH` to search for dynamically linked libraries to be loaded.

This installs ICU in the `/usr/local` directory.

**Note**

By default, ICU is installed on `/usr/local` directory. If you need to install ICU on a different directory type:

- **`runConfigureICU Linux --prefix=<install_path> --disable-64bit-libs --disable-threads`**
- Then execute the **`gmake`** commands, and set the environment variable `LD_LIBRARY_PATH` to point to the appropriate location.

**Compiling ICU 4.4 on Solaris 9 (SunOS 5.9)**

As of December 2009, ICU version 4.2 could be compiled on Solaris with the following configuration:

Operating System: *Solaris*

Version: Solaris 9 (*SunOS 5.9*)

Compilers: *Sun Studio 8 (Sun C++ 5.5)*, *GNU make (3.77+)*, *ANSI C compiler*

## Installation Instructions

1. Ensure that system environment variable PATH includes the location of all the compilers mentioned above.
2. Download the source code of ICU version 4.2 for C from <http://site.icu-project.org/download>
3. Add the following line in the configuration file source/config/mh-solaris to include the appropriate C++ runtime libraries:

```
DEFAULT_LIBS = -lCstd -lCrun -lm -lc
```

4. At the shell prompt, execute the following commands:

```
gunzip -d < icu4c-4_2_1-src.tgz | tar -xf -
cd icu/source/
chmod +x runConfigureICU configure install-sh
./configure --with-library-bits=64 --enable-threads=no LDFLAGS="-lCstd -lCrun"
gmake
gmake check
gmake install
```

5. Set the environment variable LD\_LIBRARY\_PATH to point to the location of ICU. Solaris uses the environment variable LD\_LIBRARY\_PATH to search for dynamically linked libraries to be loaded.

ICU is now installed in the /usr/local directory.



### Note

By default, ICU is installed in the /usr/local directory. To to install ICU in a different directory, use --prefix=<install\_path> with the runConfigure command.

## Compiling ICU 4.2 on Solaris 9 (SunOS 5.9)

As of December 2009, ICU version 4.2 could be compiled on Solaris with the following configuration:

Operating System: *Solaris*

Version: *Solaris 9 (SunOS 5.9)*

Compilers: *Sun Studio 8 (Sun C++ 5.5)*, *GNU make (3.77+)*, *ANSI C compiler*

## Installation Instructions

1. Ensure that system environment variable PATH includes the location of all the compilers mentioned above.
2. Download the source code of ICU version 4.2 for C from <http://site.icu-project.org/download>
3. Add the following line in the configuration file source/config/mh-solaris to include the appropriate C++ runtime libraries:

```
DEFAULT_LIBS = -lCstd -lCrun -lm -lc
```

4. At the shell prompt, execute the following commands:

```
gunzip -d < icu4c-4_2_1-src.tgz | tar -xf -
cd icu/source/
chmod +x runConfigureICU configure install-sh
./configure --disable-renaming --disable-threads --enable-64bit-libs
gmake
gmake check
gmake install
```

5. Set the environment variable `LD_LIBRARY_PATH` to point to the location of ICU. Solaris uses the environment variable `LD_LIBRARY_PATH` to search for dynamically linked libraries to be loaded.

ICU is now installed in the `/usr/local` directory.



### Note

By default, ICU is installed in the `/usr/local` directory. To install ICU in a different directory, use `--prefix=<install_path>` with the `runConfigure` command.

## Compiling ICU 3.6 on Solaris 9 (SunOS 5.9)

As of January 2007, ICU version 3.6 could be compiled on Solaris with the following configuration:

Operating System: **Solaris**

Version: **Solaris 9 (SunOS 5.9)**

Compilers: **Sun Studio 8 (Sun C++ 5.5), GNU make (3.77+), ANSI C compiler**

## Installation Instructions

1. Ensure that system environment variable `PATH` includes the location of all the compilers mentioned above.
2. Download the source code of ICU version 3.6 for C from <http://icu.sourceforge.net/download/3.6.html#ICU4C>
3. Add the following line in the configuration file `source/config/mh-solaris` to include the appropriate C++ runtime libraries:

```
DEFAULT_LIBS = -lCstd -lCrun -lm -lc
```

4. At the shell prompt, execute the following commands:

```
gunzip -d < icu4c-3_6-src.tgz | tar -xf -
cd icu/source/
chmod +x runConfigureICU configure install-sh
runConfigureICU Solaris --disable-64bit-libs --disable-threads
gmake
gmake check
gmake install
```

5. Set the environment variable `LD_LIBRARY_PATH` to point to the location of ICU. Solaris uses the environment variable `LD_LIBRARY_PATH` to search for dynamically linked libraries to be loaded.

ICU is now installed in the `/usr/local` directory.



## Note

By default, ICU is installed in the **/usr/local** directory. If you need to install ICU on a different directory type:

- **runConfigureICU Solaris --prefix=<install\_path> --disable-64bit-libs --disable-threads**
- Then execute the gmake commands, and set the environment variable LD\_LIBRARY\_PATH to point to the appropriate location.

---

## Appendix D. Building Encryption Libraries

Revision History		
Revision V5.5-000/10	28 September 2012	In Section : “Solaris 9 and 10 (SPARC)” (page 428), removed a redundant step.
Revision V5.4-002B	24 October 2011	Conversion to documentation revision history reflecting GT.M releases with revision history for each chapter.

---

### Building Encryption Libraries

FIS neither encourages nor supports the use of any specific encryption library. In order to be helpful, here is how we created the libraries for testing GT.M in the development environment.

#### Ubuntu 8.04 LTS (x86\_64)

Packages were installed from standard repositories using the package manager.

#### Red Hat Enterprise Linux 5.4 (x86\_64)

All packages except libpgpme were installed from Red Hat Network. libpgpme was built from source after specifying `./configure CC=gcc CFLAGS=-m64`

#### Red Hat Enterprise Linux 5.5 (x86)

Packages were installed from standard repositories using the package manager.

#### Red Hat Enterprise Linux 5.3 (IA64)

All packages except libpgpme were installed from Red Hat Network. libpgpme was built from source after specifying `./configure CC=gcc`

#### IBM z/OS (zSeries)

Warning: The configure scripts for the GnuPG libraries do not generate shared libraries, only static archive libraries. Owing to this limitation, the archive libraries are unpacked and recompiled back into dynamically linked libraries. The following example uses libpgp-error as the target. FIS suggests that you create a temporary directory for each DLL.

1. Unpack the archive : `ar -x /usr/local/lib/libpgp-error.a`
2. Link the archive objects into a DLL: `xlC -qascii -q64 -W l,DLL -o libpgp-error.dll *.o`
3. Copy the DLL and side deck file into the destination (assumed to be /usr/local/lib): `cp libpgp-error.dll libpgp-error.x /usr/local/lib`

## GPG-ERROR

Apply the patch `libgpg-error-1.7.patch` to `libgpg-error` sources. Then run the configuration and installation steps.

1. `./configure CC=xlc CFLAGS="-qchars=signed -qascii -q64 -qlanglvl=extc99 -qexportall -qrent -qnocsect -W l,DLL -D_XOPEN_SOURCE=600 -D_ENHANCED_ASCII_EXT=0xFFFFFFFF -D_IEEEV1_COMPATIBILITY -D_OPEN_MSGQ_EXT" LD=xlc LDFLAGS="-qascii -q64 -W l,DLL" CXX=xlc++ --enable-shared --prefix=/usr/local`
2. `make && make check`
3. (as root) `make install`

Follow the instructions above to turn the archive library into a dynamically linked library.

## GCRYPT

Apply the patch `libgcrypt-1.4.4.patch` to the `libgcrypt` sources. Before running the configure and installation steps, you need to place some symbolic links in the sources. Compiling `libgpgcrypt` on z/OS runs afoul of a limitation of the XLC compiler's include header search path and system headers. Issue the following commands, verbatim, to overcome this limitation:

```
cd mpi
ln -s ../src/mpi.h .
ln -s ../src/memory.h .
cd -
cd cipher
ln -s ../src/mpi.h .
cd -
```

Then:

1. `./configure CC=xlc CFLAGS="-qchars=signed -qascii -q64 -qlanglvl=extc99 -qexportall -qrent -qnocsect -W l,DLL -D_XOPEN_SOURCE=600 -D_ENHANCED_ASCII_EXT=0xFFFFFFFF -D_IEEEV1_COMPATIBILITY`
2. `make && make check`
3. (as root) `make install`

Follow the instructions from above to turn the archive library into a dynamically linked library.

## GPGME

Apply the patch `gpgme-1.1.8.patch` to the `gpgme` sources. Then run the configuration and installation steps.

- `./configure CC=xlc CFLAGS="-qchars=signed -qascii -q64 -qlanglvl=extc99 -qexportall -qrent -qnocsect -W l,DLL -D_XOPEN_SOURCE=600 -D_ENHANCED_ASCII_EXT=0xFFFFFFFF -D_IEEEV1_COMPATIBILITY`
- `make && make check`
- (as root) `make install`

Follow the instructions from above to turn the archive library into a dynamically linked library.

## GNUPG

Apply the patch `gnupg-1.4.9.patch` to the GnuPG sources. Before running the configure and installation steps, you need to place some symbolic links in the sources. Compiling `libpgcrypt` on z/OS runs afoul of a limitation of the XLC compiler's include header search path and system headers. Issue the following commands, verbatim, to overcome this limitation:

```
cd mpi
ln -s ../include/mpi.h .
ln -s ../include/memory.h .
cd -
```

Then:

- `./configure CC=xlc CFLAGS="-qchars=signed -qascii -q64 -qlanglvl=extc99 -qexportall -qrent -qnocsect -W l,DLL -D_XOPEN_SOURCE=600 -D_ENHANCED_ASCII_EXT=0xFFFFFFFF -D_IEEEV1_COMPATIBILITY -D_OPEN_MSGQ_EXT" LD=xlc LDFLAGS="-qascii -q64 -W l,DLL" CXX=xlc++ --without-pth --without-libassuan --without-ksba --prefix=/usr/local`
- `make && make check`
- (as root) `make install`

## IBM AIX 5.3 (pSeries)

### GPG-ERROR

```
./configure CC=cc CFLAGS=-q64 ($OBJECT_MODE=64)
```

### CRYPTO (From OpenSSL)

These instructions build OpenSSL which provides `libcrypto`.

```
./Configure aix64-cc shared # Note: it is an upper case C
.make
(as root) make install
```

### GPGME

GPGME requires a source level fix to use the proper `malloc()` that requires an include for `stdlib.h` in the include section of `version.c`. Then:

```
./configure CC="xlc -q64" --disable-asm ($OBJECT_MODE=64)/ or CC=cc CFLAGS=-q64
```

## GNUPG

GPG on AIX requires the `setuid` bit to be set. This can be done via `chmod u+s /path/to/gpg`. Please see <http://www.gnupg.org/documentation/faqs.en.html#q6.1>

```
./configure CC="xlc -q64" --disable-asm ($OBJECT_MODE=64) or CC=cc CFLAGS=-q64
```

## Solaris 9 and 10 (SPARC)

Set `$LD_LIBRARY_PATH` to include `/usr/lib/sparcv9` AND `/usr/ucblib/sparcv9` for both Solaris 9 and 10. Solaris seems to default to the 32-bit version of `ucblib` at runtime, unless the 64-bit path is ahead of the 32-bit in `$LD_LIBRARY_PATH`. However, using `-R/usr/lib/sparcv9 -R/usr/ucblib/sparcv9/` as a `CFLAG` at build time hard-codes the path into the library, removing the need for `$LD_LIBRARY_PATH`.

### GPG-ERROR

```
./configure CC=/opt/SUNWspro/bin/cc CFLAGS="-m64 -R/usr/lib/sparcv9 -R/usr/ucblib/sparcv9/"
```



### GCRYPT

```
./configure CC=/opt/SUNWspro/bin/cc CFLAGS="-m64 -R/usr/lib/sparcv9 -R/usr/ucblib/sparcv9/" --disable-asm
--with-gpg-error-prefix=/usr/local
```



### GPGME

GPGME requires a source level fix to use the proper `malloc()` that requires an include for `stdlib.h` in the include section of `version.c`.

```
./configure CC=/opt/SUNWspro/bin/cc CFLAGS="-m64 -R/usr/lib/sparcv9 -R/usr/ucblib/sparcv9/" --disable-asm --with-gpg-error-prefix=/usr/local
```

### GNUPG

```
./configure CC=/opt/SUNWspro/bin/cc CFLAGS="-m64 -R/usr/lib/sparcv9 -R/usr/ucblib/sparcv9/" --disable-asm
```

## HP-UX 11.31 (IA64)

Depending on which user runs the `configure` command, you may or may not have to specify the `--with-gpg-error-prefix` flag.

### GPG-ERROR

```
./configure CC=/usr/bin/cc CFLAGS=+DD64
```

### GCRYPT

```
./configure CC=/usr/bin/cc CFLAGS=+DD64 --with-gpg-error-prefix=/usr/local
```

### GPGME

GPGME requires a source level fix to use the proper `malloc()` that requires an include for `stdlib.h` in the include section of `version.c`.

```
./configure CC=/usr/bin/cc CFLAGS=+DD64 --with-gpg-error-prefix=/usr/local
```

## GNUPG

```
./configure CC=/usr/bin/cc CFLAGS=+DD64 --with-gpg-error-prefix=/usr/local
```

## SUSE Linux Enterprise Server 11 (x86\_64)

Packages were installed from standard repositories using the package manager. The libgcrypt packages have been upgraded using openSUSE 11.2 packages.

## SUSE Linux Enterprise Server 11 (s390x)

Packages were installed from standard repositories using the package manager.

## SUSE Linux Enterprise Server 10 (s390x)

Packages were installed from standard repositories using the package manager.

To upgrade the GPG package using the source RPM version 1.4.9-6.1 from openSUSE 11:

1. Download and install gpg-1.4.9-6.1 src

```
rpm wget
▶ ftp://ftp.pbone.net/mirror/ftp5.gwdg.de/pub/opensuse/repositories/home%3A/keutterling/openSUSE_11.0/src/
gpg-1.4.9-6.1.src.rpm
sudo rpm -i gpg-1.4.9-6.1.src.rpm
```



2. Install the required dependencies for building GPG. You can get those from the official SLES 10 CDs/DVD.

```
sudo rpm -i db42-4.2.52-20.2.s390x.rpm openssl-devel-0.9.8a-18.36.s390x.rpm
▶ cyrus-sasl-devel-2.1.21-18.11.41.s390x.rpm
openldap2-2.3.32-0.35.23.s390x.rpm openldap2-devel-2.3.32-0.35.23.s390x.rpm
```



3. Build GPG 1.4.9 RPM.

```
rpmbuild -bb /usr/src/packages/SPECS/gpg1.spec
```

4. Upgrade to the new GPG RPM.

```
sudo rpm -U /usr/src/packages/RPMS/s390x/gpg-1.4.9-6.1.s390x.rpm
```

---

## Appendix E. GT.M Security Philosophy

Revision History		
Revision V5.5-000/5	17 July 2012	Fixed a typo (stand-alone -> stand-alone) in "Recommendations" (page 432).
Revision V5.5-000/3	2 May 2012	First published revision.

---

### Philosophy

The general GT.M philosophy is to use the security of the underlying operating system, and to neither subvert it nor extend it. The purpose of this document is to discuss the implications of, exceptions to, and limitations of this philosophy (the "security model"). This appendix reflects GT.M as of V5.5-000.



#### Note

GT.M is not intended to operate robustly on a machine that is potentially subject to direct attack, such as a firewall, or a machine operating in a "DMZ."

### Normal User and Group Id Rule

GT.M processes run with normal UNIX<sup>1</sup> user and group ids. GT.M has no database daemon that needs to run with elevated privileges. Process code written in M will be able to read a database file if and only if the process has read permission for that database file, and to update that database file if and only if the process has read/write permission for that database file.<sup>2</sup>

There are two exceptions to this rule. Also, special mention is made of GT.M triggers, which require awareness of their behavior even though they comply with the Normal User and Group Id Rule.

### Exceptions

Exceptions to the Normal User and Group Ids Rule exist for:

- Shared Memory when the Buffered Global (BG) access method is used, and
- gtmsecshr.

### Shared Memory Exception for BG

With the BG access method, each open database file has a shared memory segment associated with it. This shared memory contains a pool of disk blocks (the global buffers) as well as associated control structures (for example, for concurrency control). Even a process that has read-only permission to the database file requires read-write access to the associated shared memory in order to use the control structures. It is therefore possible for a cached disk block in shared memory to be modified by one process and for the actual write of that dirty block to disk to be performed by another. Thus, a "rogue" process with read-only access to the database file but read-write access to shared memory can modify the cached copy of a disk block and effect

---

<sup>1</sup>The term "UNIX" refers to platforms on which GT.M uses a POSIX API to access the underlying operating system.

<sup>2</sup>The concept of write-only access to a database file is not meaningful for GT.M

a permanent change to the database when that block is written to disk by another process that has read-write access to the database file.

Comments on the Shared Memory Exception for BG:

- This only applies if a mumps process contains non-M code. If a mumps processes has only M code, the GT.M run-time environment will not allow a process to modify a database for which it lacks write permission.
- This only applies if a database file has mixed read-only and read-write access, that is, some mumps processes have read-only access and others have read-write access. If all processes have read-only access, although the database may appear to be temporarily modified when copies of blocks in shared memory are modified, the database file on disk cannot be permanently modified because no process will have the required permission to flush dirty blocks to disk.
- Where processes that contain C code and have read-only database access must co-exist with processes that have read-write access, GT.M will only "keep honest processes honest." [See below for recommendations where read-only access is required by processes that cannot be considered trustworthy.]

### **gtmsecshr Exception**

Processes with normal user and group ids do not have adequate permissions to effect necessary GT.M interprocess communication and cleanup after abnormal process termination. A process called **gtmsecshr** runs as root in order to effect the following functionality:

- Interprocess communication, including sending **SIGALARM** and **SIGCONT** between processes where normal UNIX permissions do not permit such signals to be sent.
- Cleanup after processes that terminate abnormally, including removing semaphores, shared memory segments, and flushing database file headers (but not database blocks) from shared memory segments to disk.

Whenever a GT.M process lacks adequate permissions to effect any of the above operations, it automatically invokes **gtmsecshr** if it is not already running. A complete list of **gtmsecshr** functionality appears in "gtmsecshr commands" (page 433) .

In order to run as root, and to be invoked by a process that has normal user and group ids, the invocation chain for **gtmsecshr** requires an executable image that is owned by root and which has the **setuid** bit turned on in its file permissions.

Once started and running, **gtmsecshr** records information in a log file **gtm\_secshr\_log** (in a directory specified by **\$gtm\_log**), creating it if it does not exist. **\$gtm\_log** is inherited from the environment of the GT.M process (**mumps**, **mupip** or **dse**) that first invokes the **gtmsecshr** process. If the environment variable **\$gtm\_log** is undefined, if its value is longer than GT.M can handle, or if it is defined to a value that is not an absolute pathname (starting with a /), **\$gtm\_log** is assumed to be the directory **/tmp** (AIX, GNU/Linux, Tru64 UNIX) or **/var/tmp** (HP-UX, Solaris).

Communication between GT.M processes and **gtmsecshr** uses socket files in **\$gtm\_tmp**, which is also inherited from the GT.M process that first invokes **gtmsecshr**. If the environment variable **\$gtm\_tmp** is undefined, if its value is longer than GT.M can handle, or if it is defined to a value that is not an absolute pathname (starting with a /), **\$gtm\_tmp** is assumed to be the directory **/tmp** (AIX, GNU/Linux, Tru64 UNIX) or **/var/tmp** (HP-UX, Solaris).

The **gtmsecshr** process receives messages via a socket file owned by root with a name of the form **gtm\_secshrnnnnnnnn**, the **nnnnnnnn** being replaced by the hexadecimal **ftok** value of the **gtmsecshr** executable file. This value is reported by the GT.M **ftok** utility on the **gtmsecshr** file, for example, **\$gtm\_dist/ftok \$gtm\_dist/gtmsecshr**

GT.M processes receive responses from **gtmsecshr** via socket files owned by the **userid** of the process with names of the form **gtm\_secshrnnnnnnnn**, where **nnnnnnnn** is a hexadecimal version of the client's process id, padded with leading

zeroes. When a client process terminates abnormally, or is killed before it cleans up its socket file, it is possible for a subsequent client with the same process id but a different userid to be unable to delete the leftover socket file. In this case, it tries to send a message to **gtmsecshr** using a slightly modified client socket file of the form **gtm\_secshrnnnnnnnnx** where x starts with "a" whose corresponding socket file does not already exist or is removable by the current client process (if all suffixes "a" through "z" are unavailable, the client process errors out with a "**Too many leftover client socket files**" message). **gtmsecshr** recognizes this special modified socket file name, and as part of servicing the client's request deletes the **gtm\_secshrnnnnnnnn** socket file and all **gtm\_secshrnnnnnnnnx** files that exist. The client process expects this file removal and creates a new **gtm\_secshrnnnnnnnn** file for subsequent communications with **gtmsecshr**.

- When there is no **gtmsecshr** process running, by starting one up with incorrect values of **\$gtm\_log** and **\$gtm\_tmp**, a **gtmsecshr** process can be made to create a file called **gtm\_secshr\_log** in any directory. Having incorrect values can also interfere with normal GT.M operations until the incorrect **gtmsecshr** process times out and terminates, because GT.M processes and **gtmsecshr** will be unable to communicate with one another.
- **gtmsecshr** can be made to delete client socket files by a rogue process. If a socket file is deleted under a running GT.M process, **gtmsecshr** will be unable to reply to the process. It will timeout, create another and proceed. Thus, while GT.M performance of a single process may temporarily be slowed, system operation will not be disrupted.

## Triggers

A GT.M trigger is a code fragment stored in the database file that all processes performing a matching update to a global variable in that file execute automatically, for example, to maintain cross-reference indexes and referential integrity. Any process that has read-write permission for a database file can change the triggers in that database file, which can in turn force other processes updating that database to execute the changed triggers.

## Recommendations

Based on the security model, the following are recommended best practices for securing GT.M.

1. Secure the machine on which GT.M operates behind layers of defenses that permit only legitimate accesses.
2. Restrict access to a system on which GT.M runs to those who legitimately need it.
3. If not all users who have access to a system require the ability to run GT.M, limit access to GT.M to a group to which all users who need access belong, and remove world access to GT.M.<sup>3</sup> If such a group is called **gtmusers**, the following command executed as root will accomplish this, if access was not restricted when GT.M was installed: **chgrp -R gtmusers \$gtm\_dist ; chmod -R o-rwx \$gtm\_dist**
4. Ensure that database file ownership (user and group), UNIX user and group ids, and permissions at the UNIX level match the intended access. If finer grained access controls than those provided by user and group ids and permissions are needed, consider using, where appropriate and available, security products layered on top of the operating system.
5. Under typical conditions, GT.M shared resources - journal files, shared memory, and semaphores - have the same group ids and access permissions as their database files, but may not be owned by the same userid, since the process creating the shared resource may have a different **userid** from the one that created the database. There are two edge cases to consider:
  - Where the owner of the database file is not a member of the group of the database file, but is a member of the group GT.M's **libgtmshr.so** file. In this case, if a process with a **userid** other than the owner were to create a shared resource, a process with the **userid** of the owner would not have access to them. Therefore, GT.M uses the group id of the **libgtmshr.so** file if the process creating the shared resource is also a member of that group. In this case it would also restrict access to the resource to members of that group. If the process creating this resource is not a member of the

## GT.M Security Philosophy

**libgtmshr.so** group, the group id of the shared resource remains that of the creating resource but the permissions allow world access. FIS advises against using a database file whose owner is not a member of the group of that file.

- Where the owner of the database file is neither a member of the group nor a member of the group of **libgtmshr.so**. In this case, GT.M uses world read-write permissions for the shared resources. FIS advises against the use of a database file whose owner is neither a member of the group of the file nor a member of the group of **libgtmshr.so**.
6. The Mapped Memory (MM) access method does not use a shared memory segment for a buffer pool for database blocks - shared memory is used only for control structures. Therefore, consider using MM if there are processes that are not considered trustworthy but which need read-only access to database files.<sup>4</sup>
  7. If MM cannot be used, and processes that are not considered trustworthy need read-only access to database files, run those processes on a replicating instance specifically set up for that purpose.
  8. If a database file does not change during normal operation (for example, it contains configuration parameters), make its permissions read only for everyone. On rare occasions when they need to be changed, shut down the application to get stand-alone access, temporarily make it read-write, make the changes, and then make it read-only once more.
  9. GT.M by default uses a wrapper for **gtmsecshr**. Source code for the wrapper is published. If processes that startup **gtmsecshr** cannot be trusted or coerced to have the correct values of **\$gtm\_log** and **\$gtm\_tmp**, modify the source code to set **\$gtm\_log** and **\$gtm\_tmp** to required values, recompile and reinstall your modified wrapper.
  10. Consider implementing layered security software if it exists for your platform, for example, SELinux, Trusted Solaris, Trusted AIX.



### Note

FIS neither endorses nor has tested any specific layered security product.

## gtmsecshr commands

Commands	Action	Comments
<b>WAKE_MESSAGE</b>	Sends <b>SIGALRM</b> to specified process.	Used to inform receiving process that a resource (such as a critical section) it awaits has become available.
<b>CONTINUE_PROCESS</b>	Sends <b>SIGCONT</b> to specified process.	Used to awake a process that has been suspended while holding a resource. <sup>a</sup>
<b>CHECK_PROCESS_ALIVE</b>	Test sending a signal to specified process. <sup>b</sup>	Used to determine if a process owning a resource still exists; if not, the resource is available to be grabbed by another process that needs it.
<b>REMOVE_SEM</b>	Remove a specified POSIX semaphore.	Used to remove an abandoned semaphore (for example, if the last attached process terminated abnormally).
<b>REMOVE_SHMMEM</b>	Remove a specified shared memory segment.	Used to remove an abandoned shared memory segment. Before removing the segment, <b>gtmsecshr</b> checks that there are no processes attached to it.

## GT.M Security Philosophy

Commands	Action	Comments
<b>REMOVE_FILE</b>	Remove a specified file.	Used to remove an abandoned socket file (for example, as a result of abnormal process termination) used for interprocess communication on platforms that do not support memory semaphores ( <b>msems</b> ); unused on other platforms. Before removal, gtmsecshr verifies that the file is a socket file, in directory <b>\$gtm_tmp</b> , and its name matches GT.M socket file naming conventions.
<b>FLUSH_DB_IPCS_INFO</b>	Writes file header of specified database file to disk.	The ipc resources (shared memory and semaphore) created for a database file are stored in the database file header. The first process opening a database file initializes these fields while the last process to use the database clears them. If neither of them has read-write access permissions to the database file, they set/reset these fields in shared memory and gtmsecshr will write the database file header from shared memory to disk on their behalf.

<sup>a</sup>Please do not ever suspend a GT.M processes. In the event GT.M finds a process suspended while holding a resource, it is sent a **SIGCONT**.

<sup>b</sup>This function is no longer needed and will be removed soon.

## Shared Resource Authorization Permissions

GT.M uses several types of shared resources to implement concurrent access to databases. The first GT.M process to open a database file creates IPC resources (semaphores and shared memory) required for concurrent use by other GT.M processes, and in the course of operations GT.M processes create files (journal, backup, snapshot) which are required by other GT.M processes. In order to provide access to database files required by M language commands and administration operations consistent with file permissions based on the user, group and world classes, the shared resources created by GT.M may have different ownership, groups and permissions from their associated database files as described below. As an example of the complexity involved, consider a first process opening a database based on its group access permissions. In other words, the database file is owned by a different userid from the semaphores and shared memory created by that first process. Now, if the userid owning the database file is not a member of the database file's group, a process of the userid owning the database file can only have access to the shared resources if the shared resources have world access permissions or if they have a group that is guaranteed to be shared by all processes accessing the database file, even if that group is different from the database file's own group. Again, although FIS strongly recommends against running GT.M processes as root, a root first process opening the database file must still be able to open it although it may not be the owner of the database file or even in its group - but it must ensure access to other, non-root processes. Some things to keep in mind:

- Even a process with read-only access to the database file requires read-write access to the shared memory control structures and semaphores.
- Creating and renaming files (for example, journal files) requires write access to both the files and the directories in which they reside.
- If you use additional layered security (such as Access Control Lists or SELinux), you must ensure that you analyze these cases in the context of configuring that layered security.

## GT.M Security Philosophy

GT.M takes a number of factors into account to determine the resulting permissions:

- The owner/group/other permissions of the database file
- The owner of the database file
- The group of the database file
- The group memberships of the database file's owner
- The owner/group/other permissions of the libgtmshr file
- The group of the libgtmshr file
- The effective user id of the creating process
- The effective group id of the creating process
- The group memberships of the creating process' user

The following table describes how these factors are combined to determine the permissions to use:

Database File Permissions	Opening process is owner of database file?	Owner is member of group of database file?	Opening process is a member of database file group?	Opening process is not owner but a member of database file group?	Execution of GT.M restricted to members of a group?
<i>Group of Resource</i>		<i>IPC Permissions</i>		<i>File Permissions</i>	
-r*-r*-r*-	-	-	Y	-	-
<i>Group of database file</i>		<i>-rw-rw-rw</i>		<i>-r*-r*-r*</i>	
-rw-rw-r*	-	-	N	-	-
<i>Current group of process</i>		<i>-rw-rw-rw</i>		<i>-rw-rw-rw</i>	
-rw-rw-rw	-	-	N	-	-
<i>Current group of process</i>		<i>-rw-rw-rw</i>		<i>-rw-rw-rw</i>	
-rw-rw-rw	Y	Y	-	-	-
<i>Group of database file</i>		<i>-rw-rw-rw</i>		<i>-r*-r*----</i>	
-r*-r*----	Y	N	-	-	N
<i>Current group of process</i>		<i>-rw-rw-rw-</i>		<i>-rw-rw-rw-</i>	
-r*-r*----	Y	N	-	-	Y
<i>Group to which GT.M is restricted</i>		<i>-rw-rw----</i>		<i>-rw-rw----</i>	
-r*-r*----	-	Y	-	Y	-
<i>Group of database file</i>		<i>-rw-rw----</i>		<i>-r*-r*----</i>	
-r*-r*----	-	N	-	Y	N

GT.M Security Philosophy

Database File Permissions	Opening process is owner of database file?	Owner is member of group of database file?	Opening process is a member of database file group?	Opening process is not owner but a member of database file group?	Execution of GT.M restricted to members of a group?
<i>Group of Resource</i>		<i>IPC Permissions</i>		<i>File Permissions</i>	
<i>Group of database file</i>		<i>-rw-rw-rw-</i>		<i>-rw-rw-rw-</i>	
<i>-r*-r*----</i>	-	N	-	Y	Y
<i>Group to which GT.M is restricted</i>		<i>-rw-rw----</i>		<i>-rw-rw----</i>	
<i>----r*----</i>	-	N	-	Y	-
<i>Group of database file</i>		<i>-rw-rw----</i>		<i>----r*----</i>	
<i>-r*-----</i>	Y	-	-	-	-
<i>Current group of process</i>		<i>-rw-----</i>		<i>-rw-----</i>	

- The resulting group ownership and permissions are found by matching the database file permissions, then determining which question columns produce the correct "Y" or "N" answer; "-" answers are "don't care".
- An asterisk ("\*") in the Database File Permissions matches writable or not writable. An asterisk in the Resulting File Permissions means that GT.M uses the write permissions from the database file.
- GT.M determines group restrictions by examining the permissions of the libgtmshr file. If it is not executable to others, GT.M treats it as restricted to members of the group of the libgtmshr file.
- Group membership can either be established by the operating system's group configuration or by the effective group id of the process.
- A GT.M process requires read access in order to perform write access to database file - a file permission of write access without read access is an error.
- GT.M treats the "root" user the same as other users, except that when it is not the file owner and not a member of the group, it is treated as if it were a member of the group.

---

## Appendix F. GTMPCAT - GT.M Process/Core Analysis Tool

Revision History		
Revision V6.0-001/2	10 April 2013	In “Usage” (page 438), added information about the --cmdfile option of --interactive.
Revision V6.0-001	27 February 2013	First published version.

---

### Overview

gtmpcat is a diagnostic tool for FIS to provide GT.M support. It gathers extensive diagnostic context from process dump (core) files and from running processes. This diagnostic information permits FIS to respond more quickly and successfully with root cause analysis of issues for which you engage with GT.M support.



#### Important

The local variables of a GT.M process may contain restricted information, such as protected health care or financial data. Under your direction, gtmpcat either ignores (the default) or accesses the contents of local variables.

The gtmpcat program is itself written using GT.M and requires:

- GT.M V5.3-004 or later when run in M mode (may misrepresent UTF-8 information if present), or
- GT.M V5.4-002 or later when run in UTF-8 mode

However, gtmpcat can analyze core files and processes from GT.M V5.1-000 or later - there need not be a relationship between the GT.M release used to run gtmpcat, and the GT.M release of a core file or process analyzed by gtmpcat.

gtmpcat requires the presence of an appropriate system debugger (for example, gdb on Linux (x86 and x86\_64) and HP-UX-IA64 or dbx on AIX and Sun SPARC Solaris). In operation, gtmpcat attaches to a system debugger, and directs it to extract information from a core file or live process.

This appendix discusses gtmpcat and how to use it.



#### Caution

Albeit small, there is a non-zero risk of premature process termination when attaching to a live process with a debugger which is what gtmpcat does when it is not directed to a core file. This risk cannot be eliminated. If problem diagnosis requires attaching to a live process, FIS recommends attaching to an expendable process, or to one in a testing environment. Use gtmpcat on a production process when you have considered all your options and carefully determined that is the best alternative.

## Usage

gtmpcat consists of two routine files -- a platform-independent main program (gtmpcat.m), and an initialization file, which is specific to the CPU architecture, operating system and GT.M version. Both the main program and the appropriate initialization file must be available in the \$gtmroutines search path. Run gtmpcat as follows:

```
$gtm_dist/mumps -run gtmpcat <options> corename|processid
```

As it executes, gtmpcat provides a running account of its actions on its stdout, as well as a detailed report in a file whose name it provides on stdout at the end of its processing.

Option values are separated from option flags by whitespace. The options are:

Options		Abbr	Description
--asver <V6.X-XXX>		-a	Specify an explicit GTM version for the core/process, if it is not the same as that specified by the -version option (which in turn defaults to that of the gtmpcat process). Note that GT.M versions are not abbreviated, for example, V6.0-001, not V60001.
--cmdfile </path/to/file-name>		-c	Specifies the path of the file containing interactive mode commands. It is ignored unless specified with --interactive (i).
--debug	*	-d	Enable interactive debugging of gtmpcat itself. On encountering an error, gtmpcat executes a BREAK command before gtmpcat either continues or exits depending on the error severity and whether an error resume point is defined. This enables some interactive debugging. Also, for a successful run, a ZSHOW dump is created with the name gtmpcat-DEBUG.zshowdump.txt. This file is overwritten/recreated as needed. The default is -nodebug.
--debuglines	*	n/a	Enables debugging of the lines written to and read from the debugger. Used for debugging gtmpcat. The default is --nodebuglines.
--devicedetail	*	-e	Include a separate section in the report to show the actual control blocks in use and their addresses.
--ignoreunexprslt	*	-u	When a GTM version is built with debugging flags, GTM can output extra lines when, for example, starting up and/or displaying the results of simple commands like .Write \$Zversion.. As part of determining what version a particular install directory has, gtmpcat executes the Write \$ZVersion command in a mumps process. If this command prints extra lines, it can cause gtmpcat initialization to fail. This option can be used to ignore the extra lines returned. The default is --noignoreunexprslt .
--interactive	*	-i	Tells gtmpcat to enter interactive mode, as described below. Use this only under the direction of FIS GT.M Support. The default is --nointeractive .
--localvar	*	-l	Include local variables, both the current local vars plus any saved (NEW'd) vars on the M stack (either explicit or implicit) in the report. Since the local variables of a process are likely to contain protected (confidential) information that is being processed, the default is ---nolocalvar to omit them. Before sharing a gtmpcat report with anyone, you must determine whether the report contains protected information and whether the recipient is permitted to view the information in the report. GT.M Support at FIS does not accept protected information.

GTMPCAT - GT.M Process/Core Analysis Tool

Options		Abbr	Description
--lockdetail	*	n/a	Include a detailed dump of M lock related control blocks showing the block addresses and relationships. The default is --nolockdetail. This option is useful only to debug GT.M itself.
--lvdetail	*	n/a	Include a detailed dump of the actual local variable structures. As this option can produce a report with protected information in local variable subscripts, please review the warnings above in the -localvar option. The default is --nolvdetail. This option is useful only to debug GT.M itself.
--memorydump	*	-m	Includes a memory map dump of all allocated storage. Only available when the core file or process is running GT.M V5.3-001 or later, and then only if \$gtmdbgvl is non-zero in the process. The default is --nodump. Use this only under the direction of FIS GT.M Support.
--mprof	*	-p	Enable M-profiling across the gtmpcat run. After loading the initialization file, gtmpcat turns on M-profiling, Just before gtmpcat completes execution, it turns off M-profiling and dumps the result in a global called ^trace. This option requires a GT.M database and global directory be available.
--msdetail		n/a	Includes additional fields from the M stack-frame. The default is --msdetail.
--mumps		n/a	the core or process is a mumps executable (default).
--mupip		n/a	the core or process is a mupip executable.
--output <file/directory>		-o	Specifies the desired output file/directory. If the value given is a directory (relative or absolute), the default file name is created in the given directory. If the value is a file, that is the file-name used to hold the report.
--tracedump		-t	Read and format the internal GTM trace table. Default is --notracedump. This is useful only to debug GT.M itself.
--version <location of the GT.M version of the core/process>		-v	Specifies the directory with the GT.M version of the core/process. The default is the version used by gtmpcat itself, that is, in \$gtm_dist.

- Abbr specifies the single character abbreviation for an option. You can combine two or more options using their single character abbreviations with a common "-" prefix and no white space in between. Specify values in the same order in which you combine the abbreviations. For example, -lov output.txt /usr/lib/fis-gtm/V6.0-001\_x86\_64 means --localvar --output output.txt --version /usr/lib/fis-gtm/V6.0-001\_x86\_64.
- \* specifies options that can be negated. To negate a single character abbreviation, use its upper case equivalent or use the full option name prefixed by "no" . For example, -P means --nomprof.

When gtmpcat runs, it needs to know how the structures and fields for a given version of GT.M are defined. There is one of these initialization files for each OS, architecture, and GT.M version. Once gtmpcat knows the architecture and GT.M version of the process/core, it invokes the proper initialization file to define the layout of everything it is interested in. The format of the gtmpcat initialization file is:

```
gtmpcat<OS>0n<architecture><gtmversion>.m
```

For example, the name of gtmpcat initialization file on Linux x86\_64 running GT.M V6.0-000 is gtmpcatLinuxOnX8664V60000.m

---

## Interactive Mode

gtmpcat has an interactive mode. Instead of producing a report, the interactive mode acts as a front-end that enhances the use of the debugger from the underlying OS.



### Caution

As interactive mode is still under development, it is likely to have rough edges. For example, M terminal IO is limited to that provided by GT.M . you can edit input using the left and right arrow keys but command retrieval is limited to the last command entered (with the up arrow key, and the down arrow key has no effect).

The *help gtmpcat* command describes currently supported commands . Any command that is not recognized in this mode, or any command prefixed with a "\" char, is sent to the native debugger and its output displayed on the console.

All of the information from the reports is available in this mode. Each "section" of the report can be printed, either the entire report or one or more at a time with the report command.

There are commands that give additional information not available in the standard report. For example:

- *cache*: The *cache* command gives information similar to the DSE CACHE command.
- *dmp*: The *dmp* command dumps a data structure, given just its address as long as the structure is defined in the initialization file.
- *dmparray*: The *dmparray* command can dump an array of control blocks in a formatted fashion. The array of control blocks are typedef structures defined in the GT.M header files and whose layouts are defined in GTMDefinedTypesInit.m<sup>1</sup>.
- *dmplist*: The *dmplist* command can dump a linked list of blocks terminating when a maximum number of such blocks is processed or if the list cycles around to the beginning or if it hits a NULL forward link.

---

## Appendix G. Packaging GT.M Applications

Revision History		
Revision V6.0-003/1	19 February 2014	First published version.

GT.M provides the **mumps -run** shell command to invoke application entryrefs directly from the shell. GT.M recognizes a number of environment variables to determine the starting characteristics of a process; these are described in the documentation and summarized in “Environment Variables” (page 17). In order to ensure that environment variables are set correctly, you should create a shell script that appropriately sets (and clears where needed) environment variables before invoking GT.M. When users should not get to the shell prompt from the application (either for lack of skill or because they are not sufficiently trusted), or when the application needs more access to the shell command line than that provided by the \$ZCMDLINE ISV, you may need to package a GT.M application using techniques beyond, or in addition to, invocation from a shell script.

Since GT.M is designed to integrate with the underlying OS, you should consider the entire range of services provided by operating systems when packaging a GT.M applications. For example, you can use the host based access control provided by TCP wrappers, or various controls provided by xinetd (including per\_source, cps, max\_load protection, only\_from, no\_access, and access\_times).

This appendix has two examples that illustrate techniques for packaging GT.M applications, neither of which excludes the other.

1. “Setting up a Captive User Application with GT.M” (page 441), such that when captive users login, they are immediately taken to the application, have no access to the shell or programmer prompt, and when they exit, are logged off the system.
2. “Invoking GT.M through a C main() program” (page 442): in addition to providing another technique for creating captive applications, invoking through a C main() is the only way that application code has access to the original argc and argv of a shell command used to start the application (the \$ZCMDLINE ISV provides an edited version of the command line).

---

### Setting up a Captive User Application with GT.M

This section discusses wholesome practices in setting up a GT.M based application on UNIX/Linux such that, when "captive" users log in to the system, they are taken directly into the application, and when they exit the application, they are logged off the system. Unless part of the application design, a captive user should not get to a shell or GT.M prompt.

The example in “Sample .profile” (page 442) is for **/bin/sh** on GNU/Linux, and may need to be adapted for use with other shells on other platforms.

At a high level, preventing a captive user from getting to a shell or GT.M prompt involves:

- trapping signals that may cause the login shell to give the user interactive access, for example, by pressing <CTRL-Z> to suspend the mumps application;
- preventing a mumps process from responding to a <CTRL-C> until the application code sets up a handler; and
- preventing an error in the application, or a bug in an error handler, from putting a captive user into direct mode.

Note that other users on the system who have appropriate privileges as managed by the operating system can still interfere with captive users. In order to secure a system for captive applications, you must protect it from untrusted other users. Users should only have credentials that permit them the level of access appropriate to their level of trustworthiness, thus: untrusted users should not have credentials to access a system with captive applications.

Defensive configuration implies setting up layers of defenses, so that an error in one layer does not compromise the system.

### Sample .profile

After initialization common to all users of a system, a login shell sources the **.profile** file in the user's home directory. A captive user's **.profile** might look something like this, where **"..."** denotes a value to be provided.

```
trap "" int quit          # terminate on SIGINT and SIGQUIT
stty susp \000           # prevent <CTRL-Z> from sending SIGSUSP

# set environment variables needed by GT.M and by application, for example
export gtm_dist=...
export gtmgbldir=...
export gtmroutines=...
export gtm_repl_instance=...
export gtm_tmp=...

# disable mumps ^C until application code sets up handler
export gtm_nocenable=1
# override default of $ZTRAP="B"
export gtm_etrp='I 0=$ST W "Process terminated by: ",$ZS,! ZHALT 1'

# set other environment variables as appropriate, for example
export EDITOR=...        # a preferred editor for ZEDIT
export TZ=...            # a timezone different from system default
export HUGETLB_SHM=yes   # example of a potential performance setting

export PATH=/usr/bin:/bin # only the minimum needed by application
export SHELL=/bin/false  # disable ZSYSTEM from command prompt

# execute captive application starting with entryref ABC^DEF then exit
exec $gtm_dist/mumps -run ABC^DEF
```

Note the use of **exec** to run the application - this terminates the shell and disconnects users from the system when they exit the GT.M application.

If an incoming connection is via an Internet superserver such as xinetd, some of these are not applicable, such as disabling <CTRL-C> and <CTRL-Z>.

---

### Invoking GT.M through a C main() program

There are several circumstances when it is desirable to invoke a GT.M application with a top-level C main() program rather than with **mumps -run**. Examples include:

- A need to ensure correct values for environment variables, and a shell script cannot be used (for example, when there is a specific operational need to install an application with the setuid bit).
- Programs that show up on a process display with meaningful names (like **friday** instead of **mumps -run monthstarting friday**, in the following example).

To compile and run the `monthstarting.zip` example, perform the following steps:

1. Download `monthstarting.zip`.

`monthstarting.zip` contains `monthstarting.m`, `month_starting.c`, and `monthstarting.ci`. To download `monthstarting.zip`, click on . You can also download `monthstarting.zip` from [http://tinco.pair.com/bhaskar/gtm/doc/books/ao/UNIX\\_manual/downloadables/monthstarting.zip](http://tinco.pair.com/bhaskar/gtm/doc/books/ao/UNIX_manual/downloadables/monthstarting.zip).

2. Run the `monthstarting.m` program that lists months starting with the specified day of the week and year range.

```
$ mumps -run monthstarting Friday 1986 1988
FRI AUG 01, 1986
FRI MAY 01, 1987
FRI JAN 01, 1988
FRI APR 01, 1988
FRI JUL 01, 1988
$
```

Notice that this program consists of a main program that reads the command line in the intrinsic special variable `$ZCMDLINE`, and calls `calcprint^monthstarting()`, providing as its first parameter the day of the week to be reported.

This step is optional as there is no need to explicitly compile `monthstarting.m` because GT.M autocompiles it as needed.

3. On x86 GNU/Linux (64-bit Ubuntu 12.04), execute the following command to compile `month_starting.c` and create an executable called `friday`.

```
$ gcc -c month_starting.c -I$gtm_dist
$ gcc month_starting.o -o friday -L $gtm_dist -Wl,-rpath=$gtm_dist -lgtmshr
```

For compiling the `month_starting.c` program on other platforms, refer to the Integrating External Routines chapter of *GT.M Programmer's Guide*

4. Execute the following command:

```
$ ./friday 1986 1988
FRI AUG 01, 1986
FRI MAY 01, 1987
FRI JAN 01, 1988
FRI APR 01, 1988
FRI JUL 01, 1988
$
```

5. You can also execute the same program with the name `monday`. In doing so, the program displays months starting with Monday.

```
$ ln -s friday monday
$ ./monday 1986 1988
MON SEP 01, 1986
MON DEC 01, 1986
MON JUN 01, 1987
MON FEB 01, 1988
MON AUG 01, 1988
$
```

The **month\_starting.c** program accomplishes this by calling the same GT.M entryref `calprint^monthstarting()`, and passing in as the first parameter the C string `argv[0]`, which is the name by which the program is executed. If there are additional parameters, **month\_starting.c** passes them to the M function; otherwise it passes pointers to null strings:

```
/* Initialize and call calprint^monthstarting() */
if ( 0 == gtm_init() ) gtm_ci("calprint", &status, argv[0], argc>1 ? argv[1] : "", argc>2 ? argv[2] :
"");
```



Prior to calling the GT.M entryref, the C program also needs to set environment variables if they are not set: `gtm_dist` to point to the directory where GT.M is installed, `gtmroutines` to enable GT.M to find the `monthstarting` M routine as well as GT.M's `%DATE` utility program, and `GTMCI` to point to the call-in table:

```
/* Define environment variables if not already defined */
setenv( "gtm_dist", "/usr/lib/fis-gtm/V6.1-000_x86_64", 0 );
if (NULL == getenv( "gtmroutines" ))
{
    tmp1 = strlen( getenv( "PWD" ) );
    strncpy( strbuf, getenv( "PWD" ), tmp1 );
    strcpy( strbuf+tmp1, " " );
    tmp2 = tmp1+1;
    tmp1 = strlen( getenv( "gtm_dist" ) );
    strncpy( strbuf+tmp2, getenv( "gtm_dist" ), tmp1 );
    tmp2 += tmp1;
    if ( 8 == sizeof( char * ) )
    {
        tmp1 = strlen( "/libgtmutil.so" );
        strncpy( strbuf+tmp2, "/libgtmutil.so", tmp1 );
        tmp2 += tmp1;
    }
    strcpy( strbuf+tmp2, "" );
    setenv( "gtmroutines", strbuf, 1 );
}
setenv( "GTMCI", "monthstarting.ci", 0 );

if ( 0 == gtm_init() ) gtm_ci("calprint", &status, argv[0], argc>1 ? argv[1] : "", argc>2 ? argv[2] : "");
gtm_exit(); /* Discard status from gtm_exit and return status from function call */
```

Note that on 32-bit platforms, the last element of `gtmroutines` is `$gtm_dist`, whereas on 64-bit platforms, it is `$gtm_dist/libgtmutil.so`. If you are creating a wrapper to ensure that environment variables are set correctly because their values cannot be trusted, you should also review and set the environment variables discussed in “Setting up a Captive User Application with GT.M” (page 441).

All the C program needs to do is to set environment variables and call a GT.M entryref. A call-in table is a text file that maps C names and parameters to M names and parameters. In this case, the call-in table is just a single line to map the C function `calprint()` to the GT.M entryref **calprint^monthstarting()**:

```
calprint : gtm_int_t* calprint^monthstarting(I:gtm_char_t*, I:gtm_char_t*, I:gtm_char_t*)
```

## Defensive Practices

The following practices, some of which are illustrated in “Sample .profile” [442], help provide layered defenses:

1. Setting the **gtm\_noceenable** environment variable to a value to specify that <CTRL-C> should be ignored by the application, at least until it sets up a <CTRL-C> handler. As part of its startup, the application process might execute:

```
USE $PRINCIPAL : (EXCEPTION="ZGOTO"_$ZLEVEL_" : DONE" : CTRAP=$CHAR(3) : CENABLE)
```

to set up a handler such as:

**DONE: QUIT ; or HALT or ZHALT**, as appropriate

2. Providing a value to the **gtm\_etrapp** environment variable, as illustrated “Sample .profile” (page 442). This overrides GT.M's default value of "B" for \$ZTRAP, which puts the application into direct mode. Of course, in a development environment, going to direct mode may be the correct behavior, in which case there is no need to set **gtm\_etrapp**.
3. Providing a value to the **gtm\_zinterrupt** environment to override the default of "IF \$ZJOBEXAM()" which causes the process to create a text file of its state in response to a MUPIP INTRPT (or SIGUSR1 signal). Such a text file may contain confidential information that the process is actively computing. Note that a user can only send INTRPT signals as permitted by the configuration of system security for the user. If your application uses INTRPT signals, review the code they invoke carefully to ensure processes respond appropriately to the signal. If any response produces an output file, be sure they have write access to the destination; restrict read access to such files appropriately. The “Sample .profile” (page 442) example does not illustrate an alternative value for **gtm\_interrupt**.
4. Setting the SHELL environment variable to /bin/false disables the ZSYSTEM command, which if executed without an argument takes the user to a shell prompt. While a correctly coded application might not have a ZSYSTEM without an argument, setting SHELL to a value such as /bin/false, as illustrated above, protects an added layer of defense against a possible application bug. Of course, if an application uses the ZSYSTEM command, then an executable SHELL is required. If your application uses ZSYSTEM to run a command, consider whether a PIPE device might provide a better alternative.
5. Setting the PATH environment explicitly to only those directories that contain executable files that the mumps process will need to execute, with a ZSYSTEM command or a PIPE device.
6. Because some text editors include functionality to run a shell in an edit buffer, setting the EDITOR variable to an editor which does not have such functionality is a way to block shell access in the event the application uses the ZEDIT command to edit a text file. Note that if an application allows users to edit text files, they can also edit GT.M program source files, and application configuration should ensure that such program files cannot be accessed by the \$ZROUTINES of the process unless that is the desired behavior.

---

## Other

Depending on application requirements, other packaging technologies for consideration include:

1. Choosing a restricted shell for login of a captive user, such as rbash, instead of /bin/sh (for example, see [http://en.wikipedia.org/wiki/Restricted\\_shell](http://en.wikipedia.org/wiki/Restricted_shell)).
2. Setting up a chroot environment for an application used by captive users (for example, see <http://en.wikipedia.org/wiki/Chroot>).
3. Using TCP wrappers to filter incoming requests (for example, see <http://www.360is.com/03-tcpwrappers.htm>).
4. Configuring mandatory access controls, such as SELinux (for example, <http://opensource.com/business/13/11/selinux-policy-guide>).

# Glossary

## I

**In-flight transactions**      *in-flight* transactions are those transactions that have been committed (or hardened, or made Durable) on an originating instance (or a propagating instance) but not yet transmitted to a replicating instance.

## O

**Originating Instance**      An *originating instance* is an instance that processes business logic and generates logical updates to the database.

## R

**Replicating Instance**      A *replicating instance* is an instance that does not execute business logic or generate database updates. It receives logical database updates from an originating instance or another replicating instance and applies them to its database.

**Receiving Instance**      Within the context of a replication connection between two instances, a replicating instance that receives updates from a source instance is referred to as *receiving instance* or receiver side. For example, in an B<-A->C replication configuration, both B and C can be referred to as a receiving instance.

## S

**Source Instance**      Within the context of a replication connection between two instances, an originating instance is referred to as *source instance* or source side. For example, in an B<-A->C replication configuration, A is the source instance for B and C.

**Supplementary Instance**      A *supplementary instance* is an instance that processes business logic, generates logical updates to the database, and at the same time receives updates from another instance.

# Index

## Symbols

@ command  
in GDE, 43

## A

Abandoned\_kills qualifier  
in DSE, 290

Access\_method qualifier  
buffered global, memory mapped, 58  
changing, 59  
in GDE, 58  
with MUPIP SET, 110

Action qualifiers, 150

Activate qualifier  
with MUPIP REPLICATE, 237

Active\_processes show option, 157

Add command  
example, 45  
examples, 282, 286  
in DSE, 281  
in GDE, 43

Adjacency qualifier  
with MUPIP INTEG, 90

After qualifier  
with MUPIP JOURNAL, 161

Alignsize journal option, 143

All command  
in DSE, 283

All qualifier  
in DSE, Dump, 301  
in LKE, 254, 257

All show option, 157

Allocation journal option, 144

Allocation qualifer  
in GDE, 59

Always fence option, 165

Apply\_after\_image qualifier  
with MUPIP JOURNAL, 163

Autorollback qualifier  
specifying , 244

AUTOSWITCHLIMIT  
journal option in GDE, 57

Autoswitchlimit journal option, 144

## B

b\*-tree, 272  
Global Variable Tree, 272

Backup command, 71  
examples, 77

Backward qualifier  
in DSE, Shift, 317  
with MUPIP JOURNAL, 154, 160

backward recovery, 154  
definition, 133

Before qualifier  
with MUPIP JOURNAL, 161

BEFORE\_IMAGES  
journal option in GDE, 57  
journaling type of, definition, 135

Before\_images option, 145

Begin qualifier  
with MUPIP LOAD, 100

BINARY, 83

bitmaps  
local, 271  
master, 272

Bkupdbjnl qualifier  
with MUPIP BACKUP, 73

Blk\_size qualifier  
in DSE, 290

Block qualifier  
in DSE, 282, 288  
in DSE, Dump, 302  
in DSE, Find, 306  
in DSE, Integrit, 309  
in DSE, Map, 309  
in DSE, Overwrite, 312  
in DSE, Remove, 315  
in DSE, Restore, 316  
in DSE, Save, 316  
with MUPIP INTEG, 82, 91

Blocks\_free qualifier  
in DSE, 291

BLOCK\_size qualifer  
in GDE, 59

Brief qualifier  
with MUPIP INTEG, 91

Broken Transaction file, 157  
definition, 136

Brokentrans qualifier  
with MUPIP JOURNAL, 163

Broken\_transaction show option, 157

Bsiz qualifier  
in DSE, 289

Buffer\_flush command  
in DSE, 287

Buffer\_flush qualifier

- in DSE, 284
- Buffer\_size journal option, 145
- Buffsize qualifier
  - with MUPIP REPLICATE, 231
- Busy qualifier
  - in DSE, Map, 309
  - in DSE, Range, 313
- Bypass qualifier
  - with Jnl\_file qualfier, 142
- Bytestream qualifier
  - in DSE, 291
  - with MUPIP BACKUP, 74

**C**

- Cache command
  - examples, 299
  - in DSE, 298
- Chain qualifier
  - with MUPIP JOURNAL, 163
- Change command
  - example, 47
  - examples
    - in DSE, 296
  - in DSE, 287
  - in GDE, 46
- Change qualifier
  - with MUPIP REPLICATE, 229
- Changelog qualifier
  - with MUPIP REPLICATE, 239, 248
- Changing the Log File
  - log interval setting , 240, 248
- Checkhealth qualifier
  - with MUPIP REPLICATE, 239
- Checktn qualifier
  - with MUPIP JOURNAL, 164
- Clear command
  - in LKE, 253
- Close command
  - in DSE, 299
- closing database, 108
- CMPC qualifier
  - in DSE, 290
- Cmplvl qualifier
  - with MUPIP REPLICATE, 235
- Collation qualifier
  - in GDE, 63
- Collation\_default qualifier
  - in GDE, 54
- Command qualifier
  - File, 50

## Index

- example, 52
  - rebuild Global Directory , 50
- Comment qualifier
  - in DSE, Save, 317
- Commitwait\_spin\_count qualifier
  - in DSE, 292
- Control qualifiers, 150
- Corrupt\_file qualifier
  - in DSE, 291
- Count qualifier
  - in DSE, Dump, 302
  - in DSE, Remove, 315
- Create command, 79
  - examples, 79
- Crit qualifier
  - in DSE, 302
  - in DSE, Find, 307
  - in DSE, Range, 313
  - in DSE, Save, 317
  - in LKE, 254, 258
- Critical command
  - in DSE, 299
- Critical section
  - in DSE, 299
- Critinit qualifier
  - in DSE, 284
- Current\_tn qualifier
  - in DSE, 292
- Cutover
  - definition, 192

**D**

- Data qualifier
  - in DSE, 282
  - in DSE, Overwrite, 312
- database crash
  - immediately after, 325
- database integrity, 324
  - preventive, 326
- database integrity check, 88
- Database qualifier
  - in DSE, 291
  - with MUPIP BACKUP, 74
- database repair, 202
  - determining cause, 327
  - repair decision tree, 334
  - safety guidelines, 329
  - with MUPIP, 327
- Db qualifier
  - with MUPIP FTOK, 88

## Index

- Dbfilename qualifier
  - with Jnl\_file qualifier, 142
- Deactive qualifier
  - with MUPIP REPLICATE, 238
- deadlock
  - resolving, 263
- Decimal qualifier
  - in DSE, 304
- Decimal to Hexadecimal
  - in DSE, Evaluate, 304
- Declocation qualifier
  - in DSE, 292
- Delete command
  - example, 47
  - in GDE, 47
- Detail qualifier
  - with MUPIP REPLICATE, 229
- Direction qualifiers, 150
- Disable journal option, 145
- Downgrade command, 80
  - examples, 80
- Downloadable Replication Examples
  - A->B, 205
  - A->P, 206
- DSE, 279
  - Cache, 298
  - changing, 287
  - changing DSE region, 307
  - Command Summary, 319
  - Critical, 299
  - Dump, 301
  - Evaluate, 304
  - Exit, 305
  - Find, 305
  - Integrit, 308
  - invoke, 279
  - Map, 309
  - Open, 311
  - Overwrite, 312
  - Page, 312
  - Range, 313
  - Remove, 314
  - Restore, 315
  - Save, 316
  - Shift, 317
  - Spawn, 318
  - syntax, 281
  - Wcinit, 318
- Dump command
  - in DSE, 301
- Dynamic\_segment qualifier
  - in GDE, 54
- E**
- Editinstance qualifier
  - with MUPIP REPLICATE, 229
- Enable journal option, 145
- Enabling/Disabling Detailed Logging
  - log interval setting , 240, 249
- Encryption, 363
  - alternatives, 366
  - building libraries, 425
  - changing encryption keys, 394
  - changing hash, 393
  - definitions, 367
  - example, 376
  - installing , 390
  - master key, 392
  - reference implementation, 387
    - architecture, 394
    - installing seperately, 398
    - using with older releases, 397
  - role of keys, 364
- Encryption qualifer
  - in GDE, 60
- Encryption\_hash qualifier
  - in DSE, 292
- End qualifier
  - with MUPIP LOAD, 100
- Endiancvt command
  - outdb qualifier, convert endian format, 80
- Enviroment variables
  - EDITOR, 17
  - GTMCCL, 23
  - gtmcompile, 23
  - gtmencrypt\_config, 24
  - gtmencrypt\_FIPS, 24
  - gtmgblmdir, 24
  - gtmroutines, 24
  - gtmtls\_passwd\_<label>, 24
  - GTMXC\_gpgagent, 24
  - gtm\_badchar, 18
  - gtm\_baktmpdir, 18
  - gtm\_chset, 18
  - gtm\_chset\_locale, z/OS only, 18
  - gtm\_collate\_n, 18
  - gtm\_crypt\_plugin, 18
  - gtm\_custom\_errors, 18
  - gtm\_dbkeys, 18
  - gtm\_db\_startup\_max\_wait, 18

## Index

gtm\_dist, 18  
gtm\_env\_translate, 19  
gtm\_extract\_nocol, 19  
gtm\_fullblockwrites, 19  
gtm\_gdscert, 19  
gtm\_gvdupsetnoop, 19  
gtm\_icu\_version, 19  
gtm\_ipv4\_only, 19  
gtm\_jnl\_release\_timeout, 19  
gtm\_keep\_obj, 20  
gtm\_lct\_stdnull, 20  
gtm\_local\_collate, 20  
gtm\_max\_sockets, 20  
gtm\_memory\_reserve, 20  
gtm\_nocenable, 20  
gtm\_non\_blocked\_write\_retries, 20  
gtm\_noundef, 20  
gtm\_obfuscation\_key, 21  
gtm\_passwd, 21  
gtm\_patnumeric, 21  
gtm\_pattern\_file, 21  
gtm\_principal, 21  
gtm\_principal\_editing, 21  
gtm\_procstuckexec, 21  
gtm\_prompt, 21  
gtm\_quiet\_halt, 22  
gtm\_repl\_instance, 22  
gtm\_repl\_instsecondary, 22  
gtm\_retention, 22  
gtm\_side\_effects, 22  
gtm\_snaptmpdir, 22  
gtm\_stdxkill, 22  
gtm\_sysid, 22  
gtm\_tprestart\_log\_delta, 20  
gtm\_tprestart\_log\_first, 20  
gtm\_trigger\_etrap, 23  
gtm\_ver, 24  
gtm\_zdate\_form, 23  
gtm\_zinterrupt, 23  
gtm\_zlib\_cmp\_level, 23  
gtm\_zmaxptime, 23  
gtm\_zquit\_anyway, 23  
gtm\_ztrap\_form, 23  
gtm\_ztrap\_new, 23  
LC\_ALL, 24  
LC\_CTYPE, 24  
LD\_LIBRARY\_PATH, 24  
old\_gtmroutines, 24  
old\_gtm\_dist, 24  
summary, 17

tmp\_gtm\_tmp, 24  
tmp\_passw, 24  
TZ, 24  
Environment variables  
  gtm\_etrap, 19  
  gtm\_tptime, 23  
  gtm\_trace\_gtm\_name, 23  
Epoch\_interval option, 145  
Error\_limit qualifier  
  with MUPIP JOURNAL, 164  
estimate global variable size, 114  
Evaluate command  
  examples, 305  
  in DSE, 304  
Exact qualifier  
  in LKE, 254  
Exclude qualifier  
  with MUPIP REORG, 104  
Exhaustive qualifier  
  in DSE, Find, 306  
Exit command  
  in DSE, 305  
  in GDE, 47  
  in LKE, 260  
  with MUPIP, 81  
Extend command, 81  
  examples, 82  
Extend qualifier  
  RESTORE, 107  
Extension\_count qualifer  
  in GDE, 60  
Extension\_count qualifier  
  used by MUPIP RESTORE, 107  
  with MUPIP SET, 111  
Extention journal option, 146  
Extention\_count qualifier  
  used by MUPIP EXTEND, 81  
Extract command, 82  
  examples, 85  
Extract formats  
  GLO, in DSE, 302  
  ZWR, in DSE, 303  
Extract formats, BINARY, GO, ZWR, 83  
Extract qualifier  
  with MUPIP JOURNAL, 153

## F

Fast qualifier  
  with MUPIP INTEG, 91  
Fence options

## Index

- with MUPIP JOURNAL, 165
- Fences qualifier
  - with MUPIP JOURNAL, 164
- Fetchresync qualifier
  - example, 249
  - examples, 163
  - specifying port number, 249
  - with MUPIP JOURNAL, 162
- file header, 265
  - changing attributes, 290
  - elements, 266
- File qualifier
  - in DSE, Open, 311
  - with MUPIP INTEG, 91
  - with MUPIP RUNDOWN, 109
  - with MUPIP SET, 110, 141, 226
- Fileheader qualifier
  - in DSE, 290
  - in DSE, Dump, 302
- Filename journal option, 146
- File\_name qualifer
  - default file name is, 61
  - in GDE, 60
- Fill\_factor qualifier
  - with MUPIP LOAD, 100
  - with MUPIP REORG, 104
- Filter qualifier
  - with MUPIP REPLICATE, 231
- Filters
  - change schema, 219
  - definition, 188
  - example, 231
  - specifying, 231
  - stopping source , 239
  - stopping source filter, 239, 244
- Find command
  - examples, 307
  - in DSE, 305
- Fixing damaged spanning node
  - example of, 354
- Flush\_time qualifier
  - in DSE, 292
  - with MUPIP SET, 111
- Format qualifier
  - with MUPIP EXTRACT, 83
  - with MUPIP LOAD, 99
- Forward qualifier
  - in DSE, Shift, 317
  - with MUPIP JOURNAL, 153, 160
- Forward Recovery

- definition, 132
- forward recovery, 153
- Free qualifier
  - in DSE, Map, 310
- Freeblock qualifier
  - in DSE, Find, 306
- Freeze command, 86
  - examples, 87
- Freeze qualifier
  - in DSE, 284, 293
  - with MUPIP EXTRACT, 84
- From qualifier
  - in DSE, Range, 313
  - in DSE, Restore, 316
- FTOK
  - examining IPCs, 405
- Ftok command, 88
  - examining IPCs, 405
- Full qualifier
  - with MUPIP INTEG, 92
  - with MUPIP JOURNAL, 165
- Fully\_upgraded qualifier
  - in DSE, 293

## G

- Gblname qualifier
  - in GDE, 63
- GDE
  - command syntax, 42
- GDS, 265
  - blocks, 273
  - compression count, 275
  - keys, 274
    - characteristics, 275
    - uses of, 275
  - records, 274
  - spanning nodes, 274
- Glo qualifier
  - in DSE, 302
- Global Directory
  - Abbreviations, 38
  - Command Summary, 64
  - creating, 35
  - Customizing, 39
  - definition, 32
  - examining, 36
  - examples of mapping, 41
  - How to customize
    - Add a Journaling Information Section, 39
  - Identify, 35

- invoke GDE, 39
- mapping, 36
- mapping guidelines, 40
- overview, 34
- Qualifier Summary, 66
- Global qualifier
  - examples, 167
  - with MUPIP JOURNAL, 167
- Global\_buffers\_count qualifier
  - with MUPIP SET, 111
- Global\_buffer\_count qualifier
  - definition, 61
  - in GDE, 61
- GO, 83
- GT.M, 1
  - configuring huge pages for GT.M on Linux, 29
  - encrypting, 363
  - gdedefaults, 17
  - general database management, 67
  - gtm, 17
  - installation prerequisites, 4
  - installing, 4
  - installing with gtminstall, 10
  - installing with Unicode support, 26
  - run, 27
  - setting an environment for, 13
  - using gtmbase, 17
  - using gtmprofile, 13
  - Utility, DSE, 3
  - Utility, GDE, 2
  - Utility, LKE, 3
  - Utility, MUPIP, 2
- gtmroutines
  - libgtmutil.so, 26
- gtmsecshr
  - considerations, 25
  - definition, 409
  - gtm\_secshr\_log, 20
  - log files, 412
  - temporary files, 22
- Gvstats qualifier
  - in DSE, 302
- Gvstatsreset qualifier
  - in DSE, 293

## H

- Header qualifier
  - in DSE, 302, 302
- Header show option, 157
- Help command

## Index

- in GDE, 48
- Helper Processes
  - definition, 187
  - Reader, 187
  - Updproc, in DSE, 303
  - with MUPIP REPLICATE , 247, 248
  - Writer, 187
- Hexadecimal qualifier
  - in DSE, 304
- Hexadecimal to Decimal
  - in DSE, Evaluate, 304
- Hexlocation qualifier
  - in DSE, 293
- Hint qualifier
  - in DSE, Find, 306
- History records, 189
- huge pages
  - improving performance with, 29

## I

- ICU
  - compiling, 414
  - used by, 26
- Id qualifier
  - with MUPIP JOURNAL, 168
- Index qualifier
  - in DSE, Range, 313
- Index\_fill\_factor qualifier
  - with MUPIP REORG, 105
- Init qualifier
  - in DSE, 300
- Initialize qualifier
  - with MUPIP REPLICATE , 245
- Instance Freeze
  - definition, 193
  - with GDE, 56
  - with MUPIP REPLICATE, 232
  - with MUPIP SET, 112
- Instance\_create qualifier
  - with MUPIP REPLICATE, 228
- Instsecondary qualifier
  - with MUPIP REPLICATE, 233, 238, 238, 239, 240, 241
- Inst\_freeze\_on\_errort qualifier
  - with MUPIP SET, 112
- Integ command, 88
  - examples, 96
  - fixing error, 325
- Integrit command
  - in DSE, 308
- Interactive qualifier

- in LKE, 255
- with MUPIP JOURNAL, 165

interrupt process, 98

Interrupt\_recov qualifier

- in DSE, 293

Intrpt command, 98

**J**

Jnlfile qualifier

- with MUPIP SET, 141

Jnlpool qualifier

- with MUPIP FTOK, 88
- with MUPIP REPLICATE, 229

Journal command, 150

- build a MUPIP JOURNAL command, 150

Journal Extract

- format, 168

Journal options, 143

- in general database management, 112

Journal Pool, 186

- selecting, 229

Journal qualifier

- Allocation, 53
- Allocation option, 57
- Autoswitch option, 57
- Before\_image option, 57
- Buffer\_size option, 57
- Extension option, 57
- File\_name, 57
- File\_name option, 57
- in GDE, 56
- in general database management, 112

Journaling, 130

- autoswitch limit, 131
- backup strategy, 136
- backward recovery, initiate, 154
- BEFORE\_IMAGE, 135
- benefits, 136
- choosing between, 135, 200
- forward recovery, initiate, 153
- Identify journaling actions, 142
- identify journaling targets, 141
- initialize database recovery, 153
- initialize database recovery for replicated database, 155
- NOBEFORE\_IMAGE, 135
- previous generation journal files, 131, 164
- rollback to a common synchronization point, 162
- rolled\_bak\*, 155
- specifying error limit, 164
- storing database update information, 131

## Index

### K

Key qualifier

- in DSE, 282
- in DSE, Find, 307

Keyrange qualifier

- with MUPIP INTEG, 92

Key\_max\_size qualifier

- in DSE, 293

Key\_size

- example, 55

Key\_size qualifier

- in GDE, 55

Kill\_in\_prog qualifier

- in DSE, 294

### L

Label qualifier

- with MUPIP EXTRACT, 84

Level qualifier

- in DSE, 289

libgtmutil.so, 26

- installing, 7

List qualifier

- in DSE, Save, 317

Listenport qualifier

- specifying , 244

LKE, 252

- CLEAR, 253
- Command Summary, 261
- EXIT, 260
- invoking, 252
- M Lock space, 271
- SHOW, 257
- syntax, 253

LMS

- application architecture, 190
- architecture, 186
- choosing between journaling type, 200
- cutover, 192
- message delivery application, 191
- transaction processing application, 190

Load command, 98

- examples, 101

load data from extract file, 98

Lock qualifier

- in LKE, 254, 258

Lock usage, 262

LOCKS command

- example, 48
- in GDE, 48

## Index

- Lock\_space qualifier
  - in GDE, 61
  - with MUPIP SET, 111
- Log command
  - example, 49
  - in GDE, 49
- Log qualifier
  - with MUPIP EXTRACT, 84
  - with MUPIP REPLICATE, 231
- Log\_interval qualifier
  - Starting the Receiver Server, 244
  - with Changelog (Receiver Server), 248
  - with Changelog (Source Server), 240
  - with MUPIP REPLICATE, 231
  - with Statslog (Receiver Server), 249
  - with Statslog (Source Server), 240
- Lookback\_limit options
  - with MUPIP JOURNAL, 161
- Lookback\_limit qualifier
  - with MUPIP JOURNAL, 161
- Lost qualifier
  - in DSE, Range, 313
- Lost Transaction file
  - definition, 136
  - format, 243
  - processing, 241
  - processing after, 242, 242
  - specifying file , 166
- Lost transaction file
  - creating, 250
- Losttncomplete qualifier
  - specifying , 242
- Losttrans qualifier
  - with -ROLLBACK, 250
  - with MUPIP JOURNAL, 166
- Lower qualifier
  - in DSE, Range, 313

## M

- manage triggers, 116
- Map command
  - in DSE, 309
- Map qualifier
  - with MUPIP INTEG, 92
- Master qualifier
  - in DSE, Map, 310
- Maxkeysize qualifier
  - with MUPIP INTEG, 93
- Monitoring
  - messages and errors, 411

- MUPIP
  - BACKUP, 71
    - example, 73
  - Commands Summary, 121
  - CREATE, 79
  - DOWNGRADE, 80
  - ENDIANCVT, 80
  - EXIT, 81
  - EXTEND, 81
  - Extract, 82
  - FREEZE, 86
  - FTOK, 88
  - INTEG, 88
  - INTRPT, 98
  - JOURNAL, 150
  - LOAD, 98
  - REORG, 102
  - REPLICATE, 226
  - RESTORE, 107
  - RUNDOWN, 108
  - SET, 109, 140
  - SIZE, 114
  - Standalone and concurrent, 69
  - STOP, 116
  - TRIGGER, 116
  - UPGRADE, 120
- Mutex\_slots qualifier
  - in GDE, 62
  - with MUPIP SET, 112

## N

- Name qualifier
  - example, 54
  - in GDE, 54
  - with MUPIP REPLICATE, 228, 229
- needrestart qualifier, 242
- NETtimeout qualifier
  - with MUPIP BACKUP (bytestream), 74
- Newjnlfiles
  - with MUPIP BACKUP, 75
- NOBEFORE\_IMAGES
  - journaling type of, definition, 135
- None fence option, 165
- Noreplace qualifier
  - with MUPIP REPLICATE, 228
- Noresync qualifier
  - with MUPIP REPLICATE , 246
- Null\_subscripts qualifier, 55
  - in DSE, 294
- Number qualifier

- in DSE, 305

**O**

- Off qualifier
  - with MUPIP FREEZE, 87
- OFF,ON journal options, 148
- Offset qualifier
  - in DSE, 282, 289, 302
  - in DSE, Map, 312
  - in DSE, Remove, 315
  - in DSE, Shift, 317
  - with MUPIP REPLICATE, 229
- On qualifier
  - with MUPIP FREEZE, 87
- Online qualifier
  - with MUPIP BACKUP, 75
  - with MUPIP INTEG, 93
- Open command
  - in DSE, 311
- orderly terminate database process, 116
- originating instance
  - propagating, 234
  - update last known synchronization point, 244
- Output qualifier
  - in LKE, 255, 258
- override qualifier
  - with MUPIP ENDIANCVT, 80
- Override qualifier
  - in DSE, 285, 294
  - with MUPIP FREEZE, 87
  - with MUPIP RUNDOWN, 109
- Overwrite command
  - in DSE, 312
- Owner qualifier
  - in DSE, 300

**P**

- Page command
  - in DSE, 312
- Partial\_recov\_bypass qualifier
  - with MUPIP SET, 110
- Passive qualifier
  - with MUPIP REPLICATE, 231
- perform journaling operations, 150
- Pid qualifier
  - in LKE, 255, 258
- Plaintextflassback qualifier
  - Starting the Source Server, 236
- Pointer qualifier
  - in DSE, 282

## Index

- Prevjnlfile journal option, 147
- Prevjnlfile qualifier
  - with Jnl\_file qualifier, 142
- PREVLINK
  - with Newjnlfiles, MUPIP BACKUP, 75
- Process fence option, 165
- Processes show option, 158
- Propagateprimary qualifier
  - with MUPIP REPLICATE, 234, 238, 238

**Q**

- Qdbrundown
  - in DSE, 294
- Qdbrundown qualifier
  - in DSE, 294
  - in general database management, 112
  - with GDE, 56
- Quit command
  - in GDE, 49

**R**

- Range command
  - in DSE, 313
- Receiver qualifier
  - with MUPIP REPLICATE, 243
- Receiver Server, 186
  - add helpers, 247
  - automatic online rollback , 244
  - changing logs, 248
  - checking progress, 249
  - enable logging, disable logging, 248
  - specify port to listen, 244
  - specifying compression level, 235
  - starting, 243
  - update process, 246
- Record qualifier
  - in DSE, 282, 289, 303
  - in DSE, Remove, 315
  - with MUPIP BACKUP, 75
  - with MUPIP FREEZE, 87
- Record\_max\_size qualifier
  - in DSE, 294
- Record\_size
  - in DSE, 294
  - in GDE, 55
- Recover qualifier
  - in DSE, 298
  - with MUPIP JOURNAL, 153
- Recovery, 325
  - from backup files, 326

## Index

- from journal files, 326
- Recvpool qualifier
  - with MUPIP FTOK, 88
- Redirect qualifier
  - with MUPIP JOURNAL, 166
- Reference qualifier
  - in DSE, 285
- Reference\_count
  - in DSE, 294
- Region qualifier
  - in DSE, Find, 307
  - in DSE, Restore, 316
  - in GDE, 54
  - in LKE, 255, 259
- Journal qualifier
  - journal options summary, 58
  - with MUPIP CREATE, 79
  - with MUPIP INTEG, 94
  - with MUPIP REORG, 105
  - with MUPIP RUNDOWN, 109
  - with MUPIP SET, 110, 141, 226
- Region Qualifiers
  - with GDE, 56
- Reg\_seqno qualifier
  - in DSE, 295
- Release qualifier
  - in DSE, 285, 300
- Remove command
  - in DSE, 314
- Remove qualifier
  - in DSE, 300
- Rename command
  - example, 50
  - in GDE, 49
- Renegotiate\_interval qualifier
  - Starting the Source Server, 237
- Renew qualifier
  - in DSE, 285
- Reorg command, 102
  - examples, 107
- reorganize, defragment, optimize, reduce database size , 102
- REPLace qualifier
  - with MUPIP BACKUP, 76
- Replace qualifier
  - with MUPIP REPLICATE, 228
- Replicate command
  - examples, 234
- Replicate qualifier
  - with MUPIP REPLICATE, 230
- replicating instance
  - accepting SI stream, 245, 246
  - identify, 230, 233
  - if previously used, 246
  - initializing SI stream, 245
- Replication
  - architecture, 186
  - changing replication state, 226
  - Filters, 188
  - implementation procedures, 203
  - LMS Group
    - SI Group, 175
  - Replication instance file, 189
  - rolling upgrades, 203
  - specifying compression level, 235
- Replication instance file
  - definition, 189
- Replication Instance file
  - display, change, 229
- Replication qualifier
  - with MUPIP SET, 143, 226
- Replication state options
  - Off, 226
  - On, 227
  - Was\_on, 227
- Replinstance qualifier, 189
  - with MUPIP BACKUP, 76
- Repl\_state qualifier
  - with Jnl\_file qualifier, 142
- Reserved\_bytes qualifier
  - in GDE, 62
  - with MUPIP SET, 113
- Reset qualifier
  - in DSE, 300
- Restart Codes, 359
- Restore command, 107
  - examples, 108
  - in DSE, 315
- Restore\_all qualifier
  - in DSE, Map, 310
- restoring incremental backup, 107
- Resume qualifier
  - with MUPIP REORG, 105
  - with MUPIP REPLICATE , 245
- Resync qualifier
  - with MUPIP REPLICATE, 250
- Resync\_seqno qualifier
  - in DSE, 295
- Resync\_strm qualifier
  - with -ROLLBACK, 250
- Resync\_tn qualifier

- in DSE, 295
- Reuse qualifier
  - with MUPIP REPLICATE , 246
- rollback
  - to a journal stream sequence number, 250
- Rollback qualifier, 249
  - with MUPIP JOURNAL, 155
- rolled\_bak\* files, 155
  - definition, 134
- Rootprimary, 192
  - example, 234
  - specifying, 233
- Rootprimary qualifier
  - with MUPIP REPLICATE, 233, 238, 238
- Rundown command, 108

## S

- Save command
  - in DSE, 316
- Secondary qualifier
  - with MUPIP REPLICATE, 230
- Segment qualifier
  - in GDE, 58
- Seize qualifier
  - in DSE, 301
- Select qualifier
  - with MUPIP EXTRACT, global specification, 84
  - with MUPIP REORG, 105
  - with MUPIP TRIGGER, 117
- Selection qualifiers, 150
- Sequence Number qualifiers, 150
- Set command, 109, 140
  - examples, 113
  - in replication, 226
  - journaling examples, 149
- SEtgd command
  - example, 50
  - in GDE, 50
- setting database characteristics, 109
- setting journaling characteristics, 140
- Shift command
  - in DSE, 317
- SHow command
  - template
    - example, 51
  - in GDE, 50
- Show command
  - in LKE, 257
- Show options
  - display journaling information, 157

## Index

- Show qualifier
  - with MUPIP JOURNAL, 157
  - with MUPIP REPLICATE, 229, 230
- Showbacklog qualifier
  - with MUPIP REPLICATE, 241
- Shutdown qualifier
  - with MUPIP REPLICATE, 237
- Sibling qualifier
  - in DSE, Find, 307
- Seize qualifier
  - in DSE, 285
- Since qualifier
  - with MUPIP BACKUP, 76
  - with MUPIP JOURNAL, 162
- Size qualifier, 114
  - with MUPIP REPLICATE, 229
- Snapshot files
  - with MUPIP INTEG, 93
- Source qualifier
  - with MUPIP REPLICATE, 230
- Source Server, 186
  - changing logs, 239, 240
  - checking progress, 241
  - deactivating, 238
  - enable logging, disable logging, 240
  - identify, 230
  - log interval setting , 231
  - logging, 231
  - Passive
    - activating, 237, 238, 238, 238, 239
    - shut down, 237
    - specifying compression level, 235
    - specifying Journal Pool size, 231
    - start in passive mode, 231
    - starting, 230
    - starting, active mode, 230
    - stopping, 241, 241, 241
    - timeout specifying, 237
- Spanning Nodes
  - special subscript, 274
- Spawn
  - in LKE, 261
- Spawn command
  - in DSE, 318
- Star qualifier
  - in DSE, 282
  - in DSE, Range, 313
- Start qualifier
  - with MUPIP REPLICATE, 230
- Starting the Receiver Server

## Index

- setting the logging interval, 244
- Starting the Source Server
  - enabling TLS/SSL replication, 236, 236, 237, 246
- Statistics show option, 158
- Statslog qualifier
  - with MUPIP REPLICATE, 240
- Stdin qualifier
  - with MUPIP LOAD, 101
- Stdnullcoll qualifier, 56
- Stdout qualifier
  - with MUPIP EXTRACT, redirect to stdout, 85
- Stop qualifier, 116
- Stopsourcefilter qualifier
  - with MUPIP REPLICATE, 239, 244
- Strm\_num qualifier
  - in DSE, 295
- Strm\_reg\_seqno qualifier
  - in DSE, 295
- Subscript qualifier
  - with MUPIP INTEG, 94
- Supplementary qualifier
  - Instance\_create qualifier, 228
  - with MUPIP REPLICATE, 228
- SYNC\_IO
  - with Newjnlfiles, MUPIP BACKUP, 75
- Sync\_io option, 147

## T

- Template command
  - example, 53
  - maintains sets of region and segment , 53
- Time qualifiers, 150
- Timeout qualifier
  - with MUPIP REPLICATE, 241
  - with MUPIP REPLICATE, shutdown qualifier, 237
- Timers\_pending qualifier
  - in DSE, 295
- TLS Replication
  - , 194
- TLSID qualifier
  - Starting the Receiver Server, 236, 246
- Tn qualifier
  - in DSE, 289
- Tn\_reset qualifier
  - with MUPIP INTEG, 95
- To qualifier
  - in DSE, Range, 313
- Total\_blks qualifier
  - in DSE, 295
- Transaction qualifier

- with MUPIP BACKUP -BYTESREAM, 77
- with MUPIP INTEG, 95
- with MUPIP JOURNAL, 168

- Trigger command, 116
  - examples, 118
- Triggerfile qualifier
  - with MUPIP TRIGGER, 116

## Triggers

- add, 118
- delete, 119
- modify, 118

- Trigger\_flush qualifier
  - in DSE, 295

- Truncate qualifier
  - with MUPIP REORG, 106

## U

- Update Process, 186
  - starting, 246
  - stopping, 248
- Updateonly qualifier, 246
  - with MUPIP REPLICATE , 248
- Updateresync qualifier
  - with MUPIP REPLICATE , 244
- Updnotok
  - specifying, 235
- Updnotok qualifier
  - with MUPIP REPLICATE, 235
- Updok
  - specifying, 234
- Updok qualifier
  - with MUPIP REPLICATE, 234
- Updproc qualifier
  - in DSE, 303
- Upgrade command, 120
  - examples, 120
- Upper qualifier
  - in DSE, Range, 314
- User qualifier
  - with MUPIP JOURNAL, 168

## V

- V4 to V5 upgrade, 120
- Value qualifier
  - with MUPIP REPLICATE, 229
- Verbose qualifier
  - with MUPIP JOURNAL, 166
- Verify command
  - in GDE, 53
- Verify qualifier

- in DSE, 298
- with MUPIP JOURNAL, 159

Version qualifier

- in DSE, Remove, 315
- in DSE, Restore, 316
- with MUPIP DOWNGRADE, 80
- with MUPIP SET, 113

**W**

Wait qualifier

- in LKE, 259

Wcinit command

- in DSE, 318

Wcinit qualifier

- in DSE, 286

Writes\_per\_flush qualifier

- in DSE, 295

**Y**

Yield\_limit journal option, 148

**Z**

ZWR, 83

ZWR qualifier

- in DSE, 303